UNIVERSITY OF JOENSUU
COMPUTER SCIENCE
DISSERTATIONS XIV

MATTI TEDRE

THE DEVELOPMENT OF COMPUTER SCIENCE

A SOCIOCULTURAL PERSPECTIVE

ACADEMIC DISSERTATION

To be presented, with the permission of the Faculty of Science of the University of Joensuu, for public criticism in Louhela Auditorium of the Science Park, Länsikatu 15, Joensuu, on October 20th, 2006, at 12 noon.

UNIVERSITY OF JOENSUU
2006

Julkaisija	Joensuun yliopisto Tietojenkäsittelytieteen ja tilastotieteen laitos
Publisher	University of Joensuu Department of Computer Science and Statistics
Editor	Erkki Sutinen
Vaihdot	Joensuun yliopiston kirjasto/Vaihdot PL 107, 80101 Joensuu Puh. 013-251 2677, fax 013-251 2691 email: vaihdot@joensuu.fi
Exchanges	Joensuu University Library/Exchanges P.O. Box 107, FI-80101 Joensuu, FINLAND Tel. +358-13-251 2677, fax +358-13-251 2691 email: vaihdot@joensuu.fi
Myynti	Joensuun yliopiston kirjasto/Julkaisujen myynti PL 107, 80101 Joensuu Puh. 013-251 4509, fax 013-251 2691 email: joepub@joensuu.fi
Sales	Joensuu University Library/Sales of publications P.O. Box 107, FI-80101 Joensuu, FINLAND Tel. +358-13-251 4509, fax +358-13-251 2691 email: joepub@joensuu.fi

ISBN 952-458-866-8 (paperback)

ISSN 1238-6944 (paperback)

ISBN 952-458-867-6 (PDF)

ISSN 1795-7931 (PDF)

Computing Reviews (1998) Classification: K.m

Yliopistopaino

Joensuu 2006

The Development of Computer Science: A Sociocultural Perspective

Matti Tedre

Department of Computer Science and Statistics
University of Joensuu
P.O. Box 111, FI-80101 Joensuu, FINLAND
matti.tedre@cs.joensuu.fi

University of Joensuu, Computer Science, Dissertations XIV
Joensuu, 2006, 502 pages
ISBN 952-458-866-8 (paperback)
ISSN 1238-6944 (paperback)
ISBN 952-458-867-6 (PDF)
ISSN 1795-7931 (PDF)

Abstract

Computer science is a broad discipline, and computer scientists often disagree about the content, form, and practices of the discipline. The processes through which computer scientists create, maintain, and modify knowledge in computer science—processes which often are eclectic and anarchistic—are well researched, but so far there is no consensus on whether studies of such processes belong to the field of computer science or not.

In this thesis the sociocultural formation of computer science and computing technology is analyzed. It is asked if there is a need to extend computer science with meta-knowledge derived from perspectives from disciplines such as sociology, history, anthropology, and philosophy.

Based on a selection of science and technology studies and case studies from the history of computing, an argument is made that understanding the social processes that create and maintain computer science is an important part of understanding computer science. An outlook on *social studies of computer science* is presented, and it is suggested that it should be acknowledged and included in the ACM Computing Classification Systems as class K.9.

Keywords: social studies of computer science; social issues; meta-knowledge in computer science

Supervisors

Professor PhD Erkki Sutinen
Department of Computer Science and Statistics
University of Joensuu, Finland

Director of Research Dr. Esko Kähkönen
International Multidisciplinary PhD Studies in Educational Technology
University of Joensuu, Finland

Associate Professor PhD Piet Kommers
Department of Behavioral Sciences
University of Twente, The Netherlands

Reviewers

Professor Dr. Tech Jari Multisilta
Department of Information Technology
Tampere University of Technology in Pori, Finland

Professor PhD Pierluigi Crescenzi
Department of Systems and Computer Science
University of Florence, Italy

Opponent

Professor Dr. Johannes C. Cronje
Department of Curriculum Studies
University of Pretoria, South Africa

Language editor

Justus Randolph
Department of Computer Science and Statistics
University of Joensuu, Finland

Acknowledgements

I wish to express my gratitude to my advisors Erkki Sutinen, Esko Kähkönen, and Piet Kommers—who gave me the opportunity to freely do my research and whose ideas were indispensable for shaping this thesis. I thank the language editor of this thesis, Justus Randolph, for his countless insightful comments on my arguments. His insistence on clarity had a substantial impact on the style of this thesis. I thank the reviewers of this thesis, Jari Multisilta and Pierluigi Crescenzi, for their comments.

I also thank the people with whom I have had the honor to write and publish: Marcus Duveskog, Ron Eglash, Pasi J. Eronen, Teppo Eskelinen, Carolina Islas Sedano, Minna Kamppuri, Jyri Kemppainen, Markku Tukiainen, and Mikko Vesisenaho. I thank everyone who has read and commented my thesis. I thank my friends and colleagues at the department of computer science and statistics, University of Joensuu. And, of course, I thank my friends and relatives for being there for me.

This work was funded by

(1) the Department of Computer Science and Statistics at the University of Joensuu, Finland

(2) the East Finland Graduate School in Computer Science and Engineering (*ECSE*)

(3) the National Institute for International Education (*NIIED*) of Korean government

(4) the Centre for International Mobility (*CIMO*) of Finnish Ministry of Education.

This work was written with OpenOffice 2.0 under Gentoo Linux, and the illustrations were made with DIA 0.94.

Joensuu, September 29, 2006,

Matti Tedre

creative commons

COMMONS DEED

Attribution-NonCommercial-NoDerivs 2.5

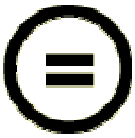
You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.



No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

Table of Contents

1.INTRODUCTION.....1

1.1.Background.....2

Computation.....5

The Separation Between Science and Humanities.....7

1.2.Research Questions.....10

1.3.Methodology, Literature, and Sources.....11

Credibility of Theoretical Research.....11

Methodology.....13

Methods.....15

Choice of Literature.....17

The Literature in Chapter Two.....17

Sources in Chapter Three.....20

1.4.Conventions Used in This Thesis.....23

1.5.Limitations and Assumptions.....25

Sociocultural Matters.....25

The Scope of This Thesis.....28

2.INTERPRETATIONS OF SCIENCE AND TECHNOLOGY.....31

2.1.Some Basic Questions About Science.....34

How Does One Come to Know Facts?.....40

Intuition of Scientific Knowledge.....41

Early Schools of Thought.....42

The Problems of Positivism.....44

Logical Truths Do Not Carry Information.....47

Inductivism.....48

Problems of Inductivism.....49

Popper's Falsificationism.....53

A Single Observation Can Falsify a Theory.....54

Progress in Science—The Falsificationist Viewpoint.....57

Criticism of Falsificationism.....59

Science as a Contract: Thomas Kuhn's Theory.....64

Some Terminology Introduced by Kuhn.....66

Scientific Progress According to Kuhn.....68

The Characteristics of a Paradigm.....70

Scientific Problems According to Kuhn.....74

The Foci And Limits of Normal Science.....75

Revolutions in Science.....79

Problems With Kuhn's Theory.....81

Pluralism in Science.....85

Freedom of Choice: Paul Feyerabend's Anarchistic Theory....89
The Anarchistic Theory of Science....90
The Critique of Feyerabend....92
Logic Cannot Explain Everything....94
Values in Science....95
The Worst Enemy of Science?....97
2.2.Modern Approaches to the Society-Technology Relationship....99
Sticking Points Between Realism and Constructionism....103
Not Everything Is a Social Construct....107
Sticking Point 1: Contingency....109
Contingency Thesis And the Definition of Successful....110
Inevitability, Contingency, And "The Mangle"....114
Sticking Point 2: Nominalism....117
Ontology in the Field of Computing....119
Brute Facts and Institutional Facts....120
Searle on Objectivity and Subjectivity....122
The Ontology of Algorithms....124
Consequences of Different Positions to Nominalism....125
Sticking Point 3: Explanations of Stability....127
Sticking Points: A Summary....133
Technological Determinism....135
Technological Momentum....138
Is Technology Value-Free?....140
Do Machines Make History?....145
Technological Progress is Immeasurable....146
The Social Shaping of Computers....147
2.3.Intermission....152
The Realist Camp....154
The Constructionist Camp....155
Middle Ground....157
3.THE DEVELOPMENT OF COMPUTING AS A DISCIPLINE.161
3.1.What Are Computational Instruments?....165
Problems With the Definition of Computational Instruments....172
3.2.What Is a Problem in Computing?....175
Different Views on the Concept of Problem....176
Three Classes of Problems....177
Authentic and Artificial Problems....179
Open and Closed Aspects of Problems....181
Open and Closed Problems in Different Problem Fields....183
Problems in Computer Science....185
Computer-Scientist-as-Bricoleur....187
Should Computer Science Borrow the Definition of Problem from Mathematics?....189
What Is a Problem in Computing: Summary....191

3.3. The Creation of Modern Computing....192
The Newton-Maxwell Gap: Before 1950....196
Analog Computing....197
Cultural Context....200
Crossing the Newton-Maxwell Gap....203
Number Crunchers....208
The Origins of Digital Computer Technology....209
Contingencies Surrounding ENIAC....211
The Birth of Business Computing: LEO I....216
The Origins of the Verb to Program....218
Summary: The Context of the Birth of Electronic Computing....219
Early Academic Computing....221
Early Entrants to the Field of Computing....221
MIT....223
Harvard....224
University of Pennsylvania....226
Columbia University....227
Princeton....228
The Birth of Programming Languages....232
The First Compiler....232
Computer Generations....233
Real-Time Computers Offer New Prospects....235
Normal Science or Pre-Science?....237
A Level Up In Abstraction: Fortran....237
Was fortran a Natural Step or a Contingent Event?....239
Algol: The Ideal Language....242
Why Did fortran Do Better Than algol?....244
Significant Factors in Language Development....247
Section Overview....250
3.4. The Creation of a Discipline....255
Struggling for Status....258
Who Is a Computer Professional?....259
Early Definitions....260
The Art and Science of Processing Information....263
The Official Birth of Computer Science....266
Shaping the Public Image of Computing....270
A Concern Over Dogmatism in Computing....273
Is the Focus the Machine (Computer) or the Phenomenon (Information)? .275
Legitimating Number Crunching as a Science....278
The Tug of War Between the Theoretical and Practical....283
Software Engineering....286
The Focus Turns to Programming....288
Separation from Mathematics....293
Emerging Interdisciplinarity....296
Shifts in the User Base, Status, and Content of Computing....297

Societal Conscience Awakens....299
The Limits of the Model Set by Turing and von Neumann....300
The Complexity of Computer Systems....305
The Sources of Complexity....309
The Semantic Gap....311
Complexity on the Language Level....313
Mastering Complexity....316
Artificial Intelligence....318
How to Bridge the Semantic Gaps?....319
Coming of Age....321
Recent Definitions....325
What Can Be Automated?....326
The Denning Report....329
A New Era of Growth....337
What's In a Name? (Part I)....340
A New Species Among The Sciences....346
Computational Science....347
What's in a Name? (Part II)....349
Research in the Discipline of Computing....352
Methodology in Computing Curricula....352
Research Approaches and Methods in Computer Science....354
Eclecticism and Opportunism....359
Meta-Research in Computer Science....360
Normative Arguments Cannot Be Based on Empirical Results....363
Section Overview....366

4.SOCIAL STUDIES OF COMPUTER SCIENCE....371

4.1.Computer Science: an Efficient Anarchy....373
Research in Computer Science....375
Scientific Statements in Computer Science....376
Social Constructionism in Computer Science....378
Anarchism in Computer Science....379
The Outcomes of Anarchism....381
4.2.Approaches to Social Studies of Computer Science....384
The Contribution of Social Studies of Computer Science....386
Social Studies of Computing....389
Three Sources of Information....390
The Sociohistorical Context of Computer Science....392
Ethnomethodology....396
Ethnographic Methods....400
Focus on Cases....403
Evaluation of Studies of Social Reality....404
Measures of Research....406
4.3.What Makes a Study Computer Science?....411
What Is the Relationship Between a Science And Its Tool?....411

To Which Field Does Social Studies of Computer Science Belong?.....415
4.4.Discussion.....418
On Normative Accounts of Computer Science.....419
On Descriptive Accounts in Computer Science.....420
"Practical" Normative Statements in Computer Science.....422
The Mangle in Computer Science.....423
Three Positions on Public Debate.....424
The Mangle in Practice.....427
The Disciplinary Implications of Social Studies of Computer Science.....428
An Essential Part of Mature Computer Science.....431
5.CONCLUSION.....435
The Philosophy of Computer Science.....436
Contingencies Surrounding Computing.....438
The Stored-Program Paradigm.....439
Technological Momentum.....440
Technological Momentum in Computer Science.....441
The Mangle of Practice.....442
Computer Science.....444
Proofs and Assertions.....447
What Should Be Automated?.....449
An Efficient Anarchy.....449
Constructionism and Intersubjectivism.....452
Computer Science in Society.....453
Social Studies of Computer Science.....455
The Promise of Social Studies of Computer Science.....457
Epilogue.....461

Index of Figures

Figure 1: Computation And Related Concepts in This Thesis.....7
Figure 2: The Research Cycle in This Research.....14
Figure 3: Internal Conceptual Coherence in a Thesis.....26
Figure 4: External Conceptual Coherence in a Thesis.....27
Figure 5: Differing Views of Facts.....46
Figure 6: Kuhn's Model of Scientific Progress.....68
Figure 7: Core Technologies of Computing.....69
Figure 8: Windows to Science.....153
Figure 9: Requirements for Computational Instruments.....172
Figure 10: Dimensions of Problems as Continua.....182
Figure 11: Some Prominent People in the Development of Early Computers.....197
Figure 12: Interdisciplinarity in Early Computing Technology.....199
Figure 13: Aspects of Culture Encouraging the Development of Early Computers. .203

Figure 14: The Influence of U.S. Military in the Early Computers.....207
Figure 15: Aspects That the World War II Brought Together.....211
Figure 16: The Context of Early Electronic Computers.....220
Figure 17: Some Characteristics of Five American Universities in the 1940s.....230
Figure 18: Timeline of Computer “Generations”.....233
Figure 19: Timeline of Programming Languages.....246
Figure 20: Some Factors in Language Development.....248
Figure 21: CS Topics Grouped According to Their Containment in CS.....269
Figure 22: Abstraction Layers at the Machine Level.....306
Figure 23: Abstraction Layers on a Language Level.....307
Figure 24: Coarse Abstraction Levels on Network Scale.....307
Figure 25: Singular Processes p1..p5 and a Class p(f(x)) of Processes.....329
Figure 26: Four Fundamental Questions in Human-Centered Computing.....336
Figure 27: Research Approaches in Computing Disciplines.....355
Figure 28: Research Methods in Computing Disciplines.....356
Figure 29: Narrow and Broad Interpretations of Computer Science.....387
Figure 30: Correspondence and Coherence in Research.....407

Index of Tables

Table 1: Different Sorts of Facts.....123
Table 2: Examples of Problem Classes.....181
Table 3: Examples of Auxiliary and Main Disciplines.....414
Table 4: Top Two K.* Sublevels in ACM CCS [1998].....433

Abbreviations

ABC.....The Atanasoff-Berry Computer
ACM.....Association for Computing Machinery
AFIPS.....American Federation of Information Processing Societies, Inc.
AI.....Artificial Intelligence
AIEE.....American Institute of Electrical Engineers
ALGOL.....Algorithmic Language
ALU.....Arithmetic Logic Unit
ANSI.....American National Standards Institute
APL.....A Programming Language
BINAC.....Binary Automatic Computer
BNF.....Backus-Naur Form (formerly Backus Normal Form)
CACM.....Communications of the ACM
CC2001.....ACM/IEEE Curricula recommendations (2001)
CHI.....Computer-Human Interaction
CISC.....Complex Instruction Set Computer

CMM.....Capability Maturity Models
COBOL.....Common Business Oriented Language
CODASYL.....Conference on Data Systems Languages
COSERS.....(NSF) Computer Science and Engineering Research Study
CPU.....Central Processing Unit
CS.....Computer Science
CS&E.....Computer Science and Engineering
CSTB.....Computer Science and Technology Board (of NRC)
DNA.....Deoxyribonucleic Acid
DPMA.....Data Processing Management Association
DUO.....Datatron Users Organization
ECC.....Electronic Control Company
EDSAC.....Electronic Delay Storage Automatic Calculator
Educom.....Inter-University Communications Council
EDVAC.....Electronic Discrete Variable Automatic Computer
EMCC.....Eckert-Mauchly Computer Corporation
ENIAC.....Electronic Numerical Integrator and Computer
ERA.....Engineering Research Associates
FORTRAN.....Formula Translating System
FPU.....Floating Point Unit
GAMM.....Gesellschaft für angewandte Mathematik und Mechanik
GNU.....GNU's Not Unix
HCI.....Human-Computer Interaction
HIV.....Human Immunodeficiency Virus
IAS.....(Princeton's) Institute for Advanced Study
IBM.....International Business Machines Corporation
ICT.....Information and Communication Technology
IEEE.....Institute of Electrical and Electronics Engineers
IFIP.....International Federation for Information Processing
IRE.....Institute of Radio Engineers
IS.....Information Systems
JACM.....Journal of the ACM
LEO.....Lyons Electronic Office Computer
LISP.....List Processing (language)
MIS.....Management Information Systems
MIT.....Massachusetts Institute of Technology
MSN.....The Microsoft Network
NCR.....National Cash Register Company
NRC.....National Research Council
NSA.....National Security Agency
NSF.....National Science Foundation
PC.....Personal Computer
PDA.....Personal Digital Assistant
PHP.....PHP: Hypertext Preprocessor
PL/I.....Programming Language One

PROLOG.....Programmation en Logique
Q.E.D.....Quod Erat Demonstrandum
RCA.....Radio Corporation of America
RISC.....Reduced Instruction Set Computer
SAGE.....Semi-Automatic Ground Environment
SETI.....Search for Extra-Terrestrial Intelligence
SIAM.....Society for Industrial and Applied Mathematics
SIG.....Special Interest Group
SIGCAS.....ACM Special Interest Group on Computers and Society
SIGCSE.....ACM Special Interest Group on Computer Science Education
SIGOPS.....ACM Special Interest Group on Operating Systems
SQL.....Structured Query Language
SSEC.....Selective Sequence Electronic Calculator
SSK.....Sociology of Scientific Knowledge
STS.....Science and Technology Studies
TC2.....(IFIP's) Technical Committee 2
TCL.....Tool Command Language
Tk.....A common graphical user interface toolkit
TSP.....Traveling Salesman's Problem
UNESCO.....United Nations Educational, Scientific and Cultural Organization
UNIVAC.....Universal Automatic Computer
USE.....UNIVAC Scientific Exchange
UTM.....Universal Turing Machine
WG2.1.....(IFIP's TC2) Working Group (on ALGOL)
VoIP.....Voice over Internet Protocol
ZISC.....Zero Instruction Set Computer

1. Introduction

This thesis is about a phenomenon called *computing*. Computing is a skill, and computing is a science; it entails actions, processes, theories, knowledge, workmanship, and artistry. Computing is a study, a method, a craft, a profession—and an academic discipline. Of all those different meanings of computing, this thesis is focused on the academic field of computing—*computer science* or *computing as a discipline*. Throughout the short disciplinary history of electronic digital computing, there has been a great variety of approaches, definitions, and outlooks on computing as a discipline. The arguments have sometimes been fierce, and the pace of the extension of the field has been unparalleled by any other science. This thesis offers one portrayal of the disciplinary history of computing.

This thesis is also about a phenomenon called *science*, or more specifically, science that is connected with computing. This thesis draws from the academic disciplines that are concerned with the phenomenon of science—the philosophy of science, the sociology of scientific knowledge, and other types of science and technology studies (STS). Scholars in the above-mentioned disciplines ask questions such as "What does it mean that something is science?", "Who defines what is science and what is not?", and "What is progress in science?". In this thesis, I discuss the different viewpoints of the questions above and weigh the pros and cons of those viewpoints. This is not a thesis about the philosophy of science, the sociology of scientific knowledge, or other types of science and technology studies, but science and technology studies are used in constructing a conceptual framework for the analysis of computing as a discipline.

This thesis is essentially about a phenomenon called *computer science*. One cannot begin an exploratory study about computer science by rigidly defining computer science—or there would not be much to study. Yet what one *can* do is study the different ways in which computer science has been characterized, analyze those characterizations, place them in a conceptual framework, and aim to understand the circumstances in which those characterizations emerged. By doing exactly that, I wish to set up an argument for extending computer science with a branch of study that aims at the disciplinary self-understanding of computer science—*social studies of computer science*.

1.1.Background[1]

This thesis deals with the sociocultural dimensions of computer science. During the last 60 years, researchers in the academic field of computing[2] have brought together a variety scientific disciplines and methodologies. The resulting interdisciplinary science, computer science, offers a variety of ways of modeling and explaining phenomena, such as computational models and algorithms. The growth of research efforts in computer science has been paralleled by the growth of the number of computing-related fields, such as computer engineering, computational science, electrical engineering, decision support systems, architectural design, and software engineering. Computers and information and communication technologies at large have also enabled a world of new socioeconomic and cultural concepts such as *e*-commerce, hacker ethics, the knowledge economy, discussion boards, and virtual communities to emerge.

There is a wealth of research on the sociocultural impact of computers and communication technologies—studies of the new economy, working culture, leisure time, new social formations, *e*-economy, and so forth[3]. Those studies most often focus on if and how new computing (and communication) technologies affect society and culture, and they also focus on the interactions of technology, society, and culture. There is less research on how sociocultural influences affect the development of theories, techniques, and instruments in computer science.

According to naïve technological determinism, technology develops independently of society[4]. Yet, science and technology studies (STS)[5] scholars generally reject naïve technological determinism. They often argue that the directions that research and development take are frequently decided on by coteries external to science, and

1 Parts of this introduction are from Tedre et al., 2006.

2 The term *computer science* was introduced long after the construction of the first fully electronic, digital, Turing-complete computers. The history of computer science as a discipline is discussed in Chapter Three.

3 Sociologist Manuel Castells has characterized various aspects of the *network society* in his trilogy *The Information Age: Economy, Society, and Culture* (Castells, 1996; Castells, 1997; Castells, 1998).

4 MacKenzie and Wajcman, 1999 :p.*xiv*.

5 Science and technology studies (STS) is often used as an umbrella term, which includes a number of research areas and approaches that concern science and technology: for example, the philosophy of science, studies of the social construction of technology (see Kline and Pinch, 1999), the sociology of scientific knowledge (SSK) (see Barnes et al., 1996), and the history of science.

that technological decisions are often based on, for instance, economic, political, or ideological arguments rather than technological arguments[6].

Several motivations can be attributed, for example, to the development of GNU/Linux and its introduction into use[7]. Arguably, GNU/Linux is advanced (a technical motivation), it is free of initial investment (an economical motivation), and its roots are in hacker ethics and the free software movement (ideological and social motivations). Also, sometimes it can emphasize a cultural or political message (e.g., IMPI Linux in South Africa has its roots in the concerns of "digital colonialism", and RedFlag Linux in China has its roots in government support for an independent operating system).

The impetus for studies of the connections between technology, academy, institutions, sciences, social milieux, human practices, economical concerns, agenda, ideologies, cultures, politics, arts, and other technological, theoretical, and human aspects of the world, arose in the wake of the constructionism of the 1960s. For instance, the strong programme in the sociology of scientific knowledge adopted the view that all beliefs, including scientific ones, are influenced by their sociocultural surroundings[8]; the philosophy of science took a new, social constructionist turn[9]; the universalist, positivist nature of mathematics was undermined[10]; historians of science and technology rejected inevitabilism and determinism[11]; and in the 1980s a new field called *science and technology studies*, which focuses on the social construction of science and technology, was formed[12].

Scientists in a number of disciplines have augmented their disciplinary understanding by exhaustive research on the methods, motives, and stakeholders' roles in their respective fields. Science and technology have been the subject of investigation of philosophers such as Karl Popper, Thomas Kuhn, Paul Feyerabend, Martin Heidegger, José Ortega y Gasset, and Imre Lakatos; of sociologists such as David Bloor,

6 See, e.g., Bijker and Law, 1992; MacKenzie and Wajcman, 1999 ; Smith and Marx, 1994; Pinch & Bijker, 1987; Bijker et al., 1987.

7 This GNU/Linux example is from Tedre et al., 2006.

8 One of the original works on the strong programme in the sociology of scientific knowledge is David Bloor's *Knowledge and Social Imagery* (Bloor, 1976).

9 Kuhn, 1996 (orig. 1962)

10 Lakatos, 1976; Barnes et al., 1996

11 Hughes, 1983; Kuhn, 1996; Heilbroner, 1967; Marcuse, 1964

12 Two influential books on social construction of technology are MacKenzie and Wajcman, 1999 (1st edition was published 1985) and Bijker et al., 1987; other similar collections are Bijker and Law, 1992 and Smith and Marx, 1994.

Donald MacKenzie, Bruno Latour, and Barry Barnes; and of historians such as Thomas P. Hughes and Lewis Mumford. Those authors, among others, have offered a variety of interpretations of why scientists are doing things as they are, why disciplines have shaped as they have, and what kinds of interconnections there are between disciplines, technologies, individuals, institutions, and other influential actors. Those explanations, or descriptions, are called *descriptive accounts of science*.

In addition, each new turn in the disciplinary self-understanding of a particular discipline has brought with it changes in prescriptions of how scientists in that discipline should work. For instance, Karl Popper's refutation of logical positivism[13] reformulated the conception of good scientific practice, and Thomas Kuhn's work gave impetus to the science wars[14]. New viewpoints of how science should be done have influenced, for instance, the methodologies, ethics, and epistemologies of science. Those prescriptions, or recommendations, are called *normative accounts of science*.

Although computer science is an innately interdisciplinary discipline, and although computer science is used as a tool for a variety of disciplines, there is uncertainty whether research that focuses on computer science and computer scientists belongs to the field of computer science. Even though there are an increasing number of studies that might be characterized as social studies of computer science, those studies are not clearly recognized as computer science. For instance, meta-research on computer science does not have a clear place in the ACM classification system for computing research[15]. In this thesis I argue that insight into the sociocultural aspects of the creation, maintenance, and modification of computer science (as a theoretical, conceptual, practical, and technical framework) is an essential part of computer science and should be included in the charter of computer science itself.

13 Popper, 1959 (orig. 1935)

14 See Kuhn, 1996 (orig. 1962). Much of the 1990s' intellectual scene was characterized by the debate between *relativist* thinkers who criticized the objectivity of science and *realist* thinkers who defended the objectivity of science (see, e.g., Bucchi, 2004, especially Chapter 6). This debate was dubbed the *science wars*, and even though the debate has lost its media appeal, neither side has given up the dispute. Most people, like me, do not admit to belong to either group.

15 See `http://www.acm.org/class/1998/` (accessed September 27th, 2006).

My argument starts with an explication of a conceptual framework[16] for this thesis (Chapter Two). My conceptual framework is typical of science and technology studies, and it derives from a number of theories in science and technology studies. I use that framework in my analysis of reports about the emergence of the discipline of computing and in my analysis of the formation of the disciplinary identity of computer science (Chapter Three). Based on my interpretation of the formation of computer science, and on the argument that social studies of science should be a part of the project of science itself[17], I propose that social studies of computer science is an important part of computer science and I outline the disciplinary implications of my proposal (Chapter Four). Before my argument begins, however, I discuss the background of my research, specify my research questions and research methodology, outline the structure of this thesis, and explicate the limitations and assumptions underlying my thesis.

Computation

The term *computation* in this thesis does not refer to the dictionary definition[18], but rather to how computation (intuitively, albeit vaguely) is often understood among computer scientists—that is, the implicit or explicit execution of algorithms[19]. This notion shifts the focus to questions such as "What is an *algorithm*?" and "What does *executing an algorithm* mean?". *Algorithm* can be defined as a finite set of instructions that operate on a finite set of symbols and can be, at least theoretically, implemented on some mechanism[20]. Executing an algorithm entails simply following those instructions.

The term *algorithm* can be defined further by arguing that algorithms should be realizable. It can be argued that in order to be realizable, an algorithm has to be unambiguous[21]. It can also be argued that an algorithm has to be useful, so it *can* have in-

16 By *conceptual framework*, I refer to the terminology, concepts, models, and characterizations of processes that are used to explain and predict a certain phenomenon. As there is significant disagreement about even fundamental concepts such as *ontology*, my conceptual framework defines the language used in this thesis. This thesis does not have a *theoretical framework*, because I have not been able to find a single theory able to fully characterize computer science.

17 Barnes et al., 1996:p.*iix*.

18 To determine by mathematics, especially by numerical methods; to determine by the use of a computer (AHD, 2004) (see p. 161 of this thesis).

19 Scheutz, 2003

20 Scheutz, 2003

21 See Knuth, 1968:pp.5-6 about *algorithms*, *computational methods* (algorithms, which may lack finiteness), and *reactive processes* (nonterminating computational methods, which interact with their environment).

put and it *must* have output. In addition, algorithms can be required to finish in a finite time, or, stricter, that the execution of an algorithm finishes within some sensible time limits.

Note that this characterization of *computation* does not define the *mechanism* that does the computing. As long as the mechanism is realizable, it can be concrete or abstract, it can be real or imagined, and it can be natural or artificial. The mechanism can be based on gears, thinking, pneumatics, magic, or hamburgers—or it can be based on an electronic computer[22]. So, *computation* does not commit one to just computation with computers. One can study computation without any connection to any existing or future machinery. Matthias Scheutz noted that a characterization of computation as executing algorithms that are realizable, useful, and finite does not yet commit one as to what computations are about or what computations are supposed to achieve[23]. However, in practice computation is often connected with some practical realms and functions, such as information and automation.

Issues such as what computation is about, what computation is supposed to achieve, and other problems that computation brings about are discussed throughout this thesis. In this thesis the term *the study of computing* refers to academic or non-academic studies on computation and the immediate phenomena[24] surrounding them, such as mechanical implementations, users, theories, data, and information. The term *computer science* refers to the academic discipline concerned with computation and its surrounding phenomena (I analyze the problems with the term *computer science* in detail in Section 3.4). In Section 3.4 I also discuss the problems involved in making a distinction between *computer science*[25], *computing as a field*[26], and *computing as a discipline*[27].

There is a plethora of concepts, definitions, and topics as well as problems, controversies, and juxtapositions connected with computation. Figure 1 maps a subset of the computation-related concepts that are discussed in this thesis. The concepts in

22 cf. Dijkstra, 1987.

23 Scheutz, 2003

24 Note that the term *phenomenon* is used in at least two meanings. The first of these meanings is "an occurrence, circumstance, or fact that is perceptible by the senses." (AHD, 2004). The second of these meanings comes from Immanuel Kant, and it refers to an object as it is perceived by the senses, as opposed to a noumenon.

25 Newell et al., 1967

26 Denning et al., 2001:p.12.

27 Denning et al., 1989

Figure 1 range from theoretical to practical, from academic to industrial, from specific to general, and from computationally fundamental to computationally peripheral.

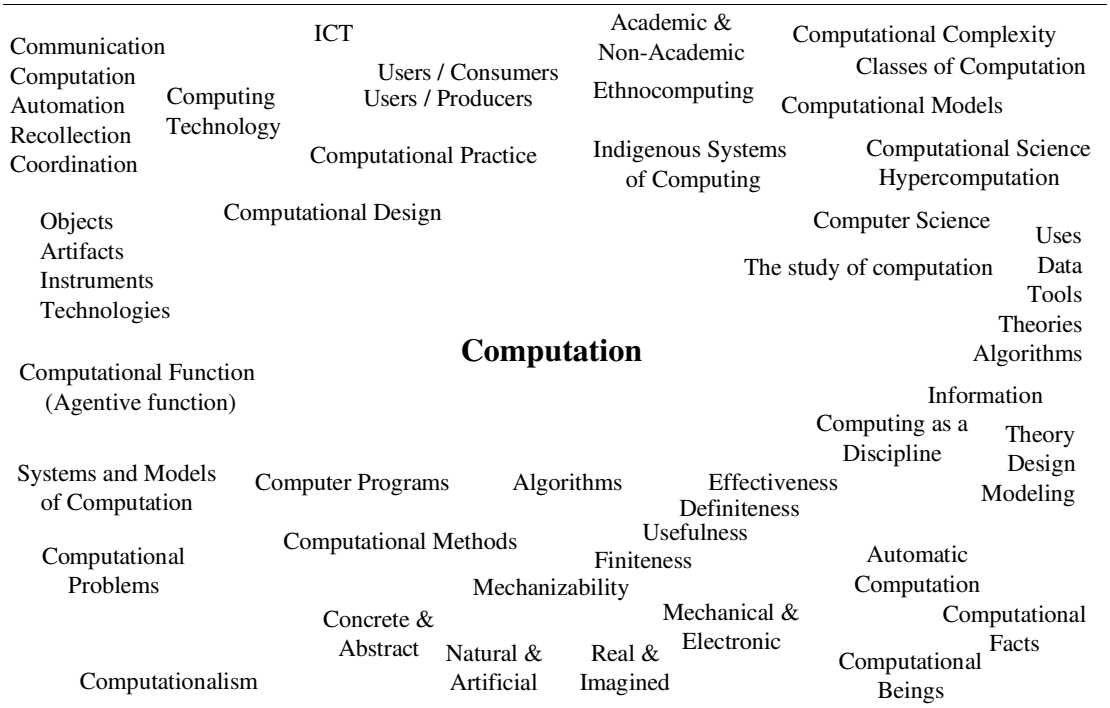


Figure 1: Computation And Related Concepts in This Thesis

Note that there are no connecting lines in Figure 1 because Figure 1 is *not* a semantic map, a theoretical hierarchic concept system, mind map, or any other kind of a *formal* representation of concepts and their relationships. Many related concepts are indeed close to each other because they have a close relationship—however, because a page of text is two-dimensional, many related concepts cannot be drawn close to each other. Perhaps one might think of Figure 1 as an organized snapshot that reflects one picture of the relationships between a number of concepts, but that does not restrict alternative organizations of concepts. After all, Figure 1 does not redefine the concepts that it presents. Figure 1 is not a portrayal of the field of computer science; Figure 1 is a portrayal of the computation-related topics discussed in this thesis.

The Separation Between Science and Humanities

As early as 1959, two influential commentators, C. Wright Mills and C. P. Snow, had already noted a significant lack of communication and understanding between

people from the humanities and social sciences and people from science and technology. That is, they noted that there is a gap between the factions who understand human issues and the factions who understand technical issues[28]. Mills argued that most people who use technological gadgets do not understand technology, and that those people who understand technological gadgets do not understand much else[29]. He argued that the scientists' work is centered around "Science Machines", which are operated by technicians, and controlled by economic and military people[30]. Snow noted the same problem, but the other way round—he argued that no more than one in ten highly educated humanities people are able to discuss, for example, mass and acceleration, which are the scientific equivalents of asking "Can you read?"[31].

The gap between technological knowledge and knowledge about human issues has been addressed by a number of computer scientists[32]. For instance, recently Ben Shneiderman claimed that linking the high-tech world more closely to the needs of people requires encouraging deeper understanding of human activities and relationships[33]. There is a large number of studies on the effects of computing on society, culture, institutions, academic disciplines, and so forth[34]. There is also research on computing from sociological, historical, anthropological, and philosophical points of view[35]. However, institutionally, computer science does not acknowledge the social studies of computer science as a branch of computer science. In the ACM Computing Classification System[36] (which is the classification system for almost all computer science publications) there is no category for research and meta-research about, for instance, how computer scientists create, maintain, disseminate, and reconstruct knowledge; how the proofs, debates, and refutations in computer science play out; how subjective assumptions, attitudes, and tacit knowledge affect computer science;

28 Mills, 1959; Snow, 1964 (orig. 1959). Snow's text has been widely quoted, but also widely criticized.
29 Mills, 1959:p.175.
30 Mills, 1959:p.16.
31 Snow, 1964:p.15.
32 See variants of this theme in, e.g., Cockton, 2004; Dertouzos, 2001; Johnson, 1998; Mahmood, 2002; Negroponte, 1995; Weiser & Brown, 1997.
33 Shneiderman, 2002:pp.3,15.
34 See, for instance, the references in Castells, 1996; Castells, 1997; Castells, 1998; Easton, 2006.
35 See, e.g., NSR Computer Science and Telecommunications Board, 1999; Campbell-Kelly & Aspray, 2004; Suchman, 1987; Smith, 1998, respectively.
36 See `http://www.acm.org/class/1998/` (accessed September 27th, 2006).

or even what computer science is[37]. Social studies of computer science is not an acknowledged branch of computer science.

Yet, computer science is done by people. No matter what ontological or epistemological standpoints one takes, one cannot escape the fact that science as an enterprise is run by scientists and all scientific statements are made by scientists. The relationship between science and social phenomena is an issue that has been debated extensively in fields such as physics and mathematics[38], but not so in the field of computer science. It has not been established, however, if understanding how computer scientists work would actually be beneficial to computer science, or if such studies might serve only sociological, historical, anthropological, or philosophical purposes[39]. In this thesis I deal with the question of whether computer science as a discipline can benefit from *social studies of computer science*.

37 With a few exceptions: K.2 (History of Computing), some topics in I.2 (Artificial Intelligence), *ethics* in K.4.1 (Computers and Society–Public Policy Issues) and K.7.4 (Professional Ethics), and–perhaps implicitly–in K.6 (Management of Computing and Information Systems).

38 The classical works in the sociology and philosophy of science concern mostly physics and mathematics; take, for instance, Popper, 1959; Kuhn, 1996; and Lakatos, 1976.

39 The importance of historical insight in computer science has been established–*the history of computer science* is a recognized part of computer science. My question is: Why is history considered to be informative about computer science while many other types of intellectual inquiry are not?

1.2.Research Questions

In order to determine what role social studies of computer science plays in the broader landscape of computer science, I needed to pose three essential questions. First, it was necessary to consider if the historical development of computer science has depended on sociocultural factors and to what extent different aspects of computer science are necessary or contingent parts of computer science. Second, it was necessary to investigate the concrete (technological), societal (historical, cultural, institutional, and political), and abstract (philosophical and theoretical) backgrounds of computing and ask which ones are within the focus and reach of current computer science. Third, it was necessary to ask what kinds of research can be considered to be computer science and what kinds of arguments support the view that understanding the sociocultural aspects of computer science is beneficial to computer science.

Because my research on the history of computer science showed that sociocultural factors indeed play an influential role in the construction of computer science, and that the benefits of sociocultural self-understanding of the discipline are warranted, I conceptualized a viewpoint of computer science that is sensitive to the sociocultural aspects of computer science. I also explored the implications of this viewpoint for computer science.

If condensed in three sentences, the research questions were:

1. Is there a need to broaden computer science with perspectives from disciplines such as sociology, history, anthropology, or philosophy?

2. If there is a need to broaden computer science, what kind of arguments support such an extension?

3. What consequences may a broad, socioculturally receptive view have on computer science?

These three research questions, especially questions 1 and 2, are intertwined. In the following section I describe how these three research questions were approached, and how the structure of this thesis relates to these research questions.

1.3.Methodology, Literature, and Sources

This is a theoretical dissertation. The purpose of theoretical research, James Day argued, is to challenge the grounds on which other acts of research take their meaning[40]. Day wrote, "*Since all research takes its impetus from theoretical suppositions, since every researcher is a theorist, the clear place of the theoretical dissertation lies in the work of reviewing and remaking the terms which govern what other researchers have done or will do*"[41]. My research is a critical synthesis of existing research and arguments about computer science, and the outcomes of my research are a characterization of computer science as a socially constructed discipline and an argument for extending computer science with some complementary viewpoints. In this section I discuss standards for judging theoretical research, methodological issues in theoretical research, the choice of literature and sources in this dissertation, and how my research questions relate to the structure of this thesis.

Credibility of Theoretical Research

As computer science is often considered to belong to the family of hard sciences, in the beginning of my research I attempted to qualify my research in terms of quantitative empirical research. I attempted to use terms such as *external validity* to refer to the applicability of my conclusions on a level more general than my research (i.e. generalizability), *construct validity* to refer to how the different parts of my research cohere, and *reliability* to refer to the degree to which, given the same readings, a number of independent authors would come to the same conclusions. But because in theoretical research—as opposed to quantitative empirical research—one does not systematically test a theory with data, using quantitative research terminology would be somewhat artificial or even misleading.

Next I attempted to describe my research in terms of qualitative exploratory research. In qualitative research, *credibility* may be a more appropriate criterion than *internal validity*, *transferability* may be preferred to *external validity*, *dependability* might be a more apt concept than *reliability*, and *confirmability* may be a better criterion than *objectivity*[42]. I attempted to use the terms *transferability* to refer to the

40 Day, 1993 in Nickerson, 1993
41 Day, 1993
42 Trochim, 2000; Guba & Lincoln, 1994 in Denzin and Lincoln, 1994

degree to which my research is limited to computer science and does not concern science at large, *dependability* to refer to the impact of my research on computer science if my argument is taken seriously, and *confirmability* to refer to the degree to which my arguments can be confirmed or contradicted by other researchers studying the same topic.

Using the qualitative terminology above, one could say that my research has limited transferability, as the aim of this dissertation is to generalize to computer science but not to generalize to science at large. That is, I make arguments about computer science, but not about all sciences. One could say that the dependability of this research is significant—I argue for an extension of the most commonly used taxonomy of computer science research with a category for meta-research on computer science. And one could say that the confirmability of this research is to be shown by other researchers who might adopt viewpoints that contradict those I describe in Chapter Two and who might analyze how well the literature and sources support those alternative viewpoints.

However, the qualitative research criteria may not be fully applicable to theoretical research either, as those concepts are related to exploratory research in which collected data is used to develop, for example, models, taxonomies, or theories. Those concepts have also been criticized as an attempt to parallel quantitative research criteria[43]. Although in a theoretical dissertation the selected theoretical literature and source texts could be understood as the "sample" or "data", that parallel is weak. In the end, a theoretical dissertation is simply different from an empirical dissertation. As M. David Merrill wrote, "*When one tries to force fit a theoretical dissertation into an empirical dissertation format the result is a misshapen monstrosity that pleases no one.*"[44].

There is, however, one term from the qualitative research tradition that is particularly apt for a theoretical dissertation: *credibility*. Egon Guba and Yvonna S. Lincoln argued that an author who works within the qualitative research tradition cannot utilize incontestable logic or indisputable evidence to force the reader to accept analyses or arguments[45]. An author can only hope that his or her argument is credible

43 Guba & Lincoln, 1994
44 Merrill, 2000:p.77.
45 Guba & Lincoln, 1994

and position is useful: Guba and Lincoln wrote, "We do ask the reader to suspend his or her disbelief until our argument is complete and can be judged as a whole."[46]. I ask the reader to be critical about my socioculturally focused interpretation of how the *status quo* in computer science has come to be, but still to judge my argument as a whole. In the end, the reader is the judge of the selection of my theoretical literature and sources, the credibility of my angle, the rigor and depth of my analyses, and the conclusions I draw.

James Day wrote that theoretical researcher needs to show *how* and *why* things have happened as they have, and to indicate *how*, and explain *why*, they might be appreciably improved when the position of the research is taken seriously[47]. This is how Day's guideline is implemented in my research: In Chapter Two I construct the conceptual framework used in this thesis and in Chapter Three I analyze computing as a discipline using the conceptual framework formed in Chapter Two. In Chapter Four I discuss the changes that should be made based on my findings, and sketch the implications to the field of computing if those propositions are taken seriously.

Methodology

The term *method* in this thesis refers to a means or manner of procedure for accomplishing something[48], like measuring the execution time of an algorithm for a given input, interviewing a group of people about an interesting phenomenon, or comparing the execution times and output sizes for a given input with two algorithms. The term *methodology* in this thesis refers to the principles and assumptions that underlie a set of methods[49]. For instance, the postpositivist methodology relies on realist ontology and dualist epistemology[50], and is incompatible with, for example, positive proofs. The term *research approach* in this thesis refers to a loose set of methods and techniques chosen for research as well as a conceptual and/or theoretical framework that defines the language of the research and how inferences and analyses are made. For instance, the term *formulative approach* has been used to refer to re-

46 Guba & Lincoln, 1994:p.108
47 Day, 1993
48 AHD, 2004
49 Note that sometimes *methodology* is understood strictly as the *study of methods*, but sometimes loosely as a *set of methods*(AHD, 2004).
50 Guba & Lincoln, 1994:p.110.

search that involves synthesizing and integrating information and then developing guidelines, models, or frameworks[51].

As this is a theoretical dissertation, my methodology is that of theoretical research. However, Chetan Bhatt noted that in a theoretical dissertation, there is likely to be overlap between the theory and methodology[52], and this thesis is not an exception. The conceptual framework in this thesis is built from the philosophical and sociological theories discussed in Chapter Two, and this conceptual framework is then used as a tool for the analysis of the history of computing in Chapter Three. In other words, the conceptual framework of Chapter Two is the lens through which the material in Chapter Three is viewed. However, that framework also guides my reading of the constituents of the framework itself.

Theoretical research is often hermeneutic—as the research goes on, the conceptual-theoretical framework of the research is developed, sources in the research are changed, new patterns are found, and the analyses of the sources are revised (see Figure 2)[53]. This research started with a number of historical accounts and characterizations of computer science, from which several sociocultural patterns arose. Those patterns were in accordance with some philosophical accounts of science, which led to the construction of an initial theoretical framework. Subsequent reading about the history of computer science, however, brought up more patterns of the creation of computer science, and the *theoretical framework*, which consisted of one philosophical account of science, diversified into a *conceptual framework*, which now consists of concepts from several sociological and philosophical accounts of science.

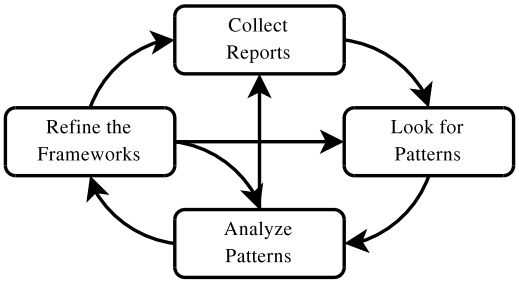


Figure 2: The Research Cycle in This Research

51 Morrison & George, 1995

52 Bhatt, 2004 in Seale, 2004

53 See especially Hans-Georg Gadamer's magnum opus *Truth and Method* (Gadamer, 1982, orig. 1960); the book *Philosophical Hermeneutics* offers a collection of Gadamer's essays (Gadamer, 1976).

This dissertation is an interpretation of the history of computer science in a certain conceptual framework. Choosing a different conceptual framework would definitely change the picture of computer science presented here. That is not, however, a thoroughly relativistic statement. The theoretical works that form the conceptual framework in Chapter Two are recognized and widely used in explanations of science and technology. Also the sample of sources in Chapter Three consists of well-recognized articles and books that have been central to the discussions of computer science in English-speaking countries during the period from 1945-present.

This thesis has characteristics of both deductive and inductive reasoning—it is difficult to say if hermeneutically oriented research is observation-driven or theory-driven[54]. The deductive character is obvious in the analysis of research material: I have a conceptual framework that I use to analyze patterns of change in computing. The inductive character is obvious in the refinement of the conceptual framework according to my analysis of computing.

Methods

In examining the sociocultural aspects of computer science, one could focus on, for instance, computer science as knowledge (abstract, intangible), computing technologies (concrete, tangible), or computer science as an activity (practical, intangible). However, technology has contributed to scientific development in the field of computer science perhaps more than in any other science. The nominally different spheres of science and technology have in fact been so interwoven throughout the history of computing, that in this thesis I often use the term *technoscience* to refer to both of them[55]. In this thesis computer science as an *activity* is coupled together with computer science and computer technology in order to create a multiperspectival view of computer science.

Because all three research questions address different matters, I separately discuss the methods that were used in this thesis for examining each of the three research questions.

---

54 cf. Bhatt, 2004

55 For instance, Donna Haraway (Haraway, 1999) has used the term *technoscience* instead of *science and technology*, because she has felt that the two have–recently, to say the least–become inseparable (MacKenzie and Wajcman, 1999).

> *Question 1:* Is there a need to broaden computer science with perspectives from disciplines such as sociology, history, anthropology, or philosophy?

To answer this question, I examined the development of electronic, digital computing and computer science as a discipline, beginning from the birth of the first electronic, digital, Turing-complete computers, and analyzed if this development included elements that fall outside the focus of the dominantly technologically oriented theoretical and conceptual frameworks of computing. This is reported in Chapter Three. I surveyed the existing social studies of computing and social studies of computer science and looked at their place in computer science; I discuss this in Chapter Four. In Chapter Four I also discuss what is needed to understand those elements, and especially if understanding those elements would benefit from concepts or theories from the humanities or social sciences[56].

> *Question 2:* If there is a need to broaden computer science, what kind of arguments support such an extension?

To answer this question, I considered arguments that have been used in other scientific disciplines to support similar extensions and analyzed the viability of those arguments in computer science. I considered both philosophical and practical arguments—both are necessary in a field as dynamic as computing. This is reported in Chapter Two. I analyzed and characterized computer science as a socially constructed discipline. This is reported in Chapter Three and Chapter Four. In addition, I considered different approaches in the humanities and social science, and analyzed the feasibility of those approaches in studies of computing. This is reported in Chapter Four.

> *Question 3:* What consequences may a broad, socioculturally receptive view have on computer science?

To answer this question I isolated a number of aspects of computing as a discipline, and analyzed the possible consequences and changes that the aforementioned ap-

56 There is overlap between the concepts *humanities* and *social science(s)*. The humanities are generally regarded as including studies in, for instance, languages and literature, the arts, history, and philosophy (Encyclopædia Britannica Online, 2004). The social sciences are generally regarded as including sociology, psychology, anthropology, economics, political science, and history (AHD, 2004). Both *humanities* and *social science* are referred to here in order to emphasize the generality of my research question. The reason why I foreground sociology, history, anthropology, and philosophy is that they are central to my argument, and they have been foregrounded in other studies on social construction of science too (e.g., Pickering, 1995:p.216).

proaches from the humanities and social sciences have on those aspects. This is reported in Chapter Four.

Choice of Literature

Similar to the concepts of reliability and validity, sampling terminology may not be appropriate in a theoretical dissertation. In a theoretical dissertation the choice of literature cannot be nailed down at the beginning of the research, but the researcher must be able to revise the literature at any point in the study. The fluidity of literature and sources is characteristic of a philosophical hermeneutic approach. It could still be said that the sample of literature in this thesis is neither a comprehensive sample nor a representative sample, but that one could interpret the choice of literature in this thesis to have the characteristics of *purposive sampling* (a.k.a. judgment sampling) and *convenience sampling*[57], for the reasons that follow.

The main body of literature in Chapter Two consists of the works of those philosophers of science that are today considered to be the key figures in the 1900s philosophy of science. Initially my theoretical framework consisted of the works central to the constructionist shift in the philosophy of science, but as my research on the disciplinary history of computer science showed inadequacies in that framework, I gradually extended my framework towards more modern accounts of scientific and technological change. I selected those accounts due to their visibility in contemporary discussion (such as in the journal *Social Studies of Science* and contemporary literature[58]). I describe my selection of sources for Chapter Three in more detail in the beginning of Chapter Three. It should be noted here, however, that in Chapter Three the articles that are available in electronic form have a better representation than articles that are hard to obtain, yet most of the journal archives in computer science are indeed available on-line.

The Literature in Chapter Two

The literature in Chapter Two consists of four main types. First, there are a number of original, classic books in contemporary (1900s) philosophy of science and modern

57 See *purposive sampling* in Bernard, 1995:p.95.

58 Pickering, Barnes, and Bloor have been visible in *Social Studies of Science*. Of contemporary literature, see, e.g., Hacking, 1999; MacKenzie and Wajcman, 1999 ; Chalmers, 1976 (1999 ed.); Rosenberg, 2000; Couvalis, 1997.

sociology, as well as a number of original, applauded books ("modern classics"[59]). Those are, in chronological order, Karl Popper's *The Logic of Scientific Discovery*[60], C. Wright Mills' *The Sociological Imagination*[61], Thomas Kuhn's *The Structure of Scientific Revolutions*[62], Berger and Luckmann's *The Social Construction of Reality*[63], Paul Feyerabend's *Against Method*[64], David Bloor's *Knowledge and Social Imagery*[65], Imre Lakatos' *Proofs and Refutations*[66], John Searle's *The Construction of Social Reality*[67], and Manuel Castells' *The Information Age*[68].

Second, there are a number of books that are original contributions or compilation works on the philosophy of science and other topics that concern the theme of this thesis, but that may not be very well-known to the readership of this thesis. Those works include, for instance, Lakatos and Musgrave's *Criticism And the Growth of Knowledge*[69], Pierre Duhem's *The Aim And Structure of Physical Theory*[70], W.v.O. Quine's *Word and Object*[71], Barnes, Bloor, and Henry's *Scientific Knowledge: a Sociological Analysis*[72], Bijker and Law's *Shaping Technology/Building Society*[73], Ian Hacking's *The Social Construction of What?*[74], Steve Fuller's *Kuhn vs. Popper*[75], Pierre Lévy's *Collective Intelligence*[76], MacKenzie and Wajcman's *The Social Shaping of Technology*[77], Luciano Floridi's *Philosophy of Computing: an Introduction*[78] and *The Blackwell Guide to the Philosophy of Computing and Information*[79], Brian

59 Because the term "modern classic" is used often, it is used here too, although the term indeed sounds like an oxymoron coined by impatient followers of some authors. The line between *classic*, *non-classic*, and *modern classic* works is a subjective one.
60 Popper, 1959 (orig. 1935)
61 Mills, 1959
62 Kuhn, 1996 (orig. 1962)
63 Berger & Luckmann, 1966
64 Feyerabend, 1993 (orig. 1975)
65 Bloor, 1976
66 Lakatos, 1976
67 Searle, 1996
68 Castells, 1996; Castells, 1997; Castells, 1998 (trilogy)
69 Lakatos & Musgrave, 1970
70 Duhem, 1977 (orig. 1914)
71 Quine, 1980
72 Barnes et al., 1996
73 Bijker and Law, 1992
74 Hacking, 1999
75 Fuller, 2003
76 Lévy, 1997
77 MacKenzie and Wajcman, 1999
78 Floridi, 1999
79 Floridi, 2004

Cantwell Smith's *On the Origin of Objects*[80], Andrew Pickering's *The Mangle of Practice*[81], and Steven Pinker's *The Blank Slate*[82].

Third, there are a number of classical, oft-quoted journal articles or book chapters, such as "Two Dogmas of Empiricism"[83], "The Mangle of Practice: Agency and Emergence in the Sociology of Science"[84], "Normal Science And Its Dangers"[85], "Consolations for the Specialist"[86], and "How to Defend Society Against Science"[87].

Fourth, there are some references to contemporary and near-contemporary discussion on the topics above by, for instance, Lewis Mumford, Greg Bamford, Philip Brey, Luciano Floridi, Donna Haraway, Robert Heilbroner, Ronald Kline, Trevor Pinch, Bruno Latour, Gary Thomas, Langdon Winner, Mario Bunge, Thomas Hughes, and Rob Kling[88], plus many other authors. The journal *Minds and Machines* deals predominantly with philosophical issues related to computing, and there are a number of recent articles from *Minds and Machines*.

In my work I do not shun the works of people once named "betrayers of the truth"[89] —Karl Popper and Thomas Kuhn—and "the worst enemy of science"[90] Paul Feyerabend. In this thesis, however, their theories are not considered as being opposed to science, but as serious criticisms of the conception of the science of their time. In this thesis Popper, Kuhn, and Feyerabend are understood as developers of scientific thinking—Popper wanted to correct a glaring misconception about the mechanisms of science, Kuhn presented a description of how science actually works, and Feyerabend wanted to free science from detrimental dogmatism. Because in a theoretical dissertation the conceptual framework is of major importance, and because I wish to keep this thesis self-contained for computer scientists, the review of and discussion about these theorists is lengthy.

---

80 Smith, 1998
81 Pickering, 1995
82 Pinker, 2002
83 Quine, 1980:pp.20-46. A solid critique of analytic *vs*. synthetic truths, and the dogma of reductionism.
84 Pickering, 1993. The "resistance" and "accommodation": a materialist contribution to social studies of science.
85 Popper, 1970. Popper's critique of Kuhn, where Popper conceded that there is an apparent weakness in falsificationism.
86 Feyerabend, 1970. Feyerabend's critique of Popper.
87 Feyerabend, 1975. On the reasons why the society must be liberated from scientific dogma.
88 Mumford, 1962; Bamford, 1993; Brey, 2003; Floridi, 2003; Haraway, 1999; Heilbroner, 1967; Kline and Pinch, 1999; Thomas, 1997; Winner, 1999; Bunge, 1979; Hughes, 1983; Kling, 1980
89 As John Horgan noted (Horgan, 1996:p.32.)
90 Horgan, 1996

Sources in Chapter Three

George Berkeley complained that philosophers are keen on creating their own problems that keep them busy afterwards: "*We have first rais'd a Dust, and then complain, we cannot see*"[91]. Although this thesis is a theoretical thesis, the content is not only theoretical. In Chapter Three I tie theoretical discussion to contemporary debates in the field of computer science. The literature in Chapter Three consists of four main types.

First, there are a large number of historical accounts of computing. Those are mainly (1) articles from the well-recognized journal *IEEE Annals of the History of Computing* and (2) history books focused on computing technology. The articles from the IEEE Annals of the History of Computing (*the Annals*) were chosen from the IEEE digital library if their abstracts led me to believe that the article dealt with sociocultural factors in computing. I read the abstracts in each issue of the Annals from 1992 to 2004 (those issues are available on-line). Note that the Annals has a large number of technically oriented articles that neither recognize nor deny a human viewpoint to the history of computing technology. The history books were written by recognized historians of computing such as Martin Campbell-Kelly, William Aspray[92], Michael Williams[93], Jean Sammet[94], Richard Wexelblat[95], Thomas Bergin, Richard Gibson[96], and Kenneth Flamm[97]. Note that the historical sources do not include primary sources, such as archival records.

Second, there are applauded, classical journal articles and books that computer scientists usually know by name and author, such as Knuth's *The Art of Computer Programming*[98] and *The Remaining Trouble Spots in ALGOL 60*[99], Brooks' *The Mythical Man-Month*[100], Simon's *The Sciences of the Artificial*[101], Gödel's *On Formally Unde-*

91 Berkeley, 1971 (orig.1734):p.5. Because of this inward-looking attitude, according to Luciano Floridi, philosophy interacts more with its own intellectual tradition than with the culture within which it develops (Floridi, 2003).
92 Campbell-Kelly & Aspray, 2004
93 Williams, 1985
94 Sammet, 1969
95 Wexelblat, 1981
96 Bergin & Gibson, 1996
97 Flamm, 1988
98 Knuth, 1997; Knuth, 1998; Knuth, 1998b (trilogy)
99 Knuth, 1967
100 Brooks, 1975
101 Simon, 1981

cidable Propositions of Principia Mathematica and Related Systems[102], von Neumann's *First Draft of a Report on the EDVAC*[103], Shannon's *The Mathematical Theory of Communication*[104], Turing's *On Computable Numbers, With an Application to the Entscheidungsproblem*[105] and *Computing Machinery and Intelligence*[106], Naur et al.'s *Report on the Algorithmic Language* ALGOL *60*[107], Knuth's *Theory and Practice*[108], Dijkstra's *GO TO Statement Considered Harmful*[109], Bush's *As We May Think*[110], Denning et al.'s *Computing as a Discipline*[111], Forsythe's *What to Do Till the Computer Scientist Comes*[112], Newell et al.'s *Computer Science*[113], and Wirth's *Program Development by Stepwise Refinement*[114]. Many of those articles triggered debates that had a significant impact on the development of computer science, and many of them belong to the "canons" of classical papers in computer science suggested by, for instance, Michael Eisenberg[115] as well as Judith Gal-Ezer and David Harel[116].

Third, there is a large number of lesser-known journal articles that are nonetheless important for this thesis. Those are mostly from *Communications of the ACM*, but there are also articles from journals such as *American Mathematical Monthly*, *Journal of the ACM*, *BIT*, *Datamation*, and *Theoretical Computer Science*. The emphasis on *Communications of the ACM* (*CACM*) is due to its long history (CACM was established in 1957) and its recognition as a forum for academic computer scientists to exchange results, knowledge, and opinions. Some of the most well-known debates in computer science have taken place on the pages of *CACM*: The ALGOL debate in the 1960s, the GO TO statement debate in the late 1960s and the following structured programming debate in the mid-1970s, computer science policies and human rights debates in the turn of the 1980s, and the computer science education debate throughout the publication's history.

102 Gödel, 1931
103 Neumann, 1945
104 Shannon, 1948
105 Turing, 1936
106 Turing, 1950
107 Naur et al., 1960
108 Knuth, 1991
109 Dijkstra, 1968
110 Bush, 1945
111 Denning et al., 1989
112 Forsythe, 1968
113 Newell et al., 1967
114 Wirth, 1971
115 Eisenberg, 2003
116 Gal-Ezer and Harel, 1998

There is a minimal number of conference articles by authoritative figures in computer science, such as Jonathan Grudin[117], Peter Wegner[118], John Backus[119], and a small number of conference articles that discuss contested or novel topics. Some articles, such as the IEEE/ACM computing curriculum 2001[120], are retrieved from the official web pages of different organizations. Finally, there are three online dictionaries used: (1) *The American Heritage Dictionary of English Language Online*, 2004 edition[121], (2) *Encyclopædia Britannica Online*[122], and (3) *Stanford Encyclopedia of Philosophy*[123]. Dictionaries are referred to only where explicitness about a definition of a word is necessary. In those parts of this thesis where "significant portions of [my] own [previous] copyrighted work"[124] are reused, references to my earlier publications are made.

117 Grudin, 1990, in which Grudin depicts "five levels of interface design".
118 Wegner, 1976, in which Wegner brought forth an early version of computing as design-modeling-theory.
119 Backus, 1959, in which Backus defined BNF.
120 Denning et al., 2001
121 AHD, 2004
122 Encyclopædia Britannica Online, 2004
123 Stanford Encyclopedia of Philosophy
124 `http://www.acm.org/pubs/plagiarism%20policy.html` (accessed September 27th, 2006)

1.4.Conventions Used in This Thesis

Following the tradition of, for instance, Popper, Kuhn, Feyerabend, and Lakatos, references in this thesis are in footnotes[125]. In references to books, the reference is to the pages indicated, for instance, "Mills, 1959:pp.19-21". In some places definite parts of texts need to be referenced, such as "Wittgenstein, 1986:last clause". In a few book references no page numbers are indicated: In those cases the reference is to the whole book and refers to the spirit of the book, such as "Popper, 1959", which refers to falsificationism at large. The year in the reference indicates the publication year of that particular edition, but whenever it is necessary, also the original publication date is added: for example, "Snow, 1964 (orig. 1959)" refers to the second edition of Snow's 1959 book. In the bibliography, the original publication dates are in brackets[126]. Journal and conference articles as well as chapters in compilation works are referred to without page numbers. The names of well-known authors are in their commonly used forms, that is, instead of "Charles Wright Mills", "C. Wright Mills" is used, and instead of "Charles Percy Snow", "C.P. Snow" is used.

In some cases, short verbatim quotations are italicized in quotation marks, followed by a footnote indicating the source ("*this is a direct quotation*"[127]). Longer verbatim quotations are italicized and framed as a separate paragraph, and in the second half of Chapter Three, quotations are accompanied with a year of the publication in brackets [1976] for the sake of chronological clarity. To keep the footnote numbers within three digits, footnote numbering starts from one in each chapter.

The terms *dissertation* and *thesis* are used interchangeably. Whenever third-person pronouns are used, they are always of the form *he or she*, *him or her*, or *himself or herself*, except for some verbatim quotations. Established terms such as the *Traveling Salesman's Problem* are used in their conventional forms, and no gender bias is assumed. The singular subject *I* refers to the author. The plural subject *we* is used only in verbatim quotations—in those quotations it usually refers to people in general (for instance, "*What values shall we choose to probe the sciences of today?*"[128]).

---

125 See, e.g., Popper, 1959; Kuhn, 1996; Feyerabend, 1993; Lakatos, 1976.

126 e.g., Kuhn, Thomas (1996 [1962]) *The Structure of Scientific Revolutions* (3rd edition). The University of Chicago Press: Chicago, USA.

127 This footnote would indicate the source of the quotation above.

128 Feyerabend, 1970

Common programming language names that are in all capitals and that are acronyms, such as FORTRAN, are written in small capitals (FORTRAN), as this is an oft-used convention[129], and this convention is arguably less distracting than using large capitals. Also some institutions are written in small capitals if it has been the writing convention in the sources that deal with them (for instance, in Peter Naur's seminal paper on ALGOL[130], he refers to the group SHARE as "SHARE" but refer to the acronym USE as "USE").

Punctuation follows the practices of structured design. In the *Chicago Manual of Style*, punctuation follows a non-structured manner[131]. For instance, according to the Chicago Manual of Style when a quotation ends a sentence, the period comes inside the quotation, "like this." Following the practices of structured design, quotation defines a block of text that begins and ends with a quotation mark. In structured design, a sentence defines a block of text, which begins with a capital letter and ends with a period. In structured design, if a block of text (quotation) is inserted inside another block of text (sentence), they are nested "like this".

I refer to fields such as mathematics and physics without capitals ("Physics"). I use the term *social studies of computer science* as a name of a field.

129 See IEEE Annals in the History of Computing–e.g. Rosenblatt, 1984. There are other conventions, too. For instance Campbell-Kelly & Aspray, 2004 and Sammet, 1969 use all capitals FORTRAN.

130 Naur et al., 1960

131 The difference between structured and unstructured punctuation is similar to the difference between HTML and XML. HTML code "........." is similar to Chicago Manual of Style of punctuation, but that code is not structured.

1.5.Limitations and Assumptions

Constraining arguments in one way or another is not uncommon in any discipline. Even in the field of algorithms and theoretical computer science, in mathematics, and in any type of quantitative research, statements are invariably bounded or qualified in one way or another[132]. In the following pages I discuss some of the limitations of, and assumptions in, my thesis. The following discussion is not meant to be a second line of defense for me, but guidelines for the reader about the applicability of my argument.

I cannot unearth all my hidden assumptions, if simply for the reason that I am not aware of all of them, but in my argumentation I have attempted to make them as explicit as possible. It may be easy to spot some implicit assumptions, such as my trust in democratic decision-making. Some other assumptions may be harder to spot, like my belief that people must be responsible for their actions. My argument about the responsibility of computer scientists for their science is based on that assumption. One can always choose to delegate responsibility to groups, opinion leaders, consumers, or other influential actors in the scientific-industrial-public complex, and such relocation changes the applicability of some of my arguments. Responsibility is not a simple matter, and it has been debated extensively in the philosophy of technology[133].

Sociocultural Matters

Everybody, every researcher, has a certain sociocultural background, a certain education, and certain experiences. The background of a researcher inevitably biases their research; every researcher interprets phenomena from a certain sociocultural, educational, and personal point of view. The statement above is an underlying assumption in this thesis—I assume that context matters and that neither statements nor interpretations are made in a sociocultural vacuum. This view is common to philosophical hermeneutics in general[134].

Essentially all research can be considered to be interpretive. All research, including logico-mathematical inference, is guided by the researcher's beliefs and interpreta-

132 For instance, "Let $n_0>0$ and $c>1$." or "$p<.005$".

133 See references in, for instance, Shrader-Frechette, 1992 and Bunge, 1979; and for a general discussion, see Scharff & Dusek, 2003

134 Gadamer, 1976:p.*xxv*.

tions about the world, as well as the researcher's view on how the world should be understood and studied[135]. Note that this standpoint about interpretivity is another underlying assumption in this thesis, and is incompatible with the naïve positivist viewpoint. A proponent of naïve positivism might argue that researchers are able to find out and prove aspects of how the world really works, and that full objectivity is possible.

Conceptual Consistency

There is serious disagreement about many terms and concepts. For instance *culture*, *society*, and *science* have been defined in a number of inconsistent ways and there are debates about which definitions are best. Many terms are used in a variety of ways in different sources, yet some consistency of terms and concepts is necessary for this thesis. However, defining terminology strictly and defining terminology loosely both have their problems.

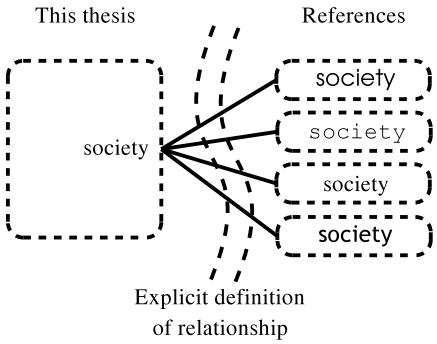


Figure 3: Internal Conceptual Coherence in a Thesis

If I defined terms (such as *culture*, *society*, and *science*) in detail for the purposes of this thesis, then it would be possible to fully establish internal conceptual coherence in this thesis. That is, I could define terms so that they would be used consistently throughout this thesis. In that case, however, maintaining conceptual consistency between terms in this thesis and terms in the references would be burdensome. In that case, the relationship between my use of each term and the use of the term in each reference should be defined explicitly (Figure 3). Because such a convention would create literally thousands of relationship definitions, and because many terms are inextricably bound to their context, I doubt the appropriateness and clarity of this approach.

135 See, for instance, Guba & Lincoln, 1994.

If I would not define terminology (such as *culture*, *society*, and *science*) at all in this thesis, then the meaning of terms in this thesis would change according to each reference (Figure 4). There would be no need to maintain conceptual consistency between the thesis and the references, but there would be no internal coherence in the uses of the concepts in this thesis. Because terms such as *culture*, *society*, and *science* have been defined in a number of inconsistent ways in my references, not defining terminology at all would lead to an excess of ambiguity of terms and concepts in this thesis.

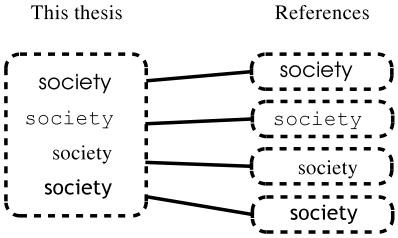


Figure 4: External Conceptual Coherence in a Thesis

Because neither a strict definition nor a loose definition seems to be appropriate, I take the middle-ground. Throughout this thesis I frequently revisit characterizations of those terms that are central to this thesis. I avoid constant redefinitions of terms, but if the author of a certain reference has a unique understanding of a certain term, I make it explicit in the text. I discuss a variety of definitions of the terms that are central for this thesis, but the final use of terminology is unavoidably subject to criticism. Regardless, I seek to make the rationales of my standpoints towards terms and concepts as explicit as possible.

Although I seek to make my theoretical and conceptual underpinnings explicit, in this thesis I present the views of a large number of authors. Many of the authors come from different disciplines, and even within specific disciplines many authors have conflicting views. Risking appeal to authority[136], I emphasize the views of authoritative figures in some disciplines at the cost of other, lesser-known authors. For instance, I elide discussion on alternative definitions of the terms *algorithm* and *normal science* and refer to Thomas Kuhn's use of the term *normal science* and Donald Knuth's use of the term *algorithm*. To make a critical reading easier for the reader, I

136*Argumentum ad verecundiam*—not "*appeal to unqualified authority*". Appeal to authority can be allowed, but strictly within some specific limits (Hurley, 2000:pp.138-140).

often introduce authors briefly[137]. Beginning from Chapter Two I also indicate the date of birth and death of each influential author.

The Scope of This Thesis

From the point of view of the *history of computer science*, this thesis is temporally limited to the post-1945 era, and spatially limited, mostly, to the American scene. The majority of my material was published in the United States, and the publication language of all my material is English. For instance Soviet, British, Israeli, Dutch, Swedish, Japanese, and Korean efforts in the field of computing are not discussed because the majority of the breakthrough innovations in electronic digital computing have been attributed to the American computing scene, and the majority of the literature is about the American scene. Although the pioneering figures in the history of computing have come from a number of countries, most of them have written in English and published in American journals. The early (pre-20th century) history of computing is excluded from this thesis despite the influence that, for instance, office machinery had on the development of electronic digital computing. The number of sources has grown gradually, and the bibliography at the moment is a snapshot of my research at one moment. Further research will always affect the sample of literature and sources.

From the *computer science* point of view, this thesis is not focused specifically on any single topic of computing, such as algorithms, complexity, artificial intelligence, software engineering, or human-computer interaction. Although all of the above-mentioned topics are nowadays considered to be parts of computer science, and although an in-depth examination of each of them might offer revealing insights into the social construction (or determinism) of computing, the sheer number of fields and subfields of computing is enormous, and the boundaries between those fields are vague. Instead of emphasizing any single topic in computer science, I discuss computer science broadly.

From the *philosophy of science* point of view, this thesis is limited to a number of influential authors. I only include a small number of well-known Western philosophers and philosophers of science such as Popper, Kuhn, Lakatos, Feyerabend, and

137 For instance, I write, “Sociologist Manuel Castells has connected...”, “Historian of technology Thomas Hughes recognized...”, or even “The 1993 Turing Award winner, Juris Hartmanis, took an opposite view...”.

Searle, and the discussion of contemporary debates is limited. The philosophy of computing is not discussed in-depth either, because most of the literature on the philosophy of computing focus on, for instance, the theory of consciousness, mental states, linguistic universes, information theory, or theories of multiple realizability, and not on the philosophy of computer science. Of the philosophical accounts of computing that play a substantial role in this thesis, Brian Cantwell Smith's meta-physical investigations of computing are discussed the most, along with the works of philosophers of computing such as Luciano Floridi, Philip Brey, and Gordana Dodig-Crnkovic.

From the *science and technology studies* point of view this thesis does not include many modern philosophers, sociologists, and media critics, including famous ones such as Deleuze, Derrida, Giddens, McLuhan, Heidegger, Foucault, and Latour. In-cluding these authors would clearly have brought depth to the analysis of computing because they have written about influential topics such as cyberspace, reflexivity, speech acts, deterritorialization, actor-networks, and even the concept of technolo-gies. Nonetheless, those topics are not within the central focus of a thesis on the definition and possible extension of the discipline of computing.

2. Interpretations of Science And Technology

Sci·ence
NOUN: **1:** A particular discipline or branch of learning, especially one dealing with measurable or systematic principles rather than intuition or natural ability; "*the science of genetics*". **2:** Ability to produce solutions in some problem domain; "*the skill of a well-trained boxer*"; "*the sweet science of pugilism*". **3:** The fact of knowing something; knowledge or understanding of a truth. **4:** Knowledge gained through study or learning; mastery of a particular discipline or area. **5a:** The collective discipline of study or learning acquired through the scientific method; **5b:** the sum of knowledge gained from such methods and discipline.
ETYMOLOGY: From French *science*, from Latin *scientia,* "knowledge", from the present participle stem of *scīre*, "to know".

In this chapter I discuss the first of the two central themes, *science*[1]. Specifically, in this chapter I focus on the views held by academic researchers who study the phenomenon of science. *Science and technology studies* (STS) is an appropriate umbrella term for the collection of literature selected for this chapter because not all of the works in this chapter are from easily demarcated areas. For instance, it can be argued that Thomas Kuhn's work[2] is from the field of philosophy or that it is from the field of sociology. Both arguments can be defended. However, most of the literature chosen for this chapter discuss the *essence of science and technology*.

The first half of this chapter deals with some aspects of the philosophy of science. The term *science* is already difficult: The term can be read in a number of ways, depending on the context. For instance, science can mean (1) a specific class of activities such as observation, description, and theoretical explanation; (2) knowledge obtained via those activities; or (3) any activity that has some special characteristics such as thoroughness or orderliness: "*I've got packing a suitcase down to a science*". Science can also refer to (4) a historical continuum: "Western science". Science can refer to (5) a societal institution: "The position of science in our society", "Humanity should be governed by science", or (6) a world view: "He believes in science". Science is also the scientist's (7) profession. Many of these definitions can have both a descriptive and a normative meaning—activities that scientists *are doing* (descriptive), or activities that recognized scientists *should be doing* (normative).

1 See the box on this page for a synthesis of the *science* entries in *Wiktionary* and *Webster's Online Dictionary* (accessed September 27th 2006).
2 Kuhn, 1996 (orig. 1962)

Furthermore, as academicians so well know, there are different aims of science, such as finding, explaining, applying, developing, refuting, verifying, and so forth—and it is not clear at all which of these aims are acceptable or desirable for science. Because many of these different readings are interconnected in complex ways, in this thesis a detailed analysis of the term *science* is elided, and different definitions are given at different contexts. However, in Chapters Three and Four I discuss the definition of *computer science*.

The second part of this chapter focuses on recent (mostly 1980s-present) discussions about the relationship of science and society. First is the discussion concerning the social shaping of technology and the social construction of science. There are a number of schools and viewpoints, ranging from a complete denial of the external world (phenomenalism) to a complete denial of free will (materialism). These extreme positions are quite easy to defend because they tend to narrow argumentation to mere denial of any opposing viewpoint (and it follows from the definition of extreme[3] that an extremist cannot agree with anything else than his or her extremist position). Extreme positions are not suitable for the purposes of this thesis, though. Therefore, at the risk of reducing coherence, a more fertile middle-ground is sought for. A number of sticking points between realism and relativism are discussed, and where possible, positions for this thesis are formed (and perhaps surprisingly, the adopted position leans more towards the realist side than the relativist side of the discussion). I also cover some of the discussions concerning the feasibility of the technological deterministic argument. There are different degrees of technological determinism, and due to the realist leaning in this thesis, some of the technological deterministic viewpoints are adopted.

In order to make this thesis self-contained (i.e., readable for computer scientists without the need to be proficient in science and technology studies) this chapter starts out by making basic concepts and divisions between philosophical schools clear. Although there cannot yet be a tight linkage with computing as a discipline, the philosophy of science is discussed with connections to and reflections on computing. Therefore, unlike the literature review chapter in conventional dissertations in computer science, this chapter is not a mere summary of what has been done in the field, but a critical explication of the conceptual framework used in later

3 "Most remote in any direction" (AHD, 2004)

chapters. To make a critical reading of my analysis in Chapter Three easier for the reader, I make my position towards each of the concepts explicit and weigh the alternatives.

2.1.Some Basic Questions About Science

Science is knowledge which we understand so well that we can teach it to a computer[4]

The philosophy of science, which is the topic of this section, deals with issues such as the foundations of science, its assumptions and limitations, its implications, and what constitutes scientific progress. Science is a complex, multifaceted entity composed of semi-autonomous fragments, and a united consensus does not exist on what counts as science and what does not. The philosophy of science is not an exception: Different schools support rival, incommensurable, and often opposing theories.

The viewpoint of a philosopher of science often reflects his or her own field of study: A mathematician and a sociologist may place different philosophical approaches above others. The philosophical accounts of science introduced in this chapter are not chosen objectively or neutrally; they are not chosen, for instance, randomly, but I have purposively selected them. Ironically, choosing the accounts randomly would also account as committing oneself to a number of philosophical and methodological standpoints.

Three basic questions that appear often in the philosophy of science are (1) the *ontological question*: "What is real?" or "What exists?", (2) the *epistemological question*: "How do we get to know about reality?" or "How can beliefs be justified?", and (3) the *methodological question*: "By which principles do we form knowledge?" or "How can we get knowledge about the world?". Norman Denzin and Yvonna Lincoln[5] noted that the choice of which one of the philosophical schools one follows leads to a choice of a set of values—there is no value-free science (and research on science itself is not an exception). The claim of many natural scientists that their research is value-free is a value statement per se—often a positivist one.

Denzin and Lincoln noted that paradigms—as overarching philosophical systems denoting particular ontologies, epistemologies, and methodologies—cannot easily be alternated between. Paradigms represent belief systems that attach the researcher to

4 Knuth, 1974c. The rest of the quotation is: "*and if we don't fully understand something, it is an art to deal with it*".

5 Denzin and Lincoln, 1994:pp.99-100. Professor Norman Denzin works in a number of fields, including sociology and cultural studies, and professor Yvonna Lincoln is from the field of education.

a particular world view[6]. This does not mean that the researcher should limit his or her study to one framework. A single phenomenon can—and should—be studied using different frameworks of explanation and from different viewpoints, thus yielding a wider view on the phenomenon and perhaps a deeper understanding of the dimensions of the phenomenon. This is often called *triangulation*, and lately *crystallization*. It has been argued that a researcher should be familiar with a variety of schools of research (or paradigms[7]), but also know that paradigms cannot mingle, or be synthesized[8]. Denzin and Lincoln called this kind of researcher, using anthropologist Claude Lévi-Strauss' term, a *bricoleur*[9].

According to philosopher of physics Mario Bunge, any authentic philosophy of science has two aims—one epistemic and one pragmatic[10]. The epistemic aim, Bunge wrote, is to understand scientific research and its findings, whereas the pragmatic aim is "to help scientists hone some concepts, refine some of theories, scrutinize some methods, unearth philosophical presuppositions, resolve controversies, and plant doubts about seemingly uncontroversial points". These two goals, Bunge wrote, complement each other.

These two aims can be seen throughout this thesis as a tension between between explanation and justification, or between *descriptive* and *normative* accounts of science. This tension, in other words, comes from the distinction between a philosophy of science that concerns classification or description (descriptive), and a philosophy of science that concerns prescribing norms or standards for science (normative). This tension is often referred to as the *is-ought problem*, and sometimes as *Hume's Law* or *Hume's Guillotine*[11]. It was raised by David Hume (1711-1776) in *Treatise of Human Nature*: Can one derive what "ought to be" from "what is"?[12] The is-ought problem is also obvious in the tenets of the authorities of the philosophy of

6 Denzin and Lincoln, 1994:pp.2-4.

7 At this point, *paradigm* is used in a very vague meaning, which is exactly as Thomas Kuhn used it (see Kuhn, 1996). The term is discussed and enlarged upon later in this thesis.

8 Denzin and Lincoln, 1994:pp.2-3,99-100.

9 Lévi-Strauss, 1966:pp.16,17. From French *bricoler*, "trifle", "tinker".

10 Bunge, 1998:p.405.

11 Rachel Cohon (2004) in Stanford Encyclopedia of Philosophy: *Hume's Moral Philosophy*. See also the discussion about the problems concerning Hume's Law, for instance, the interpretation that Hume denied ethical realism.

12 See Hume, 1739:Book III, pp.507-521. Hume says "no, you can't". Some other philosophers, notably John Searle, say "sometimes you can" (Searle, 1964).

science: witness the criticism of Thomas Kuhn's ambiguity of whether his account of science is a normative or a descriptive one[13].

Many current "naturalist" philosophers of science claim that there is no general account of science or scientific method that would apply to the development of all sciences at all historical stages[14]. Because science is not conducted in a social vacuum[15], it is necessary for a philosopher, sociologist, or historian of science to make a distinction between and understand at least five sides of science or technology: *science*, *technology*, *economy*, *polity*[16] and *ideology*[17]. In Chapter Three I focus on the aspects of science, technology, and polity, but I also touch on economic and ideological issues.

Quite a few accounts of the philosophy of science have been established, many of them based on very appealing arguments that try to combine two rival classical traditions: *rationalism* and *empiricism*[18]. In this chapter, the basic forms of a few influential philosophical viewpoints are presented and compared. Although the philosophical accounts of science introduced here do not represent a single school of thinking but several of them, the number of philosophical accounts presented here is still small. Because the ontological, epistemological, and methodological approaches in different disciplines are so various, there are many interesting, oft-used approaches that are not discussed in this thesis. For instance, approaches such as critical theory and feminist approaches are not covered here.

The philosophy of computing is not a subset of the philosophy of science. Gordana Dodig-Crnkovic noted that the philosophy of computing is a broader field than the philosophy of science, because it encompasses more than different scientific facets

13 Feyerabend, 1970

14 See, e.g., Chalmers, 1976:pp.247-253; Feyerabend, 1993:pp.18-19.

15 See, e.g., Bunge, 1998:p.406; Chalmers, 1976:pp.248-249; Kuhn, 1996.

16 *Polity* here means the form or constitution of a politically organized unit, or the form of government of a religious denomination.

17 Bunge, 1998:p.407.

18 The basic positions of rationalism, empiricism, and positivism are as follows. Rationalism regards reason as the main source of knowledge. Rationalists hold that reality has an inherently logical structure, and they assert that there are true facts that *the intellect* can grasp directly. Empiricism asserts that knowledge derives *from experience*, in the sense that justification requires experience. Positivism–especially its later branches–is related to empiricism in that positivists believe that justification requires experience and they exclude *a priori* or metaphysical speculations. The basic affirmations of positivism are (1) that all knowledge regarding facts is based on the "positive" data of experience, and (2) that beyond the realm of facts is that of "pure logic" and "pure mathematics". (See, e.g., Encyclopædia Britannica Online, 2004)

of computing[19]. Although the boundaries of both of these fields may be unclear, it is sufficient for this thesis (which is not a thesis on the philosophy of computing) to note that there are ontological and epistemological elements that are specific to the philosophy of computing and not to the philosophy of science at large. Take, for instance, the concept of a "program"—its ontological and epistemological statuses are ambiguous, as Brian Cantwell Smith noted[20]. Furthermore, the philosophy of computing not only includes both normative and descriptive components, but also an important ethical component[21].

The philosophy of computing as a discipline is not older than computer science, and the contemporary debates in the philosophy of computing reflects the trends in the philosophy of science. Currently there is no consensus about the nomenclature for a discipline that is concerned with the philosophy of things such as information theory and computation. There is research about the philosophy of information, the philosophy of computing, cyber philosophy and digital philosophy, which all refer to more or less the same discipline[22]. Perhaps the reason why there is no consensus about the term *philosophy of computing* is that there is no consensus about the term *computing*.

In this thesis, the term *philosophy of computing* is used to refer to both an analytic approach and a metaphysical approach[23] to philosophy. The philosophy of computing, as understood in this thesis, is concerned with topics such as what can be done in principle by a mechanism[24], matters of machine intelligence[25], natural and artificial realities[26], what is 'elegant' in computing[27], and other similar questions that link to issues of computation—such as to the machine and the theory.

This section on the philosophy of science supports both the epistemic and pragmatic aims of this thesis. Firstly, it forms a basis for a discussion about the predominant philosophical foundations of mainstream computer science (or the lack of foundations thereof). Secondly, as a part of a larger framework, it helps characterize the

19 Dodig-Crnkovic, 2003
20 Smith, 1998:pp.29-32.
21 Rapaport, 2005
22 See, e.g., Dodig-Crnkovic, 2003 and Floridi, 2002 for examples of these.
23 For a discussion of these two approaches, see Floridi, 2003.
24 See, e.g., Scheutz, 2003.
25 See, e.g., Dodig-Crnkovic, 2003.
26 Floridi, 2004:p.*xiii.*
27 See, for instance, Gelernter, 1998; Gelernter, 1998b.

concept of computing as a social and cultural activity (hence addressing computing as practice), and in addressing the question of whether technology is a driving force of progress.

The aim of this chapter is not to define *science*, *research*, *the scientific method* or *computer science*, but to enlarge upon these concepts and premises in this thesis. Throughout this discussion, the foci are on science as knowledge and on science as activity. The discussion includes, for instance, how people in different philosophical branches view science as knowledge; whether knowledge is produced or discovered, absolute or relative, reasoned or experienced, deductive or inductive, objective or subjective. Four different views of science are presented and discussed: the empiricist view, the falsificationist view, the "Kuhnian view", and the anarchistic view.

As the inclusion of only those four views indicates, this thesis focuses on only the major schools of the last century of philosophy of science. There is nothing original in foregrounding these authorities. The names are among those mentioned in any encyclopedia[28]: Rudolph Carnap (1891-1970) and the Vienna Circle, Karl Popper (1902-1994) who was the most prominent critic of the Vienna Circle, Thomas Kuhn (1922-1996) who described science as a contract, and Paul Feyerabend (1924-1994) whose sarcastic critique of the scientific method is as topical as ever. These authors are often mentioned as the leading figures of three dominant mainstream schools of the philosophy of science in 1900s[29]. From the field of the philosophy of computing, Brian Cantwell Smith and Luciano Floridi are foregrounded because of their pioneering work in the philosophy of computing.

Among noted philosophers who are set in the background of this discussion are Imre Lakatos (1922-1974) who modified Popper's falsificationism and whose work faces problems similar to that of Kuhn, and Bertrand Russell (1872-1970) who was a believer of the scientific method and of verification and whose work shares the general problems of positivism. Among the noted approaches that are set in the background are Bayesianism, because of its numerous shortcomings and because including it would not change my argument[30]; Ian Hacking's and Deborah Mayo's new experimentalism, mostly because it replaces the positivists' view of *senses* as the source of

28 See, e.g., Encyclopædia Britannica Online, 2004 "philosophy of science".

29 Bouillon, 1998

30 See Chalmers, 1976:pp.187-192. See also Kelly & Glymour, 2004 for a rebuttal of *Bayesianism*, i.e., the Bayesian confirmation theory.

scientific knowledge with *experiment* and thus does not offer an angle that would be of any extra use for this thesis[31]; as well as a number of often interesting variations of basic positions such as radical empiricism, radical constructivism, and scientific realism[32].

31 Moreover, "new experimentalism" does not yet exist as an established field (Chalmers, 1976:p.194–the 1999 edition).

32 See Chalmers, 1976 for more examples.

How Does One Come to Know Facts?

Foolish opinions will be readily eliminated by the shock of facts[33]

In This Section:

- ✔ A short discussion about three basic positions in the philosophy of science.
- ✔ What are the basic problems of scientific justification?
- ✔ What is the role of logic in science?

The central problem of epistemology is the problem of the *growth of knowledge*. That is, some statements sometimes become facts, whereas sometimes some facts lose their status as facts and become misbeliefs. If knowledge includes facts in the strict sense, it is hard to see how facts *proper* can change. According to sir Karl Popper, the growth of knowledge can be studied best by studying the growth of *scientific knowledge*[34]. The reason for this, Popper wrote, is that scientific knowledge is easier to analyze than common-sense knowledge—but Popper did not explicitly exclude the analysis of common-sense knowledge either. The reader is advised not to expect a definition of the *growth of knowledge*, but an exploration of viewpoints on what is knowledge and what could the growth of knowledge therefore mean. When analyzing *scientific knowledge* in the field of computing, a problem arises from the numerous arguments over what is considered to be scientific in the field of computing—indeed, what is computer *science*? More questions arise instantly: If it is a science, what does it study? If it is computer *science*, are programs *laws*? Questions such as these are discussed in detail in Chapter Three.

After having begun to achieve a disciplinary identity, the academic field of computing has seen a dramatic diversification. In the field of computing, the relationship between growth and diversification is ambiguous; it is unclear whether *diversification* is considered to be *growth* or *scattering*. It is equally unclear if diversification implies increased eclecticism (a negative connotation), increased holism (a positive connotation), increasingly wide or numerous perspectives (positive), an increasingly vague focus (negative), or something else.

33 Cohen and Nagel, 1934:p.402.
34 Popper, 1959:pp.*xix-xxi*.

Intuition of Scientific Knowledge

Because the problems with the term *computer science* are dealt with later in this thesis, it would probably be fair to start by referring to science via the *intuition of scientific knowledge* as stated by Alan Chalmers, and developing the concept of *scientific knowledge* throughout this chapter:

> *What is so special about scientific knowledge is that it is derived from the facts, rather than being based on personal opinion.*[35]

However, this intuitive idea is problematic as it turns out. It is unclear what are the *facts* in that statement. In computer science it is not certain whether facts are derived through formal methods of *reasoning*, through scientific methods, including *observation*, or through some other form of fact-gathering or fact-creation. A study by Glass et al. reveals that there are a number of methods of inquiry in the fields of computer science, software engineering, and information systems[36]. There is plenty of computer science research based on the logico-mathematical tradition, there is plenty of computer science research based on the empirical tradition, and there is plenty of computer science research based on the engineering tradition. But in computer science, there is little discussion about the mode of existence of the facts revealed by different methods of inquiry.

It can be asked, "Are facts in computer science universally true or not?". It is evident in a number of sources in Chapter Three that the universalistic stance is often supported, explicitly or implicitly. It can also be asked, "Are scientific facts timeless or do they change?—and regardless of whether they do change or not—what is so special about scientific knowledge that it can be distinguished from 'folk knowledge', personal opinions, or mythical beliefs[37]?". In fact, it has been noted that there are *folk theorems* in the field of computer science[38].

---

35 Chalmers, 1976:p.*xx*.

36 Glass et al., 2004

37 Actually, the history of science shows that what have been considered scientific facts and scientific knowledge have undergone radical changes throughout time. Keeping this in mind, the reader should every now and then stop and ask what is there so special in scientific knowledge that distinguishes it from folk knowledge or mythical beliefs.

38 See, e.g., Harel, 1980; Denning, 1980; De Millo et al., 1979.

Early Schools of Thought

In the following pages, three positions to formalize the essence of science are described: the empiricist position, the positivist position, and the inductivist position. Also the rationalist position is touched upon. Those attempts; or positions, or schools of thought; are all found in computer science, and their contact points with computer science are discussed in applicable parts of this section. Those positions are all in the spirit of a quotation from the year 1934, written by two men with a "love for truth", as they describe it (the same quote began this chapter):

> *...we are sure that foolish opinions will be readily eliminated by the shock of facts*[39]

Two early schools of thought that involved attempts to formalize a view of science that scientific knowledge is derived from facts rather than from opinions are empiricism and positivism. The British empiricists of the 17th and 18th centuries—notably John Locke (1632-1704), George Berkeley (1685-1753), and David Hume—held that all knowledge should be derived from ideas that senses have implanted in the mind. The positivists shared the view of the empiricists that knowledge should be derived from the facts of experience[40]. There is indeed so little difference between the early 20th century versions of the two schools that the terms *logical positivism* and *logical empiricism* are used synonymously[41]. Even though one might think, for reasons clarified later in this chapter, that logical positivism is dead and gone, it has been argued that many (or most) philosophically facile artificial intelligence researchers are still sometimes considered "part of the extended diaspora of Viennese logical positivists"[42].

Empiricists stand in contrast to rationalists, who believe in human, *a priori*[43] reasoning, as a source of knowledge[44]. For instance, Plato (ca. 427-347 B.C.) and René Descartes (1596-1650) held that what people know by reason alone is superior to experiential knowledge in a number of ways—it is unchanging, eternal, perfect, cer-

39 Cohen and Nagel, 1934:p.402.
40 Chalmers, 1976:p.3.; see page 36 of this thesis.
41 AHD, 2004, Encyclopædia Britannica Online, 2004: see "logical positivism" (accessed December 2nd 2004).
42 Loui, 1998
43 *A priori*: knowledge deduced without sensual stimuli or observation (using pure reason); *a posteriori*: knowledge originating from observations or experience.
44 Peter Markie (2004) in Stanford Encyclopedia of Philosophy: *Rationalism vs. Empiricism*

tain, and universal[45]. Descartes noted, though, that only knowledge that is mathematical by nature can be deducted, but other sorts of knowledge need to be justified by observations. The rationalist position and the empiricist position share similar problems when explaining the growth of knowledge—for instance, they both have a problem explaining why facts get refuted all the time. For the rationalist, it is difficult to explain how is it that many mathematical facts, once considered eternal, have been refuted. And the opposite is also true: If there are facts that are eternal, those facts can not "become" facts.

In one of the landmarks in the philosophy of science, *Proofs and Refutations*, Imre Lakatos gave a historical account of how Euler's (Leonhard Euler, 1707-1783) formula V-E+F=2 was reformulated again and again for almost two hundred years after its first statement, until it finally reached its current, stable form[46]. Lakatos' work is a criticism of dogmatist and formalist mathematicians and puts forth an idea that mathematical knowledge grows through proposals, speculation, and criticism—through proofs and refutations. According to Lakatos, even the mathematician, the researcher of the king of sciences, does not know anything for certain[47]. Similarly to some major works in the philosophy of science[48], in this thesis the philosophy of mathematics and the philosophy of science are not dealt with separately—even though they perhaps should[49].

In the context of natural sciences and the positivist school of thought, an apprehendable reality is assumed to exist, and this reality is driven by immutable natural laws and mechanisms (Egon Guba and Yvonne Lincoln noted that this ontological view is commonly called *naïve realism*[50]). Researchers can converge on the "true" state of affairs; they seek *final truths* (this epistemological view, adopted by positivists, is called dualist/objectivist epistemology). The positivists believe that objects of real-

45 Peter Markie (2004) in Stanford Encyclopedia of Philosophy: *Rationalism vs. Empiricism.*

46 Lakatos, 1976: almost the whole book discusses the matter. See also De Millo et al., 1979. *Euler's formula* concerns finite and connected planar graphs, and states that if the graph does not have any intersecting edges, and if V is the number of vertices, E is the number of edges, and F is the number of faces, then $V-E+F=2$.

47 Lakatos, 1976:pp.1-5,125.

48 Of the major philosophers of science covered in this thesis, Popper and Lakatos implicitly considered the philosophy of science and philosophy of mathematics simultaneously; Kuhn did not discuss logical and mathematical proofs; Feyerabend denied both.

49 The Quine-Putnam argument, which is seen as the best argument for mathematical realism, holds that mathematics is indispensable to science (Mark Colyvan (2004) in Stanford Encyclopedia of Philosophy: *Indispensability Arguments in the Philosophy of Mathematics*). See, for instance, Quine, 1980.

50 Guba & Lincoln, 1994:pp.109-110.

ity, questions, and hypotheses can be *verified.* A failure to verify a hypothesis does not straightforwardly lead to its rejection. Rather, the same inspection procedure is repeated, with distorting factors better controlled. The methodologies in positivism are chiefly quantitative, experimental, and manipulative.

The Problems of Positivism

According to Alan Chalmers, there are three assumptions that are central to positivism[51]:

1. Facts are received by careful, unprejudiced observers via the senses,
2. Facts are prior to and independent of theory,
3. Facts constitute a firm and reliable foundation for scientific knowledge.

Each of these seemingly commonsense claims faces unsurmountable difficulties. First of all, the positivist has to assume that when referring to, for example, sight, two observers with perfect vision would see the same thing. But that is not intuitively true. For instance, when a beginner and a seasoned microbiologist look through the same microscope at the same slide, it is often the case that the beginner cannot distinguish the cell structures that the professional can easily distinguish[52]. Even though it is evident that human biological properties play a major part in how, and what, one perceives, what one perceives is also dependent on one's previous experiences, cultural upbringing, and expectations. Briant Cantwell Smith noted that whether one sees a mountain range, forms of terrain, or a number individual mountains, is an individual experience, and there is no epistemologically correct answer[53].

If a competent, experienced programmer is given, say, a recursive algorithm, and the same algorithm is handed over to a student who has just learned the language, they both know exactly the same language syntax and see the very same code, but most likely their understanding of what the algorithm does differs. It is especially absurd to think that *statements* of fact enter the brain by means of the senses. It is absurd, because atomic statements do not make sense as such—one needs to have a linguist-

51 Chalmers, 1976:p.4.

52 Chalmers, 1976:pp.7-11. Note that these problems are discussed widely in the philosophy of science, but as the development of the philosophy of science is not the focus of this thesis, the works of philosophers such as Bertrand Russell and Ludwig Wittgenstein are not analyzed here.

53 Smith, 2002c:pp.239-240.

ic and conceptual framework for interpreting statements. To make sense, the atomic statements are anchored in and related to this framework.

If observations are always contaminated by theories and by previous experiences[54], then it cannot be that one first establishes *facts* through observation and then derives *knowledge* from those observations. I see this problem in the often cherished "data–information–knowledge–wisdom"-distinction[55]. It is common to prune these concepts to some sort of a variation of the following: *datum*[56] is a basic attribute of information; *information* is data with a conceptual commitment and interpretation; *knowledge* comes from knowing how to use the information; and *wisdom* adds the understanding of when and where the knowledge is applicable. However, I argue that this simplification is fundamentally wrong.

Data is not collected randomly, and in the data collection process, only a minor part of (infinite) available data is collected. The choice of which data to collect, and which to not collect, is done by using what one already knows about the domain. The data is collected using formal or informal data structures, which are usually conceptual aggregates that are created to model the phenomenon as well as possible. Because data structures are built according to what one already knows about the domain, data cannot be independent of one's previous knowledge[57]. Andrew Pickering's concept of "the mangle", discussed later in this thesis, offers a different viewpoint on the interrelations of data and knowledge[58]. For the sake of simplicity, the descriptions of data, information, and knowledge are in this thesis used in the meanings described above, with a remark that they are not independent of each other.

Even if one dismissed the problems of positivism described above, the positivist school of thought would be inappropriate for computer science anyhow, because computer science does not always deal with merely observable or merely logico-mathematical facts, and certainly neither with *merely a priori* nor *merely a posteriori* knowledge. Formal and empirical theories and engineering practice in computer

54 Couvalis, 1997:pp.11-13.

55 For this distinction, see, e.g., Firestone & McElroy, 2003:pp.17-18.

56 In this thesis, I adopt the convention of using the plural "data" as if it was singular. Instead of the phrase "data are collected", I use the phrase "data is collected".

57 Brian Cantwell Smith's term *inscription error* is related to this problem (Smith, 1998:pp.49-50). Joseph Firestone and Mark McElroy's work also acknowledges this: see Firestone & McElroy, 2003:p.19.

58 Pickering, 1995

science are so tightly intertwined that they are almost inseparable[59]. Some branches of computing draw on theories from other sciences, others are self-referential[60]. Note, however, that there is danger of a logical fallacy: Using theoretical frameworks or theories to assess the adequacy of experimental results, and then taking those same experimental results as supporting evidence for the theories, is clearly circular reasoning.

There are a number of ways to view the class of "facts". An illustration of some of the viewpoints of the nature of facts is presented in Figure 5. The juxtapositions in Figure 5 are by no means a comprehensive set of the positions taken in science or philosophy. A number of additional juxtapositions are presented as my argument goes along. Especially John R. Searle's division between brute facts and institutional facts is discussed in detail later in this thesis.

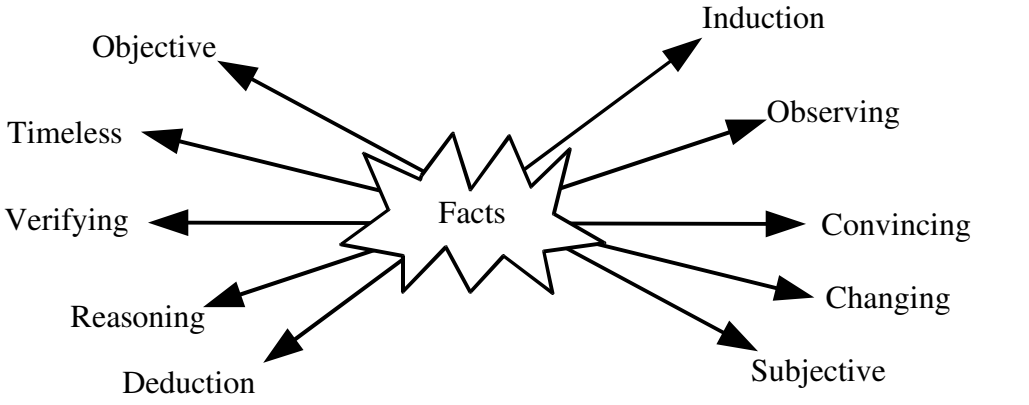


Figure 5: Differing Views of Facts

There is doubt, as explained above, as to whether observational and experimental *objective* facts could be established in science. Regardless, assuming that one could establish objective, theory-independent facts in science, the central question then becomes: "How can scientific *knowledge* be derived from those facts?". The strongest possible claim would be, as Chalmers wrote, that theory could be logically derived from facts. That is, a theory could be proven as a consequence of a given set of facts.[61]

59 For some authorities who think that theory and practice (or abstract and concrete) both belong inextricably to computer science, see Knuth, 1991; Denning et al., 1989; Forsythe, 1968; Wegner, 1976; Hopcroft, 1987. Then again, there are those who think otherwise: see, for example, just about anything by E.W.Dijkstra.

60 Glass et al., 2004

61 Chalmers, 1976:p.41.

Logical Truths Do Not Carry Information

There is a common misconception that has to be made clear. Contrary to a common (mis)belief, logical truths do not carry information. As Manuel Bremer noted, the amount of information in logical truths—seen in the light of the standard approaches to measuring or defining information content—is zero[62].

Syllogistic logic states that *if* the premises are true *and* the argument is valid, *then* the conclusion must be true. For instance, let premise p_1 be "Every human being understands the C++ programming language" and p_2 be "My grandmother is a human being". From these premises, it is a valid deduction d that "My grandmother understands the C++ programming language". In this case, it happens to be that p_1 and d are false, as my grandmother has never been interested in learning the C++ language. But this does not affect the fact that this logical argument is perfectly valid. That is to say, logic has nothing to do with the truth of deductive reasoning.

In *Critique of Pure Reason* Immanuel Kant (1724-1804) divided propositions into analytic and synthetic[63]. Analytic propositions are those that are true simply because of their meaning. For instance, "All bachelors are unmarried" is analytic because the concept *unmarried* is contained in the definition of *bachelor*[64]. Propositions that are not true because of their meaning only, such as empirically grounded scientific arguments, are synthetic[65].

According to the rules of logic, if and only if one can be certain that the premises are true, one can also be certain that everything logically derived from the premises will also be true. But affirming that premises are true can be very difficult. The fact that sun has risen every day for the last 4.5 billion years does not conclusively prove that the sun will rise tomorrow (and in fact one "day" it probably will not). If not on experiment or observation, what *can* logical deduction be based on? Deduction works only if one accepts the axioms[66] on which systems of knowledge such as mathemat-

62 Bremer, 2003

63 Kant, 1966, orig. 1781:A6-7/B11.

64 Shapiro, 2000:p.77.

65 Yet not all propositions are either analytic or synthetic. For instance, the tautology "either it rains or it does not rain" is neither analytic nor synthetic. Note that W.v.O. Quine rejected the analytic-synthetic-distinction in *Two Dogmas of Empiricism* (Quine, 1980:pp.20-46).

66 Axioms (from Greek $\alpha\xi\iota\omega\mu\alpha$: "self-evident", e.g. $\forall x: x = x$), should not be held self-evidently valid even in mathematics. To *prove* something true, it should be proven in all systems (which is, of course, impossible). From the epistemological viewpoint, the truth value of axioms is not self-evident at all either. Epistemologically, on what grounds is this x same than this other x?

ics are built—and that is purely a question of faith. Nonetheless, computer scientists do not work only within an analytic, axiomatic framework of logic, but computer scientists work also with empirically based, synthetic propositions.

Inductivism

Inductivists evaded the problem of basing the growth of knowledge on logic by basing the growth of scientific knowledge on experience. For the inductivist, the objectivity of science is a matter of degree, and the degree of objectivity of scientific facts derives from the objectivity of observation, induction, and deduction. Inductivists avoid subjective opinions by demanding three conditions from a valid *inductive argument*[67]:

1. The number of observations forming the basis of a generalization must be large.
2. The observations must be repeated under a wide variety of conditions.
3. No observation should conflict with the derived law.

The *principle of induction* can be summed up as

> *There is a form of inference of laws from the accumulated simple facts, so that from true statements describing observations and the results of experiments, true laws may be inferred.*[68]

This kind of argument has been used a lot in computing, especially in the field of information systems[69]. Also all the human-computer interaction (HCI) researchers who use Geert Hofstede's study[70] as an unquestioned basis for their research automatically acknowledge the principle of induction (although they may accept "relaxed" or probabilistic versions of induction). Hofstede's study is based on a vast number of IBM workers from more than 50 countries. From this large sample, Hofstede came up with four (and later five) dimensions of culture. He gave a score between 0 and 100 to each country, for each dimension. Note that the relaxed version of the principle of induction, "*If a large number of* **A***'s have been observed under a wide variety of conditions, and if* <u>*most of*</u> *the* **A***'s possess the property* **B***, then*

67 See Chalmers, 1976:p.46.
68 Harré, 1972
69 See, e.g., Glass et al., 2004 on the emphasis on experimental approach in IS.
70 Hofstede, 1997; used in e.g. Marcus and Gould, 2000, and numerous other studies.

all **A***'s* *probably* *have the property* **B**.", is no longer about *facts* in the strict sense. This relaxed "sophisticated" version of induction suits everyday science-making very well, but it is not really about facts *proper* anymore.

From a computer scientist's point of view it is interesting to note that if one compares problem solving using algorithms with problem solving using neural networks, one may discern features of deduction in algorithmic problem solving and features of induction in neural networks. Algorithms are based on a set of known facts (algorithm and input) that then produce new facts (output) in a *white box*-manner[71]; the whole process is transparent and explicit. Logic programming and expert systems are especially well characterized by deduction.

On the contrary, neural networks, especially ones exhibiting unsupervised learning —such as self-organizing maps—are based on a set of computational objects (neurons) that are trained to produce the desired output. Neural networks work in a black box manner; the process is probabilistic and very hard to track. One could, arguably, associate *algorithms* more with theoretical, clear-cut (pseudo-) problems, and *neural networks* more with fuzzy (real-world) problems.

Problems of Inductivism

The problems of the first condition of the inductive argument are apparent: "What is large?", "Is a sample of a hundred, a thousand, or a million 'large'?", "Does 'large' depend on the object of investigation?". Certainly, in a study of a rare disease, a "large sample" has to be different from a study of quarks. If one studies web pages to find "cultural markers", the questions are, "How large a number of observations is enough for generalization?" and "Should one study one hundred web pages or one thousand—or all of them[72]?". In algorithm research the question might be, "How many cases must one study to show that one algorithm is faster than another algorithm?". Statistics can offer some answers about probabilities, but "qualified"

71 Also called *glass box*.

72 Using studies such as Hofstede's in user interface design is a bit ill-founded. Even if the in-depth criticism of Hofstede's work (e.g., Smith, 2002; McSweeney, 2002; Hampden-Turner and Trompenaars, 1997) is ignored, a quintessential question remains: what is the connection between Hofstede's dimensions and usability? Perhaps there is no connection. It is not yet shown that, say, power distance (one of Hofstede's dimensions), would be a determining factor of usability, or that considering power distance would have *any* effect on usability. (Minna Kamppuri, oral communication, 2nd April 2005). Hofstede's work is often used as a non-problematic basis for research–it is almost as if Hofstede's results were objective facts and separable from their context.

facts, no matter how well they meet some preordained probability classes, are not facts *proper*, that is, facts that are proven correct beyond any doubt.

It is also hard to draw the line with condition 2 of the inductivist argument ("the observations must be repeated under a wide variety of conditions"). Fulfilling condition 2 requires answers to questions such as, "How much variation is enough?", "How large should the variation be?", and "What kind of variables should be taken into account?". This third question is important, because unless it can be answered exhaustively, the list of variables can be extended indefinitely by endlessly adding further variations. There is always an infinite number of untested observation settings. For instance, in the case of user interfaces, the variations added can include the size of the screen, the brand of the computer used, and the distance to the closest accordionist. Unless negligible variations can be eliminated, the conditions under which an inductive inference can be accepted can never be satisfied.

However, when defining what counts as negligible or superfluous, the researcher draws on prior knowledge of the situation to distinguish between the factors that might and those that cannot influence the system under investigation[73]. This leads to an infinite loop. Induction is based on an appeal to previous knowledge, which is based on further inductive arguments, and so on. One cannot justify all knowledge by induction.

In the early 1930s Kurt Gödel's (1906-1978) incompleteness theorem, which sets limits to intellectual achievement with axiomatic systems, shook the positivist sciences. Gödel's incompleteness theorem proved that any consistent formal system necessarily contains some propositions that cannot be proven or disproven[74]. That is, there are things that formal systems, such as mathematics, cannot tell anything about. In Brian Cantwell Smith's words, semantics can never be wholly reduced to syntax[75]. Alan Turing (1912-1954) proved later that there are problems that can be precisely formulated, but that cannot be solved by any algorithm, or any computer for that matter (undecidable problems). Furthermore, the positivist, empiricist and inductivist are bound to run into trouble when seeking rational justifications for

73 Chalmers, 1976:p.48.

74 Gödel, 1931 in Feferman, 1986. Some critical definitions have been excluded from this statement to keep it simple. The readers who notice this, will not need more thorough an explanation, and those who want to know the whole framework or to read a comprehensive analysis can find it in mathematics textbooks.

75 Smith, 2002b

every principle. This is because a rational argument cannot be provided for a rational argument itself without already assuming what is argued for. Chalmers wrote, "*Not even logic can be argued for in a way that does not commit the fallacy of begging the question*"[76].

The logical fallacy of *begging the question* (*petitio principii*) is committed whenever the arguer creates the illusion that inadequate premises provide adequate support for the conclusion by leaving out a key premise, by restating the conclusion as a premise, or by reasoning in a circle[77]. (For example, assuming in the premise of an argument something that one wishes to prove.) A prime example of begging the question comes from the question "How can induction itself be justified?" In *An Enquiry Concerning Human Understanding*, David Hume noted that there are two options: To justify it by appeal to logic (which will not do, because nothing proves *logically* that the future has anything to do with the past even though it surely appears so), or justify it by appeal to experience[78]. Alex Rosenberg wrote that one could make a deductive argument[79]:

> *1. If a practice has been reliable in the past, it will be reliable in the future.*
> *2. In the past, inductive arguments have been reliable.*
> *Therefore:*
> *3. Inductive arguments will be reliable in the future.*

This argument, Rosenberg wrote, is deductively valid, but its first premise requires justification. The only satisfactory justification would be the reliability of induction, which is exactly what the argument is supposed to establish, Rosenberg noted. One cannot argue that "induction is reliable because it has worked in the past", because one cannot justify induction with an inductive argument. Note that induction is used quite successfully in different branches of science when statistical certainties are needed, but that success does not mean that sophisticated, statistical induction could offer a short cut to facts *proper.*

An error in thinking in computer science that is close to *petitio principii* is *the inscription error*, specified by Brian Cantwell Smith[80]. Inscription error refers to a

76 Chalmers, 1976:p.52.
77 Hurley, 2000:pp.156-159. Note that "begging the question" is a broad concept and there are different forms of it.
78 Hume, 1777, orig.1748:pp.25-26.
79 Rosenberg, 2000:p.115.
80 Smith, 1998:p.50.

phenomenon unavoidable for a computer scientist or a designer of any kind of representation[81], and it consists of two parts. First, the designer imposes a set of assumptions onto a computational system (when, for instance, creating data structures, object relationships, and so forth). Second, the designer reads those assumptions or their consequences back off the system, *as if they were independent empirical discoveries or theoretical results*[82].

In conclusion, neither empiricism, positivism, nor inductivism provide a durable basis for all of computer science. Although each of those approaches to science fits some applications, they all fail to assert their supremacy as an account of science, as they all face difficulties that have not been overcome. Adopting positivism or empiricism for an account of studies of computer science would narrow the study of computational systems down to simply proving facts correct or incorrect, and building knowledge from those atomistic pieces of data. Adopting inductivism would narrow the study of computational world down to observing phenomena and making generalizations. Neither of these approaches would be able to do justice to the diversity and multidimensionality of the phenomena at hand. A look at the most serious rival to these accounts, falsificationism, is taken next.

81 Smith, 1998:p.52.

82 See Smith, 1998:pp.51-53 for examples of inscription error.

Popper's Falsificationism

No number of experiments can ever prove a theory,
but a single experiment can disprove one.[83]

IN THIS SECTION:

- ✔ How do falsificationists see growth of knowledge in science?
- ✔ Could falsificationism do as the philosophy of computer science?
- ✔ Criticism of falsificationism.
- ✔ Underrepresentation problem.

Sir Karl Popper was the most forceful advocate of an alternative to inductivism, the alternative that is heretofore referred to as *falsificationism*. Popper's *The Logic of Scientific Discovery*[84] was a groundbreaking work in the philosophy of science. Ever since the publication of Popper's book, there was a continuous clash and debate between the old school and the new school. This clash between the advocates of Rudolph Carnap, a member of the Vienna Circle[85], and the advocates of Popper was a significant feature of the philosophy of science up to until the 1960s[86].

The Vienna Circle held that claims are meaningful only if they can be *verified*. Thus, they believed there were statements that were true and they aspired to verify their truthfulness. Most of the studies of, for instance, human phenomena do not meet this sort of a criterion of science: The assertion that "under these conditions these people behaved this way" is hardly a meaningful observation because distinguishing superfluous variables from significant ones in "these conditions" is a subjective matter. That is, the choice of significant variables, drawing the line of significant correlation, and possible generalization are subjective matters.

Steve Fuller, who is a proponent of Popper, wrote that Popper was sympathetic to the positivist struggle to hold all knowledge claims responsible to a publicly accountable procedure[87]. However, Fuller continued, Popper disagreed with the positivist scientists in that for Popper, deduction was mainly a tool to compel scientists to

83 Popper, 1959 (orig. publication 1934 in German, "*Logik der Forschung*").
84 Popper, 1959
85 The Vienna Circle was a group of famous philosophers and scientists who met weekly in Vienna, in the 1920s and 1930s. Their philosophy is nowadays called *logical positivism*. In their opinion, there should be strict criteria for categorizing claims as either *true*, *false*, or *meaningless*.
86 Chalmers, 1976:p.59.
87 Fuller, 2003:pp.24-25.

test the consequences of their general knowledge claims in particular cases by issuing predictions that could be contradicted by the findings of empirical research. Any belief whatsoever may be scientific or not, depending on whether one can try to falsify it, to test the limits of its validity. In the field of software engineering, for example, the following predictions have been suggested: "GOTOS increase code entropy", "Strong typing reduces run-time errors", and "Good modularization reduces maintenance costs"[88].

Inductivists use observation and experimentation in trying to find *universals*: "Program *p* produced the correct output with input x_1; the same program *p* produced the correct output also with input x_2". An inductivist would continue this until finally, after a large enough number of tests has been done, he or she would generalize a universal conclusion: "Program *p* produces the correct output with any input *x*". However, there might always be a special case where the program would not work correctly. Edsger Dijkstra[89] (1930-2002) has often been quoted in this connection—he wrote, "program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence"[90].

Around the 1930s Popper turned the positivists' thinking around and stated that there could not be a way to prove universal truths, but observations could be used in *falsifying* claims. For example, program testing is based on the principle of falsificationism and not on inductivism. There are alternatives to the inductivist and falsificationist paradigms in programming, too; for instance, the formal verification of programs is a deductivist, positivist enterprise.

A Single Observation Can Falsify a Theory

In texts about positivism, empiricism, and inductivism, the words *true* and *truth* are commonplace—the scientists following these philosophies of science are on "a quest for truth". I noted that the *truthfulness* of universal claims cannot be deduced from single observations. However, the *falsity* of universal claims can be—and this is ba-

88 Snelting, 1998

89 Edsger W. Dijkstra (1930-2002) is one of the indubitable authorities in the field of computer science, his main interest being formal verification - "developing proof and program hand in hand". He is known for, e.g., the shortest-path algorithm *Dijkstra's algorithm*. For instance, former ACM president Peter Denning wrote about him as "one of the giants of our field and a passionate believer in the mathematical view of programs and programming" (Denning, 2004). The name is sometimes misspelled as "Edsgar" (Ensmenger, 2001; Ubiquity, 2002). However, "Edsger Dijkstra" is the correct spelling.

90 Dijkstra, 1972

sically what falsification is all about. If the claim is "program *p* produces the correct output on any input *x*", then one single observation of erroneous functioning can falsify the claim. However, using falsification, it can never be said that a theory is *true*, but it can be said that it is the best theory currently available—since it has not yet been successfully falsified. In program testing, it is acknowledged that testing can only prove the existence of bugs, not the absence of them[91].

Alan Chalmers summarized falsificationism, writing that falsificationists freely admit that observation is guided by *and* presupposes theory. Theories—which are attempts to give an adequate account of some aspects of the world or the universe—are construed to overcome the problems encountered by previous theories. Theories can be speculative or tentative conjectures or guesses freely created by the human intellect[92]. Chalmers wrote that once proposed, speculative theories are to be rigorously and ruthlessly tested by observation and experiment. For the falsificationist, the whole of science is a set of hypotheses that are tentatively proposed with the aim of accurately describing or accounting for the behavior of some aspect of the world or the universe. There is only one condition that any hypothesis must satisfy if it is to be granted the status of a scientific law or theory: If it is to form a part of science, a hypothesis must be *falsifiable*.[93]

According to Popper's definition, a hypothesis or theory is to be called *falsifiable* if it divides the class of all possible basic statements unambiguously into two non-empty subclasses[94]. The first is the class of those basic statements which contradict the hypothesis (or which it rules out, or prohibits). Popper called this the class of *potential falsifiers* of the hypothesis. The second is the class of those basic statements which do not contradict the hypothesis (or which it "permits"). In other words, there must exist a *possible* set of observation statements ("potential falsifiers") that are inconsistent with the hypothesis. If any one of these potential observations is found to be true, the observation will falsify the theory or hypothesis.

For example, the statement "*Either the students using the learning program found it useful or they found it useless in learning*" is not falsifiable, because it is tautologically true—and thus cannot constitute a part of scientific knowledge (it is fundament-

91 Dijkstra, 1972
92 Chalmers, 1976:pp.60-62.
93 Chalmers, 1976:pp.60-62.
94 Popper, 1959:pp.65-66.

ally uninformative). "*A Turing Machine cannot solve all the computable problems*" is not falsifiable either, because of the definition of "computable problems"[95]: Every computable problem can be carried out by a Turing Machine, and the computation will always produce the result in a finite number of steps. The previous statement is meaningless because if a problem would not be computable with a Turing Machine, it would not be considered computable. Note that these kinds of statements are, of course, true by definition; they are agreements between scientists about the meaning of a term—nothing more, nothing less.

One could also take Popper's example "all ravens are black", and modify it to create an unfalsifiable proposition. For instance, "at least one raven is neon yellow" can be proven true by showing a neon yellow raven, but it cannot be falsified, because there is always a possibility that somewhere there is a neon yellow raven that people have not yet found. In a similar manner, claims such as "There are people who cannot understand the desktop metaphor" and "Social studies of computer science can benefit computer science" are not falsifiable.

Tautologies aside, it is dubious if a scientist can know the potential falsifiers of a novel theory before the theory has been used, tested, and applied. Furthermore, as Lakatos' example of Euler's formulæ indicates, many theories need to be thought through over and over again before (if) they become stabilized[96]. Formulæ similar to Euler's formulæ were tentatively proposed early in the 1700s; then rediscovered by Euler; reformulated several times for hundred years; rigorously proven by Augustin Cauchy (1789-1857) in the 1800s; and then gradually incorporated into the system of knowledge approved by the scientific community.

Ironically, statements in mathematics and logic are usually not falsifiable, because most mathematical propositions are (in principle) reducible to a set of axioms that are tautologically true. Axioms are true by definition, and therefore they cannot be proven false. So if one were to adopt falsificationism as the philosophy of computer

95 *If* computability is defined as it is defined in Church-Turing thesis. However, there is no way to mathematically prove the Church–Turing thesis to be true because it is *virtually* impossible to show that all machine-computable problems can be modeled by a Turing Machine. The Church-Turing thesis is a great example for illustrating falsificationism: From the falsificationist point of view, the thesis should be treated as true, like physical laws or mathematical axioms, unless falsified at some point of time. The Church-Turing thesis has withstood quite exhaustive tests this far. Yet, this is not to say that the Church-Turing thesis would be a complete account of what can be computed with machinery. There are contrary claims, too (see, e.g., Copeland & Sylvan, 1999; Copeland, 1997; Wegner and Goldin, 2003).

96 Lakatos, 1976

science, there would need to be another, complementary philosophy for the logico-mathematical parts of computer science.

Progress in Science—The Falsificationist Viewpoint

Progress in science, as the falsificationist would see it, could be summarized as follows[97]:

> (1) Science starts with problems[98] associated with the explanation of some aspects of the world; (2) scientists propose falsifiable hypotheses as explanations or solutions to the problem; (3) these hypotheses are criticized and tested—and some of them are eliminated, perhaps leaving one to dominate; (4) when this hypothesis that has successfully withstood a wide range of rigorous tests is *eventually* falsified, a new problem, hopefully far removed from the original solved problem has emerged. This new problems calls for the invention of new hypotheses etc.[99]

The process goes on indefinitely—a survival of the fittest, one might say. It can never be said of a theory that it is true, no matter however well it has withstood rigorous tests, but it can hopefully be said that a current theory is superior to its predecessors in the sense that it is able to withstand tests that falsified its predecessors. A more sophisticated form of falsificationism gives a more dynamic picture of science; a newly proposed theory will be acceptable as worthy of the consideration of scientists if it is more falsifiable than its rival, and especially if it predicts a new kind of phenomenon not touched on by its rival.[100] From the falsificationists' point of view, this is how scientific knowledge accumulates.

The term *ad hoc solution*, which stems from the falsificationist language, seems to be a dirty word among self-respecting computer scientists. This is a bit peculiar, since from the falsificationist viewpoint, contemporary computer science with all its nuances from the social sciences, psychology, and even humanities, might be classi-

---

97 Adapted from Chalmers, 1976:p.69.

98 Apparently, the problem here means either (a) a phenomenon that defies scientific explanation or (b) an unwanted phenomenon that the scientists are unable to overcome. A historical case of the former (a) form of "problem" is, "How can bats fly in dark without bumping into things, even though they have weak vision?" (Chalmers, 1976:p.70). A historical case of the latter is Albert Einstein's problem with expanding universe, which led him to include the cosmological constant to his theory of general relativity.

99 However, Popper's theory is criticized for having no counterparts in the history of science (Kuhn, 1996:p.77). One could say that even if Popper's account of science was good as a normative theory, it is not a good descriptive account of science because of the lack of historical evidence supporting it.

100 Popper, 1959:pp.91-92; Chalmers, 1976:pp.69,74-75.

fied as a pseudo-science. There are, though, positive uses of "*ad hoc*ness" too in computer science, viz., ad hoc networks and ad hoc database queries.

In the philosophy of science, the term *ad hoc* is often used in the following (falsificationist) context: A modification in a theory, such as the addition of an extra postulate or a change in some existing postulate, that has no testable consequences that were not already testable consequences of the unmodified theory are called *ad hoc* modifications[101]. These *ad hoc*-modifications are used to evade falsification of a theory[102]. For instance, Albert Einstein's (1879-1955) addition of a "cosmological constant" to his equations served only the purpose of Einstein's preferences (afterwards he called it the biggest blunder of his life). The cosmological constant did not bring in any new testable consequences that were not testable before the addition.

The notion of *ad hoc* is criticized for making much ado about nothing; *ad hoc*ness is a commonsense idea, and certainly not a technical notion[103]. But the real problem of Popper's *ad hoc*ness in computer science comes from equivocation on the term. The growth of scientific knowledge, as Popper saw it, is based on the supposition that if a hypothesis or theory is introduced to replace some refuted hypothesis or theory, the replacing theory should have *more empirical content* than the refuted one. If a replacing hypothesis or theory does not have more empirical content than the hypothesis or theory it would replace, it is *ad hoc*[104].

Many branches of computer science are areas where progress (or perhaps the growth of knowledge) is understood as an increase in the effectiveness of processes—processes that often include complex tradeoffs and numerous interrelated attributes. In these cases the correct phrases may not be *falsification of a theory*, but *stepwise improvement* and *proof of concept*. There are plenty of examples of stepwise improvement and proof of concept in, for example, user interface research and engineering-oriented computer science. Indeed, falsificationism was, in the first place, set in the context of natural sciences, and not in that of mathematics—this should be evident in that all the examples in Popper's book are from the natural sciences. Falsificationism is not that well suited for logic or mathematics (where conceptual aggregates are made true by series of definitions) or for a science focusing on, say, stepwise im-

101 See Popper, 1959:p.60.
102 Popper, 1959:pp.19-20.
103 For further discussion and an extensive analysis of this topic, see Bamford, 1993
104 Popper, 1959:pp.130-132.

provement of complex processes, especially if that improvement entails interrelated tradeoffs.

Criticism of Falsificationism

In *Against Method* and *Consolations for the Specialist*, Feyerabend criticized the Popperian doctrine[105]. Feyerabend raised two important questions: "Is it *desirable* to live in accordance with the rules of a critical rationalism?" and "Is it *possible* to have both, science as people know it and these rules?"[106]. From the viewpoint of computer science, the first question emphasizes things such as human interests, ethics, and scientific and human freedom[107]. For the computer scientist, the second question emphasizes questions such as whether progress is limited if the scientist concedes to falsificationism.

The crucial criticism of falsificationism at large came from the field of natural sciences, where falsificationism allegedly suits best. The criticism is that a theory cannot be conclusively falsified, because the possibility cannot be ruled out that some part of the complex test situation, other than the theory under test, is responsible for erroneous prediction. This weakness in falsificationism is pointed out in what is often called the *Duhem-Quine thesis*. Pierre Duhem (1861-1916), who was a physicist and a philosopher of science, is usually credited with the thesis, but the logician Willard van Orman Quine (1908-2000) universalized and secularized Duhem's narrow and religiously biased thesis[108]. Originally Duhem wrote, "*an experiment* [...] *can never condemn an isolated hypothesis but only a whole theoretical group*"[109]. That is, an experiment cannot falsify one aspect of a theory, but the whole theoretical framework. This is because one cannot be sure if abnormal findings result from a fault in the theory, the instrument, the theory about how the instrument works, or something in the test setting[110].

---

105 Feyerabend, 1993:pp.152-153; Feyerabend, 1970

106 Feyerabend, 1993:p.153.

107 "Human freedom" in the sense that Feyerabend means it–not the "academic freedom of will" but freedom from hunger, despair, and tyranny of constipated systems of thought such as academic establishment and scientific dogmatism (cf Feyerabend, 1993:p.154.)

108 *See* Quine, 1980; Fuller, 2003:pp.61-63. Note that although the thesis is called *Duhem-Quine Thesis*, Duhem's and Quine's original theses are quite different from each other.

109 Duhem, 1977:pp.183-188 (the first edition was published 1914 in French).

110 Note that Andrew Pickering's "Mangle of Practice" deals with exactly this problem. Pickering's theory is discussed in detail later in this thesis.

In short, Quine[111] stated that falsification survives only in the supposition that each statement, taken in isolation from other statements, can admit of "confirmation or infirmation"[112]. Quine's argument was that statements about the external world face the "tribunal of sense experience" not individually but only as a corporate body[113]. Basically, according to Quine, a researcher might hold on to *any belief* by adjusting the body of premises underlying the belief. When a scientific experiment contradicts a theory or a law, nothing logically tells whether the law, theory, observation, experiment, or interpretation of the experiment was flawed.

In analytic philosophy, this thesis is called the *underdetermination thesis*, which emphasizes the idea that any body of evidence can be explained by any number of mutually incompatible theories. In that case, the theory choice is "underdetermined" by the evidence. In his later works, Popper acknowledged the problem of the Duhem-Quine thesis[114], stating that when it is impossible to decide by experiment between two theories (or which part of the whole system is faulty), the techniques of measurement have to be improved first[115]. This does not eradicate the problem raised by the Duhem-Quine thesis, though. That is, it does not eradicate the problem that if one gets abnormal findings, one can never know which part of the whole test situation and theoretical framework is faulty.

From my point of view, underdetermination is not an especially interesting problem in computer science. It is much more interesting to note that for the majority of phenomena, there is an infinite number of possible modeling schemes, all incomplete (and none perfect). Because all the models of a certain phenomenon are defective, but defective in different ways, it could be said that the phenomenon is *underrepresented* by the models. The interesting question, from the computer scientist's point of view is, "Which one of these incomplete models should be chosen?". If a number of models model and predict the different aspects of a phenomenon well, the choice is ultimately subjective and beyond formalization. The crucial question becomes, "Which aspects is it more important to model?".

111 Quine, 1980, esp. chapter II: Two Dogmas of Empiricism:pp.20-46.
112 French for invalidation.
113 Quine, 1980:pp.37-41.
114 Popper, 1959 (2004 edition: Routledge, NY and London–in Routledge Classic Series):p.56.
115 Popper, 1959:p.108.

Another problem with falsificationism, a historical one, comes from Chalmers' accusation that had the falsificationist methodology been strictly adhered to by scientists in the past, those theories generally regarded as the best examples of scientific theories would never have been developed because they would have been rejected in their infancy[116]. In the early years of its life, sir Isaac Newton's (1643-1727) gravitational theory was falsified by observations of the moon's orbit. In early versions of Niels Bohr's (1885-1962) atomic theory there were inconsistencies with classical electromagnetic theory and observations. James C. Maxwell's (1831-1879) kinetic theory of gases was falsified by Maxwell himself[117]. Paul Feyerabend commented on Popper's theory, arguing that no new and revolutionary theory is ever formulated in a manner that permits one to say under what circumstances one must regard it as endangered: *Many revolutionary theories are unfalsifiable*[118].

Feyerabend claimed that falsifiable versions of most theories do exist, but they are hardly ever in agreement with the accepted basic Popperian statements: Every moderately interesting theory is easily falsified[119]. Moreover, he wrote that theories contain formal flaws, contradictions, and *ad hoc* adjustments. The Popperian criteria are clear, unambiguous, and precisely formulated. This would, Feyerabend wrote, be an advantage if science itself was clear, unambiguous, and precisely formulated. "*Fortunately, it is not*".[120] Chalmers claimed that concerning *any* example of a classic scientific theory, whether given at the time of its first proposal or later, it is possible to find observational claims which were generally accepted at the time and which were considered to be inconsistent with the theory[121]. As Lakatos' example of the history of Euler's formula[122] shows, mathematical theorems are not different. Feyerabend noted that Popperian criteria would eliminate science without replacing it with anything comparable[123].

116 Chalmers, 1976:p.91.
117 These examples are from Chalmers, 1976:pp.91-92. Chalmers gives also additional examples.
118 Feyerabend, 1975
119 Feyerabend, 1975
120 Feyerabend, 1975
121 Chalmers, 1976:p.91.
122 Lakatos, 1976
123 Feyerabend, 1975

Conceding to this criticism, Popper answered:

> *I have always stressed the need for some dogmatism: the dogmatic scientist has an important role to play. If we give in to criticism too easily, we shall never find out where the real power of our theory lies.*[124]

Thus he at once relinquishes the falsificationist thesis. If dogmatism has a positive role to play in an account of science that is based on ruthless criticism, a question arises: Where does one draw the line? If dogmatism in science can be either accepted or forbidden depending on the case, falsificationism loses the crispness and unambiguity that were the main motivations behind it in the first place. Again, although useful in practical science-making, falsification cannot constitute a tenable, overarching philosophy of science, because of its inherent problems.

The above-mentioned examples from software engineering: "GOTOS increase code entropy", "Strong typing reduces run-time errors", and "Good modularization reduces maintenance costs"[125], are good examples of statements that are not very clearly falsifiable (if they are falsifiable at all). Although intuitive, in the falsificationist paradigm these would be classified as pseudo-science and they could not constitute a part of computer *science*.

First, GOTOS may not always increase code entropy. An example of this is the debate that Edsger W. Dijkstra's article "GO TO Statement Considered Harmful"[126] raised. After Frank Rubin's response to Dijkstra[127], almost twenty different versions of Rubin's example were published in the following numbers of the same journal (*Communications of the ACM*)—many of them for GOTOS, many against. In addition, Robert L. Glass noted that when structured programming was introduced, it soon was accepted and taken into use in almost all organizations, although no research was ever performed to demonstrate that the claimed and hyped value of structured programming existed[128].

Second, to my knowledge there are no empirical experiments that tested the "strong typing-theory" with two versions of the same language, one strongly and one weakly

124 Popper, 1970:p.55.
125 Snelting, 1998
126 Dijkstra, 1968
127 Rubin, 1987
128 Glass, 2005

typed. Such an empirical experiment could reduce superfluous variables that could be connected with the theory. Furthermore, the distinction between strongly and weakly typed languages is exceedingly vague.

Third, the problem with the "modularization argument" is its use of the term *good*. *Good* is an exceedingly unambiguous term, and without specifically specifying what is considered to be good modularization, the value of the modularization argument is dubious.

From the falsificationist perspective, these three intuitively correct "folk theorems" are bad hypotheses, and without a reformulation in a falsifiable form and without empirical testing, they could not constitute a part of computer science. The problem of computer science, from the falsificationist perspective, is that in computer science there are quite few hypotheses that are measurable and falsifiable. Much of software engineering, computational modeling, computer visualization, scientific computation, artificial intelligence, cognitive science, and human-computer interaction may not meet falsificationist criteria. That the three statements above might not be *scientific statements* does not rule out the notion that if those statements were considered to be *engineering heuristics*[129], one might argue that they belong to the body of (non-scientific) engineering methods employed in computer science.

I argue that because of the insuperable problems in the demarcation of "science" and "pseudo-science", contemporary computer science with all its nuances would be classified as a pseudo-science, but so would many other disciplines that are nevertheless considered science today. That falsificationism would not be easily compatible with the plethora of methodologies of today's computer science is one matter. It is a totally different but legitimate matter to ask why something even has to be classified as being "scientific", as discussed in the end of this chapter. Before discussing that issue, a look at science and theories as institutional structures is taken.

129 See, e.g., Koen, 1987; Koen, 2003 for discussion on the engineering method and heuristics.

Science as a Contract: Thomas Kuhn's Theory

Though the world does not change with a change of paradigm, the scientist afterward works in a different world.[130]

In This Section:

- ✔ What is a *paradigm*, strictly speaking?
- ✔ How does scientific progress happen according to Kuhn?
- ✔ Are there paradigms in computer science?
- ✔ What are the characteristics of *scientific problems*?

In light of previous chapters, it seems that in terms of the quest for shedding light on the growth of scientific knowledge or on science at large, the philosophy of science has been too confined to either theories or observations (experimentation)—or the relationship between them. By focusing excessively on the content of science, the previously introduced accounts of science all fail to acknowledge important extra-scientific influences. Scientific activity happens in a complex sociocultural framework, and understanding how science works seems to require understanding those frameworks, too. Tying science up with human frameworks—be they institutional, societal, or interpersonal—is of primary importance for alternative approaches to science, because it distances science from determinism, universalism, and generality. (Yet, frameworks can become a dogma too—a burden to free thinking).

The first reason for this new view of *theories as structures* stems from the history of science. In the early 1960s, Thomas Kuhn noted that historical study reveals that the progress of major sciences cannot be explained by inductivist or falsificationist accounts. Kuhn's thesis was especially a criticism of falsificationism, which was the prevailing account of science at the time. Kuhn boldly confronted Popper: "...*what scientists <u>never</u> do when confronted by even severe and prolonged anomalies* [is give up the paradigm.] *Though they may start to lose faith and then to consider alternatives, they <u>do not</u> renounce the paradigm that has led them into crisis*."[131]

130 Kuhn, 1996 (3rd edition, 1st ed. 1962)
131 Kuhn, 1996:p.77, underlining added.

Furthermore, Kuhn criticized Popper's denial of the existence of any verification procedures at all. Kuhn hit on a weak point in Popper's theory, when he criticized falsificationism:

> *No theory ever solves all the puzzles with which it is confronted at a given time; nor are the solutions already achieved often perfect. On the contrary, it is just the incompleteness and imperfection of the existing data-theory fit that, at any time, define many of the puzzles that characterize normal science. If any and every failure to fit were ground for theory rejection, all theories ought to be rejected at all times.*[132]

This raises a question that is as difficult as the one raised by the underdetermination thesis, yet arguably more realistic (see page 60 above for a similar discussion on *models.*) Instead of asking how to deal with a number of theories that all explain a certain phenomenon fully, a more adequate question for a theory of computation involves, how to deal with a number of theories that all explain a certain phenomenon to some degree but that all are flawed in different ways. What makes this interesting is that if the currently dominant theories in academic computing are imperfect, on what grounds is their dominance justified—and on what grounds *could* it be? What kinds of arguments are used to raise certain theories and explanations of the computational world above other theories that explain the same phenomena, also imperfectly, but that are flawed in different ways? Answers for these questions are sought in Chapter Three.

Kuhn's theory on scientific revolutions indeed has some links to the underdetermination thesis. However, there is a more general argument to be made in the context of the theory-dependence of observations. Empirical findings are always expressed in the terms and language of some theory. Consequently, observational statements will be as precise and as informative as the language in which the language they are formed is precise and informative[133].

Alan Chalmers wrote that if there is a connection between how precise the meaning of a term is and what role it plays in a theory, then the need for coherence and structure in theories would seem to follow directly from it. He continued that definitions must be rejected as a fundamental way of establishing meanings because concepts

132 Kuhn, 1996:p.146. Compare this with Popper's quotation about dogmatism and the following criticism on page 62 of this thesis.

133 Chalmers, 1976:p.105.

can only be defined in terms of other concepts, the meanings of which are given[134]. If the meanings of some concepts are not known by other means, an infinite regress will result, Chalmers noted. (For instance, dictionaries are probably just huge circular definitions.)

From the Kuhnian point of view, scientists need theories that are taken for granted and not questioned all the time. Perhaps true, but it is not certain if *any* theory should be taken for granted. Special attention should be taken in interdisciplinary studies: Witness the eclectic use of cultural theories in the field of cultural HCI[135]. Single theories might not be detached from their frameworks.

Some of the vital questions about science for the purposes of this thesis are: "How are scientific structures created, selected, objectified, and internalized?" and "How *should* they be created, selected, objectified, and internalized?". These questions are also approached in Chapter Three.

Some Terminology Introduced by Kuhn

Not only did Kuhn's work bring into the philosophy of science a repertoire of new concepts, such as "*paradigm*[136]", "*paradigm shift*", "*normal science*", "*incommensurability thesis*[137]", "*scientific revolutions*", and "*Gestalt switch*[138]", it also introduced a new, sociological perspective on the progress of scientific knowledge. The *Structure of Scientific Revolutions* was one of the most influential books on the character of (natural) science in at least the second half of the 20th century, if not the entire 20th century. Interestingly, the book just provides a general account of scientific change in about 200 non-technical, very lightly referenced pages, in the manner of an extended encyclopædia entry—as the book was in fact originally conceived[139].

---

134 Chalmers, 1976:p.105.

135 For different meanings of culture in cultural HCI, see Kamppuri and Tukiainen, 2004.

136 This term was earlier used in linguistics, but brought to the philosophy of science by Kuhn.

137 There is not always a method of determining which paradigm is "true". Because two competing paradigms may offer totally different views on the structure of the world, and paradigms themselves cannot be compared *logically*, comparing paradigms is hard and sometimes impossible. (Even though it is clear that the idea of incommensurability comes from both Kuhn and Feyerabend, it is not clear who used the term first (see Feyerabend, 1970).)

138 Kuhn referred to the psychological meaning of gestalt switch as scientists' change in their world views, "*handling the same bundle of data as before, but placing them in a new system of relations with one another by giving them a different framework*" (Kuhn, 1996:p.85). However (contrary to the psychological meaning of the word), a scientist cannot switch back and forth between the frameworks (pp.111-114).

139 Fuller, 2003:pp.18-19; Reisch, 1991: Kuhn's text was originally commissioned as a monograph for the series *Foundations of the Unity of Science.*

Kuhn's ambiguity makes it problematic to read his work: Kuhn did not make it clear whether his account is a normative or a descriptive one. Even Popper conceded that Kuhn's normal science passes as a descriptive account of scientific practice[140], but not as a normative one. Paul Feyerabend implied that Kuhn may have wanted to leave himself a second line of retreat: Those who dislike the implied derivation of values from facts can always be told that no such derivation is made and that the presentation is purely descriptive[141]. Feyerabend implicitly alluded to David Hume, who centuries ago wrote a rationalization of why one cannot mingle descriptive with normative claims (referred to as the "is-ought problem")[142].

Kuhn replied to Feyerabend: "*Surely Feyerabend is right in claiming that my work repeatedly makes normative claims*"[143]. He continued, stating that the answer is that they should be read in "both ways at once". Kuhn made his view clear: "*If I have a theory of how and why science works, it must necessarily have implications for the way in which scientists should behave if their enterprise is to flourish*"[144]. Yet, this sounds a bit dubious from the Humean point of view. Following Kuhn's logic, if it were descriptively true that totalitarian science flourishes, then the normative statement—totalitarian science is desirable—necessarily follows from that. However, as Hume wrote, one cannot derive normative clauses from descriptive statements (and the is-ought problem prevails no matter which word one uses instead of *totalitarian*, be it *free*, *independent*, or even *progressive*[145]).

In Kuhn's vocabulary the term *paradigm* means both an exemplary piece of research and the blueprint it provides for future research[146]. In later editions of his book, Kuhn explicitly made a distinction between two senses of the term *paradigm.* In one sense, a paradigm is a *disciplinary matrix*, in another sense it is an *exemplar*. The first sense of the term, disciplinary matrix (which is the sense referred to in this thesis), is a sociological one, and it refers to the entire constellation of beliefs, values, techniques, and so on shared by the members of a given community. The second sense, which is philosophically deeper than the first one, denotes one sort of a given

140 Popper, 1970; Bloor, 1971
141 Feyerabend, 1970
142 Hume, 1739:Book III, pp.507-521.
143 Kuhn, 1970
144 Kuhn, 1970
145 The notions about problems with the concept of "progress" in computing come from discussions with Teppo Eskelinen (Eskelinen, Teppo; Tedre, Matti (forthcoming) *Three Dogmas of Computing*. An article manuscript).
146 Fuller, 2003:p.19.

constellation of concrete puzzle-solutions which, when employed as models or examples, can *replace explicit rules* as a basis for the solution of the remaining puzzles of normal science.[147]

Scientific Progress According to Kuhn

The theory of the structure of scientific revolutions could be described as a circle of eras and events in science (Figure 6). In the first stage of Kuhn's model (pre-science), most of the sciences have disagreeing coteries or competing theories of explanations. In the early stage of physical optics, for example, each scientist felt forced to build his or her field anew from its foundations[148]. During that time, the resulting books were often directed as much to the members of other schools as they were to the discipline. Kuhn claimed that this pattern is not unfamiliar in a number of creative fields today, nor is it incompatible with significant discovery and invention. In this early fact-gathering phase, every candidate for a paradigm, and all the facts that could possibly pertain to the development of a given science are likely to seem equally relevant. As a result, Kuhn claimed, this early (pre-science) fact-gathering is nearly a random activity, usually restricted to the wealth of data that lie ready at hand.[149]

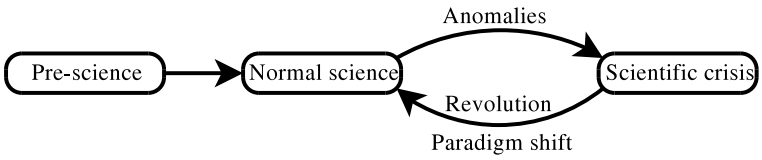


Figure 6: Kuhn's Model of Scientific Progress

Kuhn's portrayal of pre-science describes well the *ad hoc*[150]-solutions the early coders[151] frequently rigged. It is implicit in Grace Hopper's (1906-1992) memoirs that the early coders did not know that their practical work, intended to speed up coding, was laying foundations for a new science[152]. Moreover, the pattern of pre-

147 Kuhn, 1996:pp.175, 182-186 ("*disciplinary matrix*"), and 187-191 ("*exemplar*"). See also Kuhn, 1970: " *'disciplinary' because it is common to the practitioners of a specified discipline; 'matrix', because it consists of ordered elements which require individual specification*".

148 Kuhn, 1996:p.13.

149 Kuhn, 1996:pp.13-15.

150 "*Ad hoc*" in the meaning of *ex tempore* solutions that would not rely on any theoretical or philosophical framework (or paradigm, as far as Kuhn's vocabulary is concerned) but solutions that would work within the limits of the machines of the time.

151 In the early years of computing, before the 1950s, the term *programmers* had not been established (at least not in the United States), and those who programmed the computers were called *coders* (Hopper, 1978).

152 See Hopper, 1978. Grace Hopper is one of the early computer pioneers. She was the first coder for Harvard Mark I, formulated the first compiler, and worked with different kinds of compilers and compiler standards through her career.

scientific activity fits to some degree with contemporary computer science at large—or perhaps the confused state of some parts of computer science is caused by the ongoing redefinition of disciplinary boundaries. Nevertheless, it seems impossible to make researchers of core technologies of computing speak the same language if they are from areas of, say, e-commerce, human-computer interaction, algorithms, and virtual reality.

When science stabilizes enough, and the scientists working with the field have developed a strong enough consensus about the theories and tools of that particular science, that constellation of theories, beliefs, values, techniques, and so on, can be said to become *normal science*. In the course of their work, scientists every now and then come across phenomena that normal science cannot explain coherently; these are called *anomalies*. When enough anomalies accumulate, scientists cannot trust normal science anymore, and the discipline drifts into a *scientific crisis*. During the crisis state, a number of competing approaches that can explain some of the anomalies that led science to the crisis appears. Finally, one of the competing paradigms wins out, causing a *scientific revolution*. A complete *paradigm shift* happens when the opponents of the revolutionary paradigm are convinced or a new generation of scientists replaces the old one.

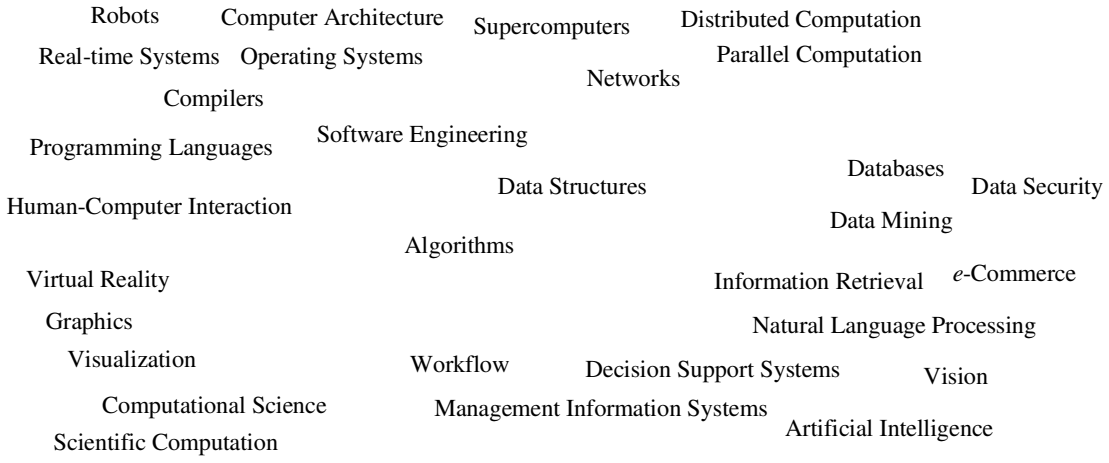


Figure 7: Core Technologies of Computing

I have constructed one kind of a visualization of the core technologies of computing, as listed by Peter Denning[153] (Figure 7). Different core technologies are highly interconnected, so every researcher would most probably come up with a chart that looks

153 Denning, 2003

different—and one researcher can make very different maps depending on the point of view he or she chooses to take. For instance, distributed and parallel computation are very closely linked with computational science, even though in this figure they are at opposing ends of the map. However controversial the arrangement may be, I prefer a chart to a list of words (atomic elements), or to an ordered list (unidimensional map) because the additional dimension gives more room for imagination and intellectual association.

The Characteristics of a Paradigm

In Kuhn's theory, if one of the pre-paradigm (or pre-science) schools seems better than its competitors, it will slowly become a paradigm for normal science. The school need not (and in fact never does) explain all the facts with which it can be confronted[154]. A mature paradigm is made up of general theoretical assumptions and laws and the techniques for their application. The scientists who adopt the assumptions, laws, and techniques of a paradigm practice normal science[155].

The focus of Kuhn's theory is evidently the natural sciences, and it is doubtful if Kuhn's theory sketches computer science well. I argue that regardless of whether one sees computing as a mature field or not, it is certainly hard to hypothesize that there would have ever been *anomalies* in the history of computing—this is discussed further in the following chapter. There have been changes in technology and theory, and some of those changes can even be called revolutionary, but in the history of computing, the revolutions have not originated from anomalies. Note that philosopher of mathematics Stewart Shapiro alluded that the Kuhnian concept of paradigm does not suit mathematics well—he noted that a contemporary mathematician needs little if any conceptual retooling to read Euclid's (325-265 B.C.) *Elements*[156].

The diverse uses of the term *paradigm* makes the term a bit hard to define; in his 1969 postscript to *the Structure of Scientific Revolutions*, Kuhn admitted that not only is his definition of paradigm sometimes circular, but also too ambiguous once in a while[157]. Thus the concept of paradigm (although given an initial definition on

154 Kuhn, 1996:pp.16-17.
155 Chalmers, 1976:p.108.
156 Shapiro, 2000:p.50.
157 Kuhn, 1996:pp.174-210. Margaret Masterman noted that it is curious indeed that before 1967 no attempt had been made to elucidate the notion of paradigm, which is *central* to Kuhn's whole view on science (Masterman, 1970–she refers to the first, 1962 edition of Kuhn's book.)

page 68 of this thesis) is further elaborated here, with some interwoven reflections on computer science.

First of all, Alan Chalmers noted that a paradigm has *explicitly stated fundamental laws and theoretical assumptions*[158]. Of the aspects of computing that some researchers take as the three fundamentals of computer science[159], (1) *theory* deals with fundamental laws and theoretical assumptions, (2) *modeling* (abstraction) promises rigorous methodology and some competing modeling schemes, and (3) *design* (engineering) relies partly on some laws and principles derived from physics and partly on heuristics from engineering[160].

Many of the human sides of computing; such as *e*-commerce, HCI, and visualization; lack uniform laws, design principles, or well-established and well-grounded guidelines. Even though there are suggestions for design principles and usability guidelines in HCI[161], not one of them seems to be universally accepted. Arguably, computer science at large, as a complex that is depicted in Figure 7, does not have a uniformly accepted paradigm; thus, in the Kuhnian framework, it is either in the pre-science state or undergoing a scientific crisis or scientific revolution. The latter seems quite improbable because it is hard to point out a bulk of accumulated anomalies in the history of computing that would have triggered such a crisis. After all, computer science is a young and interdisciplinary discipline—it is time-consuming (if possible at all) for such a discipline to find a precise form. There again, Kuhn's theory does not rule out that branches of computer science can hold to different paradigms.

According to Glass et al., in computer science there is a number of separate circles, coteries, that work within the frameworks of their distinct paradigms, and that are sometimes hostile to each others' research[162]. There is, according to Glass et al., minimal topic overlap between the subdisciplines of computing. If, as Chalmers claimed, a lack of agreement over fundamental principles is a characteristic of an immature science, then computer science might be immature, since the research paradigms of separate cliques in computer science do not match well or may even

158 Chalmers, 1976:p.109.
159 See, e.g. Wegner, 1976; or Denning et al., 1989.
160 See, e.g., Koen, 2003.
161 See, e.g., Dumas & Redish, 1999:pp.55-57.
162 See Glass et al., 2004, for an analysis of research in different coteries of computer science.

conflict with each other. Even though there is agreement on the overarching nature of some *theoretical* aspects of computing, most of the research on the core technologies of computing (Figure 7) is not focused on those theoretical topics and, therefore, the field as an umbrella entity cannot, if Chalmers' interpretation of Kuhn[163] is followed, be categorized as mature.

In defense of his own work, Kuhn remarked that in the history of science it has always turned out that some work within an era's normal science violates his characterization of a paradigm—but this does not render the concept of a paradigm untenable. Because scientists acquire their knowledge by solving standard problems and performing standard experiments, instead of learning the rules for problem-solving and experimenting, much of the normal scientist's knowledge will be *tacit*[164] instead of *explicit*[165]. From my viewpoint this is not as much a defense of normal science as it is a call for a psychological and sociological analysis of science. The questions that arise from the notion of the tenacity of normal science are, for instance, "Why is shackling innovation, imagination, and perspective with standardization understood as the tenacity of science?" and "Would explicating tacit knowledge (presuppositions, cultural references, biases, personal histories and beliefs, or microsocial relations) be detrimental to normal science?".

Second, Chalmers noted, paradigms also include *standard ways of applying the fundamental laws to a variety of types of situation*. For instance, the Newtonian paradigm will include methods of applying Newton's laws to planetary motion, pendulums, billiard-ball collisions and so on[166]. In computer science, *if* (1) there were rigorous, fundamental laws of modeling and design, *and if* (2) there would be standard ways of applying those laws, *and if* (3) they would also be applied in the computer industry, *then* software would not be so bug-ridden and hard-to-use as it often is

163 Chalmers, 1976:p.110-111.

164 "Tacit" in the sense of Michael Polanyi (Polanyi, 1964). According to Polanyi, tacit knowledge consists often of habits and culture that "we do not recognize in ourselves" (Chalmers, 1976:p.111-112). For instance, many mathematicians speak about tacit knowledge. According to Polanyi, good mathematicians often have a feeling of a result being wrong or right. He quotes, e.g., Gauss as having said "*I have had my solutions for a long time but I do not yet know how I am to arrive at them*" (Polanyi, 1964:pp.130-131). In Polanyi's opinion, tacit knowing is more fundamental than explicit knowing: "*We can know more than we can tell and we can tell nothing without relying on our awareness of things we may not be able to tell."*(Polanyi, 1964:p.*x*). In many instances, Polanyi stresses the importance of tacit knowledge also in formal reasoning, achieving scientific consensus, and even in premises of science (Polanyi, 1964:pp.129-131, 219-222,160-171).

165 Kuhn, 1996:pp.190-191.

166 Chalmers, 1976:p.109.

now. Either (1) such laws do not exist, (2) standardized ways of applying the laws do not exist, or (3) the laws are not generally applied according to those standards.

Third, according to Chalmers, paradigms include *instrumentation and instrumental techniques* necessary for bringing the laws of the paradigm to bear on the real world[167]. Instrumentation and instrumental techniques do exist in computer science. In fact, the technological side seems to be well-ahead of the human, societal, and user sides (examples of which can again be found in the areas of modeling and design).

Fourth, Chalmers noted that paradigms include some very *general, metaphysical principles* that guide work within a paradigm. For example, "any real-world computational instrument, no matter how complicated, can be presented as a Turing Machine", is such a principle (although perhaps not correct—take analog computers, for instance). One description of such principles in computing is given by Peter Denning[168]. Denning classified the "great principles of computing" in two categories: Principles of computation structure and behavior, which he called mechanics, and principles of design. Furthermore, there is also an important addition to Denning's principles—computing practices.

Denning's list of *principles of mechanics* includes topics such as algorithms, software engineering, data mining, programming languages, and human-computer interaction. The *principles of design* include abstraction, modules, separate compiling, layering, and reuse, amongst others. Denning's list has been criticized: For example, principles emphasizing cognitive modes[169], as well as principles emphasizing creativity and critical skills training[170], have been proposed as additions or alternatives to Denning's principles[171].

Fifth and finally, Chalmers noted that all paradigms contain some very *general methodological prescriptions* such as, "Make serious attempts to match your para-

167 Chalmers, 1976:p.109.

168 Denning, 2003. Denning's classification and definition owes a lot to the work he has done earlier with a group of top-class computer scientists (Douglas Comer, David Gries, Allen Tucker, etc.–see Denning et al., 1989)

169 Burnette, 2004

170 Liu, 2004 - however, Denning, in his reply (Denning, 2004b), reminds Liu that he had included "innovation" as part of the framework. It remains uncertain whether Liu and Denning have a mutual understanding of the concept of *innovation* or not. What *does* become clear, though, is that a universal consensus of the principles of computing has yet to find its form.

171 Denning, 2003

digm with nature"[172]. Examples of such methodological prescriptions in the field of computing might be, for instance, "use routines to improve the manageability of code", or "minimize complexity"[173].

Scientific Problems According to Kuhn

Kuhn took a firm stand on the term *problems*. Since the outcomes of normal research problems can be anticipated[174], often so accurately that what remains to be found is uninteresting per se, the method of achieving that outcome is often the interesting part, the unknown. "*Bringing a normal research problem to a conclusion is achieving the anticipated in a new way, and it requires the solution of all sorts of complex instrumental, conceptual, and mathematical puzzles*"[175]. Thus, Kuhn called this sort of research activity *puzzle-solving*; Sutinen and Tarhio[176] referred to this class as C-O-C problems, where the premises are known (C), the outcomes are anticipated (C), but the means of achieving the expected results are unknown (O). Kuhn did not regard practitioners of normal science as solving problems, but as solving puzzles. (The terms *puzzle* and *problem* are analyzed more deeply in the beginning of Chapter Three.)

The deliberate selection of the phrase "puzzle-solving" over "problem-solving" underscores the constrained nature of normal science. Steve Fuller argued that most scientists are narrowly trained specialists who try to work entirely within their paradigm until too many unsolved puzzles accumulate[177]. Also, Kuhn emphasized the game-like characteristic of normal science:

> *On the contrary, the really pressing problems,* e.g.*, a cure for cancer or the design of a lasting peace, are often not puzzles at all, largely because they may not have any solution. Consider the jigsaw puzzle whose pieces are selected at random from each of two different puzzle boxes. Since that problem is likely to defy (though it might not) even the most ingenious of men, it cannot serve as a test of skill in solution. In any usual sense it is not a*

172 Chalmers, 1976:pp.109-110.

173 McConnell, 1993:pp.114,397.

174 In normal science, even paradigm re-articulation cannot aim at *unexpected* results. Even though this seems like sort of a paradoxical language game (if one aims at unexpected results, one expects unexpected results, and thus they are not unexpected), it is actually a profound notion, explained further in Kuhn, 1996.

175 Kuhn, 1996:p.36.

176 Sutinen & Tarhio, 2001

177 Fuller, 2003:p.19.

> *puzzle at all. Though intrinsic value is no criterion for a puzzle, the assured existence of a solution is.*[178]

Thus, in Kuhn's theory, one of the things that acquiring a paradigm brings along is a criterion for choosing problems that (according to the paradigm) can be assumed to have solutions. In addition, those problems that are not reducible to the puzzle-form, may be rejected as metaphysical speculation, as a concern of another discipline, or sometimes as just too problematic to be worth the time. In computer science, rejections like those have for long been a part of the scientific turf wars as well as debates over the definition of computer science as a discipline, as discussed further in Chapter Three.

A paradigm can even insulate the scientific community from those *societally important* problems that are not reducible to puzzle form. It seems paradoxical that there is not much correspondence between the difficulty of a particular science's problem field, and the "hard" image of that science—quite the contrary[179]. The more ambiguity there is about the premises, methodology, and the goals, the more "soft" the science is, regardless of the difficulty of those problems. It seems that uncertainty, a characteristic which common sense would attribute to difficult problems, is typically attributed to "soft" sciences. Clarity, predictability, and an expected fit with existing knowledge (which common sense would attribute to simpler problems) are characteristics of a "hard" science. The fact that what is considered important may differ between communities and societies brings an extra flavor to socioculturally aware computing, asking, "On which terms and by whom are scientifically admissible problems chosen?".

The Foci And Limits of Normal Science

Clearly, computing or information and communication technologies in general are not panaceas that could or should be applied to *all* problems, but it is important to ask: "How are the limits of computing currently defined?" and "Are the boundaries in computing defined by the limitations of human intellect and possible human achievement, or rather dictated by normal science, and thus limited only to the narrow class of closed problems that can be reduced to puzzles?". This narrow descrip-

178 Kuhn, 1996:pp.36-37, underlining added.
179 See page 191 in this thesis.

tion of problems in normal science seems quite discouraging, and according to Kuhn, puzzle-solving activity is actually one of the reasons why normal science *seems* to progress so rapidly. Its practitioners concentrate on puzzle-problems that only their own lack of ingenuity should keep them from solving[180]. In fact, any piece of research may *seem* to progress rapidly if the solutions to the research problems are already known, and the only unknown is how to get there. If the goal is not known, progress is hard to measure.

A narrow focus is not merely detrimental to science. In Kuhn's theory, normal science owes its success to the ability of scientists to regularly select problems that can be solved with conceptual and instrumental techniques close to those already existing. Researchers conducting normal science *do not aim at novelties of fact or theory*, and, when successful, find none[181]. This sort of narrow focus enables researchers to concentrate their resources on well-constrained areas, which may lead to fast and deep development in that particular area. The researchers who practice normal science, engage throughout their careers in what Kuhn called "mopping-up" work[182]. The tendency of concentrating resources becomes apparent, from time to time, as "reverse salients"[183]. When some obstacles hinder progress in a number of topics, scientists focus their inventive efforts to overcome those critical problems, because solving them will free up a number of other areas for growth.

Kuhn wrote that there are three foci of normal science, which are neither always nor permanently distinct. The first focus is the class of facts a paradigm has shown to be particularly revealing of the nature of things. The goal of the researchers investigating this class is to expand on the phenomena with more precision and in a larger variety of situations. For instance, computational complexity is such class of facts, and the understanding of computational complexity is continuously expanded. Nowadays the number of named complexity classes is in the tens or hundreds, depending on how one counts them[184]. Perhaps in the growth of understanding in this

180 cf. Kuhn, 1996:p.37.

181 Kuhn, 1996:p.52.

182 "*Mopping up*" really *is* what Kuhn called normal science (see Kuhn, 1996:p.24). This choice of words evokes an implicit association of a few researchers walking ahead of others, finding and defining anomalies and problems, and others following behind, left with the work of the normal scientist: Proving theorems, testing predictions, and gathering more facts that support the paradigm.

183 MacKenzie and Wajcman, 1999 :pp.11-12.

184 Scott Aaronson has listed more than 460 complexity classes in *The Complexity Zoo*: `http://qwiki.caltech.edu/wiki/Complexity_Zoo` (accessed September 27th, 2006).

area lies the seed for the next scientific crisis that could eventually spur computing into a new era.

The second focus is on those facts that can be compared directly with predictions from the theories of the paradigm. Improving the scientists' agreement on facts within a paradigm or finding new areas in which this agreement can be demonstrated at all, presents a constant challenge to the skill and imagination of the experimentalist and observer. Using the predictions and theories (linear programming, heuristics, branch-and-cut) from the first class of facts has led to remarkable results in evading seemingly intractable problems. For example, the traveling salesman's problem has been solved for the 24 978 cities in Sweden[185].

The third focus of normal science is on the fact-gathering activities of science. It consists of empirical work undertaken to articulate the paradigm theory, resolving some of its residual arguments, and permitting solutions to new problems. Examples of this kind of work are determining physical constants, finding faster algorithms to analyze graph structures, or conducting usability tests. Furthermore, fact-gathering activities may incorporate ways of applying the paradigm to a new area of interest[186]. For example, computer science and computational models have been used in an astonishing number of studies conducted in a variety of fields. There are computational models in physics and chemistry, in meteorology and biology, economics and neuroscience; even models of culture[187] and sociological phenomena[188] exist. No wonder Stephen Wolfram called computational science "a new kind of science"[189]. Then again, the psychologist, economist, and the anthropologist may not yet be ready to reduce the phenomena in their fields to computation.

In conclusion, it seems that computer science is so fragmented that it can hardly be called a mature science from the Kuhnian point of view. Only the oldest branches of computer science have the characteristics that Kuhn connects with maturity. Those branches are mainly those strongly connected with mathematics and logic—algorithms, numerical methods, computational models, compilers, languages, and lo-

185 See the web site describing the solution at `http://www.tsp.gatech.edu/sweden/` (accessed September 27th, 2006)
186 Kuhn, 1996:pp.25-30.
187 Gabora, 1995
188 Brent et al., 2000
189 Wolfram, 2002

gic circuits[190]. Then again, criticism of Kuhn's theory raises doubts about whether the whole concept of mature normal science is valid anyway. This is discussed in the following section.

190 See Denning, 2003 for a list of all topics, mature or immature.

Revolutions in Science

When a piece does not fit the puzzle, is the piece or the puzzle faulty?

In This Section:

- ✔ What triggers a *paradigm shift*?
- ✔ What convinces scientists about the superiority of the new paradigm?
- ✔ Is the Kuhnian theory a relativistic theory?
- ✔ For whom is revolution a revolution?
- ✔ What are the weaknesses of Kuhn's theory?
- ✔ Was there ever a period of normal science in computer science or in any other science?

Steve Fuller claimed that a paradigm succeeds by monopolizing the means of intellectual reproduction[191]. However, new and unsuspected phenomena have been repeatedly uncovered and radical new theories have again and again been (and are) brought forth by scientists[192]. If a scientist cannot fit an observation or a whole theory to the dominant paradigm, it is generally seen as a failure of that scientist rather than as a sign of flaw in the paradigm. The problems that, according to the dominant paradigm, should have solutions but that seem to resist solution, are seen as *anomalies* (instead of falsifications of a paradigm as Popper would have liked to take them). The line between theories and empirical facts is exceedingly artificial, as seen in how anomalies are dealt with. In Kuhnian theory, when scientists encounter anomalies, they continue to explore them, and this exploration closes only when the *paradigm* has been adjusted so that the unexpected (i.e. the anomaly) has become the expected. In short, until the theory has been adjusted so that the new fact (anomaly) can be frictionlessly assimilated, the new fact, according to Kuhn, *is not quite a scientific fact at all*.[193]

As scientists find more anomalies that do not fit the paradigm, insecurity among researchers grows. The failure of theories and laws inevitably leads to a search for new ones. Often anomalies can be fit into the dominant paradigm because the theories are adjustable[194], but gradually the paradigm starts to lose status, and normal science drifts into a *crisis* state. When a paradigm has entered a state of crisis, several

191 Fuller, 2003:p.37.
192 Kuhn, 1996:p.52.
193 Kuhn, 1996:pp.52-53.
194 See, e.g. Kuhn's example of stretching the phlogiston theory to accommodate anomalies (Kuhn, 1996:p.71).

alternatives often emerge to replace the former paradigm. The history and philosophy of science indicate that a set of empirical data can always be explained in terms of more than one theoretical framework (see page 60 of this thesis about the underdetermination thesis, which argues that infinitely many theoretical constructions can always be placed upon a given collection of data).

So when scientists face anomalies, instead of immediately rejecting the dominant paradigm, a paradigm is declared invalid only if there is an alternative paradigm that explains some of the anomalies better than the dominant paradigm. A decision to reject one paradigm is always simultaneously a decision to accept another, and the judgment leading to that decision involves the comparison of both paradigms with empirical results *and* with each other, Kuhn argued[195]. In his later work, Kuhn acknowledged that falsification in fact plays an essential role in science: "*It is the strategy appropriate to those occasions when something goes wrong with normal science, when the discipline encounters crisis*"[196]. However, this is not to be read as a concession to the use of falsificationism in the era of normal science, but that it can be a tool to decide between competing theories during a state of crisis.

What makes clinging to an inadequate paradigm possible is that the proponents of the paradigm will arguably do whatever they can to hold their theory together. Kuhn claimed that they will devise numerous articulations and *ad hoc* modifications of their theory in order to eliminate any apparent conflict[197]. At some point, the proponents of the old paradigm cannot hold the paradigm together. Nonetheless, a paradigm shift cannot happen before a replacement candidate emerges:

> *Once* [, after the pre-scientific period,] *a first paradigm through which to view nature has been found, there is no such thing as research in the absence of any paradigm. To reject one paradigm without simultaneously substituting another is to reject science itself. That act reflects not on the paradigm but on the man* [sic]. *Inevitably he will be seen by his colleagues as 'the carpenter who blames his tools'*.[198]

From Kuhn's point of view, Fuller claimed, argumentation in science does more to "sway uncommitted spectators, especially if they are young or newcomers to the

195 Kuhn, 1996:p.77.
196 Kuhn, 1970
197 Kuhn, 1996:pp.77-78.
198 Kuhn, 1996:p.79.

field", and less to "change the minds of the scientific principals themselves"[199]. The sheer fact that newcomers in science have not yet personally invested in the old paradigm may be enough to make them open to a radical change in direction. Kuhn himself wrote,

> *The transfer of allegiance from paradigm to paradigm is a conversion experience that cannot be forced. Lifelong resistance, particularly from those whose productive careers have committed them to the old tradition of normal science, is not a violation of scientific standard but an index to the nature of scientific research itself. The source of resistance is the assurance that the older paradigm will ultimately solve all its problems* [...] *That same assurance is what makes normal or puzzle solving science possible.*[200]

German physicist Max Planck (1858-1947), who is the father of the quantum theory, worded the same thing in a more tragic form:

> *A new scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die, and a new generation grows up that is familiar with it.*[201]

Problems With Kuhn's Theory

It is hard to see that there would have been any scientific crises in the history of electronic computing—or that in some point of that history there would have been accumulating anomalies. Even the most decisive revolutions—the shift from mechanical to electronic computation and the conception of the stored-program computer—were not results of anomalies, yet perhaps of deficiencies in the dominant paradigm (these issues are dealt with in detail later in this thesis). Anyhow, Kuhn's *Structure of Scientific Revolutions* has come to dominate the history, philosophy and sociology of science. Arguably, it is often taken as the unproblematic foundation for its inquiries, almost as if the criticisms of Kuhn's position had never been made. Popper, for example, interpreted "normal science" as a moral failure rather than a successful adaptation strategy[202]. Because of Kuhn's dominant position, and despite the lack of correspondence with the history of computing, in the rest of section I take a standpoint towards Kuhn's theory.

---

199 Fuller, 2003:p.38.
200 Kuhn, 1996:pp.151-152.
201 Planck, 1949:pp.33-34.
202 Fuller, 2003:p.40.

Steve Fuller argued that there is a valid social critique of Kuhn's theory[203]. For Kuhn, activity is not proper science unless the community of inquirers can set its own standards for recruiting colleagues and evaluating their work. Societal or professional oversight cannot be found in Kuhn's theory of scientific change—as if scientists would be above society; untouchable, always correct. Fuller pointed out that one might think that such an élitist vision would have no place in today's world, where the costs and benefits of science loom as large as those of any public policy. Yet, Fuller continued, Kuhn managed to succeed simply by ignoring the issue, and that Kuhn left his readers with the impression—or perhaps misimpression—that, say, a multi-billion dollar particle accelerator is nothing more than a big scientific playpen. Certainly, the institutions funding science have their own interests in mind[204]—and, again, science is never value-free[205].

There also exists a problem with Kuhn's explicit statement distancing himself from relativism and his apparent relativist claims:

> *Later scientific theories are better than earlier ones for solving puzzles in the often quite different environments to which they are applied. That is not a relativist's position, and it displays the sense in which I am a convinced believer of scientific progress.*[206]

Contrary to this, it is explicit throughout Kuhn's work that scientific theories are incommensurable[207]—it is one his key claims. There are, Chalmers noted, apparently two roads that one can choose from. One could follow the path taken by sociologists and embrace and develop the relativist strand in Kuhn's thought, which among other things involves carrying out the sociological investigation of science[208]. Or, one could adopt the anti-relativist stance, but this would require an answer to the question of the sense in which a paradigm can be said to constitute progress over the one it replaces (and, incommensurability makes the comparison impossible)[209].

---

203 Fuller, 2003:pp.45-46.

204 I have discussed this in Tedre et al., 2003.

205 See page 34 in chapter 2.1.

206 Kuhn, 1996:p.206.

207 That is, because even the concept of "what is a problem" may change from paradigm to paradigm, it leaves ambiguity about what criteria there are to judge one theory superior to another–and how is scientific process measured. Moreover, as Feyerabend noted, Kuhn does not discuss the *aim* of science either (Feyerabend, 1970).

208 Chalmers, 1976:p.122-124.

209 My position towards progress is also discussed in a joint paper with philosopher Teppo Eskelinen (Eskelinen, Teppo; Tedre, Matti (forthcoming) *Three Dogmas of Computing*. An article manuscript).

The question raised by Kuhn, Feyerabend noted, is not *whether* there are limits to our reason; the question is *where* these limits are situated. Feyerabend asked: "Are the limits to our reason outside the sciences so that science itself remains entirely rational, or are irrational changes an essential part of even the most rational enterprise that has been invented by man?"[210].

Kuhn noted that the gist of the problem is that in order to answer the question "Is this development within the boundaries of normal science or is it revolutionary?", one must first ask, "For whom?"[211]. Sometimes, Kuhn continued, the answer is easy: Copernican astronomy (Nicolaus Copernicus, 1473-1543) was a revolution for everyone; the discovery of oxygen was a revolution for chemists but not for, say, mathematical astronomers unless they were interested in chemical and thermal subjects too. Later in this thesis (p.177) problems are classified according to their subject: Problems are grouped into three classes: *general*, *intimate*, and *limitary* problems. The aforementioned passage from Kuhn suggests that a similar kind of relativism applies to Kuhn's scientific revolutions too.

Although the three classes of problems are discussed in more detail later, a brief summary is in order. In computer science, *general* revolutions are those that affect the whole field of computing (and because of the nature of computer science as an intersection of theory, modeling, and design, theoretical and practical revolutions can be counted in.) General revolutions in computer science *may* include things such as the Church-Turing thesis (Alonzo Church, 1903-1995, Alan Turing)[212], the discovery of semiconductors, the construction of the stored-program computer, or the evolution of high-level programming languages. (The Church-Turing thesis and the stored-program computer can be considered to constitute the *stored-program paradigm*.) *Limitary* revolutions concern restricted fields of computing: the conception of mouse and graphical user interfaces (usability), the object-oriented paradigm (programming and software engineering), and the Chinese room argument[213] (artificial intelligence). Granted, it is hard to draw the line between general and limitary problems. Those two classes may overlap, and they may even be equal. This raises the question "*limitary* = *general*?".

210 Feyerabend, 1970
211 Kuhn, 1970
212 Turing, 1936
213 Searle, 1980

Finally, *intimate* revolutions consist of the enlightening experiences that one may gain when digging deeper into the discipline of computing. Those experiences might be classified as *tacit* knowledge (see page 72 of this thesis). For instance, digging deep into understanding GNU/Linux (and contributing to its creation) has been reported as a process where the technology comes second and personal enlightenment (i.e., intimate revolution), fun, and socializing are more important[214].

Coming back to Kuhn, what is noticeable—whether Kuhn is a relativist or not—is that many disciplines that often use Kuhn's *Scientific Revolutions* as their precept, often disregard the fact that Kuhn was a physicist. His examples and sources concern merely natural sciences such as physics, chemistry, and astronomy; and it has been claimed that, for instance, modern sociology lacks a consistent paradigm and consequently fails to qualify as a science in the Kuhnian sense[215]. On the other hand, Kuhn wrote that the typical development of a concept involves the initial emergence of the concept as a vague idea, followed by its gradual clarification as the theory in which it plays a part takes a more precise and coherent form—so perhaps the non-paradigmatic sciences need more time.

Popper and his followers identified another weakness in Kuhn's theory; Kuhnian normal science is, they claimed, a politically primitive social formation that combined the qualities of the Mafia, a royal dynasty, and a religious order[216]. Of course, for this critique to be valid, Kuhn's writing has to be read as a normative claim (see page 67 of this thesis). Kuhn stated that his theory has been misinterpreted so that "*the factors which determine what the scientific community chooses to believe are fundamentally irrational, matters of accident and personal logic. Neither logic nor observation nor good reason is implicated in theory-choice. Whatever scientific truth may be, it is through-and through relativistic*"[217].

In Fuller's words, normal science lacked the sort of constitutional safeguards that modern democracies take for granted: Those which regularly force politicians to be accountable to more people than just themselves[218]. Fuller concluded that scientists

214 Himanen, 2001:first chapter. Also visible in Linus Torvalds' preface to Himanen's book.
215 Chalmers, 1976:p.109.
216 Feyerabend too compares Kuhn's discipline to *organized crime*. He also raised the question of how paradigm changes happen; Kuhn's theory does not rule out killing the representatives of the *status quo* (Feyerabend, 1970).
217 Kuhn, 1970
218 Fuller, 2003:p.46.

should always be trying to falsify their theories, just as people should be always *invited* to find fault in their governments and consider alternatives—not simply wait until the government can no longer hide its mistakes. Yet there is a problem with lack of autonomity too. Sociologist C. Wright Mills (1916-1962) noted that if science is not autonomous, it cannot be a publicly responsible enterprise[219].

The juxtaposition of Fuller's and Mills' views is valid and deserves to be noted as one of the weaknesses in Kuhn's theory (too tight as well as too loose of a commitment to normal science would be detrimental to science). It seems that autonomity is a double-edged sword. Then again, it seems that the rivalry within the scientific community may take care of this weakness. It appears that nothing rewards a scientist better than finding fault in a fellow scientist's theory or experiment. And taking science into a state of crisis and then succeeding in convincing the scientific community of the superiority of one's own paradigm over the alternative ones makes a researcher seem immortal for a while. Kuhn himself responded; "*My critics respond to my views* [...] *with charges of irrationality, relativism, and the defense of mob rule. These are all labels which I categorically reject, even when they are used in my defense by Feyerabend*"[220].

Pluralism in Science

Pluralism, when understood as the presence and toleration of different viewpoints in science, is inherent in both Popper's and Kuhn's philosophies. Popper's pluralism is explicit in his statement "*There is no method peculiar to philosophy*"[221]. In Popper's philosophy, any falsifiable idea can be considered valid until falsified. This may be a noble idea, but Popper's idealism may not work in real life, for members of the scientific community may reject alternative hypotheses almost automatically (cf. Kuhn's notion of the stubbornness of the scientific community). In Fuller's opinion, Kuhn's pluralism emerges in the form of increasingly specialized domains of inquiry, each dominated by its own paradigm[222]—as is the case of contemporary computer science. In this sense Kuhn's description of science seems more accurate than Popper's—not as a normative but a descriptive account of computer science (not as a prescription but as a description).

219 Mills, 1959:p.106.
220 Kuhn, 1970; the Feyerabend's article Kuhn refers to is "Consolations for the Specialist" (Feyerabend, 1970).
221 Popper, 1959:p.*xix*.
222 Fuller, 2003:p.55.

In *Structure of Scientific Revolutions*, what prima facie looks like an apparent confusion, actually brings up a valuable point in Kuhn's conception of knowledge (the contradicting parts are underlined[223]):

> *Just because* [a paradigm shift] *is a transition between incommensurables, the transition between competing paradigms cannot be made a step at a time, forced by logic and neutral experience. Like the gestalt switch, it must occur all at once (though not necessarily in an instant) or not at all.*[224]

There are apparently two senses of knowledge that Kuhn includes in the sentence above. The first one is the (epistemologically) *objective* side of knowledge, which deals mostly with the paradigm in the sense of an *exemplar*. As I noted earlier in this chapter[225], an exemplar denotes one sort of a given constellation in the concrete puzzle-solutions which, employed as models or examples, can replace explicit rules as a basis for the solution of the remaining puzzles of normal science[226]. In modern science, especially in a discipline such as computer science, there is inevitably a diversity of beliefs among researchers. Many researchers are not aware of the relevance of other branches to their own work, and there exists a complexity of objective relationships between branches of the discipline independently of whether individuals are aware of that relationship[227]. When a paradigm shift occurs in one branch, it inevitably affects other branches, but this does not necessarily happen in an instant.

The second sense, the (epistemologically) *subjective* side of knowledge is apparent in, for example, Kuhn's term *gestalt switch*[228], which is a state of mind that is subjective to every researcher. If a gestalt switch occurs, it is a one-way street; one either accepts the new theory or not. Thus, a gestalt switch occurs all at once for every researcher who accepts the new theory.

223 Alan Chalmers has also contemplated this problem in Chalmers, 1976:pp.124-129.
224 Kuhn, 1996:p.150.
225 See page 67 of this thesis.
226 Kuhn, 1996:pp.175,187-191.
227 cf. Chalmers, 1976:p.126.
228 Kuhn, 1996:pp.111-114.

Paul Feyerabend made clear his negative opinion on the value of Kuhn's theory:[229]

> *Kuhn's ideas are interesting but, alas, they are much too vague to give rise to anything but lots of hot air. Never before has the literature on the philosophy of science been invaded by so many creeps and incompetents. Kuhn encourages people who have no idea why a stone falls to the ground to talk with assurance about the scientific method. Now I have no objection to incompetence but I do object when incompetence is accompanied by boredom and self-righteousness*.[230]

In addition, Feyerabend claimed that wherever one tries to make Kuhn's ideas more definite one finds that they are false. Feyerabend asked, "*Was there ever a period of normal science in the history of thought? No—and I challenge anyone to prove the contrary.*"[231]. My interpretation is that "periods of normal science" may be an illusion of history. The further back in history one looks, the less data about the era's scientific disputes is found. Is this a proof of greater scientific consensus or an indication of lack of data about disputes? Langdon Winner argued that even social constructionists disregard the possibility that there may be dynamics behind those revealed by studying the immediate needs, interests, problems, and solutions of specific groups and social actors[232]. The manner of writing history of science as series of definitive culmination points reinforces the common image of history of science and invention as a predetermined route that has passed through series of small revolutions.

Nonetheless, regardless of whether other sciences have or have not had periods of normal science, it is indeed hard to distinguish an era of normal science in the short history of computer science. There are very few things that would have been unanimously agreed upon at *any* point in the history of computer science (at least, not to the extent that it could be called a paradigm), as later chapters of this thesis show.

As a theoretical framework for the history of computer science, Kuhn's theory fails to acknowledge alternative strands of computing. Concentrating on the highlights of the history of computing reinforces the idea of the development of computing as a

229 Earlier Feyerabend called the normal science *to* revolution *to* normal science *to* revolution -pattern "[a pattern where] *professional stupidity is periodically replaced by philosophical outbursts only to return again at a 'higher level'* ". Then again, he began the same article by stating "*in my effort to be brief I do this in a somewhat blunt fashion*" (Feyerabend, 1970).

230 Feyerabend, 1975

231 Feyerabend, 1975

232 Winner, 1993

deterministic “route”. Also, it depicts history as a series of mini-fables with a number of revolutions. Following Winner's critique of social constructionism[233], a genuine understanding of computing requires an approach where alternative solutions and patterns of thought, both successful and celebrated as well as unsuccessful and discarded, are given due recognition. Thus, Kuhn's theory is not an appropriate theoretical framework of how computing works. Furthermore, Kuhn's theory sanctifies the arbitrarily set constraints of “scientific consensus”, and legitimatizes the oppression of “alternative” by “normal”, and consequently is not an eligible normative precept for computing either. A touch of Feyerabendian sensitivity about the subtle difference between science and beliefs is needed.

233 Winner, 1993

Freedom of Choice: Paul Feyerabend's Anarchistic Theory

I arranged [the text passages] *in a suitable order, added transitions, replaced moderate passages with more outrageous ones, and called the result "anarchism".*[234]

In This Section:

- ✔ What does *anarchistic theory of science* mean?
- ✔ What consequences does the incommensurability of theories have?
- ✔ Do pre-ordained "dogmatic" scientific frameworks inhibit progress?
- ✔ A critique of Feyerabend's anarchistic theory of science.

There is a large number of other philosophical accounts of science that would definitely strengthen this thesis, but I take the literature in this chapter to be a satisfactory sample of modern (1900s) mainstream philosophy of science. Surely, the works of, for instance, Deborah Mayo, William James (1842-1910), and Ian Hacking all have their strengths, but for the purposes of this thesis, the strengths and weaknesses of having a normative theory are now charted. Before concluding this section, the strengths and weaknesses of *not* having an overarching normative theory are addressed: That is, Paul Feyerabend's anarchistic theory of science.

Feyerabend's book *Against Method*[235] challenges all of the attempts to define an account of scientific method that would serve to capture "the special status of science". Feyerabend argued that there is no such method, and, indeed, that science does not possess features that render it *necessarily* superior to other forms of knowledge[236]. For me it seems that Feyerabend encouraged researchers (and educators) to do exactly what Kuhn warned them of—rejecting a paradigm without simultaneously substituting another[237]. Rejecting a paradigm does not, though, mean rejecting reason, intellect, or common sense (although one of Feyerabend's books is called *Farewell to Reason*[238]. In this chapter I discuss the reasoning behind rejecting paradigmatic science.

234 Paul Feyerabend about writing *Against Method* (Feyerabend, 1995:p.142).
235 Feyerabend, 1993 (orig. publication 1975). Paul Feyerabend and Imre Lakatos were both Karl Popper's students but whereas Lakatos radically modified Popper's falsificationism, Feyerabend rejected it completely.
236 Chalmers, 1976:p.150, note the emphasis on the word *necessarily*.
237 See Kuhn, 1996:p.79.
238 Feyerabend, 1987

The Anarchistic Theory of Science

Feyerabend's idealistic[239] thoughts were at the time of their writing (and still are by some) considered "dangerous and ill", "a bad influence", and "driving philosophers of science practically insane"[240]. My interpretation is that Feyerabend did to the scientific method what René Descartes and Zhuang Zi (莊子, ca. 369-286 B.C.) did to reality[241]. Even though the ideas of Descartes, Zhuang Zi, or Feyerabend are not exactly intuitive, they are so philosophically sound that they can not be dismissed easily. And though their ideas are not exactly practical, the doubts they raise should be noted and taken into consideration.

Feyerabend wanted to defend society from all ideologies—science included. Ideologies should, he wrote, be seen in perspective and not be taken too seriously—they should be read like ethical prescriptions which may be useful rules of thumb but which are deadly when followed to the letter[242]. Feyerabend joined with Kuhn in demanding a historical as opposed to an epistemological grounding of science, but opposed the Kuhnian (and C. Wright Mills') idea of the sociopolitical autonomy of science (which would certainly reinforce the authoritarian position of science). Feyerabend's position is well suited for the purposes of this thesis because it allows a sociocultural viewpoint of science to be taken, and criticizes the narrow view imposed by institutions, normal science, scientist authorities, dogmatic systems of thought, absolute truth, and rigid theories.

Moreover, Feyerabend had little sympathy for Kuhn's attempt to tie up history with theoretical ropes: "*A connection with theory just brings us back to what I at least want to escape from—the rigid, though chimaerical*[243] [...] *boundaries of a 'conceptual system'*"[244]. Feyerabend's thinking is congruent with (perhaps even central to) the anti-theoretical strands in postmodern thought[245]. From an ethnomethodological perspective, attempts to explain divergent computational systems in terms of another

239 See, e.g., Preston, 1997; Farrell, 2001: Feyerabend used to be an advocate of scientific realism, but *Against Method* was written around his "transitional" or "idealistic" period.

240 Feyerabend, 1975; Jacobs, 2003; Staley, 1999, respectively.

241 In the West, "I think, therefore I am", is as oft-quoted a phrase as "Zhuang Zi dreamt of being a butterfly" (莊周夢蝶) in Sinocentric cultures (see this translation of Zhuang Zi in Watson, 1964:p.45). Both refer to the same doubts about the reality and limits of knowing.

242 Feyerabend, 1975

243 A fanciful mental illusion or fabrication.

244 Feyerabend, 1993:p.213.

245 As Thomas, 1997, claimed.

conceptual system, say, in terms of academic computer science, may not be fruitful at all.

Even the most radical critics of society, Feyerabend argued, agree that people owe their increased intellectual freedom, vis-à-vis religious beliefs, to science[246]. To the extent one takes this to be true, science and enlightenment are one and the same thing; take, for instance, Galileo's (Galileo Galilei, 1564-1642) conflict with the Catholic church. However, the liberating character science once had has, as Feyerabend it sees, transformed to a dogma—an obstacle to free thinking:

> *Any ideology that breaks the 'hold a comprehensive system of thought has on the minds of men* [sic]*' contributes to the liberation of man. Any ideology that makes man question inherited beliefs is an aid to enlightenment. A truth that reigns without checks and balances is a tyrant who must be overthrown and any falsehood that can aid us in the overthrow of this tyrant is to be welcomed. It follows that 17th and 18th century science was an instrument of liberation and enlightenment. It does not follow that science is bound to remain such an instrument. There is nothing inherent in science or in any other ideology that makes it essentially liberating. Ideologies can deteriorate and become stupid religions.*[247]

From the perspective of any ethnoscience, Feyerabend's standpoint is compelling especially because it is explicit about the problematic nature of "clarification". Scientists tend to demand that any discussion of science has to be expressed in notions already familiar to them. This demand for clarification either makes it impossible to introduce—or at least narrows down the scope of—any really new theories or phenomena. The impossibility stems from incommensurability, which is closely related to inexpressibility.

By *inexpressibility* I refer to the fact that totally new concepts (or some aspects of those concepts) might not be expressible in the language of the old domain of science. Feyerabend wrote that if new concepts should be expressible using the old science, "*the course of investigation is deflected into the narrow channels of things already understood, and the possibility of fundamental conceptual discovery (or for fundamental conceptual change) is considerably reduced*"[248]. Like Kuhn, Feye-

246 Feyerabend, 1975
247 Feyerabend, 1975, underlining added.
248 Feyerabend, 1993:p.193.

rabend called this incommensurability, but unlike Kuhn, Feyerabend came to the conclusion that the concept of *truth* cannot override the freedom to choose one's frame of reference in any intellectual activity. Feyerabend's philosophy is essentially about freedom of choice.

The Critique of Feyerabend

The most obvious critique of Feyerabend stems from the fact that scientific theories are based on verifiable empirical evidence, and that makes them different from, for instance, religions that are based purely on faith. A question that has lately been debated in context of Darwinism *vs*. creationism is: Should school subjects be chosen without any scientific criteria at all? Thomas Kuhn agreed with Feyerabend in that both oppose the comparison of theories as representations of nature (as statements about "what is really out there") as well as the consequential quest for the ultimate framework of explanation[249]. The semantic conception of truth is regularly epitomized in the example:

> *"Snow is white" is true if and only if snow is white.*

If this would be merely a matter of the objective observation of nature, it would not present any insuperable problems, but as Kuhn noted, it also involves the assumption that objective observers understand the statement "snow is white" in the same way[250]. Kuhn wrote that it should not be taken for granted that the proponents of competing theories share a *neutral language* adequate for the comparison of such observational reports. Furthermore, contrary to the trivial "snow is white"-assertion, objectivity may not be obvious anymore if the assertion were, "GO TOS increase code entropy" or "Good modularization reduces maintenance costs"[251]. In the field of computing, a prominent computer scientist, Donald Knuth, has written about the dogmatism concerning the GO TO statement[252]. His concern was that if computer science goes far enough in purity, there would only be a dozen or so programs that are sufficiently simple to be allowable: Those programs would be pure, but of course they would not solve many problems.

---

249 Kuhn, 1970
250 Kuhn, 1970
251 Snelting, 1998
252 Knuth, 1974b

Both Kuhn and Feyerabend argued that there is no neutral language for science. In the transition from one theory to the next, words change their meanings or conditions of applicability in subtle ways. Linguists remind people that a perfect translation is never possible, not even with complex contextual definitions[253]. Though most of the same signs are used during different eras of science (e.g., force, mass, element, compound, cell), the ways in which some of them attach to nature has changed over time. Successive theories are thus, Kuhn and Feyerabend said, incommensurable[254]. For example, the explications of the limits of computing in the first part of the 1900s by Gödel, Turing, and Church[255] changed the face of logic and computing. There is a certain degree of incommensurability between "logic and computing" *before* Gödel's incompleteness theorems and "logic and computing" *after* Gödel's incompleteness theorems. Another example is that the creation of the first electronic computers created a number of new incommensurable concepts (as I argue later in this thesis).

Connected with the imperative of linguistic transformation, there also exists a demand for explanation or reduction of concepts—a demand that there should be continuity between concepts. Feyerabend dashed the notion that relativity is supposed to explain the valid parts of classical physics, hence it cannot be incommensurable with it, replying, *"why should the relativist be concerned with the fate of classical mechanics except as part of a historical exercise? There is only* one *task we can legitimately demand of a theory and it is that it should give us a correct account of the* world*."*[256] Denying the existence of a vocabulary adequate for neutral observation reports, Feyerabend at once concluded that there is an intrinsic irrationality to choosing one theory over another[257].

Applied to computer science the question becomes: "Why should John Presper Eckert (1919-1995) and John W. Mauchly (1907-1980) have ever been concerned with the fate of the fixed-program computer?". The stored-program computer opened an unforeseen number of new possibilities for the field of computing, but at the same

253 Feyerabend, 1970
254 Kuhn, 1970
255 Gödel, 1931; Turing, 1936
256 Feyerabend, 1970, italics in original.
257 Kuhn, 1970

time, created a number of concepts incommensurable with pre-von Neumann technologies.

The problem of reduction is found in another, technological form in the development of ICT: The demands for the downward compatibility of systems prohibit or hinder *new* conceptual or technological inventions. Yet, downward compatibility enables large scale data exchange and sharing, interconnectivity, convertibility, networking, teamwork and such[258]. From the Kuhnian viewpoint, downward compatible technology might be called "normal technology", but most of the occasional new development tracks that start from scratch could hardly be called "technological revolutions" - if anything, they might be called *technological shifts*.

Logic Cannot Explain Everything

Central to science—even to "revolutions"—is the belief that all subjects, however assembled, quite automatically obey the laws of logic. This dogmatic assertion, Feyerabend claimed, is neither clear nor is it true[259]. First of all, it is not *clear*, for there is not a single subject—*logic*—that underlies all the domains of nature, perception, the human mind, or society. Feyerabend wrote: "*There is Hegel, there is Brouwer, there are the many logical systems considered by modern constructivists*" (Georg W.F. Hegel (1770-1831); Luitzen E.J. Brouwer (1881-1966)). They offer not just different interpretations of one and the same bulk of logical "facts", but different "facts" altogether. This dogmatic assertion is not *true* either, Feyerabend argued, because there is not a single science, or other form of life that is useful, progressive, and in agreement with logical demands. Feyerabend continued that every science contains theories that are inconsistent both with facts and with other theories and that reveal contradictions when analyzed in detail. Only a dogmatic belief in the principles of the allegedly uniform discipline—Logic—will make the scientist disregard the situation. No matter how uniform the discipline of logic is, Gödel showed that the decision problem, a precisely stated problem, was unsolvable by mathematical logic[260].

I argue that even though computer science works with Boolean logic, the rules of that logic hardly apply to all of computing. Modern computing is not of the "input-

258 See, e.g., Succi et al., 1998.
259 Feyerabend, 1993:pp.195-196.
260 See, e.g., Wegner and Goldin, 2003; For the original, see Gödel, 1931.

process-output" form anymore, but a mesh of interrelated, interactive, complex, chaotic systems of actors. And many of those actors are not computers but people and natural phenomena that interact with computers through a variety of interfaces (ubiquitous or pervasive computing)[261]. I do not argue that computers are not discrete machines of logic or that they exhibit unpredictable characteristics, but I do argue that the real-world systems of which computers are a part, are non-discrete, non-quantifiable, sometimes intentional, uncertain, dynamic, and exceedingly complex.

Computational instruments and their semantic content are created for a purpose (they exist for some purpose), and they are imperfect, simplified, pruned models of reality. Programmers create these static models from a dynamic, complex, and infinite world. It seems implausible that one could make a clear cut around one aspect of a dynamic reality and paste it to a static model. Brian Cantwell Smith wrote that computer science once thought it could borrow *time* from the physical world without having to take on *space* and *energy*[262]. He noted that it worked for a while, but soon people realized what should have been predictable anyway: Time is not ultimately an isolable fragment (not an "independent export") of physics. I assume it would be equally naïve, or myopic, or hasty at the very least, to insist the same isolability about other aspects of the physical world.

Usually, the more faithfully a model depicts a dynamic phenomenon, the better that model is said to be. However, if the model is about an unisolable phenomenon in an infinite, dynamic, exceedingly complex, and interrelated world, one can improve the model infinitely—an awful task for a poor perfectionist. Finally, it would be simply absurd to insist that constantly fluctuating institutional facts[263] can be isolated, that is, to insist the social world is isolable.

Values in Science

Progress in science is often achieved by questioning the values and results of science and society, and by suggesting new, unpopular and unfounded values to replace them. This is how, for example, the Catholic church was replaced by science as the

261 I do not know which one is a lesser crime: To call this organization "*human-in-the-loop*" as, e.g., Karen Frenkel (Frenkel, 1988) did, or vice versa, "*computer-in-the-loop*", so I refuse to use either of the terms. The former one suggests that the human is subservient to the machine loop, or an external part of the computing machinery–the latter technomorphizes the human, as if the human brain or the external world would be running an infinite loop or process.

262 Smith, 2002b–Smith's *dynamic* is about norms; dynamic here is about the world of brute facts.

263 In terms of philosopher John Searle (Searle, 1996). Discussed later in this thesis.

prime authority on *the world* or *reality*[264]. The question Feyerabend raised was, "what values shall we choose to probe the sciences of today?"[265]. The answer is also imperative for the existence of any alternative views on science, for it specifically calls for courage and stubbornness (the latter cannot exist in a faithful falsificationist ideology):

> *A science that tries to develop our ideas and that uses rational means for the elimination of even the most fundamental conjectures must use a principle of tenacity together with a principle of proliferation. It must be allowed to* retain *ideas in the face of difficulties; and it must be allowed to introduce* new ideas *even if the popular views should appear to be fully justified and without blemish.*[266]

"*It seems to me*", Feyerabend wrote, "*that an enterprise whose human character can be seen by all is preferable to one that looks 'objective', and impervious to human actions and wishes*"[267]. This is how the opponents of technological determinism describe the current state of technology—impervious to human actions and wishes. But science and technology *are* human constructs, and so are all the standards they seem to impose upon scientists and people in general. It is good to be constantly reminded of the fact that science as people know it today is not inescapable and that people may construct a world in which it plays "*no role whatsoever*", Feyerabend noted.

The choice of a cosmological view is a matter of taste[268], and I see no reason why one should not be free to base his or her cosmology on the logic and axioms of mathematics. But one should also be free to reject logic and mathematics without the "justification", "rationale", or "proof" that academics so often demand. Proofs, rationales, and justifications are, after all, the core part of the mathematical and natural sciences' tradition and it would be absurd to demand a proof in terms of the system that is about to be abandoned. However, there is an important distinction to be

---

264 Granted, it has not been replaced everywhere: for example, creationism and many other explanations still lose out to Darwinism in many cultures.

265 Feyerabend, 1970

266 Feyerabend, 1970, italics in original. Make note, though, that Feyerabend's use of the term *principle* is somewhat ironic; "*Imre Lakatos loved to embarrass serious opponents with jokes and irony and so, I too, occasionally wrote in rather ironical vein. An example is the end of Chapter 1: 'anything goes' is not a principle. I do not think that 'principles' can be used and fruitfully discussed outside the concrete research situation they are supposed to affect*" (Feyerabend, 1993: preface, p.*vii*).

267 Feyerabend, 1970

268 Denzin and Lincoln, 1994:pp.99-100.

made. Although I am sympathetic to the Feyerabendian orthogonal repositioning of the philosophy of science, it is because it offers a legitimation for choosing unconventional corners for criticizing, diversifying, and perhaps cultivating knowledge about computation. It is not the case that I would *deny* the power of mathematical reasoning, the existence of brute facts, or the success of the currently dominant theory of computation.

Even though mathematics and logic are seen as the most pervasive along the whole spectrum of scientific domains, producing valid knowledge does not *necessarily* need to conform to the formalistic rules of mathematics and logic. Feyerabend argued that methodology is full of empty sophistication, and so it is impossible to fight the simple errors at the very basis of methodology: "It is like fighting the hydra—cut off one head and eight formalizations take its place"[269]. Thus, Feyerabend wrote, "when sophistication loses content then the only way of keeping in touch with reality is to be crude and superficial". It would be an error to argue that one can achieve valid knowledge about computers and computing only through formal approaches or only via the scientific method. The richness of the world defies even the most sophisticated of methodologies, and when methodology becomes a hindrance to progress, one must be able to abandon methodology without a proof.

The Worst Enemy of Science?

Feyerabend has been named "the worst enemy of science"[270], but I have a very different view on his work. My reading of Feyerabend is that his sarcastic anarchism is an attempt to defend science from rigid dogmatism—indeed, instead of attacking science, Feyerabend was defending it by attempting to break the chains that confine (incommensurable) free thinking, innovation, and novel theories. Perhaps more, Feyerabend's anarchism does not seem as much a belief of his, as an observation of progress in the history of science. My interpretation of Kuhn's description of scientific revolutions is that in practical science-making, *anything goes*[271].

---

269 Feyerabend, 1970

270 Preston, John; Gonzalo Munévar; David Lamb (eds.) (2000) *The Worst Enemy of Science? Essays in Memory of Paul Feyerabend.* Oxford University Press: New York, USA.

271 "Anything goes" (Feyerabend, 1993:p.14.) is, unfortunately a phrase that can easily be misquoted and used against Feyerabend. Feyerabend himself commented that he never meant that any scientific theory is as good as any other (Horgan, 1996:p.52).

Naturally, there is plenty of criticism of Feyerabend's position. Imre Lakatos thought that Feyerabend's epistemological anarchism combined the worst tendencies in both Kuhn and Popper[272]. But most of the critique of Feyerabend stems from the degree to which his notion of freedom is entirely negative[273]. Chalmers criticized Feyerabend's (naïve) idealism:

> *It is ironic that Feyerabend, who in his study of science goes to great lengths to deny the existence of theory-neutral facts, in his social theory appeals to the far more ambitious notion of an ideology-neutral State. How on earth would such a State come into an existence, how would it function and what would sustain it?*[274]

Chalmers wrote that criticizing Feyerabend for setting his views on science in an individualist framework involving a naïve notion of freedom is one thing, but concedes that coming to grips with the details of the case Feyerabend makes against the scientific method is another[275]. Finally, it should be clarified that I do not see that Feyerabend was a "betrayer of truth", as is sometimes portrayed, but, as Horgan put it: He believed very much in science, in fact, his skepticism was motivated by his belief[276]. Feyerabend, in a rare interview, reported that "anything goes" has never meant that any scientific theory is as good as any other—but that a researcher needs to be an opportunist, accommodating to different situations with different methodologies: "You need a toolbox full of different kinds of tools. Not only a hammer and pins and nothing else."[277]

272 Fuller, 2003:p.12.

273 Chalmers, 1976:p.157. *Negative freedom* means merely freedom from constraints (which does not guarantee equality), whereas *positive freedom* means that the necessities of freedom (education, opportunity, material resources, or such) are also present. Those two terms were initially introduced by Isaiah Berlin in his famous inaugural lecture *Two Concepts of Liberty*, delivered before the University of Oxford, October 31st 1958.

274 Chalmers, 1976:p.159.

275 Chalmers, 1976:p.159.

276 Horgan, 1996:p.33.

277 Horgan, 1996:p.52. Note that this is very similar to the concept of *researcher-as-bricoleur* that Denzin and Lincoln present (Denzin and Lincoln, 1994:pp.2-3).

2.2.Modern Approaches to the Society-Technology Relationship

In all these decisions about science [...] *there is a special set of features that has worried me since the last war. It is this—that, partly because of their inherent nature, partly because of our general education, they are made by tiny numbers of people.*[278]

There are a variety of terms referring to the diverse viewpoints of the studies of science and technology, and most of those terms are overlapping and interrelated. One of the terms that has lately become common is *science and technology studies* (STS). Traditionally, as Andrew Pickering noted, the core fields of STS are history, philosophy, and the sociology of science[279]. However, here science and technology studies is understood in its more recent meaning as an interdisciplinary research topic that encompasses a number of approaches such as feminist, hermeneutic, and critical approaches; a number of theories such as the actor-network theory, social construction of technology, and system theory[280]; and a number of fields such as anthropology, history, sociology, philosophy, and political science.

Science and technology studies is a fruitful area for this thesis for three reasons. First, in STS it is generally acknowledged that science, society, and technology are strongly interrelated subjects, and that they should be studied together. Second, in STS it is acknowledged that science and technology are multidimensional phenomena, and thus STS employs a variety of research approaches for understanding those phenomena better—for instance, approaches from psychology, history, sociology, and philosophy. Third, STS is not only restricted to academic aims but it has also practical aims. In a society where science and technology play a vital role, citizens, policymakers, business, and financiers need a firm and reliable foundation for their interpretations, decisions, investments, values, and ethics. Similarly, the recognition of the social aspects of computer science should also offer new viewpoints to stakeholders and be influential in the development of computer science and technology.

Note that there is a small number of people around the world who work in the field of the *sociology of computing*, but there seems to be no agreement whatsoever about the substance of the field. The studies seem to concern, for instance, ethics, human-

278 Snow, 1966
279 Pickering, 1995:p.1.
280 Bijker and Law, 1992:pp.12-13.

computer interaction, the organization of systems that include computing, the history of computing, the political aspects of computing, social impact, and innovation in the area of computing. None of those seem to aim at refining or examining theories or technologies of computing, but examining the implications of computing on society instead.

Throughout this thesis, I refer often to the concept of *society*, which is, as historian of technology Thomas Hughes recognized, a concept that is highly abstract[281]. Not only is there temporal disparity between societies, like the temporal disparity between the twelfth-century French society and the twenty-first century French society, but also spatial disparity, like the spatial disparity between contemporary Japanese and Saudi Arabian societies. Proposing a strict definition of *society* is neither necessary nor practical for the purposes of this thesis, and therefore, following Hughes, *society* is referred to as a *world made up of institutions, values, interest groups, social classes, and political and economic forces*. Even though this characterization is sketchy, it provides for unconventional forms of society, such as a network society, where the critical social structures and its dominant functions are organized around electronic information networks, as argued by sociologist Manuel Castells[282].

Another widely used term in this thesis, one that is notoriously hard to define, is *culture*[283]. Culture is often principally seen as a mental phenomenon (i.e., consisting of values, ideas, mental models and so forth) but it also includes material and social phenomena, which should be considered to be equally important parts of culture[284].

Cultural theories have a number of controversial characteristics. Claudia Strauss and Naomi Quinn wrote that culture can be motivating or unmotivating, enduring (in persons and across generations) or transient, shared or personal, and thematically unifying or disparate.[285] They noted that cultures are not bounded and separable: People share some experiences with other people who listen to the same music or watch the same television shows; people share other experiences with people who do

281 Hughes, 1994
282 Castells, 1998:pp.500-507.
283 I have discussed this topic in a joint paper with Minna Kamppuri and Markku Tukiainen in Kamppuri et al., 2006 and Kamppuri et al., 2006b.
284 Kamppuri et al., 2006; Kamppuri et al., 2006b
285 Strauss & Quinn, 1997:p.4.

the same work they do; and people share still other experiences with people who have had formal schooling like they do—even if they would live on opposite sides of the world.

In computer science, perhaps the most widely used definition of culture is the one by Geert Hofstede[286], in which he categorized cultures according to four *dimensions*. Even though Hofstede's research is impressive, mainly because of its substantial sample size, his outlook on culture is not well-suited for the needs of this thesis—mainly because of its rigidity and his inclination to oversimplify. Another *dimensions* alternative to Hofstede, by Fons Trompenaars,[287] is equally unsuitable for the same reasons.

Instead of characterizing cultures by a small number of fixed dimensions, I agree with C. Wright Mills in that the human variety includes not only large aggregates such as *working class* or *virtual communitarians*, but also the variety of individual human beings: "an Indian Brahmin, a pioneer farmer of Illinois, an eighteenth-century English gentleman, an Australian aboriginal, a Chinese peasant, and a Bolivian politician". For Mills, to "*write of 'man' is to write of all men and women—also of Goethe, and of the girl next door*"[288]. Mills' view is aligned with Strauss and Quinn's view of culture: Strauss and Quinn wrote, "*This makes each person a junction point for an infinite number of partially overlapping cultures.*"[289]

There have probably been critics and advocates of technology ever since the first technologies were developed. For instance, Carl Mitcham noted that skepticism about technologies is hinted at already in the ancient myths of Prometheus, Hephaestus, and Daedalus and Icarus[290]. However, because *technoscience* in this thesis concerns information and communication technologies (ICT), this chapter focuses on authors who have written about technology and society after the 1970s. Around 1970 a large number of technological, social, and intellectual changes took place. Some of those changes were the emergence of the home computer[291], the initiation of

286 Hofstede, 1997. A search for "geert hofstede" in the ACM Digital Library yielded 214 results. (Nov. 29th 2005)
287 Trompenaars & Hampden-Turner, 1997 (orig. 1993)
288 Mills, 1959:p.133.
289 Strauss & Quinn, 1997:p.7.
290 Mitcham, 1994:p.277.
291 Ceruzzi, 1999

the strong program in the sociology of knowledge[292], the breakthrough of social constructionism[293], the emergence of HCI as a discipline[294], and so forth. (It has even been argued that *the* significant breakthroughs of the *information technology revolution* took place in the United States in the 1970s[295].)

In the following section different viewpoints of technoscience are introduced. First, I argue that although many facts are socially constructed, not all of them are. Second, I discuss three cruxes in the debate between social constructionists and realists, and take positions towards those cruxes. Third, I examine different approaches towards technological determinism. Fourth, I discuss the concepts of progress, value-free technology, and the measurement of technological progress, and argue for the standpoints of this thesis. These discussions revolve around concepts such as inevitability and contingency, the inherent structures of the world, the stability of science, technological determinism, and the impact of technology on society.

292 Bloor, David (1976) *Knowledge and Social Imagery*. In this thesis, a more recent work by Barry Barnes, David Bloor and John Henry: *Scientific Knowledge: a Sociological Analysis* (Barnes et al., 1996) is utilized more.

293 Berger & Luckmann, 1966. There are a large number of other authors that could be named in the context of social constructionism, such as Lev Vygotsky and Karl Mannheim. Berger and Luckmann's role was, however, crucial because it introduced a number of concepts and a coherent framework for the field of sociology.

294 The shift from programmers turned to end users during the 1970s (Grudin, 1990; Baecker et al., 1995:pp.35-47).

295 Castells, 1996:pp.53-59.

Sticking Points Between Realism and Constructionism

The real danger is not that computers will begin to think like men [sic],
but that men will begin to think like computers.[296]

In This Section:

- ✔ What is *constructionism*?
- ✔ Is the position in this thesis a relativist one?
- ✔ What ontological position is taken in this thesis?
- ✔ Hacking's three sticking points: Contingency, nominalism, and explanations of stability.
- ✔ What is objectivity and subjectivity?

One of the most influential schools of thinking in contemporary (1960-present) sociology, philosophy, STS, cultural studies, and many other disciplines, is the *social construction of reality*, crystallized and popularized by Peter Berger and Thomas Luckmann in a book by the same name[297]. Basically, the followers of Berger and Luckmann believe that there is no external, objective reality but that people to some extent create their perceived reality themselves (I have discussed Berger and Luckmann's work in more detail elsewhere[298]).

The term *social constructionism* is used in so many different meanings and in so many different contexts, that it is meaningless to use it without a thorough explanation of what the term means for a particular author, in a particular work. Constructionism in education is different from constructionism in philosophy. For example, the philosopher Ian Hacking listed more than twenty very different titles "Social construction of *X*" that he found from the library catalog[299]. The *X* in Hacking's list includes, for instance, *danger*, *illness*, *nature*, *facts*, *knowledge*, *women refugees*, and *quarks*. Hacking argued that the term *social construction* has become a code. Those who use it favorably deem themselves rather radical—those who trash the phrase declare that they are rational, reasonable, and respectable[300].

Some clarification of basic terminology is necessary before this topic can be discussed meaningfully. The terms *constructionism* and *constructivism* (and *construc-*

296 Sydney J. Harris in Eves, H. (1988) *Return to Mathematical Circles*. Prindle, Weber, and Schmidt: Boston, USA.
297 Berger & Luckmann, 1966
298 Tedre, 2002
299 Hacking, 1999
300 Hacking, 1999:p.*vii*.

tionalism) are often used interchangeably, but the words have different meanings in different contexts. Traditionally, Berger and Luckmann's followers use *constructionism*, but when a constructionist argument is made in other fields, the choice of the term varies, sometimes implying a different emphasis. For the sake of clarity, in this thesis the term *constructionism* is used even where the term *constructivism* has been used in the material I reference (*constructivism* is used in, e.g., the philosophy of mathematics[301]). Other authors, such as Kenneth Gergen, have come to the same decision: "The term *constructionism* avoids [...] various confusions and enables a linkage to be retained to Berger and Luckmann's seminal volume, *The Social Construction of Reality*"[302]. As Hacking noted, even though there are differences of emphasis, all the construct*isms share themes and attitudes of "*things are not what they seem*", iconoclastic questioning, and finally come back to the dichotomy between appearance and reality already set up by Plato[303].

A possible disillusionment: In this thesis, I do not take the position that *everything* is socially or individually constructed. In this sense, my ontological position in this thesis is closer to John Searle's realism than Ernst von Glasersfeld's radical constructionism[304]. Although Berger and Luckmann pushed sociologists of knowledge to understand how the common reality of a society is constructed[305], their pioneering work left many things open. Searle's work goes into specifying that there indeed is a reality that is socially constructed, a reality of "institutional facts"—but that there is also a reality of facts that are not dependent on human agreement, a reality of "brute facts"[306]. Searle dissociated his realist account of social construction (note the contradiction) from anti-realists who deny any ontological objectivity[307]. The ontology that Searle's thinking is based on, is simple:

> *We live in a world made up entirely of physical particles in fields of force. Some of these are organized into systems. Some of these systems are living*

301 Shapiro, 2000:p.184.

302 Gergen, 1985, italics in original.

303 Hacking, 1999:pp.44-49. Plato made a distinction between perfect, abstract ideas and concrete, imperfect, perceived reality. This was expanded on further by Immanuel Kant, in a distinction between the noumenal and phenomenal worlds. I have discussed this division further in Tedre, 2002.

304 See, for instance, Glasersfeld, 1995:p.1: "[...] *all kinds of experience are essentially subjective, and though I may find reasons to believe that my experience may not be unlike yours, I have no way of knowing that it is the same*." ... and Glasersfeld is back to *cogito, ergo sum*–interesting, but, from my point of view, implausible.

305 Berger & Luckmann, 1966:p.30. Note, however, that sociologists of knowledge often explicitly reject the charge of anti-realism (Bloor, 1996).

306 Searle, 1996:p.2.

307 Searle, 1996:pp.196-197.

> *systems and some of these living systems have evolved consciousness. With consciousness comes intentionality, the capacity of the organism to represent objects and states of affairs in the world to itself.*[308]

However, a vast number of questions follows from this simple ontology. Some of the questions Searle raised concern social facts: "How can one account for social facts such as 'money', 'sentence', or 'restaurant' in this ontology?". Those social facts exist because people want them to exist, but there is no physical, biological, or chemical equivalent or explanation for them. Even hard-line critics of postmodernism, such as cognitive scientist Steven Pinker, agree that some categories really are social constructions: They exist only because people tacitly agree to "act as if they exist"[309]. Pinker listed money, tenure, citizenship, decorations for bravery, and the presidency of the United States as socially constructed categories. I take that computer science (computation at large, and especially computing technology) is an outstandingly difficult field for ontologists, and both extreme relativism as well as extreme realism are especially difficult positions in this difficult field. For instance, computers reproduce a variant of the mind-body problem in the form of software-hardware.

Another term that needs clarification is *pluralism*. Pluralism, when understood as the acknowledgment that reality is expressible in a variety of symbol and language systems, is evident in constructionist thinking[310]. Contrary to realist accounts of reality that take there to be a *real world independent of human mind*, some constructionists deny objective knowledge and truth entirely, and claim that there is no "real world" but only subjective, human-made perspectives, shaped by intentional minds[311]. Also the term *intentional* has specialty meanings.

Firstly, *intentional* refers to human intentions; to the ambitions, goals, and motivations of people. Secondly, *intentional* also refers to the meaning of the word in studies of the cognition, that is, "aboutness". In this sense, it refers to the capacity of the mind to represent objects and states of affairs in the world. In contrast to intentional mental states, there are human mental states that do not always refer to objects or states of affairs, and that are not intentional when they do not have that reference.

308 Searle, 1996:p.7.
309 Pinker, 2002:p.202. Note Pinker's choice of words that implies a difficulty in the definition "to exist".
310 cf. Schwandt, 1994
311 Searle, 1983:pp.1-6.

For example, anxiety and nervousness are not always "about anything"[312]. John Searle listed some mental states that can be intentional, that is, about or of something: belief, fear, hope, desire, love, hate, aversion, liking, and so on.

However, pluralism does not *need* to be accompanied by anti-realism, and often it is not. Although philosophically interesting, extreme anti-realist stands are not very useful for the purposes of this thesis because they do not accommodate some commonly accepted empiricist premises—for example, that human senses, based on certain biological processes, work in the same way for almost everyone—with some minor individual variations (that are also often explainable in terms of biology, chemistry, and physics).

The constructionist views range from acknowledging simple "everyday constructionism" to radical constructionism (and the different views certainly do not form a nice continuum, but a complex system of different combinations of different aspects, ideas, and ideologies). A simple example of everyday constructionism, or minimalist constructionism, given by Thomas Schwandt, is merely agreeing that knowing is not passive—a simple imprinting of data on the mind like on a computer hard drive—but active: Believing that mind does something with the impressions to be stored, at the very least forming abstractions or concepts[313]. Radical constructionists, according to Schwandt, reject speaking about knowledge as corresponding to, mirroring, or representing an external world. Neither of the constructionist extremes is very useful for the purposes of this thesis—the former for its lack of depth (or intellectual insight) and the latter for its total rejection of realism. It would be hard for a minimalist constructionist to explain the nature of, for instance, money, and the extreme constructionist position, when it includes relativism, has a problem with its own justification[314].

312 Searle, 1983:p.4.

313 Schwandt, 1994

314 As John Searle noted, one cannot even state relativism without denying it (Searle, 2001). Suppose one says, "there are no absolute truths". Is this argument supposed to be interpreted so that it applies to itself or not? Searle noted that either way this argument leads to inconsistency. (1) If one says there are no absolute truths except the truth that there are no absolute truths, then one has already allowed for an exception without giving a reason why there would be no other exceptions. (2) If one says there are no absolute truths including the claim that there are no absolute truths, then the argument is a contradiction in terms. See also Searle, 2001 for a more sophisticated refutation of relativism.

Not Everything Is a Social Construct

I noted above that I take the position that there are socially constructed things that do not exist independently of human beings, as well as things that exist regardless of any human beings. I take it that the piece of matter called the Earth exists no matter what people think about it. Yet I also take it that there are things that do not exist without intelligent beings, and that cease to exist when there are no intelligent beings anymore—take language, for example. Because there is a number of schools that draw the lines of social constructionism differently, a number of those schools is discussed here.

It is quite obvious that, for example, "social classes" are hardly anything else but socially constructed concepts. Yet there are much more difficult questions, such as, "Are natural sciences social constructs?", "Is mathematics a social construct?", and "Are computer science or computing at large social constructs?". At least for me, first it seems like a paradox that no-one would deny that the science of physics is made by people, but many would still argue that the science of physics does not have anything to do with society. That is, it is a part of the positivist argument that the structure of society, personal preferences, or any human aspects for that matter, do not have anything to do with the laws of physics. However, when one sees it in the positivist way, the paradox seems to vanish: If a person finds a mountain, the existence of that mountain does not have anything to do with the person, even though it was certainly a discovery made by a person. Following the same inference, physicists do not *create* laws of physics—they *discover* them. Consequently, many people can make the same discovery without knowing of each other's findings and many people can conceptualize the discovery in different ways even though they would be speaking about the same corner of reality.

As a computer scientist, I am first inclined to ask, "How seriously should one take people who claim that natural sciences are social constructs rather than discoveries about reality?". But asking that question necessarily begs another question: "How seriously should one take people who claim that natural scientists unravel objective facts about the world?" I take both people very seriously. After all, this dispute boils down to questions about objective reality and the nature of knowing that, as discussed earlier, may never be solved.

Neither of the two camps mentioned above is an esoteric, isolated group or a mere philosophical curiosity. As Ian Hacking noted, it is easy to find people who blanch when they come across the idea that the results of, for instance, physics or chemistry are social constructs[315]. The majority of natural scientists believe in objective scientific facts. Yet, as the science wars in the 1990s showed, it is also easy to find those people, including natural scientists, that agree that science is (to some degree) a social construction. There is a number of scientific disciplines that study the connection between science and social spheres, such as the sociology of scientific knowledge and science and technology studies, and many thinkers, such as Barnes et al., have argued that those disciplines should be a part of the project of science itself[316]. However, the number of different constructionist accounts of science is so large[317] that most of them cannot be dealt with in this thesis.

For this thesis I have chosen those authors of social constructionism who are most often connected with the social constructionist movement: naturally, the sociologists Peter Berger and Thomas Luckmann[318]; the sociologist David Bloor because of his seminal work with the "strong programme in the sociology of knowledge"[319]; the sociologist Barry Barnes and the historian of science and sociologist John Henry for their pioneering, philosophical work with the sociology of scientific knowledge[320]; and philosopher John Searle[321] for his unorthodox, realist account of social reality. The "Bath School" (which consists of Harry Collins, Trevor Pinch, et al.) as well as the actor-network theory (of which Bruno Latour plays a large role) are only touched on briefly in this thesis in the context of the social construction of technology[322], which unquestionably skews the thesis towards what Hacking called the "Edinburgh school"[323].

The reader should be aware of the biases in this thesis: The type of *constructionism* used in this thesis is only one of the many types of constructionism. Similarly, there is a bias with the accounts of science presented in this thesis: Chapter 2.1 offers a

315 Hacking, 1999:p.64.
316 Barnes et al., 1996:p.*iix*.
317 See, e.g., Hacking, 1999:p.65.
318 Berger & Luckmann, 1966
319 Bloor, 1976
320 Barnes et al., 1996
321 Searle, 1996
322 Kline and Pinch, 1999; Strum and Latour, 1999
323 Hacking, 1999:p.65.

very small sample of views on the philosophy of science. However, both of these samples are large enough to provide evidence that there is no clear boundary between "realists" and "postmodernists" (or constructionists, relativists, nominalists[324], idealists[325], anti-realists[326], etc.). Actually, it would probably be hard to find a typical representative of either the realist or the relativist side.

In the end, no matter how multifarious the views of realists and constructionists might be, there must be something that the constructionists and realists cannot agree with. Ian Hacking believed there to be three main areas of irresoluble differences between the two sides, which he called *sticking points*[327]. Hacking's sticking points are *contingency*, *nominalism*, and *explanations of stability*. Because these points stick to the very heart of any discussion about the significance of sociohistorical and cultural studies of computing, they all are covered here. These three sticking points mark the fronts where the researcher of social construction of computing needs to defend his or her views. Therefore, I take a position towards each of the sticking points. This analysis is lengthy, but these matters have to be dealt with thoroughly. I hope that by the end of this section, I have clarified my position towards three major bones of contention, or sticking points, in the science wars.

Sticking Point 1: Contingency

The first sticking point, *contingency*[328], can be explained through the "contingency thesis" that Hacking claimed constructionists to maintain[329]. Shortly put, the contingency thesis holds that the current state of affairs in a specific science could have developed taking a route that does not have anything in common with the route that the current science has taken. Furthermore, it states that there can be a successful alternative to the prevailing science, one that does not have to have anything in common with the prevailing science. Henceforth the terms "prevailing computing" and "alternative computing" are used in the following sense: *Prevailing computing* refers to the status quo in computing at large and *alternative computing* refers to a hypothetical alternative for the prevailing computing.

---

324 Quine, 1960:p.233.
325 Koepsell, 2000:p.19.
326 Hitchcock, 2004:p.6.
327 Hacking, 1999:p.68.
328 Contingency, in this context, refers to incidentiality, chance, and uncertainty–in, e.g., happenings or occurrences.
329 See Hacking, 1999:p.78.

I take it that applied to computing, the contingency thesis states that there could have been an equally successful alternative to modern computing; that it could have taken a route that does not include Turing Machines, von Neumann-architecture, the transistor, or such; and that this alternative computing could have been as successful as prevailing computing. Moreover, the contingency thesis in computing would state that this alternative computing could have developed in a way that the alternative computing and prevailing computing could not be explicable in each others' terms. That is, alternative computing could not be explained in terms of prevailing computing or vice versa.

Holding to the contingency thesis would actually mean that the Turing Machine is not an inevitable part of successful computing (even though it is an inevitable part of the prevailing theory of computation). For the sake of argument, I assume that there are computer scientists who think that any successful theory of computation is reducible to, or convertible to, or explicable with the prevailing theory of computation. For the sake of consistency with Hacking's terminology, I will use the term *inevitabilists* to refer to those scientists who think that *if* a successful theory of computation took place, *then* it would inevitably have taken a route similar to prevailing computing[330]. On the other side of the debate, *constructionists* think that a successful theory of computation need not take a route similar to prevailing computing. In the following pages I describe what the debate between inevitabilists and constructionists is about, and what corollaries that debate has for this thesis.

Contingency Thesis And the Definition of Successful

The first problem in the contingency debate is that claiming that a successful alternative computing would be possible, raises difficulties with the definition of *successful*. It is clear in; not only Kuhn's view, but also in Popper's view on science; that science sets its own standards, at least partially[331]. Those standards include, for instance, the definition of successful. If the standards of science are set by scientists working within the same science, they scarcely reject that science, as Kuhn and Max Planck claimed[332]. Furthermore, following the demand of reducibility, the prevailing computer science has its own representation and language, with which every altern-

330 See Hacking, 1999:p.79.
331 Fuller, 2003:pp.45-46.
332 Kuhn, 1996:pp.151-152; Planck, 1949:pp.33-34.

ative candidate should be explainable. As Chapter 2.1 discusses, both Kuhn and Feyerabend agreed that there is no "neutral system of representation" or "neutral language" that could be used to express all theories.

Now, rephrasing Hacking's question[333], if the standards of successful computer science are internal to the discipline itself, what *could* it actually mean to have an equally successful but totally alternative computer science? A problem similar to this (and a bit similar to the underdeterminacy thesis) is nicely expressed by Willard v.O. Quine[334] as the *indeterminacy of translation.*[335]

The indeterminacy of translation means, for instance, that if one meets aliens speaking Alien, how can one know that Alien is a language at all? One can know that Alien is a language at all only if one can translate it, by and large, to his or her own language[336]. The same question in computer science is, "How does one know that statements of alien computing are about computing at all?". Similarly, one can be sure that statements of alien computing are about computing at all only if the statements can be reducible to, or translatable to, or explicable with one's own language and theory of computation. Therefore, given a model of computation (or computer science), that is sufficiently different from (but equally successful with) the prevailing model, a computer scientist would not see it as computation no matter what[337].

Of course, one can say, "if this alternative model of computation can solve the same problems that our computation can, it does not matter if it cannot be expressible in our terms." This argument could be called the "equal problem solving power"-argument. However, as discussed earlier, one of the things that a scientific paradigm brings along is a criterion for choosing problems that (according to the paradigm) can be assumed to have solutions[338]. That is, science defines its own problems. As I see it, any attempt at trying to make problem-selection neutral is doomed. For the sake of clarity, in the following paragraphs all the problems in the world are denoted with $\boldsymbol{P}$, all the problems that current computer science can solve as $\boldsymbol{P}_{\mathbf{C}}$, and all the problems the alternative computer science can solve as $\boldsymbol{P}_{\mathbf{A}}$. (Perhaps the word para-

333 Hacking, 1999:p.69.
334 Quine, 1960:pp.73-79.
335 Quine gives an example of translating "neutrinos lack mass" into "jungle language".
336 Hacking, 1999:pp.74-76.
337 cf. Hacking, 1999:p.72.
338 Kuhn, 1996:pp.36-37.

digm might be used instead of science, but *science* is used here to indicate deeper incommensurability of the two systems of computation.)

First, it would be a perfectly neutral procedure to take all the *possible* problems in the world $\{\boldsymbol{P}\}$, and see which problems the two competing models of computation can solve. The winner would be the model that solves more problems than the other. However, this would lead to a question whether all problems should be considered equally important, or if there are problems that are more important to be solved than others. If one model can solve a large number of trivial problems and the other model can solve a small number of significant problems, which model is more successful? Furthermore, models of computation can set their own criteria of significance, importance, and triviality.

Second, one might try to find a "neutral set of computational problems", which begs the definition of "computational" (petitio principii[339]). Feyerabend and Kuhn noted that the definition of problem is an inseparable part of science, so the question is, "Should this neutral set be defined from the point of view of the first model of computation or from the point of view of the second model of computation?". Of course one could take all the problems that can be solved with either science $\{\boldsymbol{P}_{\mathbf{C}} \cup \boldsymbol{P}_{\mathbf{A}}\}$, or the problems that can be solved with both sciences $\{\boldsymbol{P}_{\mathbf{C}} \cap \boldsymbol{P}_{\mathbf{A}}\}$, but these would inevitably lead to the same problem with the first procedure, that is, to the question of which problems are most important to solve.

Third, one could try to aim at some kind of "content neutrality", following, for instance, Imre Lakatos' proposition of progressive and degenerating research programs[340]. Simply put, progressive research programs are characterized by the growth of new knowledge, whereas degenerating research programs are characterized by the growth of the number of ad hoc modifications to them. Even though Lakatos aimed at something universal, and intended his theory to encompass all sciences, all his examples came from the field of physics. As Alan Chalmers noted, Lakatos' theory presumes that all areas of study, if they are to be regarded as "scientific", must share

339 The problem is apparent in the following phrasing: "We can find out if the model of computation A (which defines computation in its own terms) is equally successful to the model of computation B (which defines computation also in its own terms) if they can solve the same computational problems (which are defined by a model of computation)". That is, models of computation cannot be compared without assuming the definition of computational problem, because it is an inherent part of each model of computation.

340 Lakatos, 1970 in Lakatos & Musgrave, 1970; Chalmers, 1976:pp.130-131.

the basic characteristics of physics[341]. Progression and degeneration are not content-neutral concepts.

In a word, the "equal problem solving power"-argument appears inconclusive in the choice between two computational models, because "problem solving power" is not an objective benchmark of success but it depends on the definition of problem, which depends on the selected model of computation, which leads to circularity.

It seems that it is not enough that computer scientists are allowed to define what a *successful theory of computation* is—which is, in essence, only the prevailing theory—but there are tendencies to extend the theory of computation to other disciplines and other areas of life, too. There are some attempts to explain, for instance, mind[342], universe[343], and culture[344] as sorts of computation. Paul Feyerabend used the phrase *chauvinism of science*[345] to refer to this tendency.

Scientists, Feyerabend wrote, are not content with running their own playpens in accordance with what they regard as the rules of the scientific method. Scientists want to universalize those rules, and they want them to become a part of society at large. Furthermore, Feyerabend argued that scientists use every means at their disposal from argument to ridicule to intimidation to achieve their aims. Steve Fuller wrote about the same phenomenon when he proposed that positivism was always a "made for export" philosophy[346]. In Fuller's provocative opinion, positivists have wanted to spread what they took to be the secret of physics' scientific success to the more backward disciplines.

I can agree with Feyerabend only conditionally: The phenomenon that Feyerabend depicts, is chauvinism in science only if there is a lack of symmetry. In terms of computer science, my condition for the existence of Feyerabendian chauvinism is that a computer scientist extends his or her science well over its traditional limits—but at the same time denies the probing of computer science itself with non-traditional instruments.

---

341 Chalmers, 1976:p.146.

342 Computationalists assert that persons are Turing Machines, or at least that all mental states are computational states. See, e.g., Scheutz, 2002.

343 Wolfram, 2002

344 For instance, Liane Gabora (Gabora, 1995) describes a computational model of how ideas, or memes, evolve through the processes of variation, selection, and replication.

345 Feyerabend, 1993:p.163.

346 Fuller, 2003:p.79.

A case in point is culture. If a computer scientist sees that culture can be explained as a sort of computation[347], but denies any explanation of computing as a cultural phenomenon, this asymmetry is indeed chauvinistic in the Feyerabendian sense. Chauvinism in this asymmetric sense, is surely a subjective notion. Unfortunately, this sort of chauvinism is not unknown to computer science[348]. Even though such projects are rare, computer scientists at large do not seem to find anything strange in explaining culture as computation, yet they cringe at the very thought of explaining computing as cultural.

Another example, apposite for this thesis, is that if computer scientists agree with, for instance, Brent et al.'s[349] computational approach to sociological explanations, but refuse the explanation of computer science from the sociological point of view, this asymmetry, or lack of reciprocity, is chauvinistic in the Feyerabendian sense. I *do not* argue that if society can be characterized with discrete models, then discrete models can be characterized with societal concepts. I *do* argue that if there is a lack of reciprocity or symmetry in explanations that computer science *allows* and the explanations that computer science *makes*, then computer science is a chauvinistic enterprise.

Inevitability, Contingency, And "The Mangle"

Andrew Pickering's work on quarks and high-energy physics led him to revamp the old motto of science—"science proposes, nature disposes"[350]. The old motto comes from the Popperian view that scientists can freely come up with different kinds of conjectures, subject them to thorough experimental testing, and that nature will reject the false ones. Instead of the old motto, Pickering came up with an understanding of the structure of scientific practice, which he called "the mangle"[351]. The two key words in Pickering's "mangle" theory are *resistance* and *accommodation*. New scientific theories and new instruments based on the theories do not usually work—they resist. Scientists have to give up or accommodate to the situation. Computer

347 See Gabora, 1995.
348 Wolfram, 2002; Gabora, 1995; Brent et al., 2000; Scheutz, 2002
349 Brent et al., 2000
350 Pickering, 1995:pp.38-49.
351 Pickering, 1993

science and engineering is a prime example of mangle, because theory and practice, the science and the machine, are inseparable in modern computing[352].

In Pickering's model, there are abstract scientific theories about phenomena, there are down-to-earth models of how scientific instruments work and what can be done with them, and there are the instruments themselves[353]. For instance, in computer science, the Turing Machine is an abstract theoretical construction, von Neumann-architecture is a (high-level) model of an instrument, and stored-program computers are the instruments. Alan Turing devised his theory in 1936, von Neumann architecture was conceived around 1944, and the first programmable computer was successfully built in 1949. Other possible, different routes of development can be explained by Pickering's mangle of practice[354]. Different people accommodate differently to resistance, and thus create different branches from the same roots. For instance, had John von Neumann (1903-1957), John W. Mauchly, and John Presper Eckert not been in the Moore School of Electrical Engineering during the World War II, humankind could have ended up with a different kind of computing machinery (this is further discussed in Chapter Three).

The development of a new instrument may be a long-lasting struggle between problems (resistance) and solutions to them (accommodation). The accommodation processes can include, for instance, revising a scientific theory, revising beliefs about how an apparatus works, or modifying the apparatus itself[355]. Before a robust fit between science, model, and apparatus is created, there are different accommodation strategies that, naturally, face different kinds of resistance. The mangling, Pickering wrote, is material, conceptual, and social. Even though there can be incommensurable branches of development, an inevitabilist would say that those branches would always unite somewhere along the way; a constructionist would say that they would not necessarily do so.

It seems that the way in which computing has developed, the mangle, exhibits characteristics of contingency rather than inevitability (even though exhibiting is different than proving). The history of computing may be a history of technoscientific necessity (the inevitabilist viewpoint), or it can be a history of personalities, coincid-

352 cf. Knuth, 1991; Denning et al., 1989; Forsythe, 1968; Wegner, 1976; Hopcroft, 1987
353 See, e.g., Hacking, 1999:p.71.
354 Pickering, 1993; Pickering, 1995
355 Hacking, 1999:p.71.

ences, power games, politics, and money (the constructionist viewpoint)—this should become clear by the end of this thesis. Chapter Three of this thesis discusses a large number of examples that support the constructionist viewpoint, but the reader can always adopt the inevitabilist viewpoint towards my examples, too. It is a valid and credible argument that the history of a concept is different from the concept itself. One can always adopt the point of view that no matter what contingencies characterize the history of computing, the development of computing eventually (necessarily) would have had to follow the route it has followed (or at least it would have necessarily lead to the same technoscientific outcomes).

In the end, the "contingency *vs.* necessity" debate is a matter that cannot be proven either way, so the choice of explanatory models is predicated on the credibility of each argument. If one believes that the necessity argument is stronger than the contingency argument, one can adopt it, or vice versa. Or alternatively, one can be of the opinion that both arguments are strong, disparate, and incomplete, and choose not to make a decision at all. Note, however, that technologists may not always be ready for that alternative option. Tracy Kidder claimed that since in the world of technology and engineering there are only correct and incorrect solutions, disputes among engineers must always have resolutions[356].

The position in this thesis is a constructionist one rather than an inevitabilist one. The consequence of this choice is that in examining the development of computing, the circumstances in which the development takes place are considered to influence and shape the development. That is, the environment where the researchers live and conduct their work affects the directions and forms of computing and its theories, lays foundations for the research, and indeed makes development possible in the first place. From the constructionist viewpoint, there is no "natural direction" towards which the development of computing is heading.

Adopting the inevitabilist viewpoint would mean denying the influence of the environment on the forms that computing and its theories take, or at least arguing that eventually all developmental paths would lead to the same inevitable ends. From the inevitabilist viewpoint, there is a natural path of development and no matter what historical contingencies may have affected local steps of development, natural laws

356 Kidder, 1981:p.147.

and logic would *eventually* direct researchers to the best direction. For an inevitabilist, research on computing from sociological, historical, anthropological, or philosophical perspectives does not offer insight into computing as such. For an inevitabilist, sociologists, historians, anthropologists, and philosophers do not study computing but the phenomena around it—and those phenomena have only local effects (spatially and temporally), but not lasting consequences.

Sticking Point 2: Nominalism

Hacking's second sticking point, *nominalism*, refers to one side of one of the oldest philosophical debates, and as Hacking noted[357], this sticking point is clearly visible in the science wars. Because there are problems in using the term "realism" in contrast to, for instance, nominalism[358], Hacking introduced a term *inherent-structurism*[359]. (Henceforth, for the purpose of brevity, yet risking ambiguity, inherent-structurism is referred to as *structurism*—and the adherent of inherent-structurism is, respectively, called *structurist*.[360])

The "structurism *vs*. nominalism"-discussion is central to this thesis: If algorithms and computers are a part of the hierarchical structure of the world, there is little point in studying the sociocultural and philosophical roots of computing because sociocultural and philosophical aspects are merely hurdles on the road towards understanding the universal structures of computation—structures that stem from natural laws and logic. From the structurist point of view, sociocultural and philosophical studies may, at best, reveal how to best root out all the unwanted sociocultural residue from computer science (as a body of knowledge). From the nominalist point of view, sociocultural and philosophical studies may offer insight into the essence of computer science (as a body of knowledge).

A structurist (as opposed to a nominalist) would claim that there is an inherent structure in the world, and that the purpose of science is to examine this structure. This is a nuance of Platonic ontology (even called "Platonism" by Willard v.O. Quine[361]), a

357 Hacking, 1999:p.82.

358 Already in 1960 (before the science wars), Quine noted that there are problems in the terminology (Quine, 1960:p.233).

359 Hacking, 1999:p.82.

360 The purpose of this thesis is not to discuss the nuances of philosophical terminology, so such shorthand is adopted instead of Hacking's recommendation. "Structurism" does not refer to structuralism and therefore the discussion about deconstruction is not necessary.

361 Quine, 1960:p.233.

position which states that there are abstract, universal ideas that scientists can find. A nominalist would claim that no such structure exists, and he or she—much in an Aristotelian manner—denies that there are universals. As Hacking noted, the nominalist scientist hopes to be true to experience and interaction and to the way in which, for instance, a scientific apparatus does *not* work[362]. In nominalist thinking, researchers have to accommodate constantly to the resistance of the material world. In other words: In their work, researchers constantly face difficulties in finding a robust fit between their theories, their instruments, and their understanding of how their instruments work[363]. Researchers need to accommodate to those difficulties either by revising their theories, revamping their instruments, or rethinking how their instruments work.

One more word about Quine's terminology: There are phenomena that Quine regarded as "enigmatic". For instance, "the North Pole" and "the Equator" are *abstract particulars*[364]. In Searle's framework abstract particulars might not be problematic:

(1) If all people commonly refer to one end of the axis around which the Earth rotates as "the North Pole", the existence of the North Pole is an ontologically and epistemologically objective fact. That is, the Earth has an axis no matter what people may think about it, and it is a socially established fact that the term "North Pole" refers to that explicit point.

(2) If one refers to, instead of the Earth's axis, a commonly agreed reference point on the map as the "the North Pole" (this is an abstract particular), the existence and location of the North Pole is an ontologically subjective but epistemologically objective fact.

(3) If one uses "the North Pole" as a reference to something abstract that is not shared commonly, like the position that he or she is currently thinking, the existence of the North Pole is an ontologically and epistemologically subjective fact.

(4) If one believes that the resting place of his or her forefathers' souls is at the North end of the Earth's axis (ontologically objective fact), and names this

362 Hacking, 1999:p.84.
363 Pickering, 1993; Pickering, 1995
364 Quine, 1960:p.233.

resting place "the North Pole" (instead of "Valhalla"), then the existence of the North Pole is an ontologically objective and epistemologically subjective fact. That is, in this case the exact place is an intrinsic, unambiguous feature of the Earth, but its function and meaning are observer-relative.

Ontology in the Field of Computing

The questions of ontology are rarely discussed in computer science[365]. Perhaps the reason is that ontology is not regarded as a part of computing, or perhaps because the questions are so fundamentally difficult—and fundamentally unresolved. The term *ontology* in computer science and some other computer-related fields is sometimes used in an obscure, inconsistent, and untraditional manner. A case in point is Michael Heim's *The Metaphysics of Virtual Reality*[366]. In the 145 pages of what philosopher David Koepsell called "incorrect ontology and muddled metaphysics"[367], Heim confused Ideals with brute facts[368], and mixed epistemology, metaphysics, and ontology[369]. An ACM database search for *ontology* leads to definitions of ontology with a number of very unorthodox meanings such as "[a definition of] *terms or vocabularies used within messages*"[370], "*a comprehensive knowledge model that enables a developer to practice a higher level of reuse*"[371], and so forth. There is no doubt that *ontology* as a term in computer science is nowadays understood differently from its philosophical meaning[372]. In fact, the subtitle of CACM's special issue on ontology, "Ontology: Different Ways of Representing the Same Concept" is an *epistemological* matter, if anything.

Even though ontology in its traditional meaning is not generally considered to be a part of computer science[373], there are some thorough investigations of the ontology of computing, such as Brian Cantwell Smith's *On the Origin of Objects*[374]. Unfortu-

365 This is my impression, and I may be mistaken, but for instance, a search in ACM "Guide" (more than 887,000 citations published by ACM and other publishers in the field of computing) for "nominalism" yielded 16 results, out of which few actually discuss the matter. A search for "Platonism" yielded 19 results. (1st October 2005) See `http://portal.acm.org/guide.cfm` (accessed September 27th, 2006)

366 Heim, 1993

367 Koepsell, 2000:p.22.

368 Heim, 1993:p.89.

369 Koepsell, 2000:p.23.

370 Pan et al., 2003

371 Wang et al., 2002

372 Also, for instance, Gloria Zúñiga (Zúñiga, 2001) recognizes this in her conference paper.

373 Gruninger & Lee, 2002

374 Smith, 1998 (Originally published 1996)

nately Brian Cantwell Smith's book and John Searle's *The Construction of Social Reality*[375] were published almost simultaneously, because Searle's book (which is a general account of the construction of social reality) would in all likelihood have had an impact on Smith's book (a specialized account of construction of objects), especially on Smith's *successor metaphysics*.

Despite the poverty of doctrinal discussion on the nature of objects and facts, computer scientists, generally speaking, are keen on making models about the world. This idea is at the base of the "new" conception of computing "ontologies"[376]. The creator of C++ language Bjarne Stroustrup wrote, in the context of object-oriented design, about modeling some aspects of "reality" or "concepts of application" as classes[377]. Databases or data records, too, are usually designed to represent real-world entities, parts of real-world entities, or aspects of real-world problems[378]. This tendency suggests that computer science is not different from sciences such as physics and chemistry, in which structurism (realism) is an unwritten, rarely questioned assumption. This is neither very surprising nor especially interesting. What *is* interesting, though, is the observation that computers make the ontological status of some things very vague—there are things (or facts) that could be equally well be considered either abstract or concrete. To investigate this phenomenon, which is central to the "structurism *vs*. nominalism"-debate, a brief explanation of institutional and brute facts is probably useful.

Brute Facts and Institutional Facts

John Searle's realist account of the building blocks of reality[379] offers a fruitful "middle-way" for further discussion of the nature of computational objects. His division of facts into "brute" and "institutional" allows constructionist thinking in a traditionally materialist field of study. On one hand, Searle's division allows the researcher of computing to take into account the limits of the physical world (the physical world determines some characteristics and limits of automatic (machine-) computation). On the other hand, Searle's division makes possible explanations of so-

375 Searle, 1996 (Originally published 1995)
376 See, e.g., Castel, 2002.
377 Stroustrup, 1997. See especially Part IV: Design Using C++:pp.691-790.
378 See, for instance, McConnell, 1993:p.177.
379 Searle, 1996

cially constructed phenomena that would not exist without people (e.g., money, labels, programs, and algorithms).

The most interesting objects of computing, from my point of view, are neither institutional nor brute facts, but the facts that do not fit smoothly to either class—facts such as the existence of algorithms and programs, and the objects that are not clearly tangible or intangible—objects such as *algorithm* and *program*. From my point of view, what makes computer science and computation stand as a prime case of a problem in metaphysics is the dualistic nature of the computer—not dualistic in the sense of the relationship between *mind* and *matter*, but referring to *software* and *hardware* instead. Technically computers do not differ much from any other calculators, but the programmability and plasticity of a general-purpose computer makes it much more difficult to analyze a computer than many other artifacts such as a hammer, a car, or a telescope.

Philip Brey has considered the problems of virtual entities from Searle's ontological point of view[380]. Many entities in computer environments, such as cursors, menus or windows, do not fit precisely into Searle's ontology, and therefore Brey granted them a special place: Brey claimed that they are different from physical entities, but also different from fictitious or imaginary entities. After all, Brey claimed, they do not have physical existence (mass or location in physical space), yet they can be manipulated, they respond to user's actions, and they may stand in causal relationships to other entities.

But contrary to Brey's argument, a mouse cursor *does* have a physical form—the zeros and ones (in the form of swarms and flows of electrons) in the computer memory that indicate the position, size, and shape of the cursor; the states of electronic circuits that render it on video memory; electrons that hit the phosphorescent matter on the aperture grille. Then again, one might claim that a thought also has a physical counterpart. Biological realists believe that thinking can be reduced to electrochemical phenomena in the brain. Yet, unlike a thought, the existence of the mouse cursor is a consequence of controllable and predictable, human-made processes. It seems that Brey was confused. It is as if he was saying that the beam of flashlight or a sound from a radio would not have physical counterparts. They do. It

380 Brey, 2003

is a brute fact that there are photons, electrons, gasiform compounds, and other particles that are parts of our environment. Naturally, *interpreting the meaning* of the groups of those particles as "a mouse cursor" or "a beam of flashlight" is a different matter.

Although interesting, the entities Brey focused on are not the most fascinating ones in regard to the ontology of computing. Instead of user interface elements, consider records in a database, data structures in computer memory, or processor registers. They do have a physical existence and they can be manipulated, but the fascinating part comes from the fact that they can work as referents to tangible objects as well as institutional facts (yet I do not argue that computers are intentional.) A swarm of magnetic blips on a hard drive or voltage differences in a circuit can represent money—actually, it can be used *as* money; it can replicate a conversation in different forms (video, audio, or text); it can manifest or actualize an abstract idea such as an algorithm and even make it do things in the physical world; it can refer to a tangible object such as the sun. But in the end, it is still just a swarm of magnetic blips or voltage differences.

Searle on Objectivity and Subjectivity

Before taking this discussion further, to algorithms, two crucial senses of the pair *objective-subjective* need to be distinguished. The first one, as explained by John Searle, is the epistemological sense of the words[381]. Searle wrote that epistemically speaking, *objective* and *subjective* are primarily predicates of judgments. Ontologically speaking, *objective* and *subjective* are predicates of entities and types of entities, and they ascribe modes of existence. The following list, explaining this separation, is adapted, with slight modifications, from Searle[382] (except for Table 1, which pulls together these definitions).

1) The sheer existence of a physical object does not depend on any attitudes people may have toward it: The existence of the object is a brute fact.

2) Physical objects have many features that do not depend on any attitudes people may have toward the object—a certain mass and a certain chemical composition, for instance. The object and these features are *ontologically objective*.

381 Searle, 1996:pp.7-11.
382 Searle, 1996:pp.10-11.

3) There are, however, features of objects that exist only relative to the intentionality of people—for instance, that an object is a computer. These observer-relative features are *ontologically subjective*.

4) Some ontologically subjective *features* are *epistemologically objective*. For example, a computer is not a computer only because it is Searle's opinion that it is a computer. It is a matter of objectively ascertainable fact that the object is a computer.

Table 1: Different Sorts of Facts

Ontologically Objective	Brute facts about the physical world; independent of any perceiver or mental state *A helium atom has two electrons.* *That object is made of various conducting and semiconducting elements and compounds.*
Ontologically Subjective	Statements about things whose existence depends on subject *I have pain in my lower back.* *I am thinking of a bubble sort algorithm.*
Epistemologically Objective	Statements about features that are independent of anybody's attitudes or feelings *That object is a computer.* *Algorithm A runs in* $O(2n^2+4n)$ *time.*
Epistemologically Subjective	Statements about features that depend on certain attitudes, feelings, or points of view *That object is a good computer.* *Algorithm A is more elegant than algorithm B.*

There is one part in Table 1 that needs particularly careful analysis—ontologically subjective facts. I argue that the existence of algorithms, as abstract things, are ontologically subjective facts (i.e., they do not exist independently of humans or other intelligent beings). Philosopher of computing Brian Cantwell Smith[383] noted that even though reason and mathematics are allegedly abstract, they do not escape the confines of the world (i.e., they do not exist independently of the material world). He wrote that to assume they would exist independently, would be ideologically reductionist in the sense that the mathematician would then be thought to have free access

383 Smith, 1998:p.106.

to extra-world notions, immune from theoretical scrutiny. That assumption would rely on the concept of the Platonic world of Ideals.

The Ontology of Algorithms

The argument that algorithms are ontologically subjective things, or why they would not exist independently of humans or other intelligent creatures, is rationalized in the following paragraphs. First, some definitions are necessary. An algorithm, according to Donald Knuth[384], is a finite set of rules that gives a sequence of operations for solving a specific type of problem[385]. According to Knuth's definition, an algorithm must be finite (it does not loop forever), it must be precisely defined, it may have input, it has to have output, and it must be effective. Second, a procedure that has all of the characteristics of an algorithm but that possibly lacks finiteness, is called a computational method[386]. Such computational methods, include, for instance, reactive processes (e.g., computer programs) that interact with their environment. In this thesis, I take it that the definition of algorithms concerns mechanically realizable (at least in principle) things but not naturally occurring physical or chemical reactions or biological phenomena.

Now, for the sake of example, I assume that some computer scientists may claim that algorithms (for instance, the bubble sort algorithm) are ontologically objective: That their existence is independent of humans; that they exist non-spatially and non-temporally[387]. If algorithms are ontologically objective, also computational methods, such as the method for calculating $\sqrt{2}$, are ontologically objective, because the only difference is the lack of finiteness. Note that an expression of an algorithm or a computational method in a programming language or in an executable machine code is called a program[388]. In addition, all computer programs are algorithms or computational methods.

Keeping this definition in mind, the seemingly innocent claim that algorithms are ontologically objective is actually a much more controversial claim. In effect, it is a claim that *all computer programs are ontologically objective*. To put this into a very tangible form it is a claim that, for instance, Microsoft Word exists independently of

384 Knuth, 1997:pp.5-7. See also Lewis & Papadimitriou, 1998:pp.245-246.
385 Granted, the term *algorithm* is not that straightforward–see, for instance, Cleland, 2001.
386 Knuth, 1997:pp.5-7.
387 Or, perhaps, that they exist in nature.
388 Knuth, 1997:p.5.

humans. It existed before people, and it will exist after people are gone. This claim, a corollary of the original claim, does not seem very plausible any more. Of course, a computer scientist can note that Microsoft Word is merely a string that, in theory, can be produced with a simple algorithm that produces all strings given an alphabet. This argument does not lead anywhere, though. It just maintains that one algorithm can be produced with another algorithm, but it does not reveal anything more about the ontological status of algorithms. Note also that if algorithms are ontologically objective, then all incorrect algorithms also exist regardless of people.

I assume that a hasty computer scientist could fall back to a second line of defense and specify that larger programs such as Microsoft Office are constructed out of smaller algorithms, and that those basic algorithms exist independently of humans. This defense is shaky when confronted with questions such as "What exactly is the difference in granularity that separates ontologically objective and ontologically subjective algorithms?", "Is the bubble sort algorithm within a larger program one of the basic, ontologically objective building blocks?", and "Is the `swap(a,b)` within the bubble sort algorithm one of the basic building blocks?". Perhaps one might argue that the axioms an in axiomatic definition of a programming language constitute the most atomic elements[389]. I guess that one would need to go all the way down to the most atomic operations to make a clear cut line, but a claim such as "machine instructions are ontologically objective" or perhaps even "Boolean logic is ontologically objective" is quite far from the original claim—"algorithms are ontologically objective".

Consequences of Different Positions to Nominalism

The mode of existence of mathematical facts and logic is a difficult question. Many philosophers, such as Gottlob Frege (1848-1925), Hilary Putnam, and Ludwig Wittgenstein (1889-1951), have tried to untangle this problematic field, without conclusive results[390]. Whether algorithms in their non-physical form are ontologically objective is a matter of belief, and it does not have a proven resolution[391]. One has to weigh the different sides of the question, and choose a position that he or she finds

389 This thinking is implicit in Dijkstra, 1974.

390 Shapiro, 2000

391 Alan Chalmers wrote, "*the mode of existence of* [...] *linguistic objects, as well as other social constructions such as methodological rules and mathematical systems is a tricky philosophical business. I am content to make my points at a commonsense level* [...] *This is sufficient for my purpose.*" (Chalmers, 1976:p.127).

most plausible. I have presented some reasoning for my standpoint that algorithms are ontologically subjective things, and later in this thesis I consider the implications of this standpoint to studies of computing.

Even though one would take algorithms, logic, and mathematical objects as ontologically subjective (they do not exist without humans or other intelligent beings conceptualizing them), they are, in Searle's[392] terminology, epistemologically objective: An algorithm *A* that is able to perform function *f*, is able to perform function *f* independently of anybody's attitudes towards the algorithm or feelings about it at the moment. It is a matter of objectively ascertainable fact, as Searle would call it, that the algorithm *A* performs function *f*. (Still, note that whether algorithm *A* is better, more aesthetic, or easier to implement than algorithm *B*, is an epistemologically subjective matter.)

I am not arguing that algorithms would be epistemologically subjective in the sense that their functioning, once conceptualized or created or constructed, would be dependent on a person's culture, feelings, or attitudes towards it. I am arguing that one can fairly commit to the view that algorithms do not exist without humans or other intelligent beings[393]. A proponent of an opposite view would have to either acknowledge that all software would exist without humans (in some kind of non-spatial and non-temporal, Ideal, or noumenal form), or to present a definition according to which a line between objective and subjective algorithms can be drawn—for instance, based on granularity, or functioning, or some other features.

Finally, because computer science deals not only with ontologically subjective, but also with ontologically objective phenomena (with computers, semiconductors, and such) it would be impractical, to say the very least, to deny the existence of *any* structures in the world. Therefore, in this thesis, a middle-way between the nominalist and structurist viewpoints is taken. Holding to Searle's ontology seems like a good choice to a computer scientist who wishes to be true to the physical explanations of the world, but who nonetheless considers some aspects of computing as dependent on human preferences.

392 Searle, 1996:pp.7-9.

393 Note that I mentioned earlier that naturally occurring physical or chemical reactions or biological phenomena, such as naturally occurring RNA are not considered algorithms in this thesis.

If the structurist viewpoint would be adopted for this thesis, then one would need to explain, for instance, how the functioning and the buggy parts of Microsoft Word relate to the inherent structures of the world—or even explain Microsoft Word as an ontologically objective, timeless phenomenon altogether. If the position in this thesis would be that facts in computer science are timeless and universal parts of the structure of the world, then it should also be explained why *incorrect* statements are *not* timeless and universal (which would be difficult), or state that both correct and incorrect statements are timeless and universal (which would not differ much from nominalism anymore). One could also adopt the position that Rudolph Carnap (1891-1970) took towards the ontology of mathematics—Carnap rejected metaphysical speculations and held that scientists and mathematicians should base their decisions between different mathematical frameworks on how useful the different frameworks are[394].

The nominalist viewpoint would also be difficult for this thesis, because ignoring the quite well-functioning structures that computer scientists have built would shatter the field into small esoteric units, perhaps without a common language. In the Kuhnian sense, it would erase normal science. Insofar as computer science has in any sense helped to change or can be used to positively change the human condition, it would be a pity to erase a working construct that could be directed towards improving human conditions. Although normal science has its problems, as noted in Section 2.1, it creates a common framework of understanding between researchers.

Sticking Point 3: Explanations of Stability

Hacking's third sticking point[395] between realists and constructionists concerns the difference between how proponents on either side explain how science has become stable. Whereas the problem for Kuhn and Popper was to understand revolutions in science, the problem today is to understand stability, Hacking argued. One has to understand Hacking's claim of stability with some reservations. Hacking did not claim that science would have become infallible or that science would have found ultimate answers, but he claimed that there is a common sentiment that a lot of science is here to stay.

394 Shapiro, 2000:pp.126-128.
395 Hacking, 1999:pp.84-92.

This third sticking point is not whether science deals with *facts* or *commonly held beliefs*. The third sticking point is a debate between those who believe that the stability of science (these days) is an *inherent quality* of science and those who believe that the stability of science is a result of *external forces*. The former ones, call them, for instance, "rationalists"[396], believe that most science becomes stable because of the wealth of good theoretical and empirical research. The latter ones, constructionists, hold that the stability of science involves elements external to the professed content of science[397], elements such as the human processes of making science.

My interpretation of the rationalist position is that rationalists believe that the poverty of critical anomalies in natural sciences today is due to the fact that science is currently consistent, well-reasoned, and faithful to reality (physical phenomena) in its explanation of the world. And my interpretation of the constructionist position is that constructionists believe that science today is stable not only because of its internal coherence, but also because there is a number of sociocultural aspects that support continuity in science. Included in those sociocultural aspects that support continuity in science are, for instance, those characteristics of science that Feyerabend criticized[398]: The ability of science to set its own standards, scientific dogmatism and chauvinism (i.e., the effort to extend scientific rules and inference to all sectors of society), the reluctance of scientists to denounce their own findings (which would jeopardize their careers), uniform scientific education, skewed interests, old-boy networks in the academic world, and such.

I see some aspects of the constructionist argument as more applicable to computer science than others, but the field of computing does not yet seem ripe enough that one could speculate on explanations of stability in computing. There are three major arguments for my position. First of all, computer science is not purely a mathematical, empirical, nor an engineering discipline. Computer science is perhaps all of these at the same time[399]. The definition of computer science is widely debated, and there certainly is no definition that everyone would agree upon[400]. It is not sensible

396 Hacking used the word because it draws the attention to the lineage (Hacking, 1999:p.91).

397 Hacking, 1999:p.92.

398 Feyerabend, 1993

399 cf. Denning et al., 1989

400 For different definitions and portrayals, see, e.g., Newell et al., 1967; Forsythe, 1967; Knuth, 1974; Atchison et al., 1968; Kandel, 1972; Wegner, 1976; Jehn et al., 1978; Khalil and Levy, 1978; Minsky, 1979; Gibbs and Tucker, 1986; Hopcroft, 1987; Denning et al., 1989; Lee, 1989; Hartmanis et al., 1992; Hartmanis, 1994; Brooks, 1996; Gal-Ezer and Harel, 1998; McGuffee, 2000; Brookshear, 2003:p.1; Denning, 2003.

to argue that a discipline is stable or unstable if there is no clear picture of *what exactly* is stable or unstable.

Second, if one sets aside the problem of diversity, the next problem arises from the youth of the discipline. The field of computing as a discipline is only around sixty years old, if one begins from the first electronic computer built[401]. (Also the stored-program concept was conceived around that time[402].) During the past sixty years, there have been significant changes in the "paradigms"[403] or practices of the field. There have been a number of shifts in *technology*: the transistor, the integrated circuit, optical media, microcomputers, etc.; in *theory*: fuzzy logic, public-key cryptography, Chaitin complexity, etc.; in *programming*: high-level programming languages, object-oriented programming, logic programming, etc.; in *communications*: the modem, the Ethernet, the Internet, wireless solutions, the World Wide Web, etc.; in human-computer interaction: the mouse, the monitor, speech recognition, etc.; in *uses*: computational sciences, office computing, decision support systems, games and entertainment, etc.; etc. The number of cumulative and remarkable changes is large and the pace of change has been fast. It is not sensible to argue that a discipline is stable or unstable if the *time span* of stability would be very short. In essence, anything can be called stable if the time span is chosen conveniently.

Third, unlike electrons, genes, proteins, or acids, computers are human-made phenomena. It is irrelevant whether one sees theoretical computer science as a study of abstract, universal things, or concrete, human-constructed things. Irrelevant, because theoretical computer science is only one part of computer science—in fact, the majority of the definitions of computer science note that it deals with machinery[404] (and machinery is human-made in its entirety). The proponents of universalism in computer science might argue that computers are based on theories that are universal, so they are instances or manifestations of Ideas. That would take the discussion back to the second sticking point, nominalism. Be that as it may, I suppose that the majority of scientists would not object to my claims that (1) *computers* are different from nat-

401 ENIAC became operational 1945 (Williams, 1985:p.285).

402 Mauchly, 1979; Williams, 1985:pp.298-302, 411.

403 Although the term *paradigms* (in plural form) is sometimes used in computer science, I do not consider it to be a correct term to use, at least from the Kuhnian viewpoint. A plural *paradigms* necessitates at least one paradigm shift, which necessitates at least one *scientific crisis* triggered by *anomalies*, and it is difficult to point out such course of events in the sixty-year history of modern computing.

404 See the list of references above for a sample set.

urally occurring things such as electrons, genes, proteins, or acids, and that (2) the *discipline of computer science* is different from physics, biology, or chemistry in that sense[405].

However, I expect that a counterargument to my argument might be that (1) computer scientists study theories and models of computation, and that understanding these theories and models does not require understanding computer scientists. This counterargument could be continued that (2) relying on these theories and models, computer scientists build computers, and thus understanding computers does not require an understanding of computer scientists either.

The first part of the counterargument is valid because the ontological status of theories and models of computation is fundamentally unsolved. That is, because humans cannot escape the material world, human beings have no means to know with certainty the ontological status of abstract concepts. This is not, however, to agree that theories and models of computation exist independently of intelligent beings, but just that the mode of existence of theories and models cannot be proven as being either objective or subjective. Nonetheless, I noted earlier in this thesis that the viewpoint adopted in this thesis is that theories and models of computation are human constructions.

The second part of the counterargument requires careful discussion. Andrew Pickering noted that humans differ from nonhumans precisely in that people's actions have intentions behind them, whereas the performances (behaviors) of quarks, microbes, or machine tools do not[406]. He wrote, “I find that I cannot understand scientific practice without reference to the intentions of scientists, though I do not find it necessary to have insight into the intentions of things”. Pickering was correct when he wrote about naturally occurring phenomena such as quarks and microbes. But when Pickering wrote that it is not necessary to understand the intentions of machine tools, I agree with him insofar as he did not imply that it is not necessary to understand the *intentions of the creators of machine tools* to understand machine tools.

Although computers are not intentional, and although theories and models of computation are not intentional, computers are constructed by scientists who are intention-

405 For instance Donald Knuth wrote that computer science is an *unnatural* science (Knuth, 2001:p.167).
406 Pickering, 1993

al. Unlike the actions of computers, most of scientists' actions have motivations behind them. Hence, I argue that it would be ill-advised to state that (1) computers are constructed by computer scientists, and that (2) one can understand computers without understanding the intentions of computer scientists who built them. The only way to avoid contradiction would be to argue that computers are instances of abstract (universal) theories and models of computing, but then one would have to hold to the notion that there are no practical intentions behind building computers. If there are practical purposes behind building computers, then computers cannot be understood without understanding these practical purposes[407].

Consider the hammer, for instance. Assume that intelligent beings create a hammer, and then become extinct. In addition, assume that other intelligent beings find the hammer. The question is, "Is it possible for the finders of the hammer to understand what the hammer is without understanding the intentions of the creators of the hammer?". I argue that without understanding the intentions of the creators of the hammer, the finders of the hammer can *speculate* about the hammer, and perhaps speculate correctly, but not be sure about it. But if they know that the intention of the creator of the hammer was to drive nails through wood, they can be certain that they understand what the hammer is[408].

In a sense, it would sound more odd to say that mathematical and logical truths exist without any intelligent beings than to say that they exist only with intelligent beings. It would be to argue that something immaterial exists, which contradicts what natural scientists usually argue for; that is, natural scientists tend to argue that in order to exist, things need to have mass or at least some kind of energy. No matter which way one puts it, to argue that mathematical and logic truths exist without any intelligent beings would be to argue that something immaterial and eternal can exist and that people would somehow have access to these immaterial facts.

A major part of the debate about the explanations of stability concerns the natural sciences. According to Hacking, the arguments on either side of this debate are often taken from physics; Maxwell's Equations and the Second Law of Thermodynamics appear in debates regularly[409]. That physics is used as a reference discipline re-

407 Although computers can be built for the practical purpose of studying computation, which is a special case.

408 It is a different matter altogether whether one can ever fully understand the intentions and motivations of another intelligent individual.

409 Hacking, 1999:p.84.

flects on the tone of the debate. Relying on the premise that computers and naturally occurring things are two different classes of phenomena, I make the following argument: If a natural scientist encounters problems (anomalies), the difficulties most probably stem from unfamiliar, unrecognized, or unknown complexities in the interdependencies of the physical world. The problems in the social sciences arise from the complexity of social systems. The basic units in social scientists' research literally have their own minds and intentions[410]. However, if a computer scientist encounters problems, the difficulties arguably stem from computer scientists' earlier work—computer scientists have built the complexity or definiteness of their own discipline. Earlier design choices in, for instance, control structures, architectures, languages, grammars, techniques, data structures, syntax, and semantics, define the starting points for future challenges. In other words, computer scientists are much more responsible for the level of complexity of their own discipline than physicists, chemists, or biologists.

To the extent that one agrees with the aforementioned argument, the discussion on the third sticking point, "explanations of stability" may not be very fruitful in computer science. This is due to the rationalist and the constructionist explanation coming very close to each other:

- Rationalists might say that the stability of computer science is an *inherent quality* of computer science (a science that was built by humans called computer scientists).
- Constructionists might say that stability of computer science is a result of *external qualities* (i.e., the sociocultural characteristics of computer scientists).

In a human-made science that studies human-made phenomena it is hard to see which characteristics should be attributed to external qualities and which ones to the inherent qualities of science.

My three arguments on the last six pages cast serious doubt over the rationality of a discussion about the "explanations of stability" in the field of computing. Note, however, that none of my arguments *forbids* that discussion—or any other discussion on computer science for that matter. In fact, the above-mentioned three characteristics of computer science—diversity, turbulence, and human produceness—*en-*

410 Hitchcock, 2004:p.16.

able the mode of discussion in this thesis. Therefore, I leave this question open and move on to a brief conclusion of my standpoint on Hacking's sticking points.

Sticking Points: A Summary

An analysis of the realist-constructionist sticking points in computer science shows that there are some standpoints in studies of computer science that are so fundamentally contradictory that a researcher cannot commit to one without rejecting the other. Of course, a researcher can always choose to reject both of them, and to leave the question open (or to find an alternative solution, which may be the start of a new approach to science). There are also middle-ways to these sticking points but the middle-ways tend to run in trouble with both sides, and may be extremes of their own.

The first of these aspects deal with contingency: If a computer scientist holds to the contingency thesis (i.e., the constructionist side), then he or she takes the position that none of the foundations of computer science are *inevitable* parts of successful computing. If a computer scientist rejects the contingency thesis, he or she takes the position that whatever local contingencies there may be, there is a path that any successful form of computing takes, or a set of principles that any successful form of computing *necessarily* holds to. As happens more often than not with extremes, both of the extremes (inevitability and contingency) are easy positions, because arguing them is a straightforward denial of opposing arguments. Anything between leads to difficulties in explaining which aspects of computing are contingencies, and which are necessary. This sticking point is left unresolved for now, because the following chapters of this thesis partly cover the history of computational instruments, and help in making conclusions about the question of contingency.

The second of these controversial aspects is nominalism: If a computer scientist holds to structurism, he or she takes the position that computing is built on (or exhibits, or is a part of) an inherent structure of the world and logic. If a computer scientist holds to nominalism, he or she takes the position that all the structures that computer science deals with are ultimately human creations and that there are no structures in the complexity of world. Again, neither of the extremes is particularly useful, at least not for the purposes of this thesis. The ontological middle-way that

Searle proposed offers a framework for an analysis of different sorts of computational facts, and this framework is referred to later in this thesis.

As David Bloor wrote (and Popperians should agree), all knowledge, whether it be in the empirical sciences or even in mathematics, should be treated as material for investigation[411]. Scientists oriented towards researching brute facts can use whatever method they find suitable for their research, but they should not take any of their facts as ultimate or perfect. The same applies to researchers oriented towards institutional reality. Because the dividing line between Searle's two classes is not crystal clear in computer science, one of the responsibilities of the informed researcher is to constantly keep probing the line, asking questions, trying to make it clear for all researchers alike to what extent are their facts institutional or brute.

411 Bloor, 1976:p.1.

Technological Determinism

Any sufficiently advanced technology is indistinguishable from magic.[412]

In This Section:

- ✔ What is the position towards technological determinism and social construction of technologies in this thesis?
- ✔ Is technology value-free?
- ✔ Do machines make history?
- ✔ How can one measure technological progress?

Professor Steve Fuller argued that some of the most remarkable philosophers of science of the last century, such as Popper and Kuhn, promoted excessively idealized visions of science[413]. Fuller's interpretation was that they did this in order to avoid asking whether the instruments used in experiments were inspired by and/or applied in a military-industrial setting outside the experimental context. Whatever the reason for not discussing what the possible military-industrial ties were at the time, nowadays there has been plenty of research on the external influences of science. Sociologists and historians of science, for instance, have analyzed the tacit and direct influences and motivations in different kinds of technologies[414].

A sense of technology's power as a crucial agent of change has a prominent place in the culture of modernity, claimed Leo Marx and Merritt Roe Smith[415]. They described the collective memory of Western culture as being well-stocked with lore that states that the role of mechanic arts is to initiate change. Examples of such lore in the field of computing are those that recount information and communication technologies—especially the Internet—as triggering an Information Age[416], creating the digital divide and the New Economy[417], as well as computers and networks improving education[418], changing class structures in society[419], and leading to collective

412 Often referred to as "*Clarke's Third Law*"; found in Clarke, Arthur C. (1973) *Profiles of the Future*.

413 Fuller, 2003:p.89. Fuller is a professor of sociology at Warwick university.

414 See, for instance, collections of essays by Smith and Marx, 1994; Bijker and Law, 1992; or MacKenzie and Wajcman, 1999 .

415 Smith and Marx, 1994:p.*ix*.

416 See, e.g., Manuel Castell's trilogy *The Information Age: Economy, Society, and Culture* (Castells, 1996; Castells, 1997; Castells, 1998).

417 See Irving, 2004 and Eaton, 2004 in Egendorf, 2004.

418 Honey, 2004

419 Florida, 2003

intelligence[420], etc[421]. Many people in the field of ICT mistake technology as the answer, rather than the question[422]. Yet, many technologies tend to raise more questions than they can solve[423].

Simply defined, following Thomas P. Hughes, technological determinism is the belief that technical forces determine social and cultural changes[424]. Everett Rogers noted that the opposite of technological determinism is social determinism, or the social construction of technology[425]. According to this view, technology is a product of society, and its functions and forms are defined by the norms and values of the social system. Common sense says that if one is concerned about what technology does to society, or to people, one is a technological determinist to some degree. That is, if and only if one believes that technology can have an effect, one can be worried about it.

Even though researchers might, on some level, settle for the simple definition of technological determinism mentioned above, there really is no single account of technological determinism. Instead, there are different degrees of technological determinism and different emphases. One simplistic way to outline the different deterministic views is to place them along a line between "soft" and "hard" extremes, as suggested, for instance, by Smith and Marx[426]. Merrit Roe Smith wrote that the soft view holds that technological change drives social change, but at the same time responds discriminatingly to social pressures. The hard view perceives technological development as an autonomous force, completely independent of social constraints[427]. There are variants and degrees of each: For instance, one variant of the hard view, proposed by Richard Florida, is techno-utopianism, a belief that technologies, especially "killer applications", shape the course of human events[428].

But along with many science and technology researchers, many modern sociologists disagree with hard technological determinism, claiming that *technology is not the*

420 Lévy, 1997
421 See Negroponte, 1995, for a large number of tales and visions.
422 Smith, 1994b
423 Mills, 1959:p.15.
424 Hughes, 1994
425 Rogers, 2003:p.148. Social construction of technology is depicted in, e.g., Kline and Pinch, 1999.
426 Smith and Marx, 1994:pp.*xii-xiii*.
427 Smith, 1994
428 Florida, 2003:p.26.

cause of social change at all[429]. It has been suggested that technologies are a factor in social change, but only when combined with organizational, social, and economic adjustments[430]. These views are sometimes called *social constructionism* or *social determinism*—neither of which is a really good term for the phenomenon because both terms come from a context outside the discourse on technology[431]. Better terms are, for example, *the social construction of technology*[432] or *the social shaping of technology*[433].

Notwithstanding this problem with names, there is also a gamut of views on the soft side of the spectrum. Wiebe E. Bijker and John Law wrote that the views range from (*a*) acknowledging that artifacts per se do not carry meaning but that people give them meanings (this can be paralleled with Searle's concept of observer-relative features of objects[434]), to (*b*) understanding the social, political, psychological, economic, and professional commitments, skills, prejudices, possibilities, and constraints of artifacts, to (*c*) claiming that the form, function, and shape of technologies are purely socially determined[435].

Except for their extreme forms, the social construction of technology and technological determinism are not mutually exclusive but, depending on which viewpoints are considered, they do have commonalities. This leaves room for misunderstandings, ambiguities, and criticisms against each side.

One of the criticisms based on a strict reading of the terms "technological" and "determinism", is Bruce Bimber's criticism of the soft-hard division[436]. Bimber offered an alternative model with three interpretations of technological determinism: the normative, the nomological, and the unintended consequences approaches[437]. According to Bimber, the essence of a *normative account* is that the norms and practice of technology are removed from political and ethical discourse. The *nomological account* rests on "laws of nature" rather than on social norms, and is exemplified by

429 See, e.g., Castells, 2000
430 Florida, 2003:p.26.
431 For social constructionism in sociology, see Berger & Luckmann, 1966; for social determinism in psychology, see Holstein & Gubrium, 2003:p.13.
432 Bijker, 1992
433 MacKenzie and Wajcman, 1999
434 Searle, 1996:pp.11-13.
435 See, e.g.. Bijker and Law, 1992:pp.1-14, for some viewpoints.
436 Bimber, 1994
437 Bimber, 1994

the belief that "given the past, and the laws of nature, there is only one possible future". The *unintended consequences account* relies on the uncertainty and uncontrollability of the outcomes of actions such as technological development. Of these approaches, only the nomological account is, as Bimber saw it, really both technological and deterministic: It is truly technologically deterministic.

Technological Momentum

Thomas Hughes offered another alternative to the soft-hard juxtaposition, locating the concept of "technological momentum" somewhere between the poles of technological determinism and social constructionism[438]. Hughes noted that both of those views suffer from a failure to encompass the complexity of technological *change*. Technological momentum, on the other hand, infers that social development shapes and is shaped by technology.

The temporal axis in Hughes' concept of technological momentum is important in understanding technological determinism in the field of computing. Hughes' studies indicate that younger, developing systems tend to be more open to sociocultural influences, while older, more mature systems prove to be more independent of outside influences and therefore more deterministic by nature[439]. Yet, Hughes' account is still sensitive to the complexities of history, society, institutions, values, interest groups, social classes, and political and economic forces.

Hughes' temporal axis in technological determinism gives Searle's epistemological dimension of facts an interesting twist: It seems that when an innovation is born, that innovation is epistemologically subjective, but usually[440] the older and more prevalent an innovation gets, the more epistemological objectivity it gains. In Searle's books, that some things are screwdrivers are epistemologically objective facts: It is not only Searle's opinion that they are screwdrivers. As I see it, the increase in epistemological objectivity is closely linked with those innovations becoming more rigid, less responsive to outside influences, and thus more deterministic by nature.

The temporal dimension is important for computing for a number of reasons: It is of use in proportionating (or finding the place of) computing with other, usually older

438 Hughes, 1994

439 Smith and Marx, 1994:p.101; Hughes, 1994

440 This is not always the case. Some innovations fade away. For instance, the number of people who can use a slide rule is most probably declining as it is not taught in the schools anymore.

disciplines; it helps in understanding technological change and stability in computing (although it does not help in understanding stability in the theoretical sense as used in Hacking's sticking points, discussed on page 127). It also helps in understanding computing as a cause and an effect—shaping and being shaped by society. Note, however, that by *shaping* society, I do not mean that technologies, unintentional things, would shape anything, but that by being available for people to use, technologies may increase the chance of certain decisions or changes.

Hughes wrote that as technological systems grow larger and more complex (of which intercommunicating computational systems are a prime example), systems tend to shape society more than society tends to shape systems[441]. Hughes' hypotheses help explain the positions taken by determinists and constructionists: Social constructionists have the tools for understanding analyzing young systems while technological determinists have the tools for understanding analyzing mature ones[442].

In the computer scientists' world, technological determinism is commonplace and rarely questioned, as exemplified by phrases such as: "*Technology has always been the root cause of economic and social change*"[443], "*...the natural evolutionary situation of technology*"[444], and "*The Internet has already wrought great change in our society*"[445]. In the computer science discourse, computer scientists rarely require justification of deterministic views as the ones above. That is, technological determinism is the norm. Manuel Castells' texts support my view that in academia and science (in the field of computing, at least) there is a strong culture of belief in the "*inherent good of scientific and technological development as a key component in the progress of humankind*"[446]. It is, Castells wrote, a direct continuation of the Enlightenment and modernity.

Because technological determinism is a central part of the conceptual framework in this thesis, a number of common problems around it must be discussed. There are three problems or problematic questions that I find frequently in the sociological and philosophical literature about technology. These problems are the question of value-

441 Hughes, 1994
442 Hughes, 1994
443 Spindler, 1998 in Leone et al. (eds.), 1998.
444 Levinson, 1998
445 Godwin, 1999
446 Castells, 2001:p.39.

free technology, the question of the role of technologies in history, and the question of measuring progress. In the following, I analyze and take a position towards each of these.

Is Technology Value-Free?

At this point it should be noted that my use of the term *technology* is congruent with what Carl Mitcham called *technology as object* or *as artifact*[447] and what Stephen J. Kline called *technology as hardware*. Kline argued that this is the most common usage of the term, and that it concerns manufactured articles—things that are made by humans and that do not occur naturally[448]. Other uses of the term technology are, for instance, as *knowledge*, as *activity*, as *volition*, as *sociotechnical system of manufacture*, and as *sociotechnical system of use*[449].

Marshall McLuhan (1911-1980) called claims of value-free technology the "voice of the current somnambulism"[450]. It has been argued that all technologies are shaped by and mirror the complex trade-offs that make up societies (both technologies that succeed, as well as technologies that fail)[451]. Politics; economics; theories; notions about what is beautiful, ergonomic, or worthwhile; professional preferences; prejudices; skills; design tools; and the raw materials that are available—all these are, according to Bijker and Law, thrown into the melting pot whenever an artifact is designed or built. Langdon Winner argued that because of these inherent qualities, which are fused into technologies, technologies invariably favor the interests of some over the interests of others[452]. However, I suggest that (following Hughes' theory of technological momentum[453]) the proportions and functions of the different factors in Bijker and Law's melting pot process are not static, but dynamic. When technologies are designed and built, decisive and minor factors alike can change over time; for example, professional preferences may play a decisive role in the beginning but a minor role later.

---

447 Mitcham, 1994:p.161.
448 Kline, 1985
449 Mitcham, 1994; Kline, 1985
450 McLuhan, 1975
451 Bijker and Law, 1992:p.3.
452 Winner, Langdon (1986) *The Whale and The Reactor: A Search for Limits in an Age of High Technology*. University of Chicago Press:pp.55-56, as quoted in Smith, 1994:p.32.
453 Hughes, 1994

Extending Langdon Winner's question "Do artifacts have politics?"[454], I see that there are two basic ways of stating the claim that there is a dependent relationship between political, cultural, or ethical values and information, communication, and computing technologies. The first way is a strong claim:

> The development, diffusion, or maintenance of a given technological system *requires* a particular culture, a particular social form, or a particular set of values.[455]

This claim is a central assumption in arguments such as, "the innovation or diffusion[456] of the Internet *requires* openness, a techno-meritocratic culture, virtual communitarianism, an entrepreneurial economy[457], and a hacker ethic[458]". It could be argued that without accepting those values, one cannot have the Internet, or that if one wants the Internet, one also has to accept these values[459]. The second way, although weaker, is more plausible:

> The development, diffusion, or maintenance of a given technological system is strongly *compatible with* a particular culture, a particular social form, or a particular set of values.[460]

This claim is a central assumption in arguments such as, "the development or diffusion of the Internet does not *necessarily* require openness, a techno-meritocratic culture, virtual communitarianism, an entrepreneurial economy, or a hacker ethic—however, the more these prevail among a society, the better the chances are for innovation or diffusion of the Internet". In other words, *if* the Internet is highly compatible with those values, *then* the more the values in a society differ from those values, the more problems and conflicts there will be during the innofusion process.

The second, weaker claim sounds plausible. The creators of technology usually create technologies for certain purposes. Those purposes are usually in harmony with the values that the creators hold. In a culture with values that conflict strongly with

---

454 Winner, 1999

455 cf. Winner, 1999

456 James Fleck coined the term *innofusion* when he argued that the processes of innovation and diffusion are not separate spheres, but intertwined processes (Fleck, 1999). Everett Rogers' findings are in line with Fleck's argument (see Rogers, 2003:p.167).

457 Castells, 2001:pp.36-61.

458 Himanen, 2001

459 Adapted from Winner, 1999 (Winner's example is about nuclear vs. solar energy)

460 Adapted from Winner, 1999

the purposes of technology, technology may offer less advantages because some of the major possibilities that technology makes available might not be exploited. For instance, if recording the human voice were a taboo in a certain culture, VoIP (Voice Over IP) probably would not succeed in that culture. However, it is possible that if cultural values conflict strongly with the purposes of technology, the technology may have unexpected advantages altogether. For instance, it has been reported in a non-academic publication that youngsters in United Arab Emirates use Bluetooth devices for secretly initiating conversations with the opposite sex, because it is considered impolite for a man to speak in public to a woman with whom he is neither married nor related[461]. It must be noted that neither of the above-mentioned versions of the dependency claim make statements about technology per se; they make statements only about the creation, diffusion, and maintenance of technology.

Even if one were to agree that the sociocultural environment can affect the production and use of computing technology, one can still argue, fairly, that computational instruments per se are value-free. The term *value-free technology* usually means that technologies per se do not hold any values: Technologies are not inherently good or evil.[462] In these seemingly trivial notions there is a quirk: It would surely be odd to refer to unintentional objects as having morals. But because the "inherently value-free" argument is often rationalized by an example: "A pen can be used to write a peace treaty (mostly good) or to declare a war (mostly evil)", the "inherently value-free"-clause can be understood differently.

My interpretation is that those who use the argument that a pen can be used to write a peace treaty or to declare a war, use it as an example that the pen has the *capability for* being used for good or evil. A common counterargument is that because *nuclear weapons* have no capability for good, but only a capability for evil[463], technologies are not value-free[464].

---

461 Sharp, Heather (2005) Phone Technology Aids UAE Dating. In BBC News Dubai, July 29th 2005. Available at `http://news.bbc.co.uk/1/hi/world/middle_east/4718697.stm` (accessed September 27th, 2006)

462 Admittedly, *good* and *evil* are exceedingly artificial and subjective categories. In fact, "good vs. evil" is such an inane and dichotomous juxtaposition, that I would not like to use it. However, as it has been used widely in this context, it is used here in a similar manner.

463 Note that nuclear weapons are not *designed* to protect from an enemy nuclear attack, even though they may do so. They are designed to kill people in masses, which is considered to be bad in most ethical accounts. Furthermore, they do not separate soldiers and civilians, women and men, young and old, enemy and ally, which is considered to be bad even according to the rules of war. One can still be of the opinion, however, that destroying enemy cities in masses is good.

It does not matter if nuclear weapons can be used only for evil or not, it is still a fact that nuclear weapons are inanimate, unintentional objects and they most probably do not know anything about morals. At the same time, the designers of nuclear weapons have motivations and morals and they must carry the responsibility and the pang of conscience that come from creating nuclear weapons.

Value neutrality can also be connected with the attributes of technology. Consider, for example, a simple attribute, such as *speed*, as a value-free attribute of technology[465]. It is not unheard of that an increase in the speed of computing technology is associated with technological progress. However, those who claim the attributes of technology to be value-free, do not claim an increase in speed to be either good or bad (since speed is value-free).

For those who take technology to be value-free, if technology gets faster, it gets faster, not "better". Similarly, if technology gets slower, it gets slower, not "worse". So, in fact, the notion of technological progress is thus reduced to the notion that computing gets faster or slower, without any normative notions like "better" or "worse". In this sense, *technological progress* means simply *technological change*. Only if one connects progress with a normative statement like—"speed is good"—then additional speed in computing technology is progress.

Also, those who claim that attributes of technology are *not* value-free should find it hard to label speed either good or bad. For example, an increase in computing speed has made the human genome project possible (which many people perceive as good), but it has also made new nuclear, biological and chemical weapons; the Carnivore; and the Echelon possible (which many people perceive as bad). Labeling an attribute of technology as desirable and calling development in that area *progress* is definitely not a straightforward matter. Of course one could distance oneself from consequentialism (utilitarism) and state that (1) technology is not value-free, and that (2) speed is inherently good—although then "speed=good" would be truth by definition, and alterable as such.

---

464 Although the nuclear weapon example is a common example (Williams, 1994; Winner, 1999; MacKenzie, 1999), it is an extreme example which easily leads the discussion to politics and the morals of war (an apparent oxymoron), which are not the focus of this thesis. Therefore, I will not discuss the nuclear weapon example here.

465 Parts of this discussion are from a joint work with Teppo Eskelinen, from the University of Jyväskylä (Eskelinen, Teppo; Tedre, Matti (forthcoming) *Three Dogmas of Computing*. An article manuscript).

I think that the crux of the debate is this: Although technology *as such* is value-free, the development and the use of technologies are always processes in which the developers and users of technology have a number of motivations. Unlike technologies, conscious human actions are not value-free.

That is, one has to make a distinction between two things: *technology per se*, and the *production or use of technology*[466]. Technologies are unintentional objects or concepts and unintentional objects and concepts do not have morals. Contrary to what Winner wrote, *technology does not favor anybody's interests.* The designers or users may do so, however. Technologies *as such* are no more good, evil, benevolent, or malicious than they are happy, sad, angry, or meek. Technology does not favor anybody's interests any more than it thinks or dreams. If there were a technology that could be used only for exterminating the human race, and for nothing else, the technology would still be value-free. Even though someone might want to say that such a technology favors the interests of non-human life on the planet Earth, that is not the case. The *designers* of such technology, however, cannot evade the responsibility for designing and building such technology, and its *users* cannot evade the responsibility for using such technology. The designers carry as heavy a responsibility for the outcomes of technology as the users do—without the designers' effort, the users would not have had that technology in the first place.

Technology is produced by conscious human beings, and conscious actions entail motivations. That is, the creators of technology always have a motive for creating technology—and motives are value-laden. For instance, reducing poverty is often considered to be a good motive, and greed is often considered to be a bad motive. Even thirst for knowledge is a motive. When computer scientists work on computing technology and theories, they are motivated by a number of things, be it income, passion, a thirst for knowledge, or something else. Computer scientists are also aware of a number of possible uses for their plans; some of these uses can be malignant and some benign.

Although computer scientists cannot know to which end their products will ultimately be used, and although computer scientists cannot know all the possible consequences of their products, they do have a responsibility for their work. Mario

466 This distinction was made to me by Dr. Esko Marjomaa (oral communication, February 23rd 2006).

Bunge wrote that an entire process of technology production may be morally objectionable if the goals it aims at are exclusively evil[467]. Bunge argued that "the technique of evil doing is evil itself". Although the end product, an inanimate object or concept, is value-free, the choices that computer scientists have made during the production are not value-free. Computer scientists *cannot* be held responsible for unintended consequences or unforeseen malicious uses of technology. Computer scientists *must* be held responsible for the foreseeable effects and side-effects of technology, and they *must* be able to justify their choices[468].

Do Machines Make History?

The second argument I make in the context of technological determinism is that *machines do not make history*. Although Robert Heilbroner's (1919-2005) classic article "Do Machines Make History?"[469] advocates a seriously qualified deterministic view, it is necessary to state that I take it that technologies do not impose social and political characteristics upon a society. During the turn of the twentieth century there was a common belief that technology defines the social and political characteristics of a society. For instance, Karl Marx (1818-1883) proposed that the substructures of the society (economical and material) define the superstructures of society (social and cultural)[470]. Modern sociologists, however, disagree, and indeed argue that technology is not the cause of social change at all[471]. For instance, John Fiske wrote, "*New technologies do not in themselves produce social change, however, though they can and do facilitate it*"[472]. Technologies have been claimed to be one factor in social change, but changes take place only when combined with organizational, social, and economic adjustments[473]. In other words, technologies are not a cause of change, but they can be a catalyst or a factor of change, or make change possible[474]. Machines as such do not actually do anything; they are not intentional.

467 Bunge, 1979 in Scharff & Dusek, 2003. First printed in Bugliarello, George; Doner, Dean B. (eds.) (1979) *The History of Philosophy and Technology*. University of Illinois Press: Urbana, USA:pp.262-281.

468 I concede that in reality, the choices that people make when they develop technologies are not straightforward. Kristin Shrader-Frechette listed a number of different ethical aspects that technologists are expected to weigh when they make decisions about technologies (Shrader-Frechette, 1992; first printed in Becker, Lawrence C.; Becker, Charlotte B. (eds.) *Encyclopedia of Ethics, vol. 2*. Garland Publishing, New York:pp. 1231-1234).

469 Heilbroner, 1967

470 This topic, noted in this connection by Teppo Eskelinen, is discussed in Eskelinen, Teppo; Tedre, Matti (forthcoming) *Three Dogmas of Computing*. An article manuscript.

471 See, e.g., Castells, 2000.

472 Fiske, 1994

473 Florida, 2003:p.26.

474 See, for instance, Castells, 2001:p.39.

People construct the social and political characteristics of a society, often assisted by machinery. One could rephrase Heilbroner[475] and state that computing technologies have an impact on social and cultural change, mainly by changing the material conditions of human existence.

Technological Progress is Immeasurable

The third argument I make related to technological determinism is that *technological progress at large is immeasurable*. This is perhaps the most controversial claim concerning technological determinism in this thesis, so particular clarity is needed[476]. First of all, progress, if characterized as "moving forward", relies on the assumption that there *is* a given goal towards which technological change is heading. This characterization leads back to determinism. For example, one would need to assume there is a pre-determined path that technologies will take, which is not assumed in this thesis. Furthermore, the fact that a particular technological development works well is not an indication that that technological development is progress, or even a step in the right direction. In the big picture, that step may be even a step regretted later, as exemplified by the adoption of the QWERTY keyboard.

The QWERTY keyboard worked well when the main concern of typewriter manufacturers was not having the hammers of the typewriter getting stuck together—it was designed specifically to slow down typists[477]. But nowadays that particular technology has become an example of what diffusion and STS researchers call a *technological lock-in*[478]: Its pervasive diffusion prevents competing products that are better in many senses, such as the Dvorak keyboard, to emerge.

Although progress cannot be measured if there is no specific goal, progress *can* be measured if a goal (and in some cases, a measurement unit) is specified. Towards a given goal, say, a manned space flight to Mars, or discovering a cure for HIV, there can be progress—but without a goal, the notion of "progress" is misleading. In this case, Thomas Kuhn noted that the direction of progress is *away* from something, not

475 Heilbroner, 1994

476 This is discussed in Eskelinen, Teppo; Tedre, Matti (forthcoming) *Three Dogmas of Computing*. Unpublished manuscript.

477 Rogers, 2003:p.10.

478 Arthur, 1999; MacKenzie and Wajcman, 1999 :p.20.

towards something[479]. However, because progress *away* from something can actually be harmful for future development, it is dubious if *progress* is the right term.

At this point the value-neutrality argument must be clarified. I have argued that the Internet is not necessarily a tool of liberation and democracy; it can be used as a tool for oppression as much as a tool for liberation. This claim would imply that the Internet is a value-neutral technology. Secondly, I have claimed that all technologies embody, for instance, politics, economics, theories, notions about what is beautiful, ergonomic or worthwhile, professional preferences, prejudices and skills. This would imply that there are no value-neutral technologies. The distinction is clear: Technologies are unintentional things and they do not hold values. People who design technologies do hold values. Technologies are always better suited for doing some things than other things. The values of the designers may affect technological development so that the resulting technologies are better suited for some things than others. But I also recognize what Bruce Bimber called "unintended consequences", that an element of uncertainty and uncontrollability always accompanies technological development[480]. As Bimber wrote, even willful, ethical social actors are unable to anticipate the final consequences of technological development. However benevolent the designers of technology are, technologies can be used in unforeseen ways in the future.

The Social Shaping of Computers

The Internet is an obvious example of the discussion about values and technology in the field of computer science. Freedom of expression was a central value for the early developers of the Internet[481]. I suggest that if that would not have been the case, the medium might be much more controlled. Control of the Internet can in fact be seen in countries where freedom of expression is largely restricted. For instance Saudi Arabia, China, and Iran have severely restricted Internet access by filtering foreign web sites[482].

---

479 Kuhn, 1996:p.170.
480 Bimber, 1994
481 Castells, 2001:p.55.
482 Reporters Without Borders (2006) *Freedom of the Press Worldwide in 2006. Annual Report.* Available at `http://www.rsf.org` (accessed September 27th, 2006).

After its early development between the 1960s and the 1980s, the Internet has been largely shaped by commercial uses since the 1990s[483]. E-commerce has affected technologies used on the web: For instance, PHP3 was written for *e*-commerce, Flash for the needs of the MSN portal, and Java for the convergence of consumer devices. And certainly, *e*-commerce has had a significant impact on the development of secure transactions over the net (credit card numbers) as well as on data mining techniques (e.g., "customers who bought X also bought Y").

It seems, paraphrasing Michael Smith[484], that the issue of technological determinism is not an issue about technology after all; it is a product of a curious cultural and political fetishism whereby artifacts stand in for technology, and technology in turn signifies national progress. But as C. Wright Mills aptly put it, those who use technological devices do not understand them, and those who create them do not understand much else[485]. That is, Mills concluded, why one may *not*, without great ambiguity, use technological abundance as the index of human quality and cultural progress.

It is hard to predict the prospective directions of technological change (or progress). The examples of alleged false predictions by technological leaders are numerous, the best-known probably being "I think there is a world market for maybe five computers" (attributed to Thomas Watson (1874-1956) of IBM, 1943, but never confirmed), and "640K should be enough for anybody" (attributed to Bill Gates, Microsoft, 1981). But more interesting than false predictions, are correct predictions. For instance, Vannevar Bush's (1890-1974) Memex[486], can be considered to be an early equivalent of hypertext. An interesting question—one that cannot be answered here—is if technologies are predictable or if there are just so many predictions that some of them repeatedly come true?

There is also a profound problem with the concept of "revolutionary technologies", and with predicting them. The problem with the concept is that it is not clear how the term *revolutionary* should be operationalized. The questions that arise are, for instance, "What kinds of technological innovations constitute a revolution?", "Is a technological implementation or the theory behind it a revolution?", and "Is techno-

483 Castells, 2001:p.55.
484 Smith, 1994b
485 Mills, 1959:p.175.
486 Bush, 1945

logy revolutionary because of its inherent qualities or because of its effects?". The problem with predicting revolutions is that if they are understood in the Kuhnian sense, the fact that they cannot be predicted makes them revolutions. Suppose that discussing future technology in the seventeenth century, a visionary would explain to town folks that according to his or her prediction, someone is going to invent the transistor in a few hundred years. "Transistor, what is that?" the town folks would exclaim. With tremendous difficulty, the visionary would be able to explain the transistor, only to stop and realize that he or she just invented the transistor[487]. Another interpretation is that the visionary has only predicted the transistor, but without an operational prototype, the visionary has not invented the transistor.

Of course radical innovations (or technological developments) can also be predicted a short time before the actual prototypes are built, but what seems to make them radical is that just a short time before, they were unimaginable ("short" is, admittedly, a vague modifier). There are a number of technologies that have been predicted, but not yet implemented. Take using fusion as a source of energy, for example: Even though it was theorized in the early twentieth century, there still are no functional, practical implementations—yet, such an implementation would not come as a radical innovation. Such an implementation would be regarded as an outcome of a long, cumulative research and design process.

What are called radical, unimaginable innovations, often seem to occur despite the establishment, be it academic or industrial, and not because of it[488]. It has been argued that many of the interesting breakthroughs take place at the intersection of different disciplines and cultures[489]. Note, however, that one cannot *aim at* unimaginable or unexpected innovations[490]. (In the same way, Kuhn argued that one cannot aim at unexpected findings in science.)

Pat Hughes and Graham Cosier have suggested that a clear distinction between an invention and a revolution should be made[491]. They argued that the internal combustion engine was a vital invention, but ultimately people could have done without it

487 This example, in different forms, can be found in a number of places; I have paraphrased Eskelinen, Teppo (2005) oral communication, 21st August 2005.

488 For instance, the stored-program concept and high-level languages were conceived despite the opposition of the computing establishment of the time (Flamm, 1988:p.48; Sammet, 1969:p.143; Bright, 1984).

489 Johansson, 2004

490 See footnote 174 on page 74.

491 Hughes & Cosier, 2001

(steam cars could have been used, or electric cars have been developed instead of gasoline-run cars). Electricity, on the other hand, brought about a truly revolutionary change, a change bigger than anything in business itself[492]. Although I do not agree with Hughes and Cosier in their deterministic claim that technology drove societal change, they are, of course, correct in that electricity was indeed one of the major factors in the second industrial revolution[493].

Hughes and Cosier also claimed that because eye glasses may soon become obsolete, they may not have constituted a revolution[494]. Although I do not aim at offering an alternative definition of revolution, Hughes' and Cosier's argumentation seems awkward. The fact that an innovation may become obsolete does not nullify its impact at the time of its innofusion. The Internet has enabled a number of changes in a large variety of areas of life, from economy, communications, and computing to leisure, entertainment, and love. Although the Internet may (and probably will) become obsolete one day, it has been a catalyst for a number of important changes. Hence, my point of view is that the Internet is a revolutionary technology. (Still, the Internet did not make history, but the people who built the Internet, the people who made use of it, and the people who resisted it, did.)

All the attempts to use current technologies as an example of how technologies follow a certain predetermined path are fundamentally flawed because, in retrospect, technological development naturally has "followed a path". But it is doubtful whether it has followed the only possible path. Only imaginable progress can be predicted; revolutions in technology are unimaginable and thus impossible to predict, and cannot constitute a path. It is doubtful whether there have even been technological revolutions in the same sense that there are claimed to have been scientific revolutions—this is merely a matter of definition. Alan Kay, for instance, presented an argument that the computer revolution has not even happened yet[495]. One can argue, though, that revolutions in instruments have spawned revolutions in sciences. But is the Internet a result of long-term development with a zest of innovation, or is it really a revolution as Kuhn would see it? How about the stored-program com-

492 Hughes & Cosier, 2001

493 The term *industrial revolution*, popularized by Arnold Toynbee (1852-1883) is quite vague, but usually it is used to refer to a number of technological, socioeconomic, and cultural changes (Encyclopædia Britannica Online, 2004).

494 Hughes & Cosier, 2001

495 In the abstract of his keynote speech, Kay used the history of books as a comparison. See Kay, 2000.

puter? A look at the history of computing technologies in Chapter Three sheds light on this question.

2.3.Intermission

Nihil tam absurdum quod non dicatur ab aliquo philosophorum.[496]

What would philosophers of science like Popper, Kuhn, and Feyerabend have had to say to a modern computer scientist? How would they have characterized modern computer science? How would a computer scientist take their message? To start with, the philosophers themselves do not agree. Steve Fuller, a proponent of Popper's philosophy, argued that in many respects, the postmodern condition associated with Kuhn's ascendancy marks a return to the *pre-modern* sensibility[497]. Fuller argued that what is often called relativism—be it in praise or condemnation—is simply the "ancient attitude", perhaps most clearly defended by Aristotle (384-322 B.C.), that all knowledge must be adequate to its objects. Ethnographic sociologists now speak of *context sensitivity* and cognitive psychologists of *domain specificity* to mean much the same thing[498]. The relativist side and the realist side are still both strong today. In this section, I briefly summarize this chapter. This section is lightly referenced, because all these matters have been discussed earlier in this chapter.

According to C. Wright Mills, in every intellectual age one style of reflection tends to become a common denominator of cultural life[499]. Mills reported that during the modern era in Western societies, the physical and biological sciences have been the major common denominators of serious reflection and popular metaphysics. He wrote that the cultural meaning of physical science is becoming doubtful and inadequate. Mills seems to have predicted correctly; the intangible and immaterial seem the common denominator for postmodernism. Economies have become ones of signs and space[500], societies have become knowledge-based, and sciences have diversified. Indeed, emphasis on brute facts seems like a common denominator of the modern era, whereas an emphasis on institutional facts seems like a common denominator of the postmodern era.

---

496 "There is nothing so absurd as not to have been said by a philosopher". Attributed to Marcus Tullius Cicero (106 B.C. - 43 B.C.).
497 Fuller, 2003:p.67.
498 Fuller, 2003:p.67.
499 Mills, 1959:pp.13-16.
500 As proposed by Lash & Urry, 1994.

In order to discuss the brute facts of computing together with the institutional facts of computing, a broad conceptual framework for computer science is necessary. Figure 8 shows some of the variety of concepts connected with science that have been discussed in this chapter. Because the area is so thoroughly complex, and because the complex interconnections cannot be expressed on a two-dimensional plane, the topics are mapped in the figure in a semi-random manner. That is, many related concepts are close to each other, but many other related concepts just simply cannot be drawn close to each other. In this sense, the reader should not take Figure 8 as a relational or semantic map but as a compilation of the central concepts discussed in this chapter.

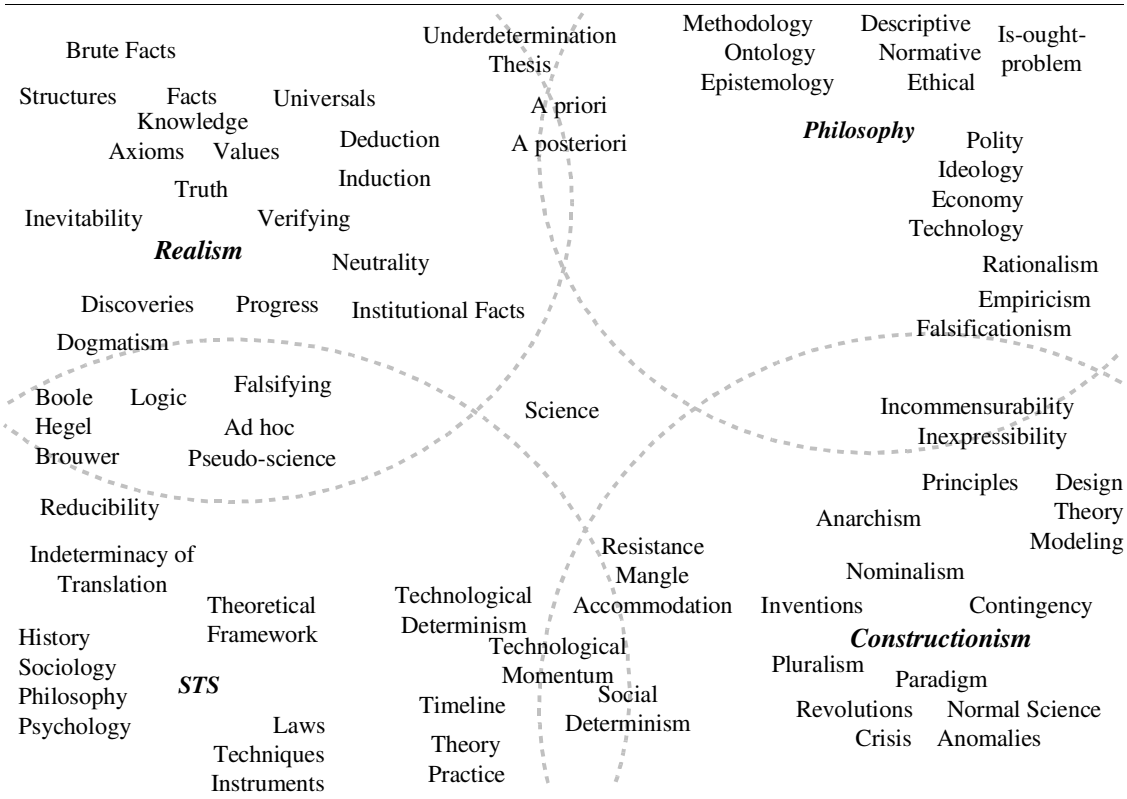


Figure 8: Windows to Science

Figure 8 presents some concepts in the philosophy of science and STS that are relevant to this thesis. There is some overlap between these two fields, but whereas the philosophy of science is more concerned with the ontological, epistemological, and methodological questions of science, STS is more concerned with viewpoints from the history, sociology, anthropology, and psychology of science and immerses itself in the interdisciplinary study of the theoretical and practical sides of science. The

subject matter of STS concerns, for instance, the scientific laws, techniques, instruments, and theoretical frameworks of science and technology. The philosophy of science focuses more on the descriptive, normative, and ethical components of science—and, for example, scientific polity, ideology, and methodology. The philosopher of science might ask, for instance, if prescriptions of how science should be done are derived, or can be derived, from descriptions of science or of the natural world (i.e., the is-ought problem). The STS researcher might ask whether the principles of design, theory, and modeling are results of contingency or necessity.

The two fields overlap, they deal with a number of similar or same questions, and many people take STS to be an umbrella term that contains the philosophy of science as a subfield. The arcs in the Figure 8 are there to sketch the quadrants related to STS, philosophy, realism, and constructionism in the figure—they are not to be taken as dividing lines or such.

The Realist Camp

Ever since early Greek philosophy, extending throughout the history of philosophy, there has been a strong rationalist line of thought; take Plato and René Descartes, for instance. Rationalists would argue that there are truths in computing that are universal; especially those in the more theoretical branches of computing. From the rationalist point of view, universally true facts are discoveries, rather than parts of a human-made system of science—actually, some think that if all axioms are known, all knowledge can be derived from those axioms. Naturally, this thinking also entails presuppositions that there are inherent structures in the world that scientists can find, and that all the new findings are inevitable: At some point progress would have led to them anyway. According to this technological deterministic viewpoint, because science and technology are parts of the inherent structures of the world, they are free from the influences of values and culture.

However, Imre Lakatos' standpoint was that all the facts that scientists deal with have formed over years of proofs and refutations (of which the Church-Turing Thesis is an example in computing). The position of Lakatos' teacher, Karl Popper, was that one would never be able to prove anything universally true. According to Popper, steadfast attempts to falsify even the most cherished theories are considered to be good scientific practice. Then again, making *ad hoc* modifications to theories is

considered to be bad scientific practice, and often pseudo-science. There would be no place for dogmatism in pure falsificationism, because any theory that fails should be abandoned.

The underdetermination thesis poses the following problem for falsificationism: "Given a number of theories that explain the data equally well, how does one determine which theory is best?". This problem does not have a general solution, but neither does the problem posed by what I chose to call the *underrepresentation problem* (a problem which is arguably more apt for computer science than underdetermination): "Given a number of models that all successfully model and predict different aspects of a phenomenon, but that all are flawed in some way(s), how does one determine which model to use?".

However, all theories most certainly are not equally valuable because the underdetermination thesis requires increasing the number of *ad hoc* modifications, which makes some theories look increasingly implausible or even fictitious. Different theories suit different situations. Questions that inevitably arise from having a number of theories that all suit different situations, are "If currently dominant theories in academic computing are imperfect, on what grounds is their dominance justified?", "What kinds of arguments raise the dominant theories above other theories that explain the same phenomena also imperfectly but that are flawed in different ways?", and "Which arguments do computer scientists use when choosing between competing models, all flawed in different ways?". All these questions are discussed in Chapter Three.

The Constructionist Camp

According to Thomas Kuhn's constructionist theory, science is an agreement among scientists, and as long as there are not too many anomalies that the current normal science cannot explain, everything is fine. If anomalies start to accumulate, the paradigmatic science may drift into crisis, and out of the turmoil of the scientific crisis, a new revolutionary paradigm that can explain the anomalies can arise. Most constructionists think that many scientific facts are facts only because scientists choose to believe so. According to the nominalist argument, which is connected with pluralism and constructionism, there are no inherent structures in the world, and

obviously, new technologies and theories are no more (and no less) than inventions —human-made things.

It has been argued that at the wake of Kuhn's triumph, science has come to be justified more by its paradigmatic pedigree than its progressive aspirations[501]. Scientific progress, in Steve Fuller's opinion, is better described by Popper's philosophy. However, in light of the history of science, Kuhn's description seems to be more valid[502].

The concept of progress is indeed dubious in the constructionist account, since after a paradigm shift, new theories are more or less incommensurable with the old ones: Radically new theories are often inexpressible with the language of old science. Two competing paradigms may not only explain the same facts in a different way, but they may explain different facts altogether. The demand of reducibility or "instant clarity" in computing is a hindrance for (at least) two reasons, one practical, one ideological.

From the practical point of view, if new or emerging embodiments of computing have to be explained in (incommensurable) terms of current academic computer science, it either makes it impossible to introduce entirely new concepts, or it reduces the power of the expression of new concepts to the power of the expression of current academic computer science. In other words, the indeterminacy of translation, accompanied with the demand for reducibility, set an upper limit for what can be expressed by theories bound to what can be expressed by current science.

From an ideological perspective, the demand of reducibility and instant clarity is a forceful occupation of intellectual territory. The demand of reducibility elevates academic computer science to the intellectual standard status. That is, any concept that cannot be clarified by showing a counterpart in current science is automatically considered a flawed concept according to computer scientists.

The incommensurability thesis that Kuhn held to, denies humans, much in a Kantian way, any access to universals. Feyerabend went further and denied that even logic is universal: "There is Boole, there is Hegel, there is Brouwer, there are the many postmodern systems of logic". In Feyerabend's anarchistic vision, there is no need to

501 Fuller, 2003:pp.6-7.

502 But not all philosophers of science think it is really valid. In addition to Fuller, 2003, see, e.g., Feyerabend, 1975.

suppress even the most "outlandish product" of the human brain. According to Feyerabend, everyone may follow his or her inclinations and science (conceived as a critical enterprise) will profit from such an activity. He wrote that one should be encouraged not just to follow one's inclinations, but to raise them, with the help of criticism (which involves a comparison with existing alternatives) to a higher level of articulation and thereby raise their defense of this criticism to a higher level of consciousness.[503] The question that Feyerabend's anarchistic philosophy of science raises in the field of computing is: Is *anything goes* a fruitful prescription for the field of computing, or should a certain amount of dogmatism be allowed?[504]

Middle Ground

Even though one might refuse constructionist, anarchist, positivist, falsificationist, empiricist, or inductivist accounts as being inadequate for providing a philosophy of science (or an account of science), some or all of the accounts may offer useful methodological tools (such as deduction and induction) for testing arguments. However, using methodological tools assumes that the researcher acknowledges the underlying philosophy. One cannot argue for an *a posteriori* interpretive methodology if one does not take a distance from universalism, at least to some extent. Similarly, it seems a bit strange (although not incorrect) for a researcher to utilize a positivist methodology but hold that results achieved from using the positivist methodology are subjective or open for subjective interpretations. It seems that there is no middle way but either one extreme or another. Yet, because all of the extremes face difficulties, none of them are philosophically adequate either. Only a variety of pragmatic accounts prevail.

There are a number of accounts that fall between constructionism and realism, or at least accounts that are credible defenses of a middle way, and I considered three of them. The first was Thomas Hughes' *technological momentum*, which relies on a temporal variable to blaze a trail between technological determinism and the social construction of science (social determinism). According to the technological momentum argument, young technological systems exhibit characteristics of social con-

503 Feyerabend, 1970

504 Actually, Feyerabend's anti-dogmatic stance has been referred to in the field of computing: *The Feyerabend Project* is an attempt to give new dynamism to computing–see De Meuter et al., 2002. Another reference to Feyerabend in the field of computing can be found in Snelting, 1998. Snelting refers to Feyerabend in a negative sense.

struction, but when they become more widespread and more established, the characteristics they exhibit are closer to those characteristics connected with technological determinism.

The second account, Andrew Pickering's *mangle of practice*, has been named "the most materialist contribution to the social studies of science to date"[505]. Pickering used an example from quantum physics to show a structure and processes between (1) theoretical models, (2) design and theory of instruments and how they work, and the (3) instruments themselves. When a researcher works, usually things do not go as planned; the world resists. The researcher accommodates this resistance by revamping some or all parts of the structure, and tries again. This process is called "the mangle"; in the end, the researcher hopes to get a robust fit between the three elements of the structure of the investigation.

The third account was John Searle's *realist* account that can accommodate *socially constructed* facts. Searle's account relies on a simple ontology: The world is made of particles in fields of force; some of these particles are organized into systems; and some of these systems are living systems that have evolved consciousness. These conscious creatures can represent objects and states of affairs in the world to themselves (*intentionality*). In Searle's theory, there are brute facts in the world, statements whose truth value does not depend on any attitudes of people. However, conscious, intentional creatures can attach particular meanings to objects or facts—or create new facts altogether. Those facts exist only because of conscious creatures, and they cease to exist when there is nobody left to hold those facts.

In this chapter I have developed a conceptual framework for this thesis. All the different accounts of science and technology introduced in this chapter concern the fields of physics, mathematics, chemistry, and other natural or mathematical sciences. I argue that this framework is also suitable for an analysis of computer science—but the framework is tested in Chapter Three.

In Chapter Three I analyze the formation of computer science using this framework as a tool of analysis. My analysis is focused on the sociocultural formation of computing—I analyze if the development of computing bears the characteristics of social construction and if it fits my framework well. That is, I ask if the development of

505 Hacking, 1999:p.71.

computing bears the characteristics of contingency, nominalism, constructionism, or other sociocultural or human characteristics that may have affected how computer science and technologies have developed.

3. The Development of Computing as a Discipline

Com·pute
VERB: **1.** Make a mathematical calculation or computation. **2.** To reckon or calculate: "*Can anyone here compute the square root of 10201?*". **3.** (*colloquial*) To make sense: "*Does that compute, or do I need to explain further?*"

Com·put·ing
NOUN: **1.** Action of using a computer. **2.** The study of computers and computer programming.
ETYMOLOGY: From Latin *computāre*: to count, to compute.
SYNONYMS: calculate, cipher, cypher, figure, reckon, work out.

In this chapter I discuss the second of this thesis' two central themes, *computing*[1]. Specifically, in this chapter I focus on the academic interpretation of computing—computer science. Even though scientific approaches to computing existed long before the advent of electronic digital computers[2]—and even though systems of numeration have existed for at least 5000 years and algorithms comparable to those of today have existed for at least 3500 years[3]—my study is limited to the era after the birth of electronic digital computing.

One can hardly argue that the terminology in computer science is fully established. Terms such as *computer science*, *computing as a discipline*, and *the field of computing* are used in a variety of competing ways. The debates about what computer science is, what computer science is about, and what computer science should become are common in the past of computer science, and not unheard of in contemporary debates either. Many disputes arise from the different meanings of the term *computing*. Instead of the restricted dictionary definition (see the box above), the term *computation* is among computer scientists more often understood as the implicit or explicit execution of algorithms[4].

Most of the early debate over the nature of *computing*, *computer science*, and *computer engineering*, (along with names such as *comptology*, *hypologi*, *Turingineering*, and *computics*[5]) revolved around the question whether computer science was a discipline separate from mathematics, electrical engineering, and applied physics[6].

1 See the box on this page for a synthesis of the *compute* and *computing* entries in *Wiktionary* and *Webster's Online Dictionary* (accessed September 27th 2006).
2 Knuth, 1974
3 Williams, 1985:p.9; Knuth, 1972
4 Scheutz, 2003
5 Ensmenger, 2001; McKee, 1995
6 Ensmenger, 2001; Aspray, 2000

Some authors claimed that computer science was a new engineering discipline[7]; some authors made a strict distinction between making programs (i.e., software engineering) and "*designing classes of computations that will display a desired behavior*"[8]; some authors claimed that "*... computer science is the study of the theory and practice of programming computers*"[9] and many authors objected to the persistent "*computer science equals programming*"-myth[10]. In this thesis, the computing activity is connected with computational instruments (which are defined in Section 3.1), and I often use the term *technoscience* to refer to both science and technology.

The tone of the debates about the essence of computer science changed during the formation of the discipline. Whereas one of the major foci of early (1950s) professionals of computing was on achieving a disciplinary autonomy, in the following decades the focus turned towards what kinds of topics should be included in or excluded from computer science. Computer science has been diversifying radically ever since the birth of electronic digital computing, and diversification is still a characteristic of computing. Diversification has kept the debate about the disciplinary identity of computer science alive. In this section I offer my interpretation of the development of computing, the formation of computer science as a discipline, and the anatomy of research in computer science.

The history of the debate around computer science is a prime example of the social construction of a scientific discipline. The definition-creating process has included several kinds of stakeholders: institutions—such as the ACM, the IEEE, and the DPMA (Data Processing Management Association); opinion leaders and authorities —such as George Forsythe, Peter Denning, Edsger Dijkstra, Richard Hamming, Donald Knuth, and Peter Wegner; professional factions—such as groups of business-, information-, and academically oriented practitioners; and societal institutions —such as universities, industries, and governments[11]. In Section 3.3 I situate the early development of modern computing in my framework, which was constructed in Chapter Two, and in Section 3.4 I analyze the disciplinary development of computer science with the same framework.

7 Loui, 1995
8 Dijkstra, 1972
9 Khalil and Levy, 1978
10 Denning, 2004; Denning et al., 1989; Ralston, 1981; Ralston and Shaw, 1980
11 Ensmenger, 2001, demonstrates the "*rich complexities hidden behind seemingly straightforward debates about professionalism*".

Because of the ambiguity of the term *computer science*, one cannot begin an investigation into the development of computer science by defining computer science. One of the hardships of defining computer science is that definitions have two functions: to tell *what a discipline is*, and to tell *what a discipline is not*. This dual function is hard to achieve, and there is often a struggle between a narrow definition and a broad definition[12], and I present examples of both in this chapter. I do not arrive at any conclusive definition of computer science because I have not been able to either find or construct a definition that would not downplay any aspects of computer science and that would still be informative about the variety of branches that are considered to belong to computer science today.

I also present a number of perspectives on what kinds of knowledge computer scientists work with. For instance, Donald Knuth described computer science as having theoretical and practical sides. He wrote, "*Theory and practice are not just two sides of the same coin. They deserve to be mixed and blended, but sometimes they also need to be pure*"[13]. However insightful, Knuth's view is a unidimensional, simplified juxtaposition that does not take into account other crucial aspects of science and technology. Practical and theoretical aspects are not the only aspects affecting the development of computer science or the everyday work of computer scientists. There is also a plethora of economic, technical, historical, sociocultural, ideological, political, philosophical, institutional, professional, and ethical factors that affect computer science and the work of computer scientists.

In this chapter I analyze my source material using the theoretical framework developed in Chapter Two. The questions I ask are, for instance, "Do these sources indicate characteristics of contingency or necessity?", "Do these sources support technological determinism or social constructionism?", "According to what scientific principles do computer scientists seem to work?", and "How are scientific controversies in computer science solved?".

In Section 3.1 I clarify what I mean by computational instruments; in Section 3.2 I expand upon the concept of problem in computing; in Section 3.3 I analyze the creation of digital electronic computing and the early developments of the field of computing; and in Section 3.4 I analyze the formation of computer science as an academ-

12 McGuffee, 2000
13 Knuth, 1991

ic discipline, analyze the different sides of the discussion about what computer science is, and present and analyze meta-research of computer science.

3.1. What Are Computational Instruments?

The computing scientist could not care less about the specific technology
that might be used to realize machines,
be it electronics, optics, pneumatics, or magic.[14]

IN THIS SECTION:

- ✔ What is a computational instrument?
- ✔ Why are not all objects that can be used in computations, computational instruments?
- ✔ What controversial issues are there about the nature of computational instruments?

The modern definition of a *computer* is that it is a machine that handles data automatically under the direction of a set of instructions specified in advance[15]. Although this definition is sufficient for most purposes, it is too short and too vague for discussing computing on the general level. A *general-purpose computer* (electronic, digital, Turing-complete computer) is a special case of a *computational instrument*—an underlying principle in a general-purpose type of computer is that strings of symbols can be interpreted both as data and as programs[16]. In this thesis, a definition for the category of *computational instruments* is necessary. This is not to say that the aforementioned definition of computer would not be good as such. However, it deserves a deeper conceptual analysis, especially in ontological and functional terms.

To begin with, many computational instruments are tools that help people do tasks that would be harder or practically impossible otherwise, at least without using any memory aids, such as a paper and pen. In principle, computational instruments do not do anything that people could not do (albeit people may do those tasks slowly). Almost anyone can do tasks that require performing very simple algorithms like adding or multiplying numbers. Furthermore, some human beings have the ability to calculate very rapidly and very accurately. A person could run complex algorithms without any tools, if he or she had sufficient memory capabilities. (For example, some *mathematical savants* have been shown to have "virtually endless" memory of series[17].)

14 Dijkstra, 1987
15 Ceruzzi, 1997
16 Denning, 1985
17 Gardner, 1993:pp.155-156.

This raises questions such as, "Is the human mind a computational instrument?", "Is the Turing Machine a computational instrument?", and "Are algorithms per se computational instruments?". A natural follow-up question is, "What is the relationship between algorithms and computer programs?"[18]. Computer programs (which can be considered to be one sort of formalization of algorithms) can be considered either abstract or concrete objects, depending on one's viewpoint, argued Brian Cantwell Smith[19]. Herbert A. Simon (1916-2001) too noted that computers are both abstract objects and "empirical objects"[20]. Also Lewis and Papadimitriou posed the question whether hardware, or software, or theory is the basis of computation[21]. The answer to this question is merely a matter of definition and a playground for endless disputation. To avoid equivocation in this matter, I use the following definition:

A computational instrument is a physical object. (Definition 1)

As a corollary, computational instruments are part of the ontologically objective world; their existence as well as their chemical and physical properties are brute facts. Furthermore, in this thesis I consider living organisms, except for those intentionally made for biological computation, to not be computational instruments (Yet this exclusion also leaves room for debate: Consider, for example, Stephen Wolfram's "universe as computation"[22], or the possible non-physical computing technologies of the future).

Systems of numeration could be counted as a sort of machinery[23] because they enable solving numerical problems beyond the limits of the human mind. So, are systems of numeration computational instruments? Systems of numeration deal with abstractions, and without these abstractions people would be able to deal only with the tangible world. Actually, I argue that the only number naturally attributable to physical objects is *one (1)*. The number *zero (0)* indicates nothingness, and there exists no *no physical object*: Physical objects can only naturally exhibit their existence ("this is a brick"). They cannot exhibit their non-existence ("this is a no-brick").

18 The relationship between a program and an algorithm is not easily, and I am not covering that discussion here. For a discussion about the problems of exactly defining algorithms, see Cleland, 2001.

19 Smith, 1998:pp.29-32 (the discussion between McMath and McPhysics).

20 Simon, 1981:pp.22-26.

21 Lewis & Papadimitriou, 1998:p.247.

22 Wolfram, 2002

23 See F.P. Barnard (1869) *Report on the Machinery and Processes of the Industrial Arts* (New York: Van Nostrand) in Williams, 1985:p.1.

The number *two (2)* involves an abstract coupling of two physical objects (one object and one object), but abstractions are ontologically subjective (although the number two is arguably an epistemologically objective thing).

Perhaps the following is a bit too trivial of a notion, but without (explicit or tacit) systems of numeration, people would be able to deal only with the number one. Based on definition 1, because systems of numeration are not physical artifacts, but theoretical constructions, systems of numeration are not considered to be computational instruments in this thesis. They are a part of the foundations that computational instruments are based on, but not computational instruments as such.

It could be defined that any physical artifact that has the capability of extending human mental capabilities (e.g., extending calculation ability) is a computational instrument. Following this definition, piles of stones, sticks, marks in the sand, or fingers would be computational instruments. They *(1)* extend human memory capabilities (for instance, 20 piles of stones can be used to extend the "immediate memory capacity" of people, which has been argued to be 7±2)[24], *(2)* they make abstractions possible (white stones might mean sheep and red stones might mean pigs, or white stones might mean tens and red stones might mean hundreds), and *(3)* they enable algorithms for at least addition and subtraction.

However, without an intentional agent giving a stone or a stick a computational function, stones and sticks are not computational instruments per se. If all physical objects were computational instruments, people would live amongst a universe of computational instruments (indeed, some argue that people do[25]). Therefore, definition 1 needs to be expanded upon. One might say,

> *A computational instrument is a physical object that has been explicitly assigned a computational function.* (Definition 2)

That is, a computational instrument is an object that is used to aid in computation. *A computational function* falls into Searle's category of *agentive functions*[26]: Agents (such as people) intentionally put objects into computational use.

24 Miller, 1956
25 Wolfram, 2002, "computational universe".
26 Searle, 1996:pp.20-21.

Pieces of wood[27], cords[28], coconut leaves[29], fingers[30], or pen and paper can meet the aforementioned constraints. Yet, only those pieces of wood, cords, coconut leaves, fingers, or pens and paper that are given a computational function can be computational instruments, and only when there is at least one intentional agent upholding that function. The demand of an intentional agent upholding a computational function of an object classifies computational instruments as being *ontologically subjective*. Although the existence of a computational instrument is a brute fact, (ontologically objective), computational instruments are not computational instruments without the intentionality of people. The ontologically objective features of objects, such as their chemical composition or their physical structure, do not yet make them computational instruments; there also must be people or other conscious or intentional beings who consider those objects to be computational instruments. When there is no intentional agent assigning a computational function to a given PC, the PC can no longer be considered to be a computational instrument. Computing is not an inherent function, not even for calculators or abaci[31].

In a definition of computing as a discipline by Denning et al.[32], computing (limited to computer science and engineering) is divided into three components. These components are *theory*, *design*, and *modeling* (abstraction). Theory derives from the mathematical sciences, design from engineering, and modeling from the natural sciences[33]. Many authors, such as Gibbs and Tucker[34], foreground (1) "the formal properties of algorithms and data structures" over (2) "tools and theory" and (3) "applications" in computing, but none of those three aspects is foregrounded in this thesis.

Theory, design and modeling are so intricately intertwined in computing that it is irrational to say that any one is fundamental. Denning et al. claimed that despite their inseparability, theory, design and modeling are distinct from one another[35]. Theory is concerned with the ability to describe relationships among objects and prove them

27 Needham, 1959:pp.68-80: Needham wrote about calculating rods in the Far East.
28 Ascher & Ascher, 1981: Ascher and Ascher wrote about the quipu of the Incas.
29 Ascher, 2002:pp.7-9: Ascher wrote about knots on coconut leaves.
30 Zaslavsky, 1980: Zaslavsky examined some African ways of using fingers to count.
31 If one interprets some naturally occurring phenomena as computation, then one might say that computing is an inherent function of those phenomena.
32 Denning et al., 1989
33 Denning et al., 1989
34 Gibbs and Tucker, 1986
35 Denning et al., 1989

correct. Design is concerned with the ability to implement specific instances of those relationships and use them to perform useful actions. Modeling is concerned with the ability to use those relationships to make testable predictions about the world. There are, though, other relationships between theory, design, and modeling: For instance, empirical modeling starts with the experiential construction of knowledge about a given domain, leading to a precise formulation or specification[36].

Using Gibbs and Tucker's and Denning et al.'s definitions of computing as a basis, a third definition of computational instruments can be stated as:

> *Computational instruments use organized structures or models to represent information.* (Definition 3)

Organized structures and models that represent information are also called *data structures*. Data structure can be formally defined, as Hansen[37] does, as a quadruple <D, F, S, A> where D and F are domain and function definitions (which define externally observable behavior), and S and A are storage structure and algorithms, which implement the functioning of the data structure. This formal definition may be best suited for the purposes of theoretical computer scientists and mathematicians, since this level of formalization is not necessary for the purposes of this thesis. Yet, Hansen's definition hints at what properties can be expected from organized structures or models that represent information.

The quadruple <D, F, S, A> can be further divided into external and internal descriptions E=<D, F> and I=<S, A>[38]. In practice, the external descriptions provide information where the data structure is applicable, and what one can do with it. The internal descriptions can be very simple or very complex: For instance, S can be bits, words, trees, or graphs. This applies well to basic computational tools, such as the abacus. The domain of the abacus is integers (fractions, too, if defined accordingly[39]). The function definitions are *storing* a number, *adding* to, *subtracting* from, *multiplying* by, and *dividing* by another number. The storage structure varies between versions of abaci[40], and the algorithms for the functions vary accordingly.

36 Beynon & Russ, 1995
37 Hansen, 1981
38 Hansen, 1981
39 See Williams, 1985:p.59.
40 Williams, 1985

The definitions of Gibbs and Tucker[41] as well as Denning et al.[42], lead to yet another definition of computational instruments:

> *Computational instruments use algorithms to manipulate organized information*. (Definition 4)

Gibbs and Tucker as well as Denning et al. emphasized the role of algorithms in their models, and it is hard to see disagreement in the claim that algorithms are central to computing[43]. The set of algorithms (A) mentioned earlier in the context of data structures—addition, subtraction, and such—implement the external functions (F) of the data structures.

An algorithm, according to Donald Knuth[44], is a finite set of rules that gives a sequence of operations for solving a specific type of problem; an algorithm must be finite (it does not loop forever), it must be precisely defined, it may have input, it has to have output, and it must be effective. In addition, a procedure that has all of the characteristics of an algorithm but that possibly lacks finiteness, is called *a computational method*[45]. Such computational methods, include, for instance, reactive processes (e.g., computer programs) that interact with their environment.

There are viewpoints in computing in which algorithms and data structures are seen to be woven together. For instance, object-oriented programming is based on such an abstraction. Alternative models of computation lead to quite different views on data structures and algorithms: For instance, quantum computing uses interdependent, probabilistic models of data; randomized algorithms aim at some level of random behavior to, for instance, speed up computation or avoid worst-case scenarios. Defining computational instruments as they are defined in this section may not do justice to all forms of computation, and the definition given here is just one of infinitely many possible definitions.

41 Gibbs and Tucker, 1986
42 Denning et al., 1989
43 See Knuth, 1974.
44 Knuth, 1997:pp.5-7.
45 Knuth, 1997:pp.5.

Yet another definition of computational instruments can be derived from what Denning et al.[46] called the fundamental question underlying all of computing, "What can be effectively automated?":

Computational instruments automate tasks. (Definition 5)

Definition 5 rules out, for example, a pen and paper as a computational instrument. Even though one can actually use a pen and paper to emulate a Turing Machine (and, accordingly, emulate any computer), a pen and paper are not computational instruments because they do not automate the task at hand: The task has to be done by a human.

The definitions 1 to 5 of computational instruments are illustrated in Figure 9. All objects regardless of their ontological status are represented by the space around the figure, and *physical objects* (def.1) are a subset of all objects. Objects that have a *computational function* (def.2) are a subset of all objects, intersecting physical objects. The set of all the objects that meet the criteria of definitions 1 and 2, is further delimited by those objects that handle *structured data* (def.3). Furthermore, the objects that handle data *algorithmic*ally (def.4), delimit the set of computational objects still more. The last intersecting ring denotes the objects that *automate tasks*. In this thesis, an artifact can be considered to be a computational instrument if it meets all the aforementioned constraints (definitions 1 to 5—see Figure 9).

I hope that the five definitions of computational instruments discussed above deepens and explains the definition of *computer* mentioned in the beginning of this subsection ("a machine that handles data automatically under the direction of a set of instructions specified in advance[47]"). Particularly, this detailed five-point definition brings up some issues that might not spring to mind from the short definition. This definition is certainly deficient, but then again—all definitions are, and all definitions leave room for debate[48]. Hence, some of the major sticking points that this definition raises are dealt with here.

46 Denning et al., 1989

47 Ceruzzi, 1997

48 If there was an unanimously agreed upon comprehensive definition, there would not be debate about the definitions. For different views about the concept of computation, see Copeland & Sylvan, 1999; Copeland, 1997. For a different view about what constitutes computing, see, for instance, Norman, 1997.

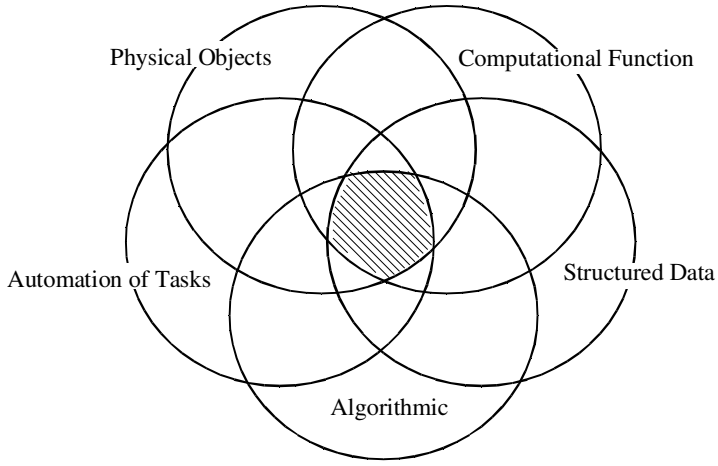


Figure 9: Requirements for Computational Instruments

Problems With the Definition of Computational Instruments

The first major sticking point here, connected with definition 1, is that this definition excludes abstract tools for computation, such as calculus and other non-physical phenomena. It even excludes computer programs that have no physical counterpart, that is, computer programs without an actual implementation running in computer memory. Determining the "physical counterpart" of an algorithm is difficult: Is an algorithm written on paper a physical counterpart?

One can give a well-grounded argument that algorithms are abstract, non-physical things. Computer programs are merely one form of expression for algorithms, so computer programs can also be understood as abstract and non-physical. On the other hand, computer programs in computer memory *do* have physical counterparts. Because this problem is well-documented by Brian Cantwell Smith[49], it is not further discussed here. (Just for the sake of clarity, as was earlier remarked, it is a computer given an agentive function[50] that does things; an algorithm, a program (on one's mind, on paper, or even on a disk), or a computer per se do not do things.)

The second major sticking point, connected with definition 2, is that the definition presented here excludes many objects that can do complex computational functions, but that are not specifically *built* for such tasks. These functions are, in Searle's taxonomy, called *non-agentive functions*[51]. When one says, "the function of the heart is to pump blood", one gives a function to a naturally occurring phenomenon as a part

49 Smith, 1998:pp.29-32.
50 Searle, 1996:pp.20-21.
51 Searle, 1996:pp.20-21.

of a theoretical account[52]. The brain can perform computational functions, but there is no intentional agent[53] that has given that function to the brain. Yet, living organisms that are genetically engineered for computational tasks, such as those organisms that researchers of strands of biological computing aim at, would still be considered computational instruments (although computational beings might be a more suitable term). In short, just that computer scientists can interpret some naturally occurring phenomena as computations does not mean that the nature would give computational functions to phenomena.

Another minor sticking point is related to the teleological-deontological[54] problem: Can computing be an inherent (not observer-relative, necessary) function of an artifact? Certainly, computing is not an inherent function of naturally occurring phenomena, such as stones or pieces of wood. But if something is built specifically for computation, does computing *become* an inherent function of that artifact (*telos*; *a purpose*)? Following Searle's[55] argument, from an outsider's (alien's) viewpoint (intrinsically speaking) there are no such things as computers—there are only things that people treat as computers. Surely, things that are built to be computers do computations much better than many other things, like socks or forks, do. But a computer is still a computer (*ontos*; *the thing*) merely because people use it as (or made it for the purpose of, or regard it as) a computer.

The third major sticking point, connected with definition 5, is that this definition excludes some technologies that intuitively would belong to the category of computational instruments, but that do not automate tasks. Take the abacus, for example. Even though the abacus fulfills all the other requirements, it does not automate calculations. Michael Williams[56], who is a historian of computing, called abaci "aids to calculation" instead of computational instruments or tools. Then again, the slide rule, another simple tool that deals with discrete and organized data, is used algorithmically (albeit the algorithm is very simple) and automates calculation, so the slide rule is here considered to be a computational instrument. Even though the automation-requirement makes determining whether some objects are computational instru-

52 Searle, 1996:pp.20-21.
53 In this thesis, I shall not analyze the possibility of a higher force such as gods giving this function.
54 *Teleological* in the sense of *goal* or *purpose*; *deontological* in the sense of *necessity.*
55 Searle, 1996:p.12.
56 Williams, 1985:p.48.

ments a bit difficult, it also clarifies the status of some objects: Fingers, brains, and rocks are debarred from the category of computational instruments.

The fourth, and perhaps the most controversial, sticking point comes from the fact that computers are nowadays ubiquitous. Certainly, a general-purpose computer is a computational instrument. But there is digital computing technology everywhere: cars, ovens, clocks, doors, phones, PDAs (Personal Digital Assistant), clothing, and so forth. The ubiquity of embedded computing technology makes it very hard to determine which of these objects fulfill the criterion of serving a "computational function". This criterion is ultimately subjective. On one hand, almost everyone can agree that even though a modern car can have twenty computational instruments, the car as such is still not a computational instrument. On the other hand, many people would probably say that a PDA is a computational instrument. But there are many things that fall between PDAs and cars and that may or may not be considered computational instruments, depending on the judge.

The lines drawn by these definitions are artificial. One can certainly find examples where two tools fall on opposing sides of the line between computational instruments and everything else, even though they both might seem equally fit or unfit for the category of computational instruments. All in all, every attempt to categorize or classify phenomena is somewhat artificial. In this thesis, this definition of computational instruments is necessary to make the concept of computational instruments as unambiguous as possible, but the definition is certainly deficient for many other uses.

3.2.What Is a Problem in Computing?

Houston, we've had a problem here.[57]

In This Section:

- ✔ Why is the concept of *problem* a controversial issue?
- ✔ Are problems the same to everyone?
- ✔ Are there problems that are more "real" than other problems?
- ✔ What is the difference between *open* problems and *closed* problems?
- ✔ How are problems approached in computer science?
- ✔ Why should not one just solve problems as he or she wants?
- ✔ Why could not computer scientists just borrow the problem-solving tools from mathematics?

The dictionary definition for "problem" is "*a question to be considered, solved, or answered*"[58]. However, in this thesis this concept ought to be analyzed further, for what is a problem to one person may not be a problem at all to another[59]. What is considered a problem in history class is not at all similar to a problem in physics class. For instance, C. Wright Mills offered a sociologist's definition of "practical problem": "*Often what is taken by the liberally practical to be a 'problem' is whatever (1) deviates from middle-class, small-town ways of life, (2) is not in line with rural principles of stability and order, (3) is not in concurrence with the optimistic progressive slogans of 'cultural lag', and (4) does not conform with appropriate 'social progress.' But in many ways the nub of liberal practicality is revealed by (5) the notion of 'adjustment' and of its opposite, 'maladjustment'.*"[60]

Although the book *Problem Solving with C++*[61] may help with computer scientist's problems, it does not help much in sociologists' problems as described by Mills. In addition, the problems presented in school differ from problems at home. The problems encountered on a national level differ from those encountered on an individual level[62]. It has also been argued that every age tends to think it is special, facing problems that have never occurred before—the "arrogance of the present"[63].

57 Apollo Spacecraft Program Office, Test Division (1970) *Apollo 13 Technical Air-to-Ground Voice Transcription: Tape 36/13*. Manned Spacecraft Center of National Aeronautics and Space Administration: Houston, Texas.
58 AHD, 2004 (American Heritage Dictionary, Online Edition)
59 Mills, 1959:p.76.
60 Mills, 1959:p.90
61 Savitch, 2004
62 For a discussion on the topic, see Tedre, 2004.
63 Hughes & Cosier, 2001

C. Wright Mills made a distinction between *troubles*, which are individual problems, and *issues*, which have to do with matters that transcend the local environment of the individual and which are public matters[64]. Karl Popper wrote that there is a difference between the problems of scientists and philosophers: When a scientist faces a problem, he or she can go "*straight to the heart of the* [structured problem]", but when a philosopher faces a problem, he or she does not face an organized structure, but rather "*something resembling a heap of ruins*"[65]. Having all these different viewpoints of the concept of *problem* raises some questions: "What is a problem?", "Are problems subjective or objective?", and "Who decides which problems are important?".

Key questions about problems, from the viewpoint taken in this thesis, are, "What is the definition of a problem?", and especially, "Can there be more than one legitimate definition for *problem* at a time?". There is arguably a number of common misuses of the term. Marcelo Borba noted that it is important to distinguish a problem from a trivial question to which the answer is known without any need for reflection[66]. For instance, the question "What color are the pants I'm wearing?" is not a problem. Another common misuse of the term is using it in a situation where "problem" is associated simply with "not knowing", Borba wrote. Not knowing how many provinces there are in Finland is not a problem for most people because they most likely do not care about knowing the answer. In the same manner, if a person is given a shuffled deck of cards, a *sorting problem* (which is a sort of a computational problem) does not exist if the person does not need an ordered deck.

Borba suggested that one valid use of the term *problem* is, "*If an obstacle occurs in the course of someone's own existence, and he/she does not know how to overcome the obstacle, then he/she has a problem.*"[67] Dermeval Saviani suggested that a problem has two sides—the *subjective side* that is the feeling of necessity, and the *objective side* that is the situation that puzzles the consciousness[68]. In other words, in each

64 Mills, 1959:pp.8-9.
65 Popper, 1959:p.1 (in the preface of the first edition, 1934).
66 Borba, 1990
67 Borba, 1990
68 cf. Saviani, 1985

problem there is an obstacle (objective aspect) that a person *wants* to overcome (subjective aspect).

It has been argued that *problem solving* (in the sense of instructions or general rules on how to overcome a problem) depicts only one aspect of the more general concept —*problem management*[69]. A solution (whether it is a definite unequivocal "answer to a problem" or, e.g., a resolution rising from a debate) is nothing more than one stage in this process. Problem management may also involve identifying, comprehending, expressing, formulating, solving, and evaluating the problem in question. C. Wright Mills noted that many scientific and technological developments actually raise more problems, both intellectual and moral, than they solve, and the problems they have raised lie almost entirely in the area of social, not physical, affairs:

> *Recent* [1959] *developments in physical science—with its technological climax in the H-bomb and the means of carrying it about the earth—have not been experienced as a solution to any problems widely known and deeply pondered by larger intellectual communities and cultural publics.*[70]

Three Classes of Problems

Generally, problems can be classified in many ways, but it seems that from the constructionist point of view it is a suitable classification method to divide classes of problems according to the size of the subject group. In the following text, I present three arguments that support this criterion of classification, and I introduce corresponding classes of problems.

First, I argue that there are problems that exist in all cultures and for all people. For instance, "What are the rules for an ideal society?" is, arguably, a question that (at least should) concern everyone from time to time—at the very least when the rules of society put the person in an awkward situation. Another example of a problem that seems to appear in all cultures is the problem of cosmology: "*the problem of understanding the world—including ourselves, and our knowledge, as part of the world.*"[71]. In this thesis this class of problems is called *general problems*.

Second, I argue that some problems exist only for one individual person and are attached to a particular context—for example, many ethical problems are highly sub-

69 Sutinen & Tarhio, 2001
70 Mills, 1959:p.15. Note that even though Mills' text is now almost half a decade old, it still seems valid.
71 Popper, 1959:p.xxiii.

jective and contextual. For instance, “Should I tell a little white lie to save someone from embarrassment?” is a question that exemplifies a subjective and contextual problem. (Hereafter these problems are referred to as *intimate problems.*) In a thesis focusing on individual-level phenomena, this might be the most interesting class of problems in this classification scheme, but in this thesis this class of problems is of secondary importance.

Third, I argue that some problems that are trivial, insignificant, or illegitimate to other cultural groups may be essential or central to others. Social philosophers may not (for various reasons) think about mathematical problems as real problems at all, whereas mathematicians may not see philosophers' problems as valid problems. According to Borba, it is fair to claim that the seemingly objective aspects of problems —obstacles—are culturally influenced, for what is considered an obstacle in one culture may not be one in another culture[72]. In this thesis, the term *limitary problems* is used to indicate this class of problems[73].

Many philosophers of science have originally been physicists or mathematicians (for example Imre Lakatos, Thomas Kuhn, Paul Feyerabend, and Pierre Duhem). That is perhaps why a problem in the philosophy of science is usually considered to be something that can be put in an explicit, structured, and deterministic form (i.e., in Popper's words, into a form of a *theoretical problem*[74]). Theoretical problems are often limitary problems. Finding the optimal solution for the traveling salesman's problem (TSP)[75] can be considered an obstacle in any culture, but not many cultural groups other than the group of theoretical computer scientists may feel the necessity to overcome it—perhaps not even traveling salesmen.

If one agrees with the three arguments mentioned above, it follows that the understanding of how a problem is defined is a socially constructed, culturally mediated, and individually interpreted concept—which necessarily means that there cannot ex-

72 Borba, 1990

73 Note that there may be other classification schemes or names for the types of problems. However, in this thesis these three terms are used: *general*, *intimate* and *limitary* problems.

74 See Popper, 1959:p.88.

75 The traveling salesman's problem is as follows: A salesman should visit a number of cities, using the shortest possible route between them and returning to the starting point. Which route should he or she choose? In reality, a salesman quite probably does not care to figure out which route is ultimately the shortest one, but he or she may choose the route by other criteria (schedules, traffic jams, speed limits), or just not care about a few extra kilometers on the road. In theoretical computer science, however, finding the optimal solution for the TSP is a central problem.

ist descriptive criteria by which some problems would be "better" or "worse" than others. Granted, some problems can be more common, taxing or abstract than others, but if for some reason it would be imperative to rank or compare problems, it still would be problematic to apply "semi-quantifiable" criteria such as prevalence, difficulty, or level of abstraction to problems. Furthermore some phenomena such as war raise both intimate problems (how to survive it, how to die with honor, how to make money out of it, etc.) and general problems (its effects upon economic, political, domestic, and religious institutions, etc.)[76].

Following Mills' thinking, computer viruses can be considered problems in two of the three aforementioned classes. Viruses are intimate problems, for computer users have to keep their virus programs continuously up to date, viruses consume computing and communication resources and sometimes destroy files or mess up whole operating systems. Viruses are also limitary problems because economies, governments, and other institutions of information societies can be put down on their knees by them. (Luciano Floridi noted that viruses can also be sources of income for some groups[77]. Be that as it may, very often a problem to one is a source of income to another.) Finally, viruses are not general problems, since the majority of the population of the world would not care less about whether a computer here or there crashes or not.

Authentic and Artificial Problems

In addition to dividing classes of problems according to the size of the subject group, problems can also be divided into two classes according to their authenticity. This division, which is essential for understanding the ambiguity of the concept of problem, is the division defined by Marcelo Borba[78] between *pseudo-problems*[79] and *authentic problems* (artificial problems). Pseudo-problems are imposed ones, and not really problems for people personally. Borba wrote that, for example, students may attempt to solve pseudo-problems only in order to get good grades. In computer sci-

76 cf. Mills, 1959:p.9.

77 Floridi, 2003

78 cf. Borba, 1990. I would rather use the term *quasi-problems*, but in this thesis, the term *pseudo-problems* is used. Using *quasi-problems* would connote *resemblance*, whereas *pseudo-problems* connotes *fake* or *hoax*.

79 Not to be confused with what Popper classified as pseudo-problems (see e.g. Popper, 1959:pp.314-315). For Popper the criterion of demarcation between science and pseudo-science is that scientific theories must be falsifiable; so in the Popperian sense (1) propositions that are not falsifiable and (2) problems based on these propositions are (1) pseudo-propositions and (2) pseudo-problems, respectively.

ence, scientific truths are generally seen as unbiased, and pseudo-problems are thought to be culturally neutral—after all, pseudo-problems mostly relate only to the science itself. This is a self-fulfilling view.

As I see it, pseudo-problems relate mostly to computer science itself, and computer science becomes increasingly applicable to these pseudo-problems. This may ultimately lead to a construction of a science that feeds only itself and fulfills only its own needs, becoming disconnected from its surroundings. Paul Feyerabend noted this phenomenon: "*The slowly emerging conceptual apparatus of the theory soon starts defining its own problems, and earlier problems, facts, and observations are either forgotten or pushed aside as irrelevant*"[80]. Using Sutinen and Tarhio's terms[81], this phenomenon may lead to the problem management principle of *what-you-know-is-what-you-get* rather than *what-you-need-is-what-you-get*. The former principle, Sutinen and Tarhio wrote, builds on a specific set of features, corresponding to a closed problem. The latter principle, they continue, states a "customer's problem" (sometimes in an obscure way), but it gives more room for innovativeness. When closed problems are introduced at schools, a teacher often expects the student to come up with not *any* solution but with a solution that the teacher is familiar with[82].

Closely related to the class of pseudo- and authentic (real-world[83]) problems is the concept of the *auxiliary problem*. An auxiliary problem (π), as defined by George Pólya (1887-1985)[84], is a problem which is not solved for its own sake—that is, for it being of interest to the researcher—but because the researcher hopes that its consideration may help solve another problem (ρ)[85]. Solving the auxiliary problem π does not solve the problem ρ that raised the auxiliary problem; the auxiliary problem π is a means by which a solution for original problem ρ is sought.

80 Feyerabend, 1993:p.155. Italics in original underlined.
81 Sutinen & Tarhio, 2001
82 Ackoff, 1978:p.9.
83 In this thesis, the terms *authentic problem* and *real-world problem* are used synonymously.
84 George Pólya was a mathematician who worked with, e.g., David Hilbert (1862-1943), especially in areas of group theory (Pólya's formula) and analytic number theory (Hilbert-Pólya conjecture). His book on general heuristics in problem solving, *How to solve it*, has sold over a million copies, yet is still considered a mathematically-sound book.
85 Pólya, 1957:pp.50-51.

Open and Closed Aspects of Problems

Because Feyerabend and Kuhn (and to some extent, Popper) noted that (normal) science carries with it a set of certain problems that are considered scientific, and that the success of normal science is judged by the success of scientists in solving the problems normal science holds, it is important to dig deeper into the concept of problem, and how it is understood in computer science. Even though there is an abundance of books in computer science that are named "Problem Solving Using *X*"[86], there has not been a lot of analysis of the concept *problem* in the area of computer science. Sutinen and Tarhio analyzed the term *problem* by composing a classification of three binary positions, similar to Table 2.

Table 2: Examples of Problem Classes[87]

Start	***Tech***	***Goal***	***Examples***
C	C	C	Routine problems[88], such as applying a formula (e.g. $E = \frac{1}{2}mv^2$)
C	C	O	Algorithm animation (with obscure learning goal)
C	O	C	Mathematics *in making*, sorting, other kinds of puzzle-solving activity[89].
C	O	O	*P=NP*?[90]
O	C	C	Designing an operating system
O	C	O	Wasting time, horoscopes
O	O	C	Preventing cheating
O	O	O	Problem of cosmology, rules for an ideal society

In this classification (Table 2), *Start* refers to the starting point, parameters, or input of the problem, *Tech* refers to the technique, method, or algorithm to solve the problem; and *Goal* refers to the result or output of the problem[91]. In Sutinen and Tarhio's

86 Replacing *X* with any programming language, for example, C, Pascal, Java, or such, and then searching for a book by that name produces plenty of results.

87 This is a modified version of Sutinen and Tarhio's table (Sutinen & Tarhio, 2001). The original was not completely suitable for this thesis; for instance, the original problem instances in O-O-O were "Spontaneous situation, improvisation, driving on a slippery road", and in C-C-O "Marital problem". Driving on a slippery road usually has predetermined goals like arriving home safely or not crashing the car. In addition, I am not convinced that marital problems have a fixed methodology or technique.

88 As Pólya would call them; see Pólya, 1957:p.171.

89 On mathematics *in making*, see Pólya, 1957:p.117. Thomas Kuhn would probably demote this class to "puzzles" instead of "problems", as I discussed earlier in this thesis (page 74), see, e.g. Kuhn, 1996:p.36. *Puzzles* are a subclass of problems. Most often puzzles cannot be solved because of the *self-imposed constraints* in them (Ackoff, 1978:p.9).

90 A central question in the theory of computation. In short, it is asked that if solutions to a decision problem (where the answer is a definite yes/no) can be *verified* in a reasonable amount of time regardless of input size, can the answers also be *computed* in a reasonable time?

91 Sutinen & Tarhio, 2001

definition these three binary positions, denoted here by <S,T,G> can be either open (O) or closed (C). *Open* means undefined, obscure, unclear, or open-ended, whereas *closed* means the opposite.

It should be noted that representing problems as three (<S,T,G>) binary positions $C \vee O$ (1) reduces the view of the world into binary statements and (2) prunes the infinite number of variables in the world to three. Considering these three positions <S,T,G>, it might perhaps be beneficial to represent the positions as continua, which would yield a three-dimensional model as depicted in Figure 10. In this model the dimensions would not have to be *either* closed *or* open, but could also be partly open and partly closed (although having only three dimensions would still be a simplification). For example, in Figure 10 point (*s, C, C*) represents a problem for which some of the parameters, or input, are open. However, representing the dimensions as three binary positions, as in Table 2, simplifies the model and makes it easier to examine different problem classes. Thus, the system introduced by Sutinen and Tarhio is used in this thesis.

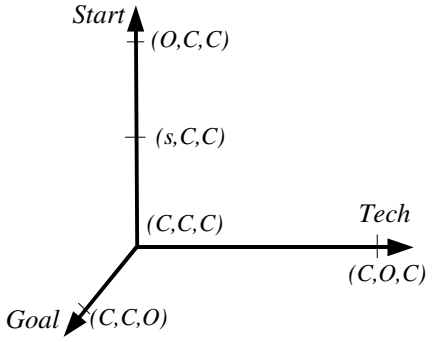


Figure 10: Dimensions of Problems as Continua

It is interesting to note that because scientists are active in seeking problems, closed problems are favored over open problems. According to Popper, "*it is we, who always formulate the questions to be put to nature: it is we who try again and again to put these questions so as to elicit a clear-cut 'yes' or 'no' (for nature does not give an answer unless pressed for it)*"[92]. If the initial conditions of a problem cannot be ascertained, and the experimental results seem to defy the attempts to formalize predictions into suitable laws, a scientist often gives up trying if the problem does not in-

92 Popper, 1959:p.280. (The term pseudo-problem still refers to artificial, imposed problems and not to Popper's use of the term.)

terest him or her much, Popper claimed[93]. In other words, Popper's statement could be interpreted as: If a scientist is not able to express an issue (i.e., the problem, the methodology, and the results) in a closed form, he or she will probably ignore the issue, and the gathered data can be used in, perhaps, estimating probabilities, but not in building a scientific base. In the following pages, the <S,T,G> classification scheme for problems is analyzed further.

Open and Closed Problems in Different Problem Fields

In mathematics, the concept of a problem seems quite straightforward. According to Pólya, a problem consists of a number of elements such as "the unknown(s)", "the data", and "the condition(s)"[94]. Some questions that Pólya used to circumscribe a problem are, for example, "Can the condition(s) be satisfied?", "Is the condition sufficient to determine the unknown?", "Is there redundancy?", and "Are there contradiction(s)?". Furthermore, there are "problems to find" and "problems to prove".

In Pólya's "problems to prove", the researcher is given a clearly stated assertion, and he or she has to answer the question, "Is this assertion true or false?". The researcher has to answer this question *conclusively*, either by proving the assertion true, or by proving it false.[95] In Sutinen and Tarhio's problem classes the "problems to prove" are of type "C-O-C"[96]. In problems to prove, the premises are clearly stated (*start*=C), the answer is definite yes or no (*goal*=C), but the researcher does not yet know how to come to the answer (*tech*=O). In Pólya's "problems to find" the aim is to find the unknown of the problem, also called "*quaesitum*"[97]. Not restricted to mathematical problems only, objects in this class of "problems to find" may be theoretical or practical, abstract or concrete, serious problems or mere puzzles, Pólya concluded.

Problem management in computer science at large (if it is defined as an inseparable synthesis of theory, design and modeling (abstraction)[98]) can concern problems to find, or problems to prove. In computing there are areas that require deductive, mathematical inference, but there are also areas that require an approach like Picker-

93 Popper, 1959:p.198.
94 For these issues see Pólya, 1957:pp.*xxxvi*, 2-3, 154-156.
95 Pólya, 1957:p.154.
96 Closed starting point, Open technique, and Closed (presupposed) results.
97 Pólya, 1957:p.154.
98 Denning et al., 1989; this is an oft-quoted definition of computer science as a discipline.

ing's "mangle"—the mangle is the process by which a scientist is ready to refine his or her theories, instruments, and theory about the instruments, accommodating to the problems along the way. From Pickering's mangle-viewpoint, the resistance of the world, that is, the problems of finding a robust fit between the theory, the instrument, and the scientist's theory about the instrument, might be considered a *problem*, and accommodation to the resistance might be considered a *solution*.

As I see it, the class (O-O-O) includes *general problems*, such as the above-mentioned "what are the rules for an ideal society?", and "the problem of understanding the world—including ourselves, and our knowledge, as part of the world". Note that whereas class (C-C-C) is considered to contain a maximum number of *controllable* variables, preferably quantitative, and a minimum number of *uncontrollable variables*[99], class (O-O-O) can be rich in both, and they can be either qualitative or quantitative.

In the first *general problem* mentioned—"What are the rules for an ideal society?"—the starting points, if taken as "the unknowns", are uncountable (think of the diversity of humankind, for example) and the goal is not known—indeed, it is not even known whether the goal exists. There are some suggested methodologies, for instance, John Rawls' (1921-2002) theory of the "veil of ignorance"[100], which emphasizes primarily negative freedom, and Karl Marx's theory, which emphasizes primarily positive freedom[101]. But because there is no general agreement of whether either of these theories—or any other theory for that matter—is "correct", and also because the competing methodologies are largely incommensurable, the methodology for this problem is open (O). In the second *general problem* that was mentioned earlier—the problem of understanding the world—the starting points are not known except for one (the oft-quoted *cogito, ergo sum*[102]), and surely, neither the goal nor the methodology are preconceived.

---

99 See, e.g., Ackoff, 1978:p.11. However, note that Ackoff stated that the *color* (of a car) is a qualitative variable, but color clearly has both qualitative and quantitative components. The color spectrum that an object reflects can be measured to a high degree (*quantitative component*–a "brute fact"), but the preferences or associations colors have are highly personal and cultural (*qualitative component*–an "institutional fact" and often epistemologically subjective fact).

100 Rawls' *veil of ignorance* is based on the idea that a person should create a social contract without knowing his or her capabilities (intelligence, vigor, values, etc.), position in the society, gender, race, or family roots. This should, according to Rawls, lead to the best possible societal contract. See Rawls, 1971:pp.136-142.

101 See footnote 273 on page 98.

102 Following René Descartes (1596-1650), the only thing one can know for certain is that his or her mind exists (if one accepts the premise that something nonexistent cannot think.

Problems in Computer Science

There are a number of definitions of computer science, but in this connection the term *theoretical computer science* refers to the "theory" part of computing as a discipline (see Denning et al.'s definition[103]), and the term *computer science* refers to the combination of theory, design, and modeling (i.e., the discipline of computing at large). Although computer science at large and theoretical computer science are not parallel concepts, they are contrasted here for a number of reasons. In the early days of computing there was a long and heated debate around the question whether computing is a mathematical discipline or if it also includes other aspects such as engineering. The common antonym pair *theoretical* and *practical* would not be a historically meaningful juxtaposition (the debate of the identity of computing as a discipline is discussed later in this thesis). In addition, theoretical computer science is an established name for one branch of the discipline of computer science, and for instance, theories of design do not belong to theoretical computer science, but they are not practical issues either. Hence the term *computer science* refers to all of computer science, and *theoretical* refers to one subpart of computer science at large.

In computer science (at large), as in any other experimental science, the initial premises may need refining, or mangling[104], during the problem management process. For example, the initial selections of a programming language or a platform may, during the process, turn out to be unsuitable for the task at hand. There rarely exists a consensus about the goal between different stakeholders. Also, requirement specifications tend to get refined throughout the process (*start*=O). In contrast, in theoretical computer science the premises (such as axioms and rules of inference) are usually given and remain in force.

In computer science if a structural design scheme was chosen initially, it may at some point turn out to be inappropriate, and the technique might be changed to, say, object-oriented design (*tech*=O). The whole process is indeed characterized by *ad hoc*ness[105]. In theoretical computer science the choices are usually similar to those in mathematics: induction, deduction, direct proof, proof by contradiction, construction, exhaustion, plus a number of other mathematical methods of proof.

103 Denning et al., 1989

104 Pickering, 1995

105 A good example of this is seen in John Tracy Kidder's "The Soul of a New Machine" (Kidder, 1981). Kidder's ethnographic study describes the whole social process of creating a new brand of a computer.

In computer science (at large), even the goals of the problem management process may change during the process. Software engineers, especially, know that what is expected of the end product tends to change throughout the development process (*goal*=O). In theoretical computer science the aim remains mainly simple and straightforward: proving an assertion true, false, or unsolvable[106].

Computer science has a large amount of problem-solving-related literature. In general, this literature seems to relate to closed problems and not to real-world problems. For example, Maureen Sprankle[107] introduced six steps in problems solving:

1) Identify the problem,

2) Understand the problem,

3) Identify alternative ways to solve the problem,

4) Select the best way to solve the problem from the list of alternative solutions,

5) List instructions that enable one to solve the problems using the selected solution, and

6) Evaluate the solution.

Sprankle's steps presuppose that there exists a *single* "best way" to solve any problem and a "best result". Sprankle wrote, "*In this book, the term* ***solution*** *means the instructions listed during step 5 of problem solving—the instructions that must be followed to produce the best results*"[108].

In programming, which is already a confined sector of computer science, what is best from one viewpoint may not be that from another. Programmers have to frequently make choices between, say, memory usage *vs.* speed, portability *vs.* efficiency, or elegance *vs.* simplicity (or elegance or simplicity *vs.* readability or ergonomics). However, some books such as Lerman's textbook[109], acknowledge the subjectivity or contextuality of "good" and "bad" when it comes to algorithms.

106 It would not be reasonable in this thesis to restrict computer science at large to "proving something to be true" or "falsifying something" even though some authorities, such as David Gries and Edsger Dijkstra, might like to see computer science that way.

107 Sprankle, 1998:pp.3-4.

108 Sprankle, 1998:p.5.

109 See Lerman, 1993:p.2.

Lerman's textbook declares explicitly that it covers only "engineering, science, management and planning", and thus delimits the parameters, premises or input of the problems to "*clear, unambiguous statements*"[110]. I argue that clear, unambiguous statements rarely exist outside the field of theoretical computer science[111]. My viewpoint is supported by some textbook authors. For example, a book by Zbigniew Michalewicz and David Fogel explicitly states that real-world situations rarely present the circumstances required by algorithmic problem-solving[112].

Michalewicz and Fogel used an example of a particular technique that is available to calculate the minimum-cost allocation of resources, and suggest that this method is almost always applied inappropriately in real-world settings because the technique is for linear functions, and in real-world settings the cost function and constraints are almost always nonlinear.

Computer-Scientist-as-Bricoleur

From Michalewicz and Fogel's point of view, problem solving has to take theory into account, but it also needs *(a)* a variety of non-specialized tools for a wide variety of purposes, *(b)* an understanding of institutional or organizational constraints, as well as *(c)* a grasp of other players in the field—which consequentially introduces personal and cultural differences. They wrote,

> *Effective problem solving requires more than a knowledge of algorithms; it requires a devotion to determining the best combination of approaches that addresses the purpose to be achieved within the available time and in light of the expected reactions of others who are playing the same game. Recognizing the complexity of real-world problems is prerequisite to their effective solution.*[113]

Taking a look at research in three distinct areas of computing—computer science, software engineering, and information systems—paints a picture of computer science as being inbred. Glass et al.'s study[114] revealed that of these three areas, computer science was 89.3% self-referential. However, software engineering was even

110 Lerman, 1993:pp.1-2.

111 cf. Paul Feyerabend: "*Popper's criteria are clear, unambiguous, precisely formulated* [...] *This would be an advantage if science itself were clear, unambiguous, and precisely formulated. Fortunately, it is not.*" (Feyerabend, 1975).

112 Michalewicz and Fogel, 2002:see pp.1-7.

113 Michalewicz and Fogel, 2002:p.3.

114 Glass et al., 2004

more so: Only 1.9% of the references in the references section of software engineering articles were to literature outside the field of software engineering itself (98.1% self-referential). Information systems (IS) was the least self-referential. It was 27.2% self-referential, and the largest external reference disciplines were management (18%), "other" (12.5%), cognitive psychology (10.7%), economics (11.1%), and social and behavioral sciences (9.0%). Even though Glass et al.'s study does not prove that IS would be any more applicable to real-world problems than the other two, one might expect it to be[115].

Anthropologist Claude Lévi-Strauss introduced the term *bricoleur* to describe the opposite of an engineer. An engineer creates and uses specialized tools for specialized purposes, whereas the *researcher-as-bricoleur* is knowledgeable with and works between and within competing and overlapping paradigms[116]. Computer scientists (especially outside the USA) are usually *not* trained in subjects other than computer science, mathematics, and perhaps the natural sciences[117]. The failure of recognizing the complexity of real-world problems may have its roots in the narrow education of computer scientists.

Mordechai Ben-Ari has argued that "*the manifestation of bricolage in computer science is endless debugging: Try it and see what happens*"[118]. This is a way of action that Edsger Dijkstra opposed fiercely throughout his career, and I do not consider Ben-Ari's bricolage to be a bricolage in the positive sense of the word, as described by Denzin and Lincoln[119]. Attributes connected with a researcher-as-bricoleur are "flexible and responsive", "technically curious and multi-competent", and "intellectually informed"—not uncoordinated, obstinate, or artless. However, Ben-Ari did not claim that bricolage would be a recognized methodology in computing, but only a thinking tool. Neither did Ben-Ari see bricolage as a normative account for computer scientists. Ben-Ari argued that students who excel at bricolage often cannot make the transition to master the thought patterns and methods that are required in abstract techniques. An abstract planning style is needed in, for instance, software

115 It is interesting to note that of these three, IS is generally funded the least.

116 See, e.g. Denzin and Lincoln, 1994:pp.2-3; Lévi-Strauss, 1966:p.17. Contrary to Denzin and Lincoln, Lévi-Strauss used the term in a somewhat negative meaning (Lévi-Strauss, 1966:pp.16-17).

117 Denning et al., 2001:chapter 9.4.

118 Ben-Ari, 2001

119 Denzin and Lincoln, 1994:pp.2-3.

engineering[120]. Nonetheless, the positive attributes that Denzin and Lincoln associated with researcher-as-bricoleur (mentioned above) can support computer scientists in broadening their competence towards the Michalewicz and Fogel's attributes (*a*)-(*c*) that begun this subsection.

Should Computer Science Borrow the Definition of *Problem* from Mathematics?

In mathematics, according to George Pólya, the three fundamental questions to be asked are explicit ("What is the unknown", "What are the data" and "What is the condition"[121]). Yet, the concept of problem in mathematics is not that straightforward. For example, the interests, persistence, acuity, and personal goals of a single researcher affect the problem. Ian Stewart wrote,

> *At research level you invent the problem yourself.* [...] *You therefore spend a great deal of time developing a feel for the problem, trying to decide what the essential ideas and concepts should be and how everything fits together.*[122]

Although the phrase "*you invent the problem yourself*" may sound non-mathematical, it has to be kept in mind that mathematics *is* human-made. Albeit being human-made, it is hard to think of a science that has not gained benefit from the field of mathematics.

Though mathematical research problems may begin with a fixed problem, the techniques for solving them may vary; to avoid getting stuck examining a problem from only one point of view, mathematicians may also *set new questions concerning a problem. Reductio ad absurdum* and indirect proof are examples of common alternative techniques[123]. Pólya wrote that new questions and new viewpoints may cause new possibilities of solving the original problem to unfold[124]. Pólya's alternative techniques approach clearly is a change in methodology that reframes mathematics as including "C-O-C" and "O-O-C" classes of problems. Mathematics presented

120 Ben-Ari, 2001

121 Pólya, 1957:pp.6-7.

122 Ian Stewart: Foreword in Pólya, 1957:p.*xxi*., underlining added. To the extent that this argument is accurate, it would render at least some of the goals of mathematics open (O).

123 See., e.g., Pólya, 1957:pp.160-171.

124 Pólya, 1957:pp.210-211.

with rigor is a systematic deductive science, but "mathematics in making" is an experimental inductive science, Pólya claimed[125].

Insistence on rigidly defined unknown(s), data, and condition(s) would not apply well to practical[126] (pragmatic, experimental) sciences such as computer science. Take the process for designing a calendar program, for example. In this case the question "*What is the unknown?*" is impracticable. The number of unknowns practically defies enumeration, and one can always add an extra variable (superfluous variables in the Duhem-Quine thesis). Those unknowns include the possible platforms, the possible features of the user interface, the possible variety of user skills, and, especially, the possible variety of user expectations and needs.

In the same calendar program the question "*What are the data?*" is as inappropriate as "*What is the unknown?*". The multiplicity of data that would affect design decisions is overwhelming: the possible portability and platform compatibility issues, the possible internationalization and localization parameters, the possible protocol variations, and, especially, when devising the metaphors and analogies for the program—the wide variety of cultural backgrounds of the users.

To use the same calendar program example with the third question: The same diversity of real-world goals makes it virtually impossible to answer the question "*What are the conditions?*" extensively. The possible economic restraints, the possible deadlines, the possible management issues, the possible inter-personal tensions, the unpredictability of people—the list of conditions is endless.

Norman Matloff claimed that the often-praised CMM (Capability Maturity Models) that are used to evaluate companies by the maturity of their *production processes*, do not actually tell the whole truth about a company[127]. Matloff argued that CMM merely assesses a company's project management techniques, not the quality of its personnel. As one official in the CMM project at Carnegie Mellon University[128] noted (according to Matloff): "*You can be an* [highest CMM-rated] *organization that produces software that might be garbage"*.[129]

125 Pólya, 1957:p.117.
126 Pólya, 1957:p.149.
127 Matloff, 2004
128 Carnegie Mellon University's Software Engineering Institute is *de facto* authority in CMM.
129 Matloff, 2004

What Is a Problem in Computing: Summary

In the beginning of this section, I asked if there can be more than one legitimate definition for the term *problem* at a time. I proposed that there are a variety of equally useful definitions of problem, and I sketched an extended characterization of the concept of *problem*.

First, a problem consists of an obstacle in someone's own existence (*objective aspect*), and of a feeling of necessity to overcome the obstacle (*subjective aspect*). An important aspect of the definition is that if a person does not feel the necessity to solve a problem, it cannot be considered a problem for that person. If a person does not feel the necessity to solve the traveling salesman's problem, it is not a problem for that person.

Second, problems can be divided to *general*, *intimate*, and *limitary*. General problems concern most people (though not necessarily most of the time). Intimate problems are problems for single individuals; they are subjective and contextual. Limitary problems concern confined groups of people. Problems for some may be advantages for others. Based on these statements, I further argued that the understanding of how *problem* is defined is a socially constructed, culturally mediated and individually interpreted concept. A relativist stand was taken regarding the concept of problem. There does not exist criteria by which some problems would be better or worse than others, although some problems can be more common than others.

Third, there are different sorts of problems, for example (but not limited to these), *authentic*, *pseudo-*, and *auxiliary* problems. Authentic (real-world) problems correspond to general, intimate, or limitary problems, and they include both an objective and a subjective aspect; for instance, the feeling of necessity to overcome the problem. Pseudo-problems include an objective aspect, but the subjective aspect is weak or artificial. Auxiliary problems are not solved for their own sake but as a means to solve another problem.

Fourth, eight *problem classes* are defined as a triplet <S, T, G>, where S refers to the starting point, parameters, or input of the problem; T refers to the technique, method, or algorithm to solve the problem; and G refers to the result or output of the problem. Each parameter of <S, T, G> is either closed (C) or open (O), and the corresponding classes are marked as C-C-C, C-C-O, ... , O-O-O.

3.3.The Creation of Modern Computing

New Machine Solves Problems Faster than
Mathematicians Can Make Them[130]

The forces and motivations behind the development and diffusion of computing technology are not certain. It is not certain if the development and diffusion of computing technology have benefited from certain sociocultural settings; what interrelations between the technological, institutional, professional, and social aspects of computing there have been (if any); or if the history of computing has been contingent or if it has followed a certain path. This section is an exploration of the sociocultural aspects that have affected early computing technology and related areas. In the end of this section, I offer my interpretations about the uncertainties above. If the forms and directions that computing technology takes are influenced by sociocultural phenomena, then the research on the sociocultural aspects of computing development sheds light on technology itself. If the forms and functions of technology are shaped by extra-technological and extra-scientific (sociocultural) forces, then one cannot understand technologies only by looking at the current state of technology; one also has to understand the sociocultural environment and the history of technological development. The discussion in this section builds my case for the importance of social studies of computer science.

The topics covered in this section are not divided into subsections chronologically. They are not divided by, for instance, decade, because decade per se has no significance[131]. Instead, subsections are divided into *themes*. The theme of the first subsection is the shift from electromechanical computation to electronic computation, the theme of the second subsection is the early development of electronic computation, the theme of the third subsection is early academic computing, and the theme of the fourth subsection is the birth of programming languages. Because the themes are interrelated, there is some chronological and thematic overlap between the subsections.

The history of electronic computing is quite short. In this section the term *early computing technology* refers to a period of thirty years: From the crossing of the Newton-Maxwell gap in the 1940s to the advent of computing as a discipline in the

130 New York Herald Tribune, January 28, 1948, p.7, as quoted in Martin, 1993.
131 cf. Sammet, 1991

1960s. The crossing of the Newton-Maxwell gap; that is, the shift from electromechanical devices to fully electronic devices; took place during the 1940s, and the first *fully electronic*, *digital*, *Turing-complete* computer, ENIAC, became operational in November, 1945 and was inaugurated in February, 1946[132]. After the completion of ENIAC there have been a number of major breakthroughs in computing in Western societies. Take, for instance, breakthroughs such as the advent of programming languages in the 1950s; the development of time-sharing in the 1960s; the emergence of personal computing in the 1970s; the innovation of the graphical user interface in the 1980s; the diffusion of the world wide web in the 1990s; and a diversification of computing technologies, so that they extend to nearly all aspects of the urban Western middle- and upper-class life, in the 2000s. The themes in this section span over the first thirty years after the construction of ENIAC, from the 1940s to the 1960s.

The constraints in the development of technology are not always just technical ones[133]. All technologies have been argued to embody the physical, intellectual, and symbolic resources of the society that constructs them[134]. It has been argued that the constricting and enabling factors of the development of early electronic computing include, for instance, institutional and organizational factors that support a certain practice or tradition in computer designs[135], inter-personal relationships[136], visionaries and strong leaders[137], and a diversity of cultural aspects[138]. For instance, it has been argued that much of the devotion invested in the development of high-speed electronic computing was due to the Second World War[139]. Note that the phrase *technologies embody resources of the society* should not be understood as a position that technologies would, after their construction, hold the values of their constructors. Technologies are unintentional things. But values, among other things, may af-

132 Campbell-Kelly & Aspray, 2004:pp.85-86. Note that there had been earlier computers that fulfilled *some* of the criteria that ENIAC met: The ABC computer and Colossus were *digital* and *fully electronic*, but they were not Turing-complete (Williams, 1985:pp.270-271;294-296); Konrad Zuse's Z3 was *digital* and *Turing-complete*, but it was not fully electronic because it used relays (Williams, 1985:pp.220-221); Harvard Mark I was *digital* but it was electromechanical, not fully electronic (Williams, 1985:pp.243-244).

133 Marcus and Akera, 1996

134 Hughes, 1983:p.2.

135 Marcus and Akera, 1996. (This is also supported by the Kuhnian view of science; see Kuhn, 1996).

136 Williams, 1985:p.303.; Aspray, 2000.

137 Aspray, 2000; also Tracy Kidder's book *The Soul of a New Machine* implies the importance of leadership figures (Kidder, 1981).

138 Flamm, 1988:p.136, Bowles, 1996.

139 Flamm, 1988:p.48; Campbell-Kelly & Aspray, 2004:p.73.

fect technological development so that the resulting technologies may be better suited for some things than others.

Although computing technology was, from early on, developed in a number of countries[140], most of the major technological breakthroughs in early electronic computing technology were made in the U.S.[141] The themes in the following subsections are focused on the development of computing in the U.S., and the sources are chosen accordingly.

There is an abundance of research about the history of computing in the U.S. One of the premier journals in the history of computing is the (U.S.-based) *IEEE Annals of the History of Computing* (hereafter *The Annals*). The Annals is highly esteemed; for instance, major textbooks in the history of computing regularly cite the Annals[142]. It has been noted, though, that one of the drawbacks of the Annals is that it has a strong orientation towards the English-speaking world[143]. The sources in this section include a number of books by recognized historians of computing such as Jean Sammet, Michael R. Williams, Martin Campbell-Kelly, and William Aspray, and a large number of articles from the Annals. In this chapter the history books and journal articles are used as sources that offer historians' interpretations of the history of computing.

In this section I analyze the source material in terms of the framework developed in Chapter Two. For instance, I investigate whether these selected historical studies of computing indicate that the development of computing technology shows characteristics of *contingency*. I also investigate whether these historical studies support the view that the concepts, theories, and technologies of computing are a product of a *mangle* of testing and revamping different aspects of computing. What is more, I investigate if these historical studies back up the idea of *technological momentum*—that in the beginning of the development of a system, the system shows characteristics of social construction, and that as the system matures it tends to show increasing characteristics of technological determinism. I use my interpretations in this section

140 See, for instance, examples of computing in the Soviet Union in Trogemann et al., 2001; in Great Britain in Croarken, 1992 and Ferry, 2003; in Germany in Rojas, 1997; in Sweden in Petersson, 2005; and in Japan in Flamm, 1988:pp.172-202.

141 See Pugh and Aspray, 1996, for a large number of early computing innovations in the U.S.

142 See, e.g., Flamm, 1988:pp.31, 32, 38, 40, 42, etc.; Ferry, 2003:p.201ff.; Williams, 1985:pp.193, 194, 262, 264, 407, etc.; Campbell-Kelly & Aspray, 2004:p.301ff.

143 Lee, 1996

to support my argument about the nature of computer science and to my rationalization of social studies of computer science in Chapter Four.

The reader is advised to take into consideration the fact that the history of computing presented in this section is doubly interpreted. Firstly, the historians of computing who have written the texts selected in this chapter offer interpretations of their source material. Secondly, I have selected those parts of the historians' texts that are relevant to my research questions, and analyzed those texts in terms of the framework developed in Chapter Two.

The Newton-Maxwell Gap: Before 1950

30-ton Electronic Brain at University of Pennsylvania
Thinks Faster than Einstein[144]

IN THIS SECTION:

- ✔ Who were the pioneers of electronic computing?
- ✔ Which disciplines were central to the birth of electronic computing?
- ✔ Was the development of early electronic computing culturally neutral or was it influenced by any cultural aspects?
- ✔ What was the role of World War II in the development of early electronic computing?

Technology is not developed in a sociocultural vacuum. For instance, public attitudes, politics, and social structures are claimed to influence technological development[145]. It is quite clear that the diffusion of certain technologies can be sped up by political decisions. For instance, the government of Finland has decided to end analog TV broadcasts by the fall of 2007. This decision promotes (even forces) the diffusion of digital television technology. It is also clear that the development of certain technologies can be a political decision: The mission to the Moon is a case in point[146]. Also, culture matters in technological development. Sociologist Manuel Castells has connected four cultures with the development of the Internet—the techno-meritocratic culture, the hacker culture, the virtual communitarian culture, and the entrepreneurial culture[147]. The creator of the Linux kernel, Linus Torvalds, noted that Linux was born out of a need for socialization and entertainment, not from an economic need[148]. Wiebe Bijker and John Law argued that technologies are born out of conflict, difference, and resistance, and out of controversies, disagreements, and difficulties[149].

This section deals with the early history of computing starting from the shift from analog computers to digital computers, during the Second World War. The aspects that are discussed include, for instance, the events that brought together the people who built the first digital computer, the cultural atmosphere that surrounded techno-

144 *Philadelphia Evening Bulletin*, February 15th, 1946, as quoted in Martin, 1993.
145 Flamm, 1988; Winner, 1999; Bowles, 1996, respectively.
146 See, for instance, Sadeh, 2002.
147 Castells, 2001:p.37.
148 See Linus Torvalds' foreword in Himanen, 2001:p.15.
149 Bijker and Law, 1992:p.9. Bijker and Law are recognized figures in the field of social studies of technology.

logy at the time, the influence of governmental institutions, the Second World War, and the contingencies that led to building the first business computer.

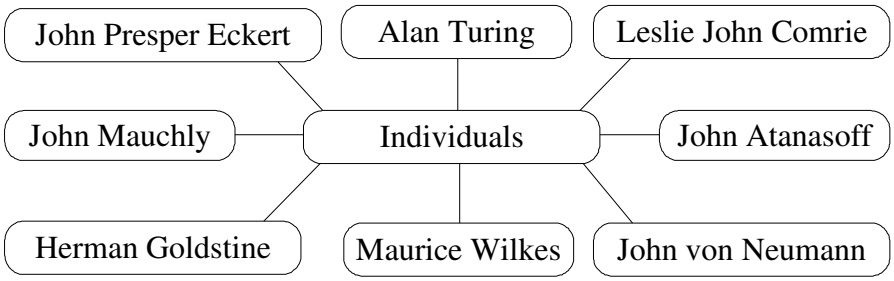


Figure 11: Some Prominent People in the Development of Early Computers

To help readers recognize the important names in the text, a number of people who had a strong influence in the development of early computers are presented in Figure 11. I recognize that presenting this kind of a list of people is a source for endless disputation, so I wish to make clear that I do not claim that these eight people would be the eight most important people at the time. Nonetheless, all these people did play an important role in the history of computers at early 1900s, and their names appear frequently in the history of computing.

Analog Computing

One of the problems that has vexed applied mathematicians over the years has been that of calculating the area under a given curve, that is, the integral of a given function, or $\int f(x)\,dx$. The integral is easy to calculate for elementary functions that can be integrated analytically, but almost impossible if the function *f(x)* is either not known in analytic form or is not well-behaving[150]. Beginning from the early 1820s, many attempts to build devices for measuring the integral were made[151], but the first *differential analyzers* that actually worked were constructed by Vannevar Bush at MIT (Massachusetts Institute of Technology), USA, around 1927.[152]

In England, Douglas Hartree (1897-1958), J.B. Bratt, and John Lennard-Jones' (1894-1954) *Manchester Differential Analyzer* was built in 1935 after Hartree, Bratt, and Lennard-Jones had studied Vannevar Bush's differential analyzer[153]. However, neither the British scientific community nor the British Press showed much interest

150 Williams, 1985:pp.206-212; Bowles, 1996; Polachek, 1997.
151 Campbell-Kelly & Aspray, 2004:p.45.
152 Williams, 1985; however, Polachek, 1997, claimed the exact year of the production-scale machine to be 1931.
153 Croarken, 1992

in Hartree's machines[154] for the reasons discussed later in this chapter. In the U.S., the final version of Bush's machine, known as the *Rockefeller Differential Analyzer #2*, was used extensively during the Second World War by the armed services of both America and Britain for the calculation of ballistic firing tables[155]. However, the reasons for building the machine were not only scientific or practical reasons.

Susann Puchta, who is a historian of mathematics, noted that the role of interdisciplinarity in the development of computing technology has been given too little attention[156]. Susann Puchta, Michael R. Williams (historian of computing), and Mark Bowles (historian of technology and science) listed a number of sectors that needed Vannevar Bush's computing machinery: for instance, electrical engineering, mathematics, sciences (especially physics), radio technology, and warfare[157]. Puchta argued that breaking the disciplinary boundaries was not only necessary to achieve the goals that the device was built for, but that the interdisciplinary development work enabled more efficiency and creativeness within the traditional disciplinary boundaries, too.[158]

Bush's end product combined knowledge from fields that had not had much interplay before: for example, pure or formal mathematics, mechanical engineering, innovations from seafaring, and logic (Boolean algebra)[159]. In addition, a number of other fields such as astronomy, navigation, and meteorology had an impact on the development of early computers[160]. Of the people in Figure 11, Turing was a logician and mathematician, Leslie Comrie (1893-1950) was from the field of astronomy, John Atanasoff (1903-1995) was from the fields of electrical engineering and mathematics, John von Neumann[161] worked on a number of mathematical fields, Wilkes and Mauchly were from physics, Lt. Herman Goldstine[162] (1913-2004) was from math-

154 Bowles, 1996

155 Williams, 1985:pp.206-212.

156 Puchta, 1996

157 Puchta, 1996; Bowles, 1996; Williams, 1985:p.209.

158 An interplay of different cultures, domains, and disciplines that allows established concepts to clash and combine, ultimately forming a multitude of new, groundbreaking ideas, has recently been named the *Medici Effect* (Johansson, 2004:p.2).

159 Williams, 1985:p.209; Puchta, 1996; Bowles, 1996.

160 Naur, 1992:pp.596-597; Campbell-Kelly & Aspray, 2004:pp.52-59.

161 John von Neumann was a student of renowned mathematicians such as David Hilbert and George Pólya. In *How to Solve It*, Pólya tells that he had shown his class an unproven theorem and told the class that it may be difficult. After five minutes, von Neumann had written a proof of the theorem on the blackboard. Pólya wrote, "After that I was afraid of von Neumann." (Pólya, 1957:p.xv).

ematics, and John Presper Eckert[163] was a student at the Moore School of Electrical Engineering. Figure 12 shows the main branches of science that had an impact on the development of early computers.

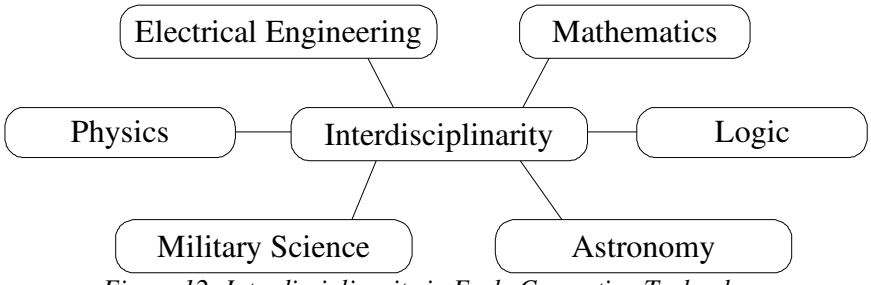


Figure 12: Interdisciplinarity in Early Computing Technology

The development of early computing technology was a point of intersection of a number of different ideas. My interpretation is that what is essentially fruitful in the clash and combination of different cultures, domains, disciplines, experts, and concepts, is the anarchism of that situation. That is, when sufficiently many cultures, domains, disciplines, experts, and concepts are put together, the demands for instant clarity or reducibility have to be dropped. In practice, every person cannot be trained to be knowledgeable about every other person's art or science. An eclectic combination of incommensurable arts and sciences creates an ontological, epistemological, and methodological anarchy in the sense that no ontology, epistemology, or methodology can be claimed superior over others[164].

Insofar as this is true, epistemological and methodological anarchy inevitably inhibits dogmatism (a characteristic of monodisciplinary coteries) because dogmatic views are based on faith about the superiority of a certain belief system. When the people in an anarchical interplay of cultures, domains, disciplines, and concepts, have a common mission, they need to communicate some aspects of their art or science to the others—which should lead to the people explaining their art or science in the clearest and simplest way possible. That is, they need to present their art or sci-

162 Lieutenant Herman H. Goldstine, a mathematics PhD from University of Chicago, was an officer at the Ballistics Research Laboratory, assigned to the Moore school of electrical engineering at the University of Pennsylvania (Campbell-Kelly & Aspray, 2004:p.76).

163 Eckert was characterized as "The brightest graduate student around at the time", "undoubtedly the best electronic engineer in The Moore School" (Winegrad, 1996). Not to be confused with Wallace John Eckert from Columbia University.

164 When Pickering wrote about eclectic multidisciplinarity, he called this phenomenon the *balance problem* (Pickering, 1995:p.215). I do not share Pickering's view that it is a *problem* that there are no rules for specifying which of the multiple set of epistemic and social factors is dominant in a multidisciplinary situation.

ence without the theoretical or metatheoretical issues, the counterarguments, the alternatives, or the disciplinary controversies between different coteries or cliques.

Superficial knowledge about powerful ideas enables a person to utilize concepts or innovations without getting mired in field-specific debates. In addition, ignorance about traditional boundaries of an art or science may enable unexpected crossings of those boundaries (especially boundaries of applicability). Yet, superficial knowledge about powerful ideas can also be counter-productive. If researchers utilize a powerful idea without knowledge about its theoretical and metatheoretical issues, counterarguments, alternatives, or disciplinary controversies, they may get mired in black spots that an expert would avoid.

Cultural Context

Both Puchta and Bowles emphasized that the context in which Vannevar Bush's analyzer was developed provides information about the intellectual, interdisciplinary, and social dimensions, as well as sources, of modern computing[165]. Whereas Puchta focused on definite intellectual and disciplinary contexts, Bowles argued that the techno-utopistic, technologically enthusiastic, and practical atmosphere of the U.S. enabled American scientists to obtain the funding needed for development. Historians of computing Martin Campbell-Kelly and William Aspray called this "America's love affair with office machinery"[166]. Kenneth Flamm[167] even claimed that instead of a lack of funding or technological knowledge, tradition-bound culture was *the* crucial factor that made British scientists unable to compete with their American colleagues in building the differential analyzer.

Bowles, too, stated that the differences between U.S. and British cultural contexts were the main reason for the difference between the U.S. and British results. His findings suggest that there were four contextual factors that were the main sources of differences in the development of computing machinery between the U.S. and Britain[168]. Some of the aspects that Bowles dealt with are supported by the texts of Mary Croarken, a historian of computing and expert on British scientific computing[169]. Bowles' four contextual factors are discussed below.

165 Puchta, 1996; Bowles, 1996
166 Campbell-Kelly & Aspray, 2004:p.19.
167 Flamm, 1988:p.136.
168 Bowles, 1996
169 Croarken, 1992; Croarken, 1993

First, Bowles argued that the practical, professional values of American scientists were reinforced by popular and professional culture. He continued that the British scientific community, in contrast, resisted and ignored practical research because they believed that a practical engineer's machine would be incapable of solving theoretical problems for scientists. An example of this different emphasis can be seen in Vannevar Bush's exclamation, "I'm no scientist, I'm an engineer"[170]. The British valued theory over practice, the Americans valued practice over theory.

Second, because the British university scholarship sponsors valued theory and downplayed practice, the best students went to universities to study theoretical subjects, and technical schools got mediocre students who were not taught advanced topics such as differential equations[171]. In the U.S. the predominant professional style, according to Bowles, was "practical enthusiasm". In the U.S., the traditional universities had their technical schools, such as Pennsylvania's Moore School of Electrical Engineering and MIT's laboratories. Aspray argued that the departmental environment in American universities was critical to the success of computing, especially with respect to whether the discipline of computing was able to grow[172]. In addition, individual people had an influence on the success of computing—for instance, Puchta noted that Vannevar Bush's role as a mediator between the subcultures of science, mathematics, and engineering had an influence on MIT science policy[173].

Third, Bowles claimed that the public status of engineers in the U.S. was in complete contrast to the status of engineers in Britain. Whereas the American engineer was the hero of the new century, the British engineer was a second-class citizen[174]. While American engineers continually had to fight for status within the scientific community, they were beneficiaries of a strong *public* appreciation for their expert skills and financial support for their services[175]. Aspray noted that engineering universities in the U.S. got financial support both from the government and industry—and that external support was critical to becoming and remaining strong in the computing field[176].

---

170 Kevles, 1987:p.293.
171 Interview with Arthur Porter, Douglas Hartree's graduate student, Feb. 20, 1995, in Bowles, 1996.
172 Aspray, 2000
173 Puchta, 1996
174 Bowles, 1996; Bush, Vannevar (1970) *Pieces of the Action*: Morrow Press (as quoted in Bowles, 1996).
175 Bowles, 1996
176 Aspray, 2000

Fourth, Bowles argued that the American press played an important part by reinforcing an anthropomorphic, laudatory image of computational instruments (the American press used terms such as "the robot Einstein"[177], "the super-brain", and "man-made mental giant"[178]). The American press should not be blamed, though. The designers of early electronic machines were the ones using anthropomorphic terminology in the first place[179]. In contrast to the American press, the British press considered machinery important mainly because it reduced "donkey work". The British and U.S. public were given very different cultural representations of computing machinery.

It seems that cultural (media) representations are highly important in the development of any technology. Communications theorist and philosopher Marshall McLuhan wrote that new technologies extend the physical and mental properties of people, and that any extension affects the whole psychic and social complex[180]. If computing machines are presented as what Lewis Mumford (1895-1990) would call "extensions of [a person's] own organism"[181], the emergence of new machinery changes those extensions, and forces people to redefine their relationship to, for example, technoscience, nature, institutions, and each other. In fact, the more technologies are allowed to de-categorize[182] and recategorize people, the harder it gets to understand and perpetually redefine one's place in the network of relationships between, for instance, technoscience, nature, individuals, and institutions.

The culmination of de-categorization would be a point where traditional communities, societies, classes, strata, tribes, and such would be gone. In this kind of a culmination of de-categorization, people would constantly redefine their (molecular[183] or atomic) relationships with respect to everyone and everything they come in contact with. Currently people use molar aggregates (such as communities, societies, and strata) as pre-defined categories to guide their actions and define their place in a community. Elements of ongoing de- and re-categorization are visible in, for in-

177 Bowles, 1996
178 Martin, 1993
179 Grier, 1996
180 McLuhan, 1975:pp.3-4.
181 Mumford, 1962:p.321.
182 In the terminology of philosopher and anthropologist Pierre Lévy (Lévy, 1997:pp.37,55).
183 A term from Pierre Lévy (Lévy, 1997:pp.39-55).

stance, Himanen's, Florida's, and Castells' accounts of the Information Age, but especially in Pierre Lévy's texts on collective intelligence[184].

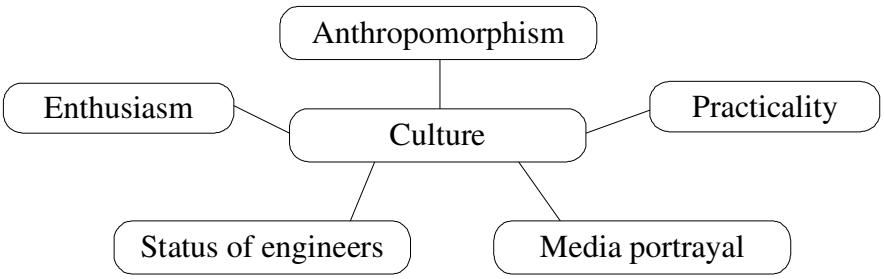


Figure 13: Aspects of Culture Encouraging the Development of Early Computers

In a culture, like the American culture that is portrayed by Bowles and Flamm, where technologies are glorified and portrayed as extensions of people, technological development modifies the interrelations of technoscience, people, businesses, institutions, individuals, and so forth. In contrast, consider the portrayal of British culture by Bowles and Flamm; a culture in which computing technology is merely a means of getting tedious jobs done. In comparison with the American culture, the changes that technology brings to the British culture are much more minor: A technology that is not anthropomorphized is not a rival to mankind, except for perhaps technologies that may compete with people in the labor market. If technology is not anthropomorphized, it does not force people to redefine their places in the socio-industrial-institutional complex, much less the universe. The effect of computing technology in these two media cultures is radically different. Technology is leading, guiding, as well as de- and re-categorizing in the former culture, but servient and preserving the *status quo* in the latter culture. The different aspects of culture that, arguably, encouraged the development of computing technology in the early- to mid-1900s are portrayed in Figure 13.

Crossing the Newton-Maxwell Gap

The inaccuracy and slowness of the Differential Analyzer actually led to the Ballistic Research Laboratory of the U.S. Army to agree to fund a new high-speed computer, starting from May 1943[185]. This agreement in part led to the beginning of the construction of ENIAC[186] at the Moore School of Electrical Engineering at the University of Pennsylvania. This change is sometimes called crossing the Newton-Max-

184 Himanen, 2001; Florida, 2003; Castells, 2001; Lévy, 1997

well gap because it marks a paradigm shift from mechanical computation (governed by Newton's laws of motion) to electronic devices (governed by Maxwell's laws of electromagnetic radiation)[187].

In the 1940s, computing was not performed for its own sake, but always as a means to an end[188], and at that time the end was dictated by world politics. The lack of effective calculating technology was a major bottleneck to the deployment of new weapons[189] and to the calculation of trajectory tables[190]. The U.S. Army took a gamble on an untested technology, bypassed the established scientific community[191], and decided to fund a new technology proposed by Eckert and Mauchly at the Moore School. This gave John Mauchly and John Presper Eckert the chance to realize their ideas for an electronic computing machine.

However, the established scientific computing community, headed by Vannevar Bush, fiercely opposed the ENIAC project and Eckert and Mauchly's choice of electronic circuit elements[192]. It was the Army's willingness to gamble on a radically new approach, and Eckert and Mauchly's stubborn defiance of the scientific establishment, that led to the birth of the first all-electronic computers. This is but one of the examples where computing technology has taken a route opposed by the scientific establishment. The question of contingency raised by Ian Hacking's first sticking points of constructionism remains[193]. That is, it is not certain if computing technology would have taken another, equally successful path (the one favored by the establishment) without the influence of Eckert, Mauchly, and the U.S. Army.

If one puts the Eckert-Mauchly project in Pickering's "mangle"[194] framework, it seems that automatic computing could have turned out differently without Eckert and Mauchly. Eckert and Mauchly encountered a number of clashes between 1) the

185 Leslie Comrie stated that the accuracy of a differential analyzer was roughly $1/x$, where x is the number of pounds (money) one is prepared to spend (Comrie, 1944). About the slowness, Comrie noted that where the computer (human) would need weeks to achieve the result, the analyzer can obtain the result in a few days (Comrie, 1944).

186 Polachek, 1997; Winegrad, 1996

187 Ceruzzi, 1997

188 Campbell-Kelly & Aspray, 2004:p.70.

189 Flamm, 1988:p.48; Campbell-Kelly & Aspray, 2004:p.73.

190 Williams, 1985:pp.272-273.

191 Flamm, 1988:p.252.

192 Flamm, 1988:p.48.

193 Hacking, 1999:p.78. See page 109 of this thesis.

194 Pickering, 1995; Pickering, 1993

theory of computation, 2) their theory about how computers should work, and 3) the computer itself. They had an idea of how computers should work, but the world "resisted". Eckert and Mauchly accommodated to the clashes between their expectations and the resistance of the world by revising their theory about how their computer should work and by revamping their computer. For instance, the original plan of ENIAC that contained 5,000 vacuum tubes and cost $150,000 escalated to one that contained 18,000 vacuum tubes and cost $400,000; the operating voltages were dropped to make the life-span of vacuum tubes feasible; and quite early in the construction of ENIAC, Eckert and Mauchly (and later von Neumann) became convinced that the design of ENIAC was insufficient for generic computation[195]. The understanding of the *stored-program concept* grew out of accommodations to the problems in building ENIAC. This is a prime example of *the mangle*[196].

The Eckert-Mauchly project raises a question concerning the contingency thesis: Were the accommodation strategies of Eckert and Mauchly inevitable or contingent? On one hand, if their brilliant strategies and ideas were inevitable, and if other researchers too would have eventually come to the same conclusions, there is not much sense in celebrating Eckert and Mauchly's ingenuity. Then the development of electronic computers was only a matter of time—it was inevitable. On the other hand, if their strategies and ideas were independent and original, then the *status quo* of computing is a result of contingencies—computing developed as it did because of the ideas and strategies of people who came together due to certain sociopolitical circumstances, and among those people were Eckert and Mauchly. There is a number of other viewpoints to the question of contingency, and I address here the most interesting ones for the purposes of this thesis.

If one believes that the scientific community was wrong and Mauchly and Eckert were right, this case is an example of the fallibility and dogmatism of the scientific community, thus supporting Kuhn's and Feyerabend's descriptions of science. There is a large variety of scenarios that can follow from assuming that Eckert and Mauchly were wrong and the scientific community was right, or that they all were wrong, or that they all were right. However, those scenarios are not the focus of this thesis.

195 Campbell-Kelly & Aspray, 2004:pp.76-83.
196 Pickering, 1995; Pickering, 1993

It would be difficult to accommodate many events and development routes of the Eckert-Mauchly project into the inevitabilist framework (in which technological development follows an inevitable route defined by natural laws[197]). Many events and developments in the Eckert-Mauchly project surely *seem* contingent. Inevitabilists cannot explain those events in Kuhn's terms, because the Kuhnian account is incompatible with inevitabilism[198]. They cannot resort to falsificationist and positivist accounts of science, either, since a dutiful falsificationist would not have continued developing the computer given the critique and refutations of the reliability and plausibility of Eckert and Mauchly's machine[199]. A practical falsificationism that would allow some dogmatism (as suggested by Popper[200]) would be closer to Feyerabend's anarchistic theory of science than to original falsificationism. If dogmatism is allowed inconsistently, then there is the problem of who decides when dogmatism is allowed and when is it not. If one would not be ready to accept the Feyerabendian position, he or she would need to explain the scientific and technological progress of Eckert and Mauchly in terms other than those discussed in this thesis. (Granted, there are numerous deterministic accounts of technoscientific progress that are not discussed in this thesis.)

Notwithstanding the different explanations of how progress in technoscience occurs, one important contributor to this progress is certain. The U.S. Army played a decisive role in the development of early electronic computing machinery. This is not a social deterministic statement—it is not to say that the form and function of Eckert and Mauchly's computer was defined by the Army. Yet, the role of the Army cannot be ignored. Some of the changes that the military brought in are depicted in Figure 14.

Pugh and Aspray[201] listed technologies, which are central to modern computers and which have been directly funded by customers with national-security funding and priorities: the stored-program computer itself (U.S. Army funding), magnetic-wire storage device (U.S. Army funding), mylar-based magnetic tape (U.S. defense contractors), ferrite-core memory (Department of Defense and U.S. Government, for the

197 Hacking, 1999; or the nomological interpretation of technological determinism in Bruce Bimber's terms (Bimber, 1994).
198 Hacking, 1999:p.99.
199 Flamm, 1988:p.48; Williams, 1985:p.275; Winegrad, 1996; Campbell-Kelly & Aspray, 2004:pp.79-80.
200 Popper, 1970:p.55.
201 Pugh and Aspray, 1996

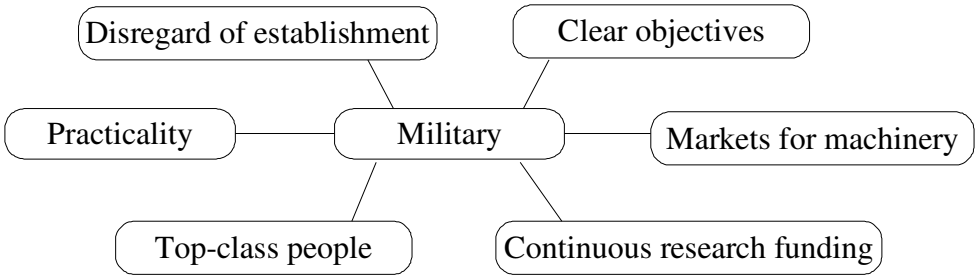


Figure 14: The Influence of U.S. Military in the Early Computers

SAGE project), real-time operation (SAGE), overlapping of computation and input-output functions (SAGE), cathode-ray-tube displays with light pens (SAGE), duplexed operation for improved reliability (SAGE), digital data transmission via phone lines (SAGE), magnetic-drum storage devices (ERA with government contract), and semiconductor device technologies (NSA among other national-security customers).

The traditional cost-sensitive customers would not have funded such experimental projects[202], but for national security purposes, especially in the post-World War II and Cold War years, cost was not an important factor[203]. It is not clear, however, how the military and national security funding affected the development of computers in this early phase—for instance, it is unknown whether development would have been slower without military funding, or whether researchers working without military funding would have been able to develop much cheaper alternative solutions.

202 With one exception: the British catering and food-manufacturing company J. Lyons, which financially supported Cambridge University in building EDSAC, and also built its own, business-oriented version of EDSAC, LEO I– finished December 1953 (Land, 2000).

203 Flamm, 1988:p.2.

Number Crunchers

"*Computing Super-Brain Aids Army.*"[204]

In This Section:

- ✔ What factors were crucial to the construction of ENIAC?
- ✔ What was the role of contingency in the birth of electronic computing?
- ✔ What motivated or discouraged the business sector to participate in the development of early electronic computing?

At the Moore School of Electrical Engineering at the University of Pennsylvania, John Mauchly was in the midst of a very strong research and training center for electrical engineering, which meant that he had both mentors above him and students beneath him—both of whom he could enlist into a project like ENIAC[205]. The Army had moved a number of top scientists to the Moore School (including Goldstine[206]), and the academic programs were accelerated by eliminating vacations[207]. Mauchly was not the one to initiate everything—Mitchell Marcus and Atsushi Akera argued that it took Herman Goldstine's initiative to launch the ENIAC project, suggests that the birth of electronic computing owed itself as much to various trends in the historical context as the ability of Mauchly to draw together various pieces of the puzzle himself[208].

The designers and constructors of ENIAC had a grand vision for their machine: David Grier wrote that they believed ENIAC would change the nature of science and establish a new scientific method based on electronic computation[209]. To make this change clear, the researchers attempted to delineate a clear boundary between the new world of electronic computers and the older world of human computers by using words that could not be connected with human computing (e.g., they talked about "program" instead of "calculation"). I argue that another reason for the new terminology was the incommensurability of electronic computers with previous knowledge, but I discuss this matter later in this thesis in connection with von Neumann's anthropomorphic terminology.

---

204 *Newark Star Ledger*, February 15th, 1946, as quoted in Martin, 1993.
205 Marcus and Akera, 1996
206 Flamm, 1988:p.47; Campbell-Kelly & Aspray, 2004:p.76.
207 Campbell-Kelly & Aspray, 2004:p.71.
208 Marcus and Akera, 1996
209 Grier, 1996

Despite the opposition in the beginning, when ENIAC was completed it was the first large-scale electronic computer[210]. Marcus and Akera argued that it was a crucial machine that convinced many institutions and people—scientists, military officials and industrialists alike—to commit to the rapid development of electronic computing. ENIAC was also the *only* electronic computer in the world for three years[211]. According to Marcus and Akera, the roots of high-speed electronic computing had multiple origins, which fundamentally came together only because of the particular circumstances surrounding Word War II[212].

The first origin came from the development in the elements of electronic computing itself (for the purposes of different kinds of measuring equipment, not for computers as such). The second origin came from the advances in theoretical and experimental physics, notably by John Atanasoff and John Mauchly. Third, the particular situation of the Moore School of Electrical Engineering[213] provided a fertile soil for the innovative ideas which Mauchly and others brought to it.

The keywords of Marcus and Akera's origins of electronic digital computing are

(1) technological prerequisites (instruments),

(2) a scientific base (theory), and

(3) sociocultural issues (culture).

Flamm, on the other hand, named four origins of computing[214]:

(1) mechanical calculating machines (instruments),

(2) differential analyzers used to model physics (modeling),

(3) new components (instruments), and

210 However, ENIAC was not a stored-program computer (where the program as well as data are symbols located in the memory), but for each different kind of computation, a manual rewiring of wiring patterns was needed–and this took hours or days to accomplish. Therefore, some argue that it should rather be called a *programmable calculator* (Pugh and Aspray, 1996). In 1948 rewiring was made easier by introducing switch banks so that reprogramming could be accomplished by re-setting the switches (Pugh and Aspray, 1996).

211 Mauchly, 1979

212 Marcus and Akera, 1996

213 Marcus and Akera noted that firms such as Philco, RCA, and Atwater Kent were located close together, and major research facilities such as Bell Laboratories were not so far away either. The undergraduate, as well as masters level program at the Moore School, was specifically designed to supply this regional industry with well-trained engineers. The educational program provided both the faculty and students, as well as the laboratory facilities, that were essential to the development of ENIAC (Marcus and Akera, 1996).

214 Flamm, 1988

(4) the abstract conceptualization of information and information processing (theory).

In addition, the military roots of computing are emphasized throughout Flamm's book. These themes—instruments, theory, modeling, and culture—occur in a large number of texts on the early history of the computer, with different emphases. They are referred to as, for instance, (*a*) "engineers", "theoreticians", and "attitudes"[215]; (*b*) "engineers", "logicians", and "war effort"[216]; and (*c*) "design", "construction", "people", and "science"[217]. The same issues are addressed in my early definitions of *ethnocomputing*[218]: *data structures*, *algorithms*, *tools and theory*, and *uses*. *Data structures* are inherent in Flamm's (4); *algorithms* are a part of Marcus and Akera's (2) as well as Flamm's (4); *tools and theory* are a part of Marcus and Akera's (1) as well as Flamm's (1) and (3); and *uses* is clear in Flamm's (2).

Pugh and Aspray wrote that World War II also brought about the shift of emphasis from private companies' demand for *cost-effectiveness* to national-security agencies' demand for function, performance, and availability at *any cost*[219]. Dilys Winegrad argued that the theoretical and practical developments at the University of Pennsylvania would at any other time, in all likelihood, have been dismissed as interesting, but impractical. Winegrad claimed that the projects at Penn would have undoubtedly been rejected for the simple reason that they cost too much[220].

Mary Croarken argued that due to World War II the importance of mechanizing and centralizing computation was recognized in both government and academic circles[221]. In England, World War II had profound effects on the immediate and later work of the Cambridge Mathematical Laboratory. At the local level, the laboratory's premises and equipment were leased by the Ministry of Supply, and in the wider context, the war accentuated the growing need for scientific research and, with it, the increased need for automatic computation[222]. Figure 15 shows a composite of some

215 Bowles, 1996
216 Campbell-Kelly & Aspray, 2004
217 Williams, 1985
218 Tedre, 2002
219 Pugh and Aspray, 1996
220 Winegrad, 1996
221 Croarken, 1992
222 Croarken, 1992

of the aspects that were brought together by the Second World War, aspects that had a fundamental effect on the birth of computing in the early 1940s.

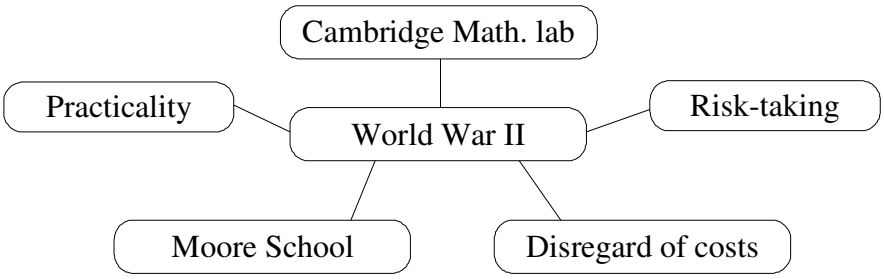


Figure 15: Aspects That the World War II Brought Together

William Aspray, however denied that World War II had a uniformly positive effect on the development of computing in America. He stated that, for example, before the war, MIT had a much broader and more active computer program than is generally recognized[223]. According to Aspray, the war actually brought a wide variety of research projects to an end, including research on digital techniques, electronic elements, and a general-purpose digital computing machine.

Contingencies Surrounding ENIAC

Unfortunately, as David Grier noted, ENIAC became a burden before it was even completed; It became operational just as its developers began to appreciate and understand stored memory programs and serial arithmetic machines[224] (ENIAC was a parallel machine). To the developers of ENIAC (among them, Eckert and Mauchly), the *stored-program computer*[225] concept (conceived apparently around January 1944[226]) was so important and so powerful that they wanted to put aside their old work and ENIAC, and to begin work on new machines. Grier wrote that they quickly discovered that they could not abandon their old ideas as soon as they wished because they had made large intellectual investments in the old techno-

223 Aspray, 2000; *NB*: the term "general-purpose" is a bit misconceived. Perhaps "multipurpose" would be a better term. If there really were a general-purpose machine design, there would be no need for any other design (cf. Sammet, 1991, about the use of "general purpose language" and "multipurpose language"). However, because the term "general-purpose" is so firmly established, it is used in this thesis.

224 Grier, 1996

225 In a stored-program computer the instructions on how to do the computation and the data required or generated by the computation reside all in the memory of the computer, and the computer can be reprogrammed for different tasks just by uploading different instructions into the memory–no rewiring is needed as was the case with ENIAC.

226 Mauchly, 1979; Williams, 1985:pp.298-302, 411.

logy[227], and because the sponsors needed to get their money's worth. John von Neumann, however, published Eckert and Mauchly's ideas in his landmark text *First Draft of a Report on the EDVAC*[228] in June 1945, and thus, in a twist of fate in computer history, the architecture became to be known as *von Neumann architecture*[229] (discussed further in this chapter).

It was also a historical contingency that the ideas generated by the people associated with ENIAC were, after the unveiling of the machine, freed of their military security classifications (for instance, the British Colossus computers were not[230]). The entire development of ENIAC was done secretly. No papers could be published, and discussion was limited to initiates[231]. Von Neumann has been criticized for using anthropomorphic language in connection with computers[232], but actually his (1) use of neurological rather than engineering terms enabled him to circumvent military security[233]. Pugh and Aspray noted that other reasons for the loosening of military security classifications were (2) the failure of ENIAC to be completed in time to demonstrate its military value during the war, (3) the perception that ENIAC was a general-purpose calculator rather than a specialized military machine, and (4) the desire of the army and the University of Pennsylvania to publicize their accomplishments in electronics[234].

Note that even though Dijkstra claimed that the use of anthropomorphic terminology when dealing with computer systems is a "symptom of professional immaturity",[235] anthropomorphic language is not shunned today. Everyday computing terminology

---

227 Grier, 1996

228 Neumann, 1945

229 The widely-adopted term "*von Neumann-architecture*" ignores the developers of ENIAC, that is, Mauchly and Eckert among others who also devised the stored-program concept when they understood the limitations of ENIAC. This conflict, together with a patent rights issue, lead both Eckert and Mauchly to leave the Moore School. (Grier, 1996; Williams, 1985:pp.302-303; Mauchly, 1979). Mauchly noted that von Neumann himself had stated that he had done the design as an accommodation for the Moore School group, but Herman Goldstine "inadvertently distributed" the report with John von Neumann as the sole author (see e.g. Mauchly, 1979; Pugh and Aspray, 1996; Williams, 1985:p.302).

230 Williams, 1985:pp.287-296.

231 Winegrad, 1996

232 Eckert and Mauchly were careful to shun anthropomorphic terms to describe their work. Though they did resort to the term "instructions", they never talked of "instructing" a machine. George Stibitz was more direct in attempting to distance ENIAC from anthropomorphic terms, asking the press to better understand what they were trying to do (see Grier, 1996; Stibitz, 1946)

233 Pugh and Aspray, 1996; see Neumann, 1945 for terms such as *sensory* or *afferent neurons* and *motor* or *efferent neurons*.

234 Pugh and Aspray, 1996

235 Dijkstra, 1975b in Dijkstra, 1982:pp.129-131.

includes terms such as neural networks, master and slave, sleeping, killing, dying and being alive (processes), artificial intelligence and memory, agents, viruses, parents, siblings and relatives (trees), and so forth[236]. As I see it, there is an ongoing anthropomorphizing of technology and simultaneous technomorphizing of humans ("software of the mind"[237], characterizations of the brain as central processor, the human memory as hard drive, and other technomorphisms). This tendency is well-supported by those strands of a modern, scientific world view where people are seen as complex machines. A number of new philosophical trends, such as transhumanism[238], well from this contraction of the human-machine gap.

As I noted earlier in this section, I suggest that incommensurability with earlier knowledge was one reason for differing opinions on the terminology of the new machinery. My interpretation is that John von Neumann attempted a *metaphor transfer* from neuropsychology to computing technology—namely, referring to an article published in the *Bulletin of Mathematical Biophysics* (1943), von Neumann made parallels from neuropsychology (neurons, synapses, axons) to EDVAC[239]. Furthermore, he called input and output devices of the machine *organs*: "*The device must have organs to transfer* [numerical information] *from R into its specific parts C and M*"[240]. It is also my interpretation that in contrast to von Neumann, Mauchly and Eckert wanted to disassociate from earlier science by introducing new terminology, such as *a program*. This argument is supported by David Grier: "*the researchers attempted to delineate a clear boundary between the new world of electronic computers and the older world of human computers*"[241].

I agree with Grier's argument that the stored program model was not a paradigm shift in the Kuhnian sense because there was no crisis instigated by accumulating anomalies[242]. Nevertheless, I argue that EDVAC still epitomized an intellectual and

236 There is also plenty of culture-specific terminology and terminology that may rouse different feelings among people: take, for instance, terms such as *daemon*, *Trojan horse*, *divide and conquer*, *greedy algorithm*, *trapdoor* and *backdoor*, *worm*, *abort*, *terminate*, and *execute*. This is not to say that it would be bad or good to use such terminology, just to notice that a phenomenon of attributing anthropomorphic terms is common.

237 Hofstede, 1997

238 *Transhumanism* is an umbrella term for a number of philosophies that principally include the view that new technologies can take humankind to a new level of consciousness or being, as well as add to the quality and meaning of human life. These philosophies include, for instance, posthumanism.

239 Neumann, 1945

240 Neumann, 1945

241 Grier, 1996

242 Grier, 1996

technological paradigm shift without Kuhnian prerequisites. The development of EDVAC entailed a number of concepts without an analogy to the normal science of the time (*stored program*, *memory addresses*, *registers*, and such[243]). The concepts were incommensurable, and therefore, following both Feyerabend and Kuhn, one could not demand a continuation between old and new concepts[244]. Such a demand for continuation (or explanation, or reduction) of concepts simply could not have been realized. Von Neumann's metaphor transfer from neuropsychology may have been an attempt to ease the innofusion of EDVAC by contextualization[245] but in retrospect it seems a bit awkward.

As already mentioned, the idea of a stored-program computer was first embodied in EDVAC designs (First Draft of a Report on the EDVAC[246]) created at the University of Pennsylvania late in 1944. The war-related projects led to ENIAC, and from ENIAC emerged the EDVAC concept[247]. The "First Draft of a Report on the EDVAC"[248] created considerable interest[249]. In May 1946, Leslie John Comrie returned to England from a visit to America with a copy of von Neumann's draft, and as soon as Maurice Wilkes from Cambridge had read the document, he was determined to build a stored-program computer[250]. Wilkes' Mathematical Laboratory had sufficient funds to get the project started, and the nature of the project was well within the mission statement of the Mathematical Laboratory. The Moore School had offered a famous series of lectures about everything that was needed for constructing a computer, and Wilkes, with Douglas Hartree and David Reese, soon put the Moore School lectures into practice[251]. Wilkes also had the support of the Mathematics faculty of Cambridge. The next few years were spent in intense activity—setting up an electronics laboratory and building the EDSAC (electronic delay storage automatic calculator)[252].

---

243 These are the modern day equivalents for the EDVAC terms.
244 Feyerabend, 1970; Kuhn, 1970
245 *Contextualization* here means situating a new concept into a familiar context or expressing a new idea in terms that are familiar to the audience.
246 Neumann, 1945
247 Croarken, 1992
248 Neumann, 1945
249 Pugh and Aspray, 1996
250 Croarken, 1992
251 Williams, 1985:p.303.
252 Croarken, 1992

Just before the turn of the decade, the world's first stored-program computer became operational, but two computers have been suggested to have been first. EDSAC in the Cambridge Mathematical Laboratory performed its first automatic calculation on May 6, 1949[253], but John W. Mauchly claimed that the BINAC, or the first of the two computers that became the BINAC, had undergone testing early in 1949, and that it had run nonstop for 44 hours in April 1949[254]. Mauchly wrote that the test was then interrupted so that the engineers could get on with other work. Other sources claim that the BINAC ran its first program in the late summer of 1949[255].

The disagreement most probably derives from the difference between the testing-debugging phase and the actual announcement of a new computer[256]. On one hand, if one thinks of running "44 hours nonstop" in the testing and debugging phase as the début of a machine, then BINAC might have been the first one. On the other hand, if one thinks of constructing a full-sized machine capable of solving realistic problems and introducing it to regular use as the début of a machine, then EDSAC was the first[257]. This question is merely a matter of definition: "When assessing which computer was the first, should one use the earliest "gleam in the eye", first test usage date, completion of testing, or first installation to a customer[258]?".

History-writing based on "firsts" often runs into trouble because of the questions above. The BINAC/EDSAC-controversy is just one example of such problems. In the larger picture, the whole issue of "firsts" does not really matter that much. For instance, Mary Croarken suggested that regardless of which computer was actually the first, it was more important that Maurice Wilkes of Cambridge had created a young and vibrant team at the Cambridge Mathematical Laboratory[259]—the construction of EDSAC was just the beginning of a continuing program of computer science research and teaching. Wilkes' work on EDSAC led directly to the construction of the first business computer, LEO I.

253 Worsley, 1950
254 Mauchly, 1979
255 Williams, 1985:pp.361-362.
256 See, e.g. Kidder, 1981, for a description of the different phases in building a computer.
257 Williams, 1985:pp.333-334.
258 cf. Sammet, 1991; a good example of the fuzziness in where to draw this line can be found in Kidder, 1981.
259 Croarken, 1992. The laboratory was later to be called *Cambridge Computer Laboratory*.

The Birth of Business Computing: LEO I

The enthusiasm of Wilkes, Hartree, and Reese for manufacturing computers in Britain was not matched with an enthusiasm of Britain's old-fashioned businesses for using computers[260]—however, there was an exception. In May 1947, two senior company executives from the British company J. Lyons visited the USA, meeting a number of pioneers in electronic computing, and getting a demonstration of ENIAC[261]. The company executives recognized quickly that the characteristics of the machines they saw could be modified to provide the necessary capabilities to solve some of the problems of business data processing[262]. Here lie the roots of the world's first office computer. According to Georgina Ferry, before the visit by Lyons' people, Goldstine had never thought about using computers in the office, but now he instantly saw the point and became enthusiastic about it[263].

Hinted by both Aiken in Harvard and Goldstine in Princeton, on their return to England, the J. Lyons people visited Cambridge where they were introduced to Maurice Wilkes[264]. The work on the EDSAC (electronic delay storage automatic calculator) had already started. The J. Lyons people were most impressed with the advances in technology, well beyond anything they had seen in the U.S.[265], yet contrary to the American researchers, for whom the money was not a problem, when the J. Lyons executives had visited Cambridge they wrote,

> *They have all their plans drawn and the hold-up is purely due to lack of money.* [...] *We were told that given £2000 they could complete much more rapidly. Both Professor Hartree and Dr Wilkes were willing and keen to co-operate with us, in particular they are interested in applying their machine to any clerical job we may suggest.*[266]

The computer to be built for J. Lyons was dubbed LEO I (Lyons Electronic Office computer One). A member of the LEO team, a professor of information management Frank Land, noted that the company J. Lyons could have played a passive role and waited until such machines would have become commercially available. Land

260 Campbell-Kelly & Aspray, 2004:p.95.
261 Ferry, 2003:p.42.
262 Land, 2000
263 Ferry, 2003:pp.42-43.
264 Land, 2000; Ferry, 2003:p.43.
265 Land, 2000; Ferry, 2003:p.64.
266 Thompson, T.R. and Standingford, O. *Report on Visit to USA*, archives of J. Lyons and Co., May/June 1947, as quoted in Land, 2000 and mentioned by Ferry, 2003:p.65.

argued that the company wanted to have some influence in the design so that the new machine would not be built in a form more suited to handling mathematical and census calculations, but that it would suit business interests better[267]. The uses of LEO were to include payroll, stock control, sales invoicing, and whatever else could be effectively envisaged[268]. Ferry and Flamm argued that the British electrical and office machine companies had not even begun to think about electronic calculating machines[269]. Ferry continued that the post-war government was also unable to initiate anything like this expensive and risky proposal. It took a private company to initiate such a step.

Finally, the J. Lyons board members agreed to support the Cambridge venture, but at the same time to work towards building their own machine with the help of Wilkes and his team[270]. The board also agreed to help finance the work at Cambridge with a grant of £3000. In addition, Lyons offered to make one of their best technical people, Ernest Lenaerts, available to Cambridge[271]. As it turned out later, Lenaerts, a former business clerk, became an important contributor to EDSAC and later a key person in the design and building of Lyons' computer[272]. In return for the Lyons help package, Cambridge agreed to help Lyons design and build their own computer and to help recruit a chief engineer, Dr. John Pinkerton, to head the technical side of the project[273]. A project team was established and work commenced in 1949 to design and build the computer. The collaboration between academy and industry, between Cambridge and J. Lyons, might be the first instance of a user-driven innovation process in the field of computing.

Frank Land argued that there are a number of factors that affected Lyons' pioneering move into business computing[274]:

(1)*The nature of Lyons' business* was characterized by a very large range of food products and food provision services. Their trade was mass transactions with relatively low average values, so their competitiveness depended on advantageous

267 Land, 2000
268 Aris, 2000
269 Ferry, 2003:p.67; Flamm, 1988:p.146.
270 Land, 2000; Ferry, 2003:pp.67-68.
271 Williams, 1985:p.333; Land, 2000; Ferry, 2003:p.69.
272 Land, 2000
273 Ferry, 2003:p.92.
274 Land, 2000

pricing of its products. Computers were able to deliver an unparalleled fit between cost and income.

(2)*Personnel and leadership*; the company recognized the need for top-class management early on. Lyons recruited top-class academic people and permitted them to take leadership roles in the management of the company. Academic people understood the possibilities of experimental technologies.

(3)*Self-sufficiency*; the strategy of Lyons was to be as self-sufficient as possible. They had the confidence and experience to believe they could provide the relevant goods or services more effectively than any other contractor.

Similar reasons are also given by another member of LEO group, John Aris[275], but he argued that the reliance on personnel also accounted for the demise of LEO.

In light of the sources used above, the founding of the office computer business in Britain seems like quite an incidental happening. It was a result of a number of social, economic, and technological aspects mixed with extraordinary foresight, or simple contingency. There was a group of people with a vision and creative imagination[276], an economic situation that encouraged adventurous implementation[277], and enough technological knowledge (yet no experience) of how to build a computer[278]. The simultaneous construction of LEO I and EDSAC marks two turning points—the beginning of electronic office computing and user-driven development.

The Origins of the Verb *to Program*

Many of the terms of today's computing were coined around the time of the building of ENIAC. In the year 1944 the term *computer* was, instead of the current sense, associated with people whose job was to do calculations. Take, for instance, the comment by Leslie J. Comrie: "*The chance of turning anyone into a good computer decreases rapidly from the time they leave school or college to vanishing point five years later—or even earlier.*"[279] Desktop calculators were in high demand during World War II by the thousands of human computers who did calculations for the

275 Aris, 2000
276 Ferry, 2003:p.*ix.*
277 Aris, 2000
278 Aris, 2000
279 Comrie, 1944

design and operation of military equipment[280]. Because most of these human computers were women, things like time saving were referred to accordingly: "*The work done was the equivalent of 4 to 10 girl-years*"[281].

Irrespective of the actual beginning of the construction of ENIAC, the verb *to program* was introduced by John Mauchly in his 1942 paper on electronic computing (however, it was introduced in the context of ENIAC, *the programming device*[282], not in its current meaning). Mauchly had borrowed the term *to program* from contemporaneous electronic control engineering, but the word came from the Greek $\pi\rho o$ (before) and $\gamma\rho\alpha\phi\varepsilon\omega$ (to write) and was first used in 1633 to mean an official public notice[283]. The modern meaning of the term *program* first emerged during the summer of 1946 at the Moore Lectures, the six-week-long summer class on electronic digital computers[284] (of course, to name an idea is not the same as inventing that idea[285]). Because of the inconsistent use of the term in the early years, it is hard to say who really was the one who introduced the term *programming*. Nonetheless, the need for totally new terminology suggests that there were significantly new concepts and activities. Those new concepts and activities were not explainable by single words in the old vocabulary. I interpret the need for new vocabulary as a signal of a significant conceptual novelty, sometimes even a paradigm shift. I discuss later whether the development of ENIAC/EDVAC was a paradigm shift or not.

Summary: The Context of the Birth of Electronic Computing

All authors in the history of computing recognize the obvious: The political situation (*World War II*) and one branch of government (*military*) were the driving forces in the development of computing machinery towards electronic computers. An aggregate of factors that contributed to the development of early electronic computers is portrayed in Figure 16. The advancements in, for instance, theoretical physics (*science*) and engineering (*design*) that lead to new innovations (*instruments*) are also agreed on as having a direct connection with the development of computers. The interplay of theories, methods, and concepts across different sciences (*interdisciplinar-*

280 Pugh and Aspray, 1996
281 Stibitz, 1946, in Campbell-Kelly & Williams, 1985.
282 Mauchly, 1942
283 Grier, 1996
284 See, for instance, Eckert, 1946.
285 Grier, 1996

ity) and the decisive role of a few people (*individuals*) is also widely accepted as having played a role in the process. It is difficult to say which of those were more important than others, and which of those were sufficient or necessary conditions for the birth of electronic computing.

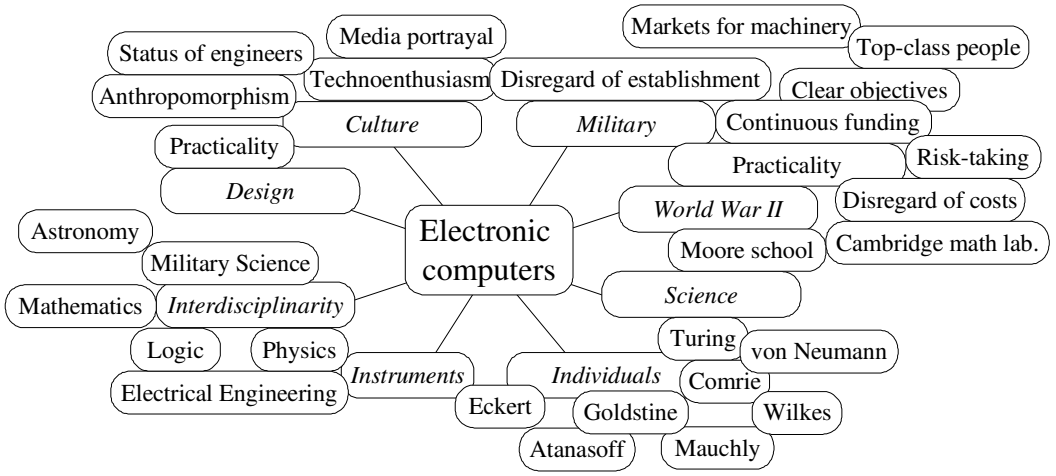


Figure 16: The Context of Early Electronic Computers

Furthermore, many sources suggest that coincidence, such as freeing ENIAC of military classification or risks taken by Lyons (*contingency*), was an important aspect. Interestingly, economic aspects did not seem to play much of a role—since human computers were cheap to use, electronic computers were developed *despite the costs*, rather than their potential for cost-saving (except for LEO I). Finally, the sources indicate that a number of sociocultural aspects (*culture*) had some degree of influence on the development of early computing machinery.

Early Academic Computing

"Army 'Robot Brain' Does 100 Years Math in 2 Hours:
New Device to Cut Research Blocks"[286]

In This Section:

- ✔ Was an early entry to the field of computing an advantage in the academic world?
- ✔ Was the academic world receptive to the opportunities of automatic computing?
- ✔ Did academic research in computing follow an orderly plan?

Early academic computing emerged from a mixed collection of disciplines, and for a long time it was not certain what computing as a discipline exactly was. It was uncertain whether computing was a theoretical subject, like mathematics, or a practical, design-oriented subject, like engineering. It was not certain either if the value of computing technology was merely instrumental, or if computers and the phenomena surrounding them were worthy of academic study. While these questions were still unclear, a number of universities made an early start in the field of computing. Although some of them were able to keep their head start, some lost their advantage along the way. This section discusses five American universities that made an early entry in the field of computing and the characteristics that made a difference in their later success.

Early Entrants to the Field of Computing

William Aspray, who is a historian of computing, argued that those American universities that entered the computing field in the 1940s[287] gained significant advantages from their early entry[288]. He wrote that among those advantages were (1) established reputations in the computing field, (2) good ties with potential employers for their graduates, (3) the ability to garner a large portion of what limited government and industry support and funding was available, (4) a decisive advantage over later entrants in recruiting faculty from the minuscule pool of mathematicians and engineers with computer experience (as well as in attracting students), (5) setting the

286 *New York Journal-American*, February 15th, 1946, as quoted in Martin, 1993.

287 The first independent university departments of *computer science* did not emerge before the 1960s, so *computing field* or *field of computing* is used here instead.

288 Aspray, 2000. William Aspray has written a number of books on the history of computing, among them one with Martin Campbell-Kelly, another famous historian of computing (Campbell-Kelly & Aspray, 2004). Aspray is also the author or co-author of a notable number of journal articles on the early history of computers.

design standards not only for the academic sector but also for the computer industry, and (6) being able to set the research agenda in the emerging discipline in ways that were to their own advantage.

Points (1) to (4) in Aspray's list are practical concerns, yet they most likely have an influence on issues that are beyond their immediate scope. They have such influence because an established reputation can earn the university an authority position on scientific practices (point 1); because the amount of funding can also give the university power also outside academic boundaries (point 3); and because the famous names can make *ad hominem* or prestige-based value assessments more probable (point 4). Point (5) is a case of a Kuhnian *exemplar*—universities are able to set models and examples that later replace explicit rules as a basis for the solutions to the problems of normal science[289]. Finally, point (6) speaks for itself: Merely being first can be enough to shape the discipline. In terms of the definition of problems presented earlier in this thesis (p.191), early entrants are able to set the research problems so that the *starting points* are most favorable for them, the *outcomes* are anticipated, and only the *methodology* to achieve the results is unknown. This sort of problems are of class C-O-C, and in terms of Kuhn, they would be referred to as *puzzles*[290].

Points (5) and (6) are explicit in Kuhn's and Feyerabend's accounts of science. If researchers set the research agenda according to their own expertise, then there is a congruence of agendas and expertise, but perhaps not a connection between agendas and the most important problems. This is explicit in Kuhn's description of scientific practice. In addition, standards and agenda partly dictate the language of science. According to Kuhn and Feyerabend, there is no neutral scientific language[291], and the norm of scientific language (in the Kuhnian sense) is set by a consensus of researchers, which in this case would be the early entrants to the field of computing.

Aspray noted that there are also disadvantages to early entrance. The disadvantages of early entrance are, for example, (*a*) the need to devote time away from computer projects to develop peripherals, component technologies, and test equipment, (*b*) the need to spend a great deal of effort educating governmental officials about the value

289 Kuhn, 1996:pp.175,187-191; Kuhn, 1970.
290 Kuhn, 1996:p.36.
291 Feyerabend, 1970; Kuhn, 1970

of computing, and (*c*) great uncertainty and many dead-ends with the technical direction of computers.

Note the resemblance of disadvantages (*a*) and (*c*) with Pickering's "mangle"[292]. In the face of a great uncertainty, pioneers attempt to construct instruments. They have their theories of a domain and theories of the equipment they are constructing, but things usually do not go as planned—the world resists. Through repeated rewriting and rebuilding attempts, the pioneers hope to get a robust fit between three things: the theory, the instrument, and the theory of how the instrument works[293]. The theories and cutting-edge technologies that are considered pioneering are the ones that go through the mangle; the followers of the pioneers can avoid the troubles of the pioneers. In the following pages, the starting positions of five American universities, which pioneered in computer research, are described. This early history of academic computing is a prime example of sociocultural and institutional influences in the development of an academic discipline.

MIT

The Massachusetts Institute of Technology, MIT, to begin with, was shaped by three factors in the interwar years, William Aspray argued[294]. In the early decades of 20th century, MIT had already become the leading institution in mathematically intensive, science-based engineering in the USA. First, MIT, especially the electrical engineering section, had strong *ties to industry*. In MIT, Vannevar Bush and Harold Hazen carried out substantial research for industry with the calculating machines they built[295]. Second, Bush and his colleagues had relatively strong *ties to the government and the military* at a time when there was relatively little contact between universities and the federal government. Third, electrical engineering was one of the most important departments at MIT, and members of the department's faculty were frequently recruited for senior *administrative posts* at the university. Aspray stated,

> *These three factors reinforced one another. Faculty members interested in computing received institutional support from the department and the university's central administration; they received strong financial support from the industrial and government sectors; and they had close contact with the*

292 Pickering, 1995
293 Pickering, 1995; Hacking, 1999:p.71.
294 Aspray, 2000
295 Campbell-Kelly & Aspray, 2004:pp.53-54.

most sophisticated users and developers of computing technology. This situation was unparalleled among American universities in the interwar period.[296]

Harvard

Harvard's entry into computing was attributed completely to Howard Aiken (1900-1973), Aspray wrote. Aiken got Harvard and IBM to sign an agreement where IBM would conduct the engineering and construction of an automatic computing machine (known as Harvard Mark I) for Harvard to use in its scientific research. However, Michael Williams, a historian of computing, wrote that a major quarrel occurred between IBM's Thomas Watson (Sr.) and Aiken at the dedication ceremony of Harvard Mark I, in the summer 1944[297]. Harvard's people thought of Aiken's functional specifications for the machine as being the most important, while IBM regarded the most important work to be the design, engineering, and construction. Campbell-Kelly and Aspray wrote that people at IBM claimed that Harvard did not give due credit to IBM for its generosity[298]. Campbell-Kelly and Aspray blamed this conflict on Aiken's youthful arrogance. The end result was an abrupt termination of IBM support for computing at Harvard[299] and a fall out between the two men that was to last the rest of their lives[300]. But although Harvard Mark I was greeted with technoenthusiasm—the press called it "Harvard's Robot Super-Brain" and declared that "Robot Mathematician Knows All the Answers"[301]—computing was seen in Harvard as having a second-class status.

Aspray, as well as Flamm, wrote that after the IBM incident, Aiken convinced the Navy that the calculator could contribute to the war effort, and co-operation between the Bureau of Ships and Harvard begun[302]. Even though this co-operation ended at the end of the war, the Navy Bureau of Ordnance continued to provide operating funds for the laboratory until 1948, when the Harvard Mark II was completed and shipped to the Navy laboratory. After the Navy funding ended 1949, a funding crisis

296 Aspray, 2000
297 Williams, 1985:pp.240-242.
298 Campbell-Kelly & Aspray, 2004:p.64.
299 Aspray, 2000
300 Williams, 1985:p.242.
301 American Weekly and Popular Science Monthly, as quoted in Campbell-Kelly & Aspray, 2004:p.64.
302 Flamm, 1988:p.41.

arose in the laboratory, and Aiken had to struggle each year to finance his operation[303].

Computing, with its practical bent and the widely held view that it was a service operation, was especially suspect. Aiken's interpersonal skills and management styles also distanced him from his colleagues; Aspray argued that Aiken "*just didn't know the meaning of cooperation*"[304], and having a strong sense of hierarchy, he was called "*the commander*" and "*the boss*" around the laboratory. Contemporaries agree that he was a difficult person to work with—and "*he just could not tolerate anybody not considering him the most eminent computer engineer in the world*"[305]. Aiken's name even appeared at the top of every report produced by the laboratory.

During 1950s, before Aiken retired at 1961, the problems arising from his character had deteriorated Harvard's computing program:

> *While Harvard had been on the cutting edge of computing in the 1940s, it had become an intellectual backwater by the time of Aiken's retirement. Aiken was increasingly seen outside as a technological reactionary, preaching a conservative approach to machine design, mistrust of the reliability of vacuum tubes, and lack of commitment to the stored-program concept.*[306]

Aspray argued that when Aiken retired from Harvard, the laboratory was in poor financial shape, his laboratory was in isolation, there was a leadership vacuum, the educational and research programs needed refreshment, the relationship to the faculty and administration were strained, and the program was highly inbred[307].

Although some historians claim that Aiken failed to grasp the universality of the computer, Aiken has also been defended in this matter[308]. Aiken's computing laboratory is an extreme example of where authoritative leadership can work—for both good and bad. Aiken had both built the program, and created relationships in the process, and brought it down, and destroyed relationships in the process. Furthermore, Aiken's era in Harvard is an example of how large an influence one person can have on a whole field of research in a certain university.

---

303 Aspray, 2000
304 Evident also in a defense of Aiken by Cohen, 1998.
305 Cohen, 1998
306 Aspray, 2000
307 Aspray, 2000
308 See Copeland, 2004 for accusations and defense.

University of Pennsylvania

The University of Pennsylvania, with its development of high-speed calculating devices, is perhaps the most famous university in the modern history of computing. Aspray wrote,

> *Penn should have been extremely well positioned to become one of the academic leaders in the computing field. In 1946 it possessed the only electronic calculator in the world, held government funding to build the prototype stored-program computer, had close working relations with the military groups that had a continuing interest in funding the development of computers in the Cold War era, and employed a talented staff with unequalled experience.*[309]

However, three factors impeded the Moore School of the University of Pennsylvania from becoming a dominant player in postwar academic computer science. First, some of the *principal people* involved in the ENIAC and EDVAC projects had come to the Moore School only because of the war[310]. After the war, von Neumann, for example, started a project at the Institute for Advanced Study, taking with him several people from Penn[311]. Second, the university administration was uneasy about receiving military support in peacetime[312]. George McClelland, who was the president of the university, was not sure whether such funding would advance the *university's objectives* or corrupt them instead. The third factor was the *university's regulations* concerning faculty with commercial interests. Particularly, quarrels about patent rights made some key persons, Eckert and Mauchly among them, leave the university at the end of March 1946[313]. Many from the Moore School left after Eckert and Mauchly's departure[314].

A new patent policy was adopted in 1946 to protect the “intellectual purity” of the research that the University of Pennsylvania sponsored[315]. The patent quarrel with the dispute over the term *von Neumann architecture* caused Eckert and Mauchly to resign in March 1946—had they stayed beyond that, the new policy would have re-

309 Aspray, 2000
310 Williams, 1985:pp.273-274; Aspray, 2000; Winegrad, 1996.
311 Flamm, 1988:pp.50,52.
312 Aspray, 2000
313 Flamm, 1988:pp.50,51; Aspray, 2000.
314 Flamm, 1988:p.50.
315 Aspray, 2000

quired them to assign all their patents to the university[316]. Instead, they formed the Electronic Control Company (ECC), later called the Eckert-Mauchly Computer Corporation (EMCC). Historian Kenneth Flamm called Eckert and Mauchly's move a "daring bet"[317]: It was the first company to be established for the sole purpose of designing, manufacturing, and marketing electronic stored-program computers. When UNIVAC was accepted by the Bureau of Census in 1951, EMCC also became the first company to produce an electronic stored-program computer for the commercial market[318].

The changing of University of Pennsylvania policies led—directly or indirectly—to the beginning of the commercial computer industry (in the UK, LEO became an industrial product later). One may argue that it was just a matter of time before the commercial computer industry would emerge, but the fact that two brilliant minds, Eckert and Mauchly, were the ones to initiate it, derived from the changes of university policies at the University of Pennsylvania.

Columbia University

Columbia University and IBM announced in February 1945 that they would jointly establish the Watson Scientific Computing Laboratory, on the Columbia campus. Flamm wrote that Thomas Watson Sr., the chairman of IBM, was a long-time friend of Columbia and a trustee of the university since the 1930s[319]. *Personal attachment* was, however, not the only reason for the co-operation between IBM and Columbia. Because the co-operation with Harvard had ended abruptly in the summer of 1944, due to a slight by Howard Aiken[320], IBM did not have other research laboratories at the time.

The *academic-industrial collaboration* was most obvious with Columbia University. For example IBM 805, the International Test Scoring Machine, was constructed for Columbia, and only afterwards developed into a commercial product. Columbia, especially Wallace Eckert (1902-1971), had been IBM's main contact with the scientif-

316 Pugh and Aspray, 1996; Williams, 1985:p303.
317 Flamm, 1988:p.29.
318 Pugh and Aspray, 1996. The J. Lyons' LEO I computer in Great Britain was built in collaboration with Cambridge University (Land, 2000; Ferry, 2003), not by a computer company.
319 Flamm, 1988:p.31.
320 Williams, 1985:p.242.

ic community[321]. In 1945, Watson hired Wallace Eckert to direct a newly-formed the "IBM Department of Pure Science". The department's mission was to help IBM understand the needs of the scientific community and find opportunities for IBM in the academic market. IBM trusted this academic-industrial collaboration to such an extent that they paid Wallace Eckert two and a half times his previous salary, donated an impressive battery of IBM's machines to Eckert's laboratory, furnished a library, and funded Eckert's graduate students.[322]

The work of the laboratory and the academic-industrial collaboration had some bearing on the design of IBM's new electronic products[323]. For example, the Selective Sequence Electronic Calculator (SSEC), designed at Columbia and built by IBM, was intended to give IBM a "super calculator" that would be more powerful than Aiken's machine or any subsequent machines that would evolve from it[324]. Campbell-Kelly and Aspray, as well as Flamm, attributed the development of SSEC partly to IBM's Watson's desire for revenge on Howard Aiken for the argument over Harvard Mark I[325]. The success of SSEC gave IBM engineers useful experience for the design of the IBM 650 calculator, one of IBM's most important products in the 1950s[326]. Williams argued that despite their commercial successes, IBM was by no means a leader in computing, but they were immensely successful in following the product developments of other, smaller firms[327].

Princeton

For *Princeton*, computing came as a result of John von Neumann's wartime experience at the Moore School. Von Neumann began in Princeton in 1936[328] by researching applied mathematics—fluid dynamics in particular. This research led, during the war, to von Neumann becoming a consultant for the National Defense Research Committee, the Navy Bureau of Ordnance, and the Manhattan Project. Von Neumann learned about the ENIAC project through Herman Goldstine, and quickly

321 Campbell-Kelly & Aspray, 2004:pp.100-101. Wallace Eckert should not be confused with J. Presper Eckert.
322 Aspray, 2000
323 Aspray, 2000
324 Flamm, 1988:pp.61-62.
325 Campbell-Kelly & Aspray, 2004:p.64; Flamm, 1988:p.61.
326 Aspray, 2000
327 Williams, 1985:p.382.
328 Flamm, 1988:p.34.

gained access to the classified project[329]. Flamm wrote that in 1945, before ENIAC was operational and before the war had ended in Japan, von Neumann began to plan for the postwar construction of a computer that would be devoted exclusively to scientific research at Princeton's Institute for Advanced Study (IAS)[330]. Note that the purpose was not computer science, but applied science using computers as a tool. Von Neumann's purpose for developing computing was chiefly instrumental.

In order to succeed with his plan, von Neumann needed to overcome a number of challenges. The faculty and administration of the IAS was interested in pure, theoretical issues in mathematics and physics. Similar to Harvard, most of IAS's faculty regarded computing as a practical subject matter that was not worthy of their investigation[331]. Also, to build a computer would require the purchase and construction of extensive laboratory facilities and equipment. Aspray named three major obstacles von Neumann faced:

> [1] *The institute was a place carefully groomed for the scientific elite, and there was great concern among the faculty over having to share their hallowed grounds with engineers, technicians, coders, and operators.* [2] *There was also a serious problem of funding, both to build the computer, and to maintain it and the research scientists who would be brought to the institute to use it in their research.* [3] *Later there was a political problem, when von Neumann's use of the computer for weapons design ran counter to the born-again pacifist views of Robert Oppenheimer, who had been appointed director of the institute.*[332]

According to Aspray and Campbell-Kelly, von Neumann's political deftness, scientific reputation, and military contacts, enabled him to overcome these challenges. Von Neumann managed to persuade his colleagues and the institute administration to permit a computer project to be started if funding was found. A number of people from the Moore School joined von Neumann's computer project[333]. Likewise, funding, both direct and in-kind, was, after all, not so difficult to obtain[334]. Aspray wrote that von Neumann convinced Princeton to become a partner, to help with the engineering of the machine, and to have their scientific faculty use it for their research.

329 Williams, 1985:pp.299-301.
330 Flamm, 1988:p.49.
331 Aspray, 2000
332 Aspray, 2000, underlining added.
333 Campbell-Kelly & Aspray, 2004:p.86.
334 Aspray, 2000

Von Neumann secured RCA (Radio Corporation of America), which had recently moved its research laboratories to the Princeton area, to become a partner and undertake the design of the computer's memory device. He persuaded the military to underwrite most of the costs of the project, arguing that it would be to their benefit to have additional calculating capability and to explore alternative computer designs to those being pursued at Harvard and the Moore School at Pennsylvania University. Finally, von Neumann won the political obstacle:

> *Despite the reliance on military funding, von Neumann was able to retain his objective to restrict the institute computer to scientific research by agreeing that the computer would serve as a prototype and that copies of the computer would be built for military use at other locations (Argonne, Los Alamos, and Oak Ridge laboratories; RAND corporation; and Aberdeen Proving Grounds).*[335]

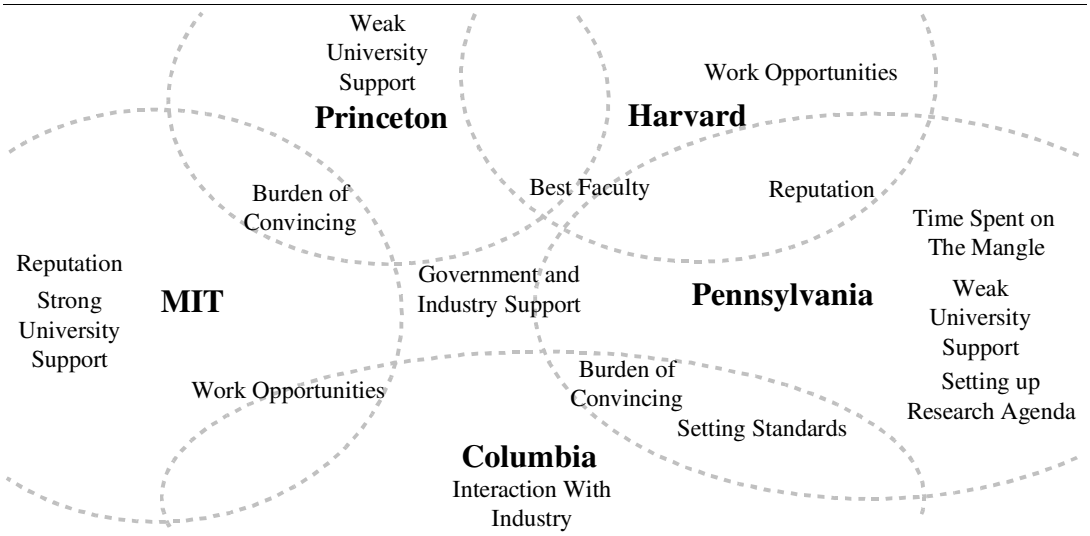


Figure 17: Some Characteristics of Five American Universities in the 1940s

In Figure 17 some characteristics of the academic computing field in the USA after the 1940s are illustrated. Kuhn's descriptive account of science is most obvious in the case of the Moore School of the University of Pennsylvania. The second world war had given the Moore School such a strong leadership position in the field that the school had a decisive influence on the research agenda in the field of computing.

Also Andrew Pickering's theory of "the mangle" can best be seen in the work done in the Moore School: Although the Moore School was *the* vanguard of technological and theoretical development, it arrived at many dead-ends. Sometimes the research-

335 Aspray, 2000

ers at the Moore School had to even knowingly arrive at dead ends. One example is the Moore School's building of ENIAC despite the researchers knowing its limitations and despite the researchers knowing how to overcome them.

The Birth of Programming Languages

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense.[336]

IN THIS SECTION:

- ✔ Why were programming languages developed?
- ✔ Are computer generations eras of *normal science* and technological revolutions *paradigm shifts*?
- ✔ Was the development of high-level languages contingent or determined?
- ✔ What are the reasons behind the major diffusion of FORTRAN and the minor diffusion of ALGOL?
- ✔ What are the most significant factors in language development?

The decade of the 1950s clearly marks the birth of programming languages. The only programming language developed before the 1950s was Konrad Zuse's (1910-1995) *Plankalkül*, but it was never implemented[337]. Despite its age, Plankalkül contains features that are standard in today's programming languages[338]. Plankalkül was, in some ways, a more elegant and advanced programming language than those that appeared 10 to 15 years later[339]. In this section the development of programming languages is paralleled with the development of computer technology. One question in this section concerns computing from the Kuhnian perspective: The history of the computer is divided into computer "generations", but can these generations be considered eras of normal science? Other questions in this section concern, for instance, the contingency thesis and technological momentum in computer science, factors in language development and diffusion, and the origins of high-level programming.

336 Dijkstra, 1975b
337 Bauer & Wössner, 1972; Sammet, 1991
338 Sammet, 1991
339 Backus, 1981:p.28. in Wexelblat, 1981; Bauer & Wössner, 1972.

The First Compiler

One of the early programming pioneers, Captain Grace Murray Hopper, wrote that during the 1950s, the whole establishment of computing was firmly convinced that the only way to program a computer was using octal notation[340]. The pessimists presumed that high-level languages could never generate code that was efficient either in the use of time or the use of memory[341].

In 1951 Hopper began building a set of mathematical subroutines for UNIVAC I, and to standardize them for general use. Hopper wrote that she soon noticed that a few things were badly wrong with the way subroutines were being written at the time[342]. All the subroutines started at line zero, and when copying a subroutine to another program, all the lines should be added to existing addresses—and, as Hopper wrote, programmers were (and perhaps are) lousy adders. Hopper also mentioned that programmers were lousy copyists: Very often *4* would turn into *A*, or *4* would turn into Δ or even *B* into *13*. Out of these problems came the A-0 compiler that Hopper wrote between 1951 and 1952[343]. Robert W. Bemer suggested that if Grace Murray Hopper and the other people working with languages A-0 and A-2 had been given the type of support IBM gave to FORTRAN, they could have gotten a lot farther, much faster with their early start[344].

Computer Generations

Computing machinery advanced quickly from the very beginning; the rapid pace of innovation after 1950 makes it difficult to connect the history of modern computing with what happened before the 1950s. Paul Ceruzzi wrote that the division of computing history into "generations" (see Figure 18 for the author's interpretation of these generations) reinforces the notion that everything that happened before the 1950s was only a prologue to the story[345]. However, nearly all the major players in electronic computing's early years—IBM, NCR, Burroughs, and Remington Rand—

340 Hopper, 1978:p.7. See Knuth, 1998:pp.194-209 for some history and theory of positional number systems, including radix-8 and radix-16.
341 Bright, 1984
342 Hopper, 1978:p.10.
343 Hopper, 1978:p.10.
344 Bemer, 1984
345 Ceruzzi, 1997

had deep roots in the office appliance industry, supplying mechanical or electromechanical equipment to businesses since the late 19th century[346].

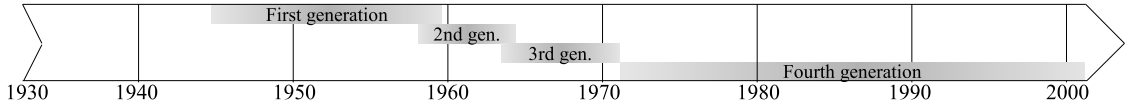


Figure 18: Timeline of Computer "Generations"

From my perspective, dividing computing history into computer generations reinforces the notion that the history of computing equals the history of the computer, which is simply incorrect. Computer generations (Figure 18) are usually segmented as follows: electronic computers (ca.1946-1958), transistor-based computers (ca.1958-1964), the use of integrated circuits (ca.1964-1972), and the use of large-scale integration such as the microprocessor (after ca.1972). The idea of *generations* is based on single technological shifts (which are not revolutions in the sense discussed on page 149 of this thesis).

An approach to the history of computing that would not equate computer history with computer generations would be to take the history of computing as gradual transitions in, for instance, *architectural design*, *programming*, *usability*, *pervasiveness*, *research fields*, *applications areas*, and *modes of operation*. All these factors can be safely included under the title "computing history", because they are interrelated and, to a large extent, inseparable parts of the development of computing. For instance, *pervasiveness* depends on miniaturized architectures, interoperable programming standards, and interactivity (usability); it arises from a number of research fields, such as communications and ubiquitous computing; it has its own application areas (e.g., wearable computing); and it defines new modes of operations (e.g., sentient computing). Pervasiveness also affects other aspects of computing such as systems design (e.g., distributed systems), HCI (e.g., ubiquity and invisibility), communications (e.g., wireless communications), and programming (e.g., protocols). Although there are indeed technological breakthroughs that could warrant the term *generations*, from my point of view small-scale shifts in the different aspects of computing accumulate to be more important in computing than the four large technological shifts called *computer generations*.

346 Cortada, 1993 as quoted in Ceruzzi, 1997.

My concern over writing the history of computing as computer generations is supported by a comparison between the early history of computers and the early history of programming. While computers got faster, programmers did not: They were still writing programs with awkward octal code or call words. The progress in computing machinery could not have been properly utilized without progress in programming methods. The leaps between computer generations are not well-aligned with the leaps in other aspects of computing. Hopper noted that the number of people who were interested in using (the few available) computers to solve problems, but who were unwilling to learn octal code and bit manipulation, grew ever after the 1950s[347]. Hopper stated that in her opinion, things started to already go wrong by the early 1950s:

> [1978] *I'm hoping that the development of the microcomputer will bring us back to reality and to recognizing that we have a large variety of people out there who want to solve problems, some of whom are symbol-oriented, some of whom are word-oriented, and that they are going to need different kinds of languages rather than trying to force them all into the pattern of the mathematical logician.*[348]

Although there were large shifts in computer technology, the shifts in how computers were used were much slower and the development steps in programming languages, coding conventions, utilization of the processing power, and such were not aligned with the computer generations. Dividing computing history into periods marked by the four generations in Figure 18 is a technocentric oversimplification, a choice which promotes a technological deterministic view rather than a social constructionist view.

Real-Time Computers Offer New Prospects

The 1950s marked the construction of the first real-time computer. Even though Project Whirlwind at the Massachusetts Institute of Technology began as early as 1943 with U.S. Navy funding, the system became operational in the beginning of 1950, and was completed in 1951[349]. Designed especially for real-time applications,

347 Hopper, 1978:pp.10-11.
348 Hopper, 1978:p.11.
349 Aspray, 2000; Williams, 1985:pp. 372-378.

such as flight simulation[350], Whirlwind was the first real-time computer in operation, and it incorporated (and yielded) numerous innovations, including magnetic core memory, the graphical display, and the light pen[351]. Whirlwind also served as the prototype for the computers that IBM built for the U.S. government's semiautomatic ground environment (SAGE) system[352]. As a result, Project Whirlwind was separated from the MIT Servo Lab, and absorbed by Lincoln Laboratories, which was funded by the Air Force[353].

Perhaps because real-time computing was born in the vacuum tube era, its birth may escape one's attention as one of the most important technological changes in computing. ENIAC and its successors were used to processing tasks that could have been done using mechanical technology too, and often electronic computers did not offer an advantage over the mechanical punch-card machines in speed or costs[354]. The case was different with Whirlwind: Analog machines were not able to react fast enough for the purposes of Whirlwind[355]. In addition, real-time systems made things possible that were unachievable earlier with batch processing. Batch processing systems are rare today, but real-time systems, of different complexities, are nowadays ubiquitous. The emergence of real-time computers, not the emergence of the electronic computer alone, was able to provide a platform for changes in business and academical practices.

Although it may not be considered a technological revolution (see page 149 of this thesis), Whirlwind is an outstanding example of a combination of academic, military, and industrial co-operation. The experience that IBM obtained from this academy-designed, military-funded project, played a big part in IBM's subsequent commercial line of computers[356], providing it an apparent advantage over other companies. The 24 installed Whirlwinds (the Air Force named them AN/FSQ7[357]) were the central air defense radar system of the U.S. government until the early 1980s[358].

350 Rosen, 1969
351 Campbell-Kelly & Aspray, 2004:pp.144-148; Williams, 1985:pp. 372-378.
352 Williams, 1985:pp. 372-378; Flamm, 1988:p.56.
353 Flamm, 1988:p.56.
354 Campbell-Kelly & Aspray, 2004:p.141.
355 Williams, 1985:p.373; Campbell-Kelly & Aspray, 2004:p.143.
356 Williams, 1985:p.378.
357 "AN" designates a complete system, "F" designates a fixed ground system, "S" designates a special system, and "Q" designates a special or combination purpose.
358 Williams, 1985:p.378.

MIT got steady funding for its research, was provided an IBM Model 704 computer free of charge[359], and became one of the most active groups in the early history of electronic computers[360].

There was another development in machinery in the early 1950s: Whereas EDVAC used bit-at-a-time (serial) arithmetic, the IAS computer, constructed at *Princeton University* with John von Neumann, was a parallel machine[361]. John von Neumann's reports on the IAS (Institute for Advanced Study) computer soon became one of the most important tutorial documents in the early development of electronic computers[362]. The word-at-a-time (parallel) arithmetic that the IAS computer used, gave it an advantage over other computers at the time[363]. The IAS computer executed instructions 40 bits at a time, which the IAS team called a *word.* This is argued to be the first use of the term *word* to describe the aggregate of binary digits that a computer would handle at a time[364]. The IAS design, retaining sequential instruction fetching and instruction executing, but doing arithmetics a whole word at a time, became the norm. The designers of business computers often chose the "serial approach", but by the 1960s, the single-bit-at-a-time design had vanished[365].

Normal Science or Pre-Science?

When speaking about the computing of the 1950s, it seems unsubstantiated to speak about *normal science* in the Kuhnian sense. The discipline of computing did not have a form, its content was debated, and the difference between computing and other disciplines, such as mathematics and electronic engineering, was disputed. The only phase in Kuhn's theory that correctly portrays the computing of the 1950s is *pre-science*. In the pre-science stage of Kuhn's theory, a discipline has disagreeing coteries or competing theories[366]—computing had both. In Kuhn's theory, in this early fact-gathering phase, every candidate for a paradigm is likely to seem equally relevant—in computing there was a number of research directions, none of which

359 Aspray, 2000
360 Rosen, 1969
361 Ceruzzi, 1997
362 Rosen, 1969
363 Rosen, 1969
364 Ceruzzi, 1997 At about the same time, IBM researchers specifically used the concept *externally programmed* to mean that their machine took instructions from a stream of punched cards (Grier, 1996)–the verb *to program* had stabilized within the language of computing.
365 Ceruzzi, 1997
366 Kuhn, 1996:p.13.

had asserted its superiority over the others. Because it is difficult to find evidence for the existence of a well-defined era of normal science in the early development of computer *hardware*, a look at the development of *software* is taken next.

A Level Up In Abstraction: FORTRAN

Before 1954 almost all programming was done in machine language or assembly language[367]. Jonathan Grudin has portrayed this era as "interface as hardware": The typical users were engineers and programmers, and hardware was the central part of the user interface[368]. At that time the term *automatic programming* was used to refer to what are now called *high-level languages* (and *compilers*)[369]. Computer scientist John Backus wrote that automatic programming was not taken seriously mostly because early experiments had been discouraging. This had led the computing community to reason that efficient programming was something that could not be automatized[370]. Although there were doubts about the feasibility of high-level languages, writing machine language or assembly language was laborious, and there was pressure to ease this burden[371].

In addition to the labor-intensiveness of machine language, another influence on the development of the high-level language FORTRAN (FORmula TRANslating system, later also FORmula TRANslator) was the economics of programming in 1954. The cost of hiring programmers was usually at least as great as the cost of the computer itself. Computer scientist John Backus[372] wrote that programming and debugging accounted for as much as three quarters of the cost of operating a computer, and obviously, as computers got cheaper, this situation got worse.

The labor-intensiveness of computing led Backus to propose the FORTRAN Project in late 1953. Another reason for recommending the development of FORTRAN was, in Backus's own words, "*FORTRAN may apply complex, lengthy techniques in coding a problem which the human coder would neither have the time nor inclination to de-*

367 *Machine language* refers to machine instructions that are executable on a specific architecture as such, and *assembly language* (or symbolic assembler) refers to the notation used for making machine language readable.

368 Grudin, 1990

369 Backus, 1981:p.25.; Sammet, 1969:p.13. Note that the term *high-level language* does not refer to the supremacy of these languages over machine language or assembly language but to a higher abstraction level. Note also that also the term *higher-level language* has been used to refer to high-level language (Sammet, 1969:p.1).

370 Backus, 1981:p.26.

371 Campbell-Kelly & Aspray, 2004:p.168.

372 Backus, 1981:pp.26-27. Backus was originally from the field of mathematics.

rive or apply"[373]. From today's perspective it is interesting, as historian of computing Jean Sammet noted, that Backus' group really needed to justify the development of FORTRAN[374]. However, there was plenty of pessimism about whether FORTRAN could ever construct code that was efficient both in time and in space[375].

Even though researchers at the Mathematics Department at Bettis Laboratory (for nuclear power research) were pessimistic about the still-fetal FORTRAN, it produced *correct* code—and the amount of labor required to debug and maintain the code (and even to change it substantively)—was remarkably small[376]. Herbert Bright recalled that one of the researchers had written a program specifying the gamma of tau in FORTRAN in *one afternoon*. Bettis Laboratory researchers estimated that it would have taken about two weeks to have written that amount of code in assembly language and another two weeks to debug it[377]. Bright's recollection reinforces the view that in computer science progress takes place through demonstrations instead of proofs or experiments.

Was FORTRAN a Natural Step or a Contingent Event?

Here Ian Hacking's first sticking point[378], contingency, comes in. The question that the history of FORTRAN poses is, "Was FORTRAN or a FORTRAN-like language a natural, necessary step or was it a contingency without which other modes of programming could have developed?". Compilers as a class existed *before* FORTRAN[379]. There has been a large number of significant conceptual changes *after* FORTRAN. So what indeed is the significance of FORTRAN? The two aspects that need to be discussed concerning the development and standardization of FORTRAN are Hacking's "contingency *vs*. necessity-sticking point" and Hughes' "technological momentum"[380]. The contingency *vs*. necessity debate raises a number of questions, such as: "Were the concepts of programming languages discoveries, as Peter Wegner's choice of words suggests[381], or were they inventions? Did this process display forms of technological determinism or social construction?".

373 Backus, 1981:p.30.
374 Sammet, 1969:p.143.
375 Bright, 1984
376 Bright, 1984
377 Bright, 1984
378 Hacking, 1999:p.68.
379 Bemer, 1984
380 Hughes, 1994

In the time before the standardization of FORTRAN, programming was a skill that few had, and that skill was tied to a few types of computers from even fewer computer companies. A high-level language such as FORTRAN was considered to be just another tool for working with disconnected varieties of computers[382]. My interpretation is that because programming was being done in machine language at the time, there were no structures that would have imposed any conceptual models on programming (except for the concepts dictated by the computer architecture, such as random access, single storage for data and instructions, separation of memory and processor, input and output, and such).

Assembly languages[383] have a very special place among computer languages. Anything that can be done in any high-level language, can be done in assembly language, but not everything that can be done in assembly language, can be done in every high-level language (e.g., direct memory manipulation). Consider the scenario where a researcher would take a programmer who knows only the Java language, a programmer who knows only the C language, a person who has no programming experience, and programmers who know only, for instance, LISP and Prolog, and the researcher would teach all of these programmers the assembly language. It would be interesting to know if their program constructs in assembly language would differ from each other because of the different mental models they may have about programming.

The chaotic state of programming was restrictive in many senses; it barred non-specialists, separated computer brands, and hindered portability[384]. This chaos seems like a technological counterpart of Kuhn's *pre-science* state. Standardization, however, changed the computer world. Martin Greenfield argued that the standardization of FORTRAN brought along with it a stamp of approval for high-level languages[385]. He continued that this approval led to all systems large enough to support a FORTRAN compiler having a FORTRAN compiler. According to Greenfield, a number

381 Wegner, 1976b. However, I would like to suppose that Wegner's sentence "*The 1950's were concerned primarily with the discovery and description of programming language concepts*" is a lapse and not a stand on the contingency debate. The rest of Wegner's article suggests that it is a lapse.

382 Greenfield, 1984

383 Assembly language can be considered the most basic programming language, which is translated (almost) directly to machine instructions. Basically assembly language is a set of human readable (students today might disagree) symbols that correspond to machine instructions. Assembly languages are not dead: They are still needed in some time-critical systems and embedded systems that have very limited resources, as well as in reverse engineering (e.g., virus research).

384 cf. Greenfield, 1984

385 Greenfield, 1984

of key barriers were broken by FORTRAN: (*i*) it took programming out of the hands of the few, (*ii*) programming skills became portable and not tied to a single system, (*iii*) vendors could have compatibility between their models, and all this contributed heavily to the expansion of the computer industry. The shift from machine language programming to high-level language programming still does not constitute a *paradigm shift* in the Kuhnian sense. First, machine language programming was not refuted, but a higher-level alternative for it was built. Second, there were no *anomalies* that would have led to a *crisis*. Although there was clearly a *revolution*, it was not a revolution in the Kuhnian sense. If the shift from machine language programming to high-level languages has to be labeled as a shift in the Kuhnian sense, it is a shift from pre-science to a scientific paradigm.

The making of FORTRAN clearly bears the characteristics of social construction. First, there was the disbelief of many, the optimism of some, and the final acquiescence of the people of IBM[386]. The decisions were dependent upon groups of *opinion leaders*. Second, there were several kinds of "recruitment activities"[387] that were carried out in order to *convince* experts who were open-minded and venturous. Bruce Rosenblatt's recollection that the early user community was small enough to form support groups and attend meetings[388] suggests that there is a parallel between early FORTRAN advocates and Everett Rogers' description of innovators[389]. In Rogers' description, *socially active* gathering of outside influences and the forming of cliques are characteristic of innovators. Third, Peter Wegner argued that FORTRAN is evidence that the model of implementation *in the mind of language designers* may strongly affect the design of the language[390]. That is, a programming language is designed according to how its designer thinks about the computer and its programming. The design of FORTRAN was dependent on the "virtual machine" in the mind of John Backus, who designed FORTRAN. Fourth, one of the strongest motivations was money: John Backus, the FORTRAN project leader, made the case for FORTRAN mainly on *economic grounds*[391].

386 Sammet, 1969:p.144; Bright, 1984; Wegner, 1976b; Campbell-Kelly & Aspray, 2004:p.170.
387 Rosenblatt, 1984
388 Rosenblatt, 1984
389 Rogers, 2003
390 Wegner, 1976b
391 Campbell-Kelly & Aspray, 2004:p.169.

But as FORTRAN got standardized and as it diffused well, it also started to institutionalize and become rigid. Rosenblatt called this phenomenon *setting up defenses* and *perpetuation*[392]. FORTRAN was taught in colleges, and taken as the notation system for many scientific publications. It seems that as the language got more widespread the more deterministic characteristics it assumed. Since large libraries of technical and scientific programs and subroutines were written in FORTRAN, it was fairly natural to build FORTRAN compilers for all of the new machines because they were introduced to the scientific and technical industry[393]. FORTRAN was constructed from a fairly open (i.e., flexible) set of needs, designed according to the views of a small number of people, and as FORTRAN gained technological momentum, it developed the characteristics of technological determinism. It achieved such an institutional status that it is even today regarded as the lingua França of scientific computing[394]. This is a prime example of Thomas Hughes' technological momentum in computing. (Note, however, that although FORTRAN is the origin of a large number of programming languages, there are plenty of languages that are not descendants or relatives of FORTRAN at all, such as LISP, APL, and ALGOL[395].)

ALGOL: The Ideal Language

Jean Sammet wrote that the years 1958 and 1959 were the most influential and prolific in the development of programming languages. She claimed that no other two-year period contained developments that had as much long-range significance[396]. Sammet attributed the convergence of so much significant work in those particular years to a growing interest in software in general and programming languages in particular. From the interface design-perspective, the reason for such enthusiasm is clear: users were freed from having to know about the hardware[397].

In 1957 the ACM and the GAMM (Gesellschaft für angewandte Mathematik und Mechanik) appointed a committee to study, and recommend action for, the creation

392 Rosenblatt, 1984
393 Rosenblatt, 1984
394 Campbell-Kelly & Aspray, 2004:p.169.
395 It is a matter of definition whether ALGOL is a descendant of FORTRAN or not. For instance, Jean Sammet does not mention such a relationship (Sammet, 1969).
396 Sammet, 1991
397 Grudin, 1990

of a "universal" programming language (in a GAMM meeting Peter Naur[398] had proposed an algorithmic language-machine code translator as early as 1951[399]). Soon IFIP's (International Federation for Information Processing) technical committee 2 (TC2) founded a working group on ALGOL (WG2.1)[400], supported by USE (UNIVAC Scientific Exchange), DUO (Datatron Users Organization), and SHARE[401].

Alan Perlis (1922-1990)[402], who was one of the people active in designing ALGOL (ALGOrithmic Language), wrote that at the time it seemed that every new computer and even each programming group was spawning its own algebraic language or cherished dialect of an existing one[403]. There is one part of the GAMM-ACM letter that particularly stands out to me: Perlis wrote, "*The situation would not be improved by the creation of still another nonideal language.*" I do not know the exact meaning of this phrase, but apparently (some of) the researchers of the GAMM and the ACM believed that an ideal language could be developed. However, the opening speaker of the GAMM meeting had already urged the group not to waste the precious days in a search for the perfect language and to heed Wittgenstein's observation that all too often "the best is the enemy of the good"[404]. Today there is no widespread delusion of an ideal language: it is accepted that different types of language suit different needs [405].

ALGOL grew out of this GAMM-ACM joint effort[406]. ALGOL was sophisticated, and it also met two of the three initial objectives set by the ALGOL committee[407]. It was (1) possible to use it for the description of computing processes in publications ("publication language"), and it was (2) mechanically translatable into machine programs ("hardware representations")[408]. However, the third objective, that ALGOL should be

398 Peter Naur is a pioneer in the areas of software engineering and software architecture.
399 Naur, 1981:p.93.
400 Lindsey, 1996 in Bergin & Gibson, 1996.
401 SHARE was allegedly the first organization of computer professionals, formed around the IBM 704 scientific computer. SHARE is an organization to distribute and share software for free. SHARE is not an acronym: According to an early SHARE manual, the founders of the organization chose the name SHARE, hoping to find suitable words to match the initials, but "nobody was really that smart", and so "each member is free to interpret the initials in his [or her] own way" (Edson et al., 1956:p.01.01-02).
402 Perlis' BSc degree was in chemistry and MSc and PhD degrees in mathematics.
403 Perlis, 1981:p.76.
404 Perlis, 1981:p.78.
405 Denning, 2003
406 Sammet, 1969:pp.173-174.
407 Naur et al., 1960
408 Naur, 1981:p.113; Sammet, 1969:p.175.

as close as possible to standard mathematical notation and readable without further explanation ("reference language"[409]), was unattainable. Peter Naur noted that the part of the notation of ALGOL that was similar to standard mathematical notation was confined to simple arithmetic and Boolean expressions[410].

ALGOL was the pet child of a number of the world's top computer scientists of the time. Although there was a clear and common vision of the purpose and general form of ALGOL, its details were grounds for fervent debate and criticism. In his October 1967 *CACM* article, "The Remaining Trouble Spots in ALGOL 60", Donald Knuth wrote that if any other language comparable to ALGOL would undergo such detailed scrutiny as ALGOL had undergone, it would be impossible to publish the list of trouble spots because the number of trouble spots would be a full order of magnitude greater[411]. Knuth finished his article by writing, "*The author* [Knuth] *has tried to indicate every known blemish in* [the ALGOL report[412]]*; and he hopes that nobody will ever scrutinize any of his own writings as meticulously as he and others have examined the ALGOL report*"[413]. A massive intellectual effort was invested in developing ALGOL to be the ultimate programming language.

ALGOL 60 had a significant effect on the field of programming language design in several ways[414]: (1) ALGOL was a language suitable to be used as a programming language and *independent of any implementation*; (2) ALGOL was suitable for the expression of algorithms in pure *human-human communication*; (3) as a side product of ALGOL, a *new style of language description* was demonstrated (known as BNF, Backus-Naur Form[415]); (4) ALGOL was a demonstration of a combination of *generality and economy* of concept as well as *precision and description*; and (5) ALGOL included a *number of novel ideas*, especially in the areas of blocks and procedures.

409 Sammet, 1969:p.175.
410 Naur, 1981:p.113.
411 Knuth, 1967
412 Backus et al., 1963
413 Knuth, 1967
414 Naur, 1981:p.93.
415 See Backus, 1959. Originally it was called "Backus Normal Form", but Donald Knuth (Knuth, 1964) pointed out that calling it "Backus Naur Form" would have several advantages: (1) it gives proper credit to both developers Backus *and* Naur; (2) it preserves the established abbreviation BNF, and (3) it does not call "Form" a "Normal Form". The name *Backus Naur Form* replaced Backus Normal Form soon after Knuth's note (Knuth, 1964).

Why Did FORTRAN Do Better Than ALGOL?

The limited diffusion of ALGOL, which was carefully designed to be a superior language to FORTRAN, into the programming world at large is an indicator that by the time ALGOL was ready to be introduced, FORTRAN had already become an institution. Although ALGOL was designed and backed up by a large number of the most prominent figures in the field of computing[416], and it was endorsed by, for instance, *Communications of the ACM* (*CACM*), it never gained ground in programming practice at large. Because IBM did not want to add ALGOL support to FORTRAN, ALGOL never really had the chance to become the standard language for IBM machinery[417] (recall that IBM was the leading computing corporation of the time). Yet, ALGOL became the base of reference for much of the subsequent programming language development[418].

Although ALGOL was, since February 1960, the publication language used in *CACM*, for most people in the 1960s the only way to check the correctness of *CACM* algorithms in practice was to translate them into FORTRAN because relatively few installations in the U.S. at that time had ALGOL compilers[419]. In the vocabulary of Thomas Hughes[420], as FORTRAN gained *technological momentum*, it became more a shaper of its environment than shaped by it. ALGOL was not able to gain similar momentum and, under the circumstances, did not become a similar shaper of computing practice. Whereas FORTRAN was the most important *practical* milestone in programming language development, ALGOL has been proposed as being perhaps the most important *conceptual* milestone[421].

At the end of the decade, business computing finally got its own language. In 1959 a number of representatives from user groups, government, computer manufacturers, and other interested parties formed the COBOL (COmmon Business Oriented Language) language specifications[422]. The CODASYL[423] Committee behind COBOL was the first *intercompany* committee in computing, which consisted primarily of competing computer manufacturers and two government agencies (the U.S. Air Force and the

416 See, for instance, Sammet, 1969:pp.172-196.
417 Perlis, 1981:p.83.
418 Perlis, 1981:p.90.
419 Sammet, 1969:pp.176-177.
420 Hughes, 1994
421 Wegner, 1976b
422 Sammet, 1969:p.330.
423 COnference on DAta SYstems Languages

Navy)[424]. Captain Grace Murray Hopper noted that, in fact, it was the U.S. Navy that ordered her to work with the first computer, that the Navy made it possible to develop COBOL, and that the Navy made it possible for her group to develop the COBOL test routines[425].

Although COBOL entailed a number of new concepts; such as natural language style programming, record data structures, and file description and manipulation facilities[426]; COBOL is not discussed here further. Its origins do not bring in any essentially new characteristics to the birth of programming languages.

A large number of different languages have been developed over time. Figure 19 shows a timeline of the creation of a number of influential languages. Note the long period between FORTRAN and any of its competitors.

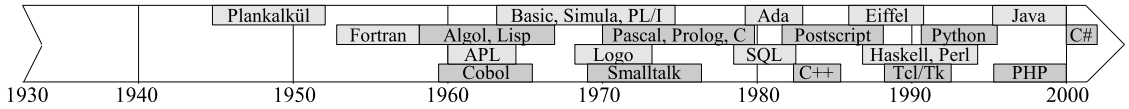


Figure 19: Timeline of Programming Languages

FORTRAN had a good head start on other high level languages. The reason for the difference between the impacts of FORTRAN and ALGOL is clear. FORTRAN was a response to a number of needs that arose from practice, but ALGOL was planned devoutly, for a long time, by a large number of the most prominent people in the field, and retrospectively, it seems that it was more of an object of study than a solution itself. The participants of this "great programming language debate" were at first called ALGOL lawyers and later ALGOL theologians[427].

The construction of ALGOL is an example of the social construction of technology par excellence. Whereas FORTRAN was designed to meet needs rooted in practice (descriptive aspects), the designers of ALGOL had clear intellectual intentions that were agreed upon together (normative aspects). The designers wanted it to be a publication language that would be as close as possible to mathematical notation and wanted it to be readable without further explanation. However, generally speaking, both of the languages are a result of specific sets of needs that have affected the design and implementation of those languages.

424 Sammet, 1969:p.330.
425 Hopper, 1978:p.20.
426 Wegner, 1976b
427 Wegner, 1976b

When one looks at the programming languages used during the past fifteen years, it is notable that FORTRAN has survived while many others have disappeared. Certainly, it was the first high-level programming language to come into common use, but there are also other reasons for the persistence of FORTRAN. First, Bruce Rosenblatt wrote that one reason for the survival of FORTRAN was that IBM was farsighted enough to put support people in the field; the users could easily get support[428]. Second, FORTRAN III, which used in-line assembler language within FORTRAN code, was not released[429]. So the language remained machine-independent.

Third, Rosenblatt continued that FORTRAN was (and is) easy to use; the self-teaching course that IBM gave took about 20 hours—about a fourth of the time that was estimated to teach or learn COBOL or PL/I. Soon after FORTRAN's introduction, it was also used as a notational method in many scientific publications[430]. Fourth, the synergistic, snowballing effect of users helped to build large, shared libraries of programs and subroutines that could be quickly put on new machines as they were introduced, providing those new machines had a FORTRAN compiler. Thus the FORTRAN compiler was a standard program. Fifth, and probably the most important thing about FORTRAN has been its adaptability. Subroutines, on-line computer programming, operating systems, structured programming, and object-oriented programming have all been successfully added to successive versions of FORTRAN.[431] The latest version of FORTRAN at the moment is Fortran 2003[432].

Significant Factors in Language Development

Since programming languages do not appear spontaneously, a major factor in their conception are the people and organizations who bring them into existence[433]. Jean Sammet named five significant social, institutional, and human factors in language development[434]: First, (1) there are *organizations*, such as profit making companies and universities; second, (2) an organization, as such, does not create anything, but *people* do; third, (3) language development depends not only on the employers of the individuals doing the work, but also on *sponsors* (e.g., CODASYL for COBOL and

428 Rosenblatt, 1984
429 Backus, 1981; Rosenblatt, 1984
430 Rosenblatt, 1984
431 Rosenblatt, 1984
432 Note the writing convention change from FORTRAN to Fortran between FORTRAN 77 and Fortran 90.
433 Sammet, 1991
434 Sammet, 1991, underlining added.

IFIP TC2 for ALGOL 68); fourth, (4) for languages developed by individuals, such as Pascal, it is worth taking *an individual's background* into consideration; and fifth, (5) a significant factor in language development is the *motivation* of the organization or individual. Figure 20 shows an outline of the factors discussed here, with a few samples of each of them.

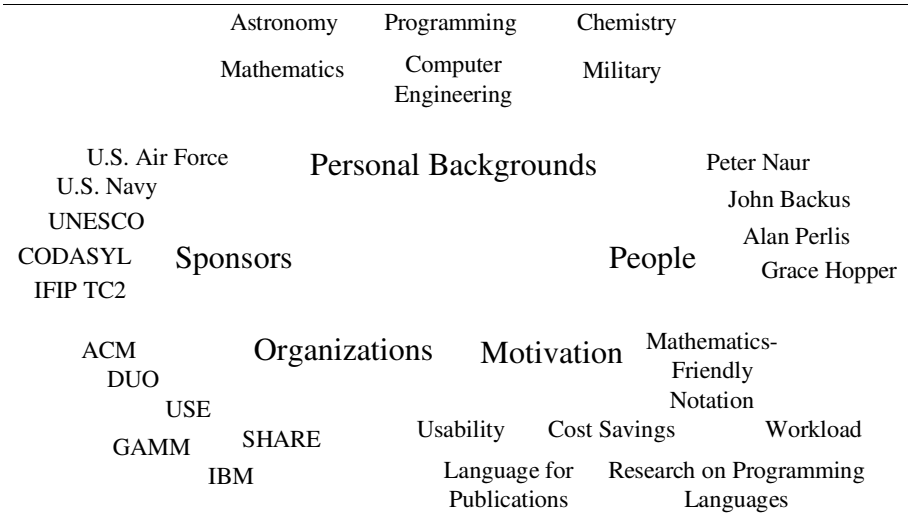


Figure 20: Some Factors in Language Development

In Figure 20, there are four people who played a major role in the early development of programming languages. They have been selected for inclusion in Figure 20 because they appear in this section. The examples of aspects of personal backgrounds in Figure 20 concern only the professions of people involved, but a thorough understanding of a person's background would require understanding that person's socio-economic status, education, political and ideological leanings, upbringing, home culture, significant happenings in the person's life, motivations, and other things that may play a part in one's actions and behavior. Sponsors include, for instance, UNESCO, which sponsored the conference in which the BNF was introduced and the open ALGOL debate appeared[435]; IBM, which was a sponsor of FORTRAN; and the U.S. Navy, which was active in creating COBOL. Organizations include, for instance, SHARE and USE organizations, which established ALGOL working groups[436], and the ACM, which was very active in enforcing ALGOL. Motivations include, for instance, the above-mentioned motivations driving ALGOL, such as the need for a publication

435 Sammet, 1969:p.175.
436 Naur et al., 1960

language, programming language research, and a notation close to mathematics, as well as the cost-saving issues that drove the creation of FORTRAN and COBOL.

Sammet wrote that in examining programming languages, one should also look at the different types of uses and functionalities that each language was designed to incorporate[437]. She noted that it sometimes occurs that a new functionality is often more a matter of style and personal preference than meeting real needs that had not been met before. In fact, Sammet listed personal needs and functional needs as the two *most important incentives* for programming language development.

By the end of the 1950s the number of uses for computing had grown rapidly. Computing speeds grew and high-level languages enabled non-devoted specialists to utilize computers for their business, research, statistical, or other purposes. There was no such term as *computer scientist*, though. People who worked with computers were mathematicians, physicists, electronic engineers, military officers, specialists from different businesses, and scientists. However, there was an increasing number of scientists whose primary subject area was computers, and there was an increasing number of topics that did not belong clearly to any other science (topics such as computability, algorithms, grammars, automata, modern cryptography, artificial intelligence, and so forth). Computer scientists as a group did not belong clearly to any existing discipline, but there was no disciplinary identity for computing professionals either. In the next section, the difficulties of forming a new discipline are examined.

437 Sammet, 1991

Section Overview

I began this section by noting that I would offer my interpretations about the sociocultural influences in the development of early computing; about the interrelations between the different technological, institutional, professional, and social aspects of computing; about the possible contingencies in the development of computing; and about the technological momentum of computing technology. In this subsection I summarize my interpretations of the development of early computing technology.

The role of interdisciplinarity was crucial in the shift from electromechanical computation to electronic computation[438]. I suggested that a combination of mutually incommensurable crafts and sciences creates an ontological, epistemological, and methodological anarchy, which inhibits dogmatism and thus allows for unexpected directions of development. In addition to interdisciplinarity, a techno-enthusiastic culture was one of the main reasons for the U.S.'s success in terms of the development of early computing machinery[439]. Culture has been argued to affect the amount of funding a field gets, the valuation of theory and practice, the foci of research, the popularity of technological disciplines, and public support[440], which are all central to development of a field.

Historians of computing note rather unanimously that the best people in the field of automatic computation became associated with the Moore School because of the U.S. Army's increased need for automatic computation, which was due to the Second World War[441]. The U.S. Army took a gamble on an untested technology that no private investor would have taken in a normal situation—especially when the untested technology was opposed by the established scientific community[442]. Note that the scientific community has not always been a facilitator of progress; for instance, the scientific establishment resisted ENIAC, which is now considered to be one of the most important milestones in the history of electronic computing. Interestingly,

438 For instance, Puchta, 1996; Williams, 1985:p.209; Bowles, 1996; Naur, 1992:pp.596-597; and Campbell-Kelly & Aspray, 2004:pp.52-59, have noted a number of fields that have contributed to the development of early computing.

439 Bowles, 1996

440 Campbell-Kelly & Aspray, 2004:p.19; Flamm, 1988:p.136; Bowles, 1996; Aspray, 2000.

441 Marcus and Akera, 1996; Flamm, 1988; Campbell-Kelly & Aspray, 2004; Williams, 1985

442 Marcus and Akera, 1996; Pugh and Aspray, 1996; Winegrad, 1996; Croarken, 1992

economic aspects did not seem to play a decisive role in the birth of electronic computing—human "computers" were cheap to hire, and electronic computers were developed despite the costs, rather than their potential for cost-saving[443]. Two wars—the Second World War and the Cold War—fueled an unprecedented scientific mobilization of the U.S., which in turn resulted in massive investments, innovations, and developments in high-technology[444].

The work headed by Eckert and Mauchly (and von Neumann) shows characteristics of *the mangle of practice*: When the ENIAC was being built, the developers of ENIAC had (1) a theory of computation (Turing's formalization of computation[445]), (2) a theory of how electronic computers should work, and (3) ENIAC itself. During the construction of ENIAC both the theory of how electronic computers should work and ENIAC itself were revised and revamped many times, but finally Eckert, Mauchly, and von Neumann came up with the concept of the *stored-program computer*[446] (However, the stored-program computer took several more years and a number of modifications of the original plan before it became complete).

After the Second World War, the stored-program idea and EDVAC designs were, against all odds, declassified[447], which meant that academics around the world were able to build on the work done at the Moore School. In Britain, the example of EDVAC, combined with favorable social and economic conditions as well as risk-taking, led to the construction of the first stored-program computer, EDSAC, and the world's first office computer LEO I[448]. Because there was great uncertainty about the research directions and paradigms of electronic computing, the first twenty years of electronic computing saw a great variety of fundamentally different competing technologies[449].

My interpretation is that in the early years of computing, computing technology did not yet bear the characteristics of technological determinism, but that there was an uncertainty about research directions and design practices. Most computing ma-

443 Campbell-Kelly & Aspray, 2004:p.141.
444 See Flamm, 1988:p.2. However, Aspray denied that World War II had a uniformly *positive* effect on computing in the U.S. (Aspray, 2000).
445 Turing, 1936
446 Campbell-Kelly & Aspray, 2004:pp.76-83.
447 Pugh and Aspray, 1996; Williams, 1985:pp.287-296; Winegrad, 1996.
448 Aris, 2000; Ferry, 2003; Land, 2000; Williams, 1985:p.333.
449 Williams, 1985

chines that were built during the first ten years of digital computing were different from each other in their architecture, design, constraints, or working principles. It seems that computing technology gained technological momentum quite slowly, and it is my interpretation that only the birth of the first unified computer architecture, the IBM System/360 in 1964[450], was a clear signal of technological determinism in the computing machinery.

The IBM System/360 series introduced a computer architecture that was common to a family of computers—computers from small business machines to large scientific machines were run on the same operating systems, using one set of software[451]. Before the System/360, all computers ran different software packages, so compatibility was hard to achieve. Campbell and Aspray noted the following gearing effect: Given m different computer models, each requiring n different software packages, a total of $m \times n$ programs had to be developed and supported before the unified System/360[452].

The American universities that entered the field of computing earlier than other universities gained some advantages, but also some disadvantages, because of their early entry[453]. The studies of the competition between Harvard, Princeton, MIT, Columbia, and University of Pennsylvania suggest that success and failure in the field of computing resulted from a combination of military-industrial-academic relationships, institutional decisions, academic reputations, interpersonal skills, university policies, politics, and geo-economic location[454]. Also, the quality of leadership affected the success of competing research groups[455]. And because there was no common understanding of how computers should be built, pioneering groups faced a number of technological and theoretical dead-ends that the groups that followed were able to avoid[456].

The first programming languages were shunned by the computing establishment but eventually some people, most notably Grace Hopper and John Backus, succeeded in

450 Campbell-Kelly & Aspray, 2004:pp.122-129; Silberschatz et al., 2002pp.801-802.
451 Silberschatz et al., 2002pp.801-802.
452 Campbell-Kelly & Aspray, 2004:pp.122-129.
453 Aspray, 2000
454 Marcus and Akera, 1996; Aspray, 2000; Williams, 1985:pp.242, 299-301; Flamm, 1988:pp.50-62; Campbell-Kelly & Aspray, 2004:pp.100-101; Pugh and Aspray, 1996; Winegrad, 1996; Cohen, 1998.
455 Williams, 1985:pp.242, 350-353; Campbell-Kelly & Aspray, 2004:p.64; Aspray, 2000.
456 Aspray, 2000

selling the concept of programming languages to administrative, managerial, and technical people by arguing that programming languages would result in economic savings[457]. Although the creation of FORTRAN had the characteristics of social construction[458], FORTRAN gained technological momentum as it matured, and it soon turned into an institution itself. FORTRAN was followed by programming languages much more elegant than FORTRAN, but the newcomers were too late: FORTRAN became the *de facto* standard of computing for a long time[459].

Although high-level languages changed the face of computing, the shift from machine-language programming to high-level language programming constitutes neither a revolution nor a beginning of a paradigm *in the Kuhnian sense* (see pp.-241). I argue that the shift from an era of pre-science to a scientific paradigm happened with the advent of stored-program computers. This paradigm might be called, for instance, the *stored-program-paradigm* (not the "von Neumann-paradigm" because a historically accurate name would also have to acknowledge the influences of Eckert, Mauchly, and Turing). The new *programming paradigms* such as object-oriented programming are not paradigms proper (in the Kuhnian sense), because new programming paradigms do not render pre-existing programming paradigms obsolete. If anything, new programming paradigms should be called new programming viewpoints or new programming approaches. If there is a programming paradigm in the Kuhnian sense of the word *paradigm*, it is the *stored-program-paradigm*, and that paradigm has persisted for almost sixty years now. If there is to be a *paradigm shift* in the Kuhnian sense in programming languages, the new paradigm will not merely offer another perspective of the stored-program-architecture, but it will *replace* the stored-program-architecture or make all other programming languages redundant.

A number of non-technological factors can be attributed to the development of early electronic computing. The factors that have been found to steer the development of computing include aspects of *organizations* (institutional sponsors, profit-making companies, universities, government branches), *people* (characters, individual's backgrounds, motivations—as well as quarrels, visionaries, opinion leaders, reputa-

457 Sammet, 1969:pp.4, 143; Bright, 1984; Backus, 1981:pp.26-30.
458 Sammet, 1969:p.144; Bright, 1984; Wegner, 1976b; Campbell-Kelly & Aspray, 2004:p.170.
459 Rosenblatt, 1984; Campbell-Kelly & Aspray, 2004:p.169; Sammet, 1969:pp.176-177; Wegner, 1976b.

tions, contacts), *contingencies* (security lapses, coincidental convergences of common interests, misunderstandings, the snowball effect), *interdisciplinarity* (electrical engineering, mathematics, logic, theoretical and experimental physics, weapons research), and *culture* (techno-enthusiasm, practicality, techno-utopianism, risk-taking, political situation, regulations, policies, élitism).

Although the above-mentioned non-technological factors can be attributed to the development of computing, every source text in the history of computing included in this thesis also discusses technological or theoretical aspects of computing. No matter what motivations, interests, or contingencies have affected the development of computing, the technological or theoretical aspects of computing affected the development of computing too. However, it seems that sociocultural factors had a stronger impact in the early development of electronic computing than in the later developments. It can be said that when computing as a field matured, computing technologies gained technological momentum.

The early history of electronic computing shows two signs of an increase of technological momentum, that is, two signs of a shift from social construction towards technological determinism. As knowledge about computing grew, computing systems also became more complex and more interdependent. Firstly, in the early days of computing there was uncertainty about the directions of computing, but as knowledge about computing accumulated, researchers increasingly followed the path set by earlier researchers. This is traditionally called the *growth of knowledge*. Secondly, early designers in the field of computing were able to start with a *tabula rasa*, but as the number of computer installations grew, the design decisions of computing had to be increasingly based on pre-existing systems. This is, using Hughes' terms, an example of an increase in *technological determinism,* or a growth of *technological momentum*[460].

460 Hughes, 1994

3.4.The Creation of a Discipline

In our early days we were, in part, evangelists with a message about computing machinery. Whatever our own contributions may have been, the gospel is certainly widespread at the present time.[461]

The history of computing as an academic discipline and as a profession is not shorter than the history of electronic computing itself. The first societies for professionals in automatic computing were founded at the same time that ENIAC was unveiled. It is difficult to trace the earliest discussions of computing as a distinct discipline, but in the April 1958 issue of *Communications of the ACM* (*CACM*) the editors of DATA-LINK asked the editor of CACM,

> [1958] *What is your reply when someone asks your profession? Computing Engineer? Numerical Analyst? Data Processing Specialist? To say "Computer" sounds like a machine, and "Programmer" has been confused with "Coder" in the public mind (if your particular segment of the public knows what you are talking about at all!)*[462]

The editors of DATA-LINK then asked for suggestions for the name of the discipline, noting that a brief, definitive, and distinctive name would help *the profession* to be widely recognized (yet they do not further specify *the profession*). After these early attempts to characterize the discipline of computing, there has been a dizzying amount of debate about the form and content of the field. In this section, I take a look at the changes in computing as a discipline since the 1958.

The reason for tracing the changes of computer science throughout its disciplinary history is that it is necessary for a philosopher, sociologist, or historian of science to make a distinction between and understand a number of different aspects of science and technology[463]. In social studies of computer science changes in computing are situated in their scientific, technological, social, historical, cultural, linguistic, political, economic, institutional, personal/individual, and other sociocultural contexts. In this section I discuss the effect of those contexts on the development of computer science.

---

461 The president of the ACM, Alston S. Householder, in his 1955 presidential address to the ACM, published in the January 1956 issue of *Journal of the ACM* (Householder, 1956).

462 The editors of the DATA-LINK (Los Angeles ACM Chapter Newsletter) in Letters to the Editor, April 1958 issue of *Communications of the ACM*.

463 cf. Bunge, 1998:p.407.

The subsections in this section are arranged chronologically. The accounts of computing as a discipline are, for the most part, arranged chronologically. In this section, all full quotations begin with the year of the quotation in brackets (e.g., [1976]). This convention is adopted to help the reader with the timeline of this section. The first subsection discusses the birth of computing as an academic and professional field; the second subsection discusses the diversification of the field; the third subsection discusses recent definitions of the field; and the fourth subsection characterizes and discusses the methodologies of computer science.

The focus in this section is not explicitly on computing in the U.S., although implicitly there is an American bias because most of the citations are from U.S.-based publications, such as *CACM* and *IEEE Computer*, and because the language in this dissertation is English. Then again, it can be argued that the U.S. has been central to the development of computing from the 1950s onwards. In addition, the ACM and the IEEE Computer Society are the two largest societies for professionals in the field of computing, and *CACM* and *IEEE Computer* are the flagship publications of these societies[464]. Other journals that have been used as sources include *Science*, *American Mathematical Monthly*, *Journal of the ACM*, *Theoretical Computer Science*, *Datamation*, *BIT*, plus a number of other journals and books.

Most of the articles included are written by computer scientists, but the target audiences vary. Although many of the articles are addressed to computer scientists, some are addressed to mathematicians[465], and some to the whole scientific community[466].

Because curricula specify what competent or qualified professionals should know and what they should be able to do, official curricula offer one viewpoint of the discipline at each era. Therefore I include a number of academic computing curricula in the sources in this section. It should be noted that the curricula included are writ-

464 *Communications of the ACM* writes, "Communications of the ACM is the flagship publication of the ACM and one of the oft-cited magazines in the computing field". (`http://www.acm.org/pubs/cacm/about_cacm/`) (accessed September 27th, 2006). *IEEE Computer* writes that it is "The flagship magazine of the IEEE Computer Society". (`http://www.computer.org/computer/`) (accessed September 27th, 2006). The website of *CACM* reports the readership to be about 85,000 and the website of *IEEE Computer* reports its readership to be about 86,000.

465 Knuth, 1974; Dijkstra, 1974.

466 Newell et al., 1967

ten by computer scientists, not scholars in the field of education, and that I look at the various curricula as a computer scientist, not as an educator.

The reader should take note of the bias that the selection of source material causes. Although the publications above belong to the most oft-quoted and most widely circulated journals and magazines in the field, it is not certain if they are representative of the discussion in the field at large. For instance, the readership of the publications above is highly educated, and just belonging to their readership indicates an interest in certain computing-related issues[467]. The reviewing process creates a double bias in the publication of articles. Firstly, some screening-out may take place implicitly because the authors know that there is a double-blind review process, and secondly, the review process itself often follows a preordained policy set by the publication board. This double bias is implicit in Langon Winner's criticism of social constructionism—From Winner's point of view it would be an error to take the published, public discussions and other social activities of "relevant actors", analyze them, and argue that one has unpacked or uncovered the whole phenomenon[468].

Winner argued that the visible, surface phenomena are just the tip of the iceberg. He wrote that focusing on "relevant actors" often disregards those dynamics of technoscientific change that are not revelead by studying surface (public) phenomena[469]. However, I am not offering a comprehensive picture of the development of computer science but I am taking a number of interesting cases that demonstrate some aspects of the debate about computer science. In sampling terms, my choice of the articles is a purposive sample: I have attempted to select an array of interesting accounts of computing as a discipline, and my aim is to offer a good variety of viewpoints of computer science.

467 For instance, IEEE Computer and CACM report that about 65% of their readers hold at least a M.Sc degree.
468 Winner, 1993
469 Winner, 1993

Struggling for Status

Programming is one of the most difficult branches of applied mathematics; the poorer mathematicians had better remain pure mathematicians.[470]

IN THIS SECTION:

- ✔ How was computing as an academic and professional field born?
- ✔ What controversies were there in the early years of the field?
- ✔ How was computer science as a discipline born?
- ✔ What were the main controversies in the debate about computing as a discipline?

The same year ENIAC was unveiled, 1946, the first society for computing professionals, the *Subcommittee on Large-Scale Computing of the American Institute of Electrical Engineers* (AIEE) was founded[471]. Five years later, the *Institute of Radio Engineers* (IRE) formed its *Professional Group on Electronic Computers*. Then, in 1963 the AIEE and IRE merged, forming today's *Institute of Electrical and Electronics Engineers*, the IEEE. This merging also led to the *IEEE Computer Society*. The *Association for Computing Machinery* (ACM) was founded 1947, and the specialization of IEEE and ACM emerged early in their history: The IEEE Computer Society focuses on standards (IEEE) and hardware, whereas the ACM focuses on theoretical computer science and applications[472]. Although the early professional societies of AIEE, IRE, and ACM were clearly focused on computing, it is difficult to say if the early professionals in those societies considered themselves primarily as *computing professionals* or perhaps engineers or mathematicians who focus on computing. For instance, in some of the articles in the early issues of *Journal of the ACM* (*JACM*) the professional identity of the people who work with computing is still unclear[473].

Journal of the ACM was first published in 1954, and in the inaugural issue the president of the ACM, Samuel B. Williams, sketched the development of ACM[474]. Williams noted that the membership of the *Eastern Association for Computing Machinery,* renamed *ACM* in January 1948, had grown quickly from a 78-member in-

470 Dijkstra, 1975b
471 Wood, 1995
472 Williams, 1954; Householder, 1956
473 See Williams, 1954; Householder, 1956; Householder, 1957; Carr, 1957.
474 Williams, 1954; a more thorough discussion on the early history of the ACM can be found in Alt, 1962.

formal group of people interested in computing (May, 1947) into a 1200-member international group of people (January, 1954). In the early years of the ACM there were already debates about whether the ACM should concern itself with hardware or theory[475]. *Communications of the ACM* was established in 1958 to be a forum for timely information in the field, whereas *Journal of the ACM* was for articles less temporal by nature[476].

Who Is a Computer Professional?

In the early 1950s, the disciplines that are known today as computer science and software engineering existed only as a loose association of institutions, individuals, and techniques[477]. According to Gibbs and Tucker, early academic programs in computer science (even though they were not called that yet) were of two types[478]: One type of academic programs involved short *non-credit programming courses*, offered by college and university computer centers, for students in the scientific disciplines. The other type of academic programs were *graduate programs* in computer science, which emerged in selected large universities that had significant research interests in computing. The programs offered were diverse in content and standards.

Since the late 1950s[479] there has been debate over the qualifications required of a computing (or computer) professional. The questions asked are, for instance, "Who qualifies as a computing professional, with whose accreditation, having exactly how much experience, knowing which areas, having what kind of skills, and belonging to which organization?". The seemingly straightforward debates about professionalism in the diverse field of computing are actually complex confrontations between various actors[480]. When the stored-program digital computers (programmable computers) emerged in the early 1950s[481], the debate was whether computing is a respectable scientific field or if it is an applied science—a branch of engineering. For example, Edsger Dijkstra stated in his 1972 Turing Award Lecture,

---

475 Householder, 1957
476 Bauer et al., 1959
477 Ensmenger, 2001
478 Gibbs and Tucker, 1986
479 Ensmenger, 2001. See, e.g., Denning, 2004; Lethbridge, T.C., 2000; Pour et al., 2000 of recent debate
480 Ensmenger, 2001
481 Although the first stored-program computer to be conceptualized was EDVAC, described by von Neumann in 1945, it was not operable until 1952 (Aspray, 2000). EDSAC and BINAC were both run in 1949.

> [1972] [In 1952] *I had to make up my mind, either to stop programming and become a real, respectable theoretical physicist, or to carry my study of physics to a formal completion only, with a minimal effort, and to become ..., yes, what? A programmer? But was that a respectable profession? After all, what was programming? Where was the sound body of knowledge that could support it as an intellectually respectable discipline?*[482]

Dijkstra's statement can, in part, explain his lifelong aspirations to formalize computer science. If Dijkstra held that computer science is second to theoretical physics because computer science lacks the theoretical, formal body of knowledge that physics has, then it is only natural that Dijkstra aspired to elevate the status of computer science by formalizing the core knowledge of computer science.

During the last years of the 1950s, the terminology in the field of computing was discussed in the *Communications of the ACM*, and a number of terms for the practitioners of the field of computing were suggested: *turingineer*, *turologist*, *flow-charts-man*, *applied meta-mathematician*, *applied epistemologist*[483], *comptologist*[484], *hypologist*[485], and *computologist*[486]. The corresponding names of the discipline were, for instance, *comptology, hypology*, and *computology*. Later Peter Naur suggested the terms *datalogy*, *datamatics*, and *datamaton*[487] for the names of the field, its practitioners, and the machine, and recently George McKee suggested the term *computics*[488]. None of these terms stuck, but I take the discussion about the name of the field as a clear indicator that by the turn of the 1960s the search for a disciplinary identity had begun.

Early Definitions

As the field matured, and as model curricula and textbooks were developed, the programs at various universities increasingly came to resemble one another[489]. Although a common agreement on the content of the field developed, throughout the

---

482 Dijkstra, 1972
483 Weiss & Corley, 1958
484 Correll, 1958
485 Zaphyr, 1959
486 The December 1958 issue of DATA-LINK attributes the term *computology* to Edmund C. Berkeley. The editors write, "*We like the name "Computology" for our profession, as suggested by E.C. Berkeley in the November issue of COMPUTERS AND AUTOMATION. A member of the profession would therefore be a "Computologist."*"
487 Naur, 1966
488 McKee, 1995
489 Aspray, 2000

1960s computer specialists continued to wonder at the "*almost universal contempt*"[490] (or at least "*cautious bewilderment and misinterpretation*"[491]) with which programmers were regarded by the general public[492]. If there was an image of computing, it seems to have been somewhat negative[493]. In the November-December 1959 issue of *Datamation*, the leading periodical of the era, Herb Grosch was concerned that information processing was being defined narrowly because it is "as broad as our culture and as deep as interplanetary space"[494].

In 1967 *computer science* as a term was still quite new (for instance, the first department of *computer science* was established at Purdue University in 1962[495]). In the 1960s professors from fields other than computer science often asked, "what exactly is computer science?". Three prominent computer scientists—Allen Newell (1927-1992), Alan J. Perlis (1922-1990), and Herbert A. Simon (1916-2001) gave a public answer. They elaborately defended computer science in an article published in the September 1967 issue of *Science*;

> [1967] *Wherever there are phenomena, there can be a science to describe and explain those phenomena. Thus, the simplest (and correct) answer to "What is botany?" is "Botany is the study of plants". And zoology is the study of animals, astronomy the study of stars, and so on. Phenomena breed sciences.*
>
> *There are computers. Ergo, computer science is the study of computers. The phenomena surrounding computers are varied, complex, rich.* [...] *Computer science is the study of the phenomena surrounding computers.*[496]

Newell et al.'s definition is clearly a descriptive one (i.e., it describes what professionals, researchers, and teachers actually do). It also leans towards the empiricist tradition because it emphasizes knowledge creation from description and explanation. C. Wright Mills has given a similar argument that *social science* is defined by what duly recognized social scientists do and have done[497]. Richard Hamming (1915-1998) pointed out the difficulty of defining fields precisely when he wrote

490 C.J.A., 1967
491 Datamation, 1962
492 Ensmenger, 2001
493 C.J.A., 1967; Datamation, 1962; Ensmenger, 2001
494 Grosch, 1959
495 Rice and Rosen, 2004
496 Newell et al., 1967
497 Mills, 1959:p.19.

that "*Mathematics is what mathematicians do.*" followed by "*Mathematicians are people who do mathematics.*"[498].

Despite its eloquence, Newell et al.'s definition is criticized as being a circular definition that seems flippant to outsiders[499]. Nonetheless, the definition has been widely quoted ever since[500]. Since the computer at that time was—and still is—a novel and complex instrument, it cannot be said to be subsumed under any other science as an instrument (unlike instruments such as the electron microscope or the spectrometer). Newell et al. noted that the study of computers does not lead to user sciences, but to the further study of computers. They continued that the computer by this definition is not just an instrument but a phenomenon as well, requiring description and explanation—and phenomena define the *focus* of the science, not its *boundaries*[501]. Note that in Newell et al.'s definition, the boundaries form around the computer, not computing.

Although the computer is not an instrument of one *single* field, it does work as an instrument for numerous sciences. The development of computer hardware and software—especially nowadays—is often not an end in itself. Thus, from the academic view of computer science, a question has been raised: "*Of what use is computer science in the real world?*"[502]. This question could be answered in the spirit of Newell et al.'s definition: If computer science is the study of the phenomena surrounding computers, then the more instrumental uses the computer has, the more diverse computer science is. Consequently, the more instrumental uses the computer has, the more use computer science has in the real world. Be that as it may, Newell et al.'s definition inevitably includes many cases that most people would not consider to be computer science[503]. If one sticks to Newell et al.'s definition and reads it loosely, today essentially everything could be considered computer science.

498 Dijkstra, 1987; Hamming, 1969
499 Denning et al., 1989
500 See e.g., Knuth, 1974, Wegner, 1976, and McGuffee, 2000 among numerous others.
501 Newell et al., 1967
502 MacKinnon, 1988
503 McGuffee, 2000

The Art and Science of Processing Information

George Forsythe (1917-1972)—whom Donald Knuth argued to be responsible for the rapid development of computer science in the world's colleges and universities more than anyone else[504]—wrote in the January 1967 issue of CACM,

> [1967] *I consider computer science, in general, to be the art and science of representing and processing information and, in particular, processing information with the logical engines called automatic digital computers.* [... A] *central theme of computer science is analogous to a central theme of engineering science—namely, the design of complex systems to optimize the value of resources.*[505]

> [1968] *Computer science is at once abstract and pragmatic. The focus on actual computers introduces the pragmatic component: our central questions are economic ones like the relations among speed, accuracy, and cost of a proposed computation, and the hardware and software required. The (often) better understood question of existence and theoretical computability—however fundamental—remain in the background. On the other hand, the medium of computer science—information—is an abstract one. The meaning of symbols and numbers may change from application to application, either in mathematics or in computer science. Like mathematics, one goal of computer science is to create a basic structure in terms of inherently defined concepts that is independent of any particular application.*[506]

Forsythe brought a number of important aspects of computing to light. These aspects, which I discuss below, are the dichotomy between arts and science, the activities of computer science, real-world constraints, and the juxtaposition of practice and theory.

First, Forsythe noted the dichotomy between art and science. Although Forsythe has not used *art* in the context of computing, another famous computer scientist, Donald Knuth, contemplated the term in his Turing Award lecture[507]. In Knuth's words, "*Science is knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it.*" Referring to C.P. Snow (1905-1980), Knuth described the scientific approach with terms such as *logical*, *systematic*, *impersonal*, *calm*, and *rational*, and described the artistic

504 Knuth, 1972b
505 Forsythe, 1967
506 Forsythe, 1968
507 Knuth, 1974c

approach with terms such as *aesthetic*, *creative*, *humanitarian*, *anxious*, and *irrational*[508]. Knuth wrote, "*It seems to me that both of these apparently contradictory approaches have great value with respect to computer programming.*"[509] Richard Hamming also noted the lack of insight into the artistic aspects of computing in the January 1969 issue of the Journal of the ACM;

> [1969] *To parody our current methods of teaching programming, we give beginners a grammar and a dictionary and tell them that they are now great writers. We seldom, if ever, give them any serious training in style.*[510]

One must remember that neither science nor art are solely positive categories. Paul Feyerabend wrote that terms such as *science* and *art* are temporary collecting-bags containing a great variety of products, some excellent, others rotten, and all of them characterized by a single label[511]. But collecting-bags and labels, Feyerabend wrote, do not affect reality. They can be omitted without changing what they are supposed to organize. Of course, Knuth never implied that all science and all art would be equally valuable. Knuth's notions that art can be transformed into science and that theory improves artistry[512] are important. Art and science are not rigid labels or collecting-bags—ideas can move between art and science.

Second, Forsythe noted that *representing* and *processing* information are the activities of computer science. He also distinguished between a broad sense (*general*) and a narrow sense (*particular*) of these two activities. My interpretation of Forsythe is that in the broad sense, *representing* refers to data representations or data structures, be they trivial or complex, or be they intuitive (art) or structural-categorical (science). Furthermore, I assume that *processing* refers to informal (art) or formal (science) ways of manipulating those representations of information (algorithms). In my interpretation, Forsythe's narrow sense of the concepts confines these actions to those achievable with a digital computer.

---

508 Note that neither of these definitions (*art* nor *science*) is adopted for this thesis because of their apparent flaws. The characterization of science as something that can be taught to a computer is definitely too narrow, and the characterization of art as anxious and irrational does not do much justice to art (Knuth did not claim it to do much justice to art, either). The text Knuth refers to is Snow, 1964.

509 See Knuth, 1974c.

510 Hamming, 1969:p.10.

511 Feyerabend, 1994

512 Knuth, 1974c; Knuth, 1991. Dijkstra, on the other hand, noted only one direction to this: transforming the Art of Programming into the Science of Programming (Dijkstra, 1968b).

Third, Forsythe foregrounded *design* and *resources*.[513] Forsythe's computer science is not situated in the ideal, infinite world of mathematics, but it is situated within the finite boundaries of available resources. Design, as defined by Denning et al., is rooted in engineering and deals with constructing systems or devices to solve a given problem[514]. Design, as an engineering activity, has to cater not only to material resources but also to human constraints. In addition, whereas the Turing Machine does not have space constraints[515], Forsythe's version of computing takes into account the limits of computing resources of actual computers.

Fourth, Forsythe recognized that there are *abstract* and *pragmatic* sides to computer science. However, in Forsythe's language the abstract side does not refer to the theory of computation as one would expect. Forsythe explicitly pushed the theory of computation to the background. Forsythe's abstract side to computer science is *information* (i.e., symbols and numbers). When Forsythe noted that one task of computer science is to create an *application-independent symbol system*, he definitely took sides with the reductionist information theorists. The reductionists (or *inherent-structurists* as Hacking[516] called them) believe that all kinds of information are ultimately reducible to some *Ur*-concept, "the mother of all instances"[517]. Note that there are plenty of opposing, *nominalist*-siding viewpoints to information, and even Claude E. Shannon (1916-2001), who laid the foundations of the mathematical theory of communication, noted that it is hardly to be expected that a single concept of information would satisfactorily account for the numerous possible applications of the theory of information[518]. In fact, Shannon's use of the term *information* is very specific—from Shannon's engineering viewpoint, the *meaning* of information was not relevant at all[519].

Although Forsythe's definitions are unorthodox and practical in some aspects, such as pushing the theory of computation to the background, his definitions are also very positivist or structurist in other aspects, such as aiming at universalist symbol systems in information. On the whole, Forsythe offered an early definition of computer

513 Forsythe wrote, "...the design of complex systems to optimize the value of resources" (Forsythe, 1967).
514 Denning et al., 1989
515 See Turing, 1936–his example machines yield infinitely long results. See also Turing, 1950, where Turing explicitly talks about infiniteness of the Turing Machine.
516 Hacking, 1999
517 Floridi, 2004b
518 Shannon, 1950, as reprinted in Sloane & Wyner, 1993:pp.180-183.
519 Shannon, 1948, as reprinted in Sloane & Wyner, 1993:pp.5-83.

science that tries to accommodate both the pragmatic and the abstract sides of computer science under the same umbrella term, instead of dividing computing into the fields of science and engineering (or into abstract and concrete, or into pure and applied).

An early characterization of *programming* by an authoritative figure in computing can be found in Donald Knuth's 1968 classic *The Art of Computer Programming Vol. 1*:, which states, "*The notion of an algorithm is basic to all computer programming.*" An algorithm is a precisely defined sequence of rules that tells one how to produce specified output information from given input information in a finite number of steps.[520] Knuth's seminal 1968 book is a prime example of the rationalistic research tradition, but six years afterward, in 1974, Knuth took another viewpoint to computer science in the April 1974 issue of *American Mathematical Montly*, when he referred to the Newell et al.'s definition from 1967. Knuth wrote,

> [1974] *When I say that computer science is the study of algorithms, I am singling out only one of the 'phenomena surrounding computers', so computer science actually includes more. I have emphasized algorithms because they are really the central core of the subject, the common denominator which underlies and unifies the different branches.*[521]

The notion of algorithm is indeed central to programming, and Knuth's 1974 argument "*computer science is the study of algorithms*" seems to hold still today in many characterizations of computer science. For instance, each of Denning's thirty core technologies of computing[522] (see Figure 7 on page 69) includes studies of algorithms in one way or another.

The Official Birth of Computer Science

The ACM had begun working on a recommendation for academic programs in computer science as early as 1962 (the same year as Purdue University launched the first study program actually called *computer science*[523]). The ACM Curriculum Committee became an independent committee of the ACM in 1964, and released their first

520 Knuth, 1968:pp.1-7.
521 Knuth, 1974
522 Denning, 2003
523 Rice and Rosen, 2004

draft in 1965[524]. The final version was completed in 1968, and it characterized the subject areas of computer science as follows:

> [1968] *The subject areas of computer science are grouped into three major divisions: "information structures and processes", "information processing systems" and "methodologies".*[525]

Computer science, according to this view, is the study of *information structures*[526]. A modern computer scientist would probably see this definition as suitable for the academic discipline of *information systems*. Nevertheless, this definition can be read either as a normative framework (i.e., aiming at defining the discipline by listing the topics computers scientists *should be* doing) or a descriptive one (i.e., describing what computer scientists *actually* do). The report describes and discusses twenty-two courses that were found in the computer science curricula at universities in the U.S., and the report uses two dozen pages of suggested readings for these courses. In the '68 report, the ACM people clearly take some distance to subjects they believe should not belong to academic computer science. They wrote:

> [1968] *...these recommendations are not directed to the training of computer operators, coders, and other service personnel. Training for such positions, as well as many programming positions, can probably be supplied best by applied technology programs, vocational institutes, or junior colleges.*[527]

Similarly to Knuth's 1968 definition, the ACM's 1968 definition reflects a mathematical research tradition since algorithms and information structures are two abstractions of the phenomena that computer science is concerned with[528]. But unlike Knuth, the ACM definition excludes computer-related work that is not considered academic, such as programming. At the time this position was already criticized for being too academic, theoretical, and narrow[529]. The critics demanded a more practitioner-oriented view, hands-on laboratory work, and inclusion of other computer-related areas into computer science education (this issue is discussed later in this thesis).

---

524 Conte et al., 1965
525 Atchison et al., 1968
526 Wegner, 1976
527 Atchison et al., 1968, underlining added.
528 Wegner, 1976
529 Wishner, 1968; Hamming, 1969

In the June 1974 issue of *CACM*, the president of the ACM at the time, Bernard A. Galler, announced that the National Science Foundation had passed the following resolution:

> [1974] *Resolved that the Computer Science and Engineering Advisory Panel of NSF affirms the distinction of Computer Science from all other science or engineering disciplines and recommends that the National Science Foundation make this manifest in its statistical and programmatic activities.*[530]

This resolution was greeted as an important step, because it was expected that the distinction of computer science as an autonomous discipline would increase funding, give computer science its own student and research fellow quotas, and grant computer science its own representation on the National Science Board and similar policy-making committees and boards[531].

Neither the theoretical camp nor the practical camp in computer science were undivided. There existed a clear dichotomy within the ranks of mathematically oriented computer scientists because the mathematical aspects of computer science include not only *computability* and *complexity*—which touch upon logic, combinatorics, and probability theory—but also include *numerical analysis*, which entails intensive computations[532]. The practitioners' side was even more scattered—the practical sides of computer science included, for instance, architectural design, coding, computer engineering, and program design.

Lotfi A. Zadeh, who has been credited as being the father of fuzzy logic, wrote in 1968 that computer science consists of subjects which belong to computer science to different degrees[533]. In Zadeh's opinion, *fuzzy sets of topics* that play a central role in computer science, such as programming languages, operating systems, and data structures, have almost full containment in the *fuzzy set of computer science*. Those topics that are more peripheral, such as mathematical logic, have "less containment in computer science". Zadeh's (sets of) topics of computer science are grouped in Figure 21 according to their containment in (the set of) computer science[534].

---

530 Galler, 1974
531 Galler, 1974
532 Gal-Ezer and Harel, 1998
533 Zadeh, 1968
534 Adapted from Table 1, *Containment Table for Computer Science*, in Zadeh, 1968.

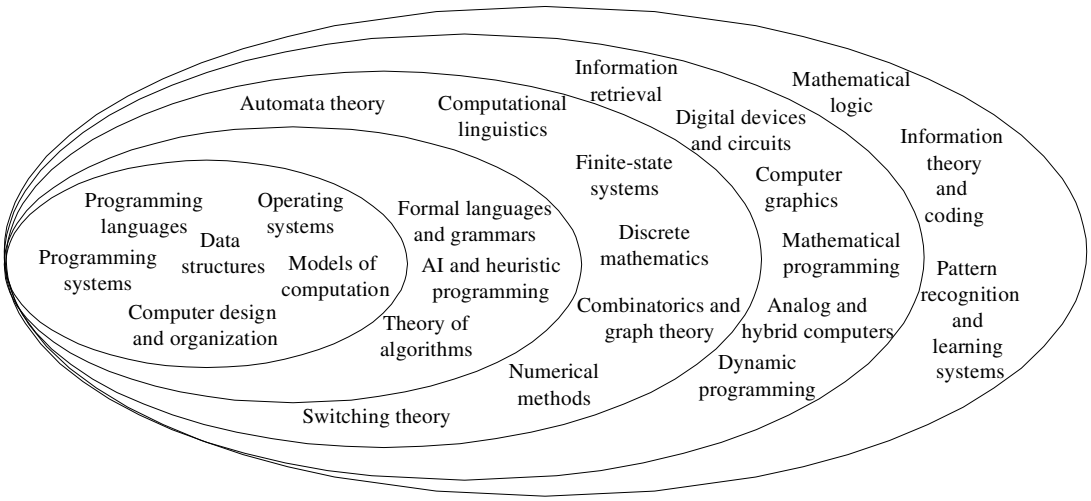


Figure 21: CS Topics Grouped According to Their Containment in CS

The topics of computer science on the innermost circle of Figure 21 are those that Zadeh regarded as having a "*degree of containment equal to unity*". Zadeh gave these topics a degree of containment equal to 1 (on a scale from 0 to 1). The topics that Zadeh graded to have a degree of containment in computer science equal to 0.9 are grouped on the second innermost circle in Figure 21. The topics on the third, fourth, and fifth circles have degrees of containment equal to 0.8, 0.7, and 0.6, respectively.

There is substantial overlap between Zadeh's list of computing topics from the year 1968 and Denning's *core technologies of computing* from the year 2003 (Figure 7, page 69 of this thesis). Although the two lists are not fully comparable, because Denning's list consists of *technologies* whereas Zadeh's list consists of *topics*, many comparisons can be made. Many topics such as *programming languages*, *operating systems*, and *data structures* are found in both Zadeh's and Denning's lists under exactly same names, which implies that much of modern computer science was already in place in 1968. However, *analog and hybrid computers*, found on Zadeh's list, are not even implicitly on Denning's list. Furthermore, there are a number of new topics on Denning's list that are not even implicitly on Zadeh's list—*e-commerce*, *workflow*, *virtual reality*, *robots*, *software engineering*, *management information systems* (MIS), and *human-computer interaction*. Although much of computer science was in place in 1968, the field has diversified substantially between 1968 and 2003.

Shaping the Public Image of Computing

In his 1968 Turing Award lecture, Richard Hamming stated his occasional frustration with the debate around "*What is computer science currently? What can it develop into? What should it develop into? What will it develop into?*"[535]. Still he considered it very important not to ignore the discussion over definitions of computing to just "get on with doing it", and he brought to light a very important point—the point that the public image of a science affects how a science is formed;

> [1969] *the picture which people have of a subject can significantly affect its subsequent development. Therefore, although we cannot hope to settle the question definitively, we need frequently to examine and to air our views on what our subject is and should become.*[536]

In Hamming's sense of the term *computer science*, at the heart of computer science lies a technological device, the computing machine. Without it, Hamming argued, almost everything that computer scientists do would become idle speculation, hardly different from that of the notorious Scholastics of the Middle Ages[537]. Hamming noted that the work of computer scientists is guided by expenses, workload, and the popular image of computer science (which also affects the amount of money granted to science);

> [1969] *So much of what we do is not a question of can it be done as it is a question of finding a practical way. It is not usually a question of can there exist a monitor system, algorithm, scheduler, or compiler, rather it is a question of finding a working one with a reasonable expenditure and effort.*[538]

That is, in Hamming's opinion, the theoretician's question "Can there be *x*?" is less frequent than the practitioner's question "What is the most cost-effective way of building *x*?". Therefore, in Hamming's vision, the focus should not be on computation, but on the computer.

---

535 Hamming, 1969

536 Hamming, 1969

537 Hamming, 1969. Scholastics relied on books by renowned scholars, studying them thoroughly, learning the theories of the authorities to the letter. Scholastics' work relied solely on the books, not on developing their own theories. In this mode of working there cannot be growth of knowledge, but the amount of knowledge can only diminish or stay the same.

538 Hamming, 1969, underlining added.

Hamming also brought forth an important point when he raised a question about the distinction between *undirected research* and *basic research*. The point of the question concerns what kind of activity is considered to be progressive research and what is not. He wrote:

> [1969] *This brings me to another distinction, that between undirected research and basic research. Everyone likes to do undirected research and most people like to believe that undirected research is basic research. I am choosing to define basic research as being work upon which people will in the future base a lot of their work.*[539]

The line between these two kinds of research is not always clear: "*While one cannot be certain that a particular piece of work will or will not turn to be something to be built upon, one can often give fairly accurate probabilities on the outcome*"[540]. Hamming questioned the academic and professional skills of many of his contemporaries; he argued that very few people are capable of doing basic research. Unfortunately, Hamming did not define undirected research vis-à-vis basic research in any of his articles available in the ACM Digital Library[541], and the aforementioned definition is ambiguous. However, the chain of Hamming's logic seems to be as follows: There is some (probabilistic) criteria that can be used for measuring whether a research study will turn out to be basic or not. So, one can predict whether his or her research study will be basic research or not. Why do some people continue, then, doing research that they know is undirected and will not help in building the scientific base? Because they cannot do basic research.

Regardless of whether Hamming's comment was actually meant in the way I interpreted it or not, one way of understanding the concepts of undirected research and basic research is from the Kuhnian viewpoint. From Kuhn's point of view basic research is a *puzzle-solving activity*, and undirected research is just unparadigmatic non-research. Note that Kuhn was unequivocal in that no research can aim at unexpected results, not even research that aims at paradigm re-articulation. Hamming may have taken a Kuhnian-kind of a point of view when he noted that the problem solving *method* defines whether research is basic or undirected. If he did take the

539 Hamming, 1969:p.10.

540 Hamming, 1969

541 A search in ACM Digital Library (Nov. 12, 2005) for "`undirected research`" produced two articles: Denning, 1992 and Hamming, 1969. Denning used the term to refer to basic research.

Kuhnian point, his comment does not mean much—Hamming would be comparing research to non-research. Yet, I think that the gist of this matter is not about whether research is done within the area of normal science or not.

Rather, it seems to me that the term *undirected research* is an oxymoron. As an idiom, the term *undirected research* has been widely used synonymously with *non-mission research*, that is, basic academic research that is not driven by applications[542], but this cannot be the sense in which Hamming used the term. *Undirected* means most commonly *having no object or purpose*; *not guided*; *having no prescribed destination*[543]. Furthermore, *research* means usually something like *investigation*, *inquiry*, or *study*. All synonyms of *research* are *about* something, or *of* something. There is no research that does not aim at *something*, be it theory-refinement or exploring unknown phenomena. Research always aims at something, and no research can aim at *unexpected* results (see footnote [174] on page 74 of this thesis).

I noted earlier that problem-solving, also of the scientific kind, has starting points (premises), goals, and methodologies (see page 191 of this thesis). Respectively, if all research aims at something, then Hamming's juxtaposition of basic research and undirected research can be understood either as a criticism of free choice of starting points, as a criticism of free choice of goals, or as a criticism of free choice of methodologies. The Feyerabendian response to all three is roughly the same.

First, if science (as an institution) is rigid about the *premises* of research, it may close its doors to valuable insights. Take, for instance, the insistence of the geocentric world view in the age of Copernicus. Second, if science (as an institution) is rigid about the *goals* of research, it may insulate the scientific community from a whole universe of important problems. Take, for instance, *verification* as the goal of research versus *explanation* as the goal of research. If the goal of science is verification, scientists can tackle a very different set of problems than if the goal of science is explanation. Third, and similar to the second point, if science (as an institution) delimits the *methodologies* of research to a number of approved ones, it may close the doors to a number of important approaches and to whole fields of inquiry. Take, for instance, *a priori* methodologies versus *a posteriori* methodologies as normative accounts for research. The only approach to research that does not restrict progress

542 See, e.g., Asner, 2004 in Grandin et al., 2004:pp.4-5; Denning, 1992; National Research Council, 1999:p.213.
543 AHD, 2004

in any way is absolute freedom from any kinds of constraints. (Yet this does not mean that freedom of constraints would *further* research the best, or that one should not care about, for instance, the ethics of research.)

It is certain that there was a wide divergence of opinion on what computer science was before the 1970s—and on what computer science should have become. There were those who saw computer science as a mathematical science, separated from mundane computing machines, then there were those who took computer science as an engineering science, and finally there were those who wanted to see an interplay of a variety of aspects of computing. At the turn of the 1970s, as computing machinery continued to spread to academic and business circles, the debate gained intensity.

A Concern Over Dogmatism in Computing

In his 1970 Turing Award lecture, published in the April 1970 issue of *Journal of the ACM*, Marvin Minsky expressed his concern about computer science's having an obsession with form instead of content[544]. For instance, programming languages and the theory of computation, Minsky argued, had an excessive preoccupation with formalism: Minsky wrote,

> [1970] *To build a theory, one needs to know a lot about the basic phenomena of the subject matter. We simply do not know enough about these, in the theory of computation, to teach the subject very abstractly. Instead, we ought to teach more about the particular examples we understand thoroughly, and hope that from this we will be able to guess and prove more general principles.*
>
> [The] *syntax* [of programming languages] *is often unnecessary. One can survive with much less syntax than is generally realized.* [...] *What is a compiler for? The usual answers resemble "to translate from one language to another".* [...] *For the future, a more ambitious view is required. Most compilers will be systems that "produce an algorithm, given a description of its effect.*[545]

Minsky's concern was related to his claim that "*there are many ways to formulate things and it is risky to become too attached to one particular form or law and come*

544 Minsky, 1970
545 Minsky, 1970

to believe that it is the real *basic principle*"[546]. As the underdetermination thesis states, there are indeed infinitely many ways to formulate a theory about a phenomenon. A caution about detrimental dogmatism is reflected in Minsky's text (which was published before Feyerabend's book *Against Method*[547]). Minsky is anxious that too strong of an adherence to normal science (or too strong of a belief that the normal science of a particular era is the culmination of progress) may blindfold the researcher from seeing alternative paths. In Minsky's opinion, concentrating on content instead of form would free people to "*build, in their heads, various kinds of computational models*" instead of wasting their resources on the form[548]. I interpret this as a call for intuitiveness, creativity, and *ad hoc*ness instead of a call for concentrating on preordained forms. The content *vs*. form-juxtaposition can be interpreted as a juxtaposition between actively *shaping* new forms *vs*. *adapting* existing forms into different contexts.

The concern Minsky had about programming languages is also quite an unorthodox one, given the year he stated it. Some researchers were quite happy with the state of programming at the time, and Peter Wegner; who was an editor-in-chief of one of the major ACM publications, *Computing Surveys*; even wrote in *IEEE Transactions on Computers* 1975, that "*it may well be that programming language professionals did their work so well in the 1950's and 1960's that most of the important concepts have already been developed*"[549]. Note that afterwards there have been developments in web design languages, virtual machines, visual programming, design patterns, and so forth. Although some of them were developed as early as the 1960s, they had not been included in Wegner's article. It is unlikely that by *most important concepts* Wegner referred to the *stored-program-paradigm*, because the *stored-program-paradigm* was already in place in the late 1940s.

It is difficult to see how programming languages could cease to change. The different styles of programming languages such as SQL, Java, Perl, and C++ are a result of differences in the application domains that inspired their creation in the first

546 Minsky, 1970. Note the resemblance of this statement to underdetermination thesis (see p.60).

547 Feyerabend, 1993 (orig. 1975). Feyerabend's early papers on the topic were published around 1970 (see, e.g., Feyerabend, 1970), but both Preston and Farrell have argued that this was between Feyerabend's realist and anti-realist periods (Preston, 1997, Farrell, 2001).

548 Minsky, 1970; see also Minsky's earlier text *Why Programming is a Good Medium for Expressing Poorly Understood and Sloppily-Formulated Ideas* (Minsky, 1967 in Krampen & Seitz, 1967).

549 Wegner, 1976b

place[550]. Insofar as programming languages are application-dependent, their change would cease when application domains would cease to change. Minsky was far-sighted enough to understand that the application domains of computing are not yet exhausted. The number of calls for new approaches; such as empirical modeling[551], resilient systems[552], stochastic and quantum computation; is as large as ever. Minsky was also wise enough not to think that the 1970's abstraction level in programming languages would have been the highest level possible. Yet, because Minsky's statement above does not exclude universals, his angle seems close to that of Donald Knuth in the sense that Minsky argued that working with particulars may help in finding universals. (Similarly, Knuth wrote, "*We should continually be striving to transform* every *art into a science: in the process, we advance the art.*"[553]) Although both Minsky and Knuth seem to be on the structurist side of the structurist-nominalist debate, neither Minsky nor Knuth mentioned, though, how universals are to be found, if ever.

Is the Focus the Machine (Computer) or the Phenomenon (Information)?

The International Federation for Information Processing (IFIP) World Conference on Computer Education convened in 1970 in Amsterdam, the Netherlands, and accepted the term *computer science* as "*the study of computing machines (actual or potential)*"[554]. It is unlikely that this definition could have been expected to make much of a difference in the computing community. It does not comment on what subfields are included in the science, neither does it explain in a sufficiently exact way what is it that computer scientists do except for "study". From almost any point of view, this definition suffers from excessive generalization (which may be a direct result of working in a committee).

Much more interesting than the IFIP Conference Recommendation Committee's defining of computer science is their defining of a parallel term—*informatics*—which the French championed. Aaron Finerman wrote that inherent in the definition of informatics is the notion that information processing by computer is a *syntactic process* involving symbol strings (much in the same manner Shannon defined *informa-*

550 Denning, 2003
551 Beynon & Russ, 1995
552 Frankston, 1997 in Denning & Metcalfe, 1997.
553 Knuth, 1974c
554 Finerman, 1970

tion[555]), while information processing by humans is a *semantic process* involving word and phrase images. Finerman wrote in the November 1970 issue of CACM,

> [1970] *Informatics is the science of the systematic and effective treatment (especially by automatic machines) of information seen as a medium for human knowledge and for communication in the technical, economic, and social fields.*[556]

That is, *informatics* studies how information can be processed automatically. It is unclear, however, whether the committee implied that computers might be able to process information semantically (as humans do), or whether there is some sort of a fundamental dissimilarity between information as a medium for human knowledge and information as machine-processable phenomenon.

Even today, computer scientists from different regions of the world name and define the field of computing in a different ways. Gal-Ezer and Harel wrote in the September 1998 issue of CACM,

> [1998] *In fact, there is no clear agreement even on the name of the field. In European universities, the titles of many of the relevant departments revolve around the word 'informatics', whereas in the U.S. most departments are 'computer science'. To avoid using the name of the machine in the title,* [...] *some use the word 'computing' instead. Other department names contain 'information systems' or 'computer studies'.*[557]

There is a variety of names for the field in languages other than English. For example, Finnish universities use the term *computer science* in texts in English, but the Finnish word for the discipline, *tietojenkäsittelytiede*, is translated word for word as "information processing science"[558]. In Swedish the discipline is *datavetenskap* or *informationsteknologi*: "data science"[559] or "information technology", respectively. In Dutch the term is *informatica*, which refers to the junction between information and automatic (processing)[560]. Similarly, in Russian, the basics of computer science

555 Shannon, 1948; Shannon, 1950
556 Finerman, 1970
557 Gal-Ezer and Harel, 1998
558 However, it is ambiguous if in Finnish the word *tieto* refers to data, information, or knowledge.
559 Yet, *data* in *datavetenskap* is often understood as *dator* (computer), but to use *data* and *dator* synonymously is obviously incorrect.
560 Dr. Piet Kommers (Aug 5th 2004, oral communication).

is информатика—"informatics" or "information science"[561]—but on the advanced level, the field is divided into subfields and characterized in more specific terms such as вычислительная техника ("computer techniques", or "computer engineering")[562]. Gordana Dodic-Crnkovic has noted that interestingly, the British term *computer science* with its empirical orientation and the German and French terms *informatics* with their abstract orientation, correspond to the eighteenth- and nineteenth-century characters of British Empiricism and Continental Rationalism[563]. One should also note that in the early history of modern computing, Americans were the advocates of *practical* work whereas the British were the advocates of *theoretical* work.

In 1966 Peter Naur suggested using the term *datalogy* to replace *computer science*[564] —Naur made a distinction between *data* and *information*. Edsger Dijkstra also argued that *computer science* is an entirely wrong term: "*Primarily in the U.S., the topic became prematurely known as 'computer science'—which actually is like referring to surgery as 'knife science'*."[565]. Instead of the computer, or computing technology, Dijkstra wanted to emphasize the abstract mechanisms that computing science uses to master complexity. Contrary to Dijkstra's opinion, Frederick Brooks Jr. wrote that "*our namers got the 'computer' part exactly right*"[566].

The importance of naming the field is discussed further later in this thesis (page 341ff.), but it ought to be noted here that it has been argued that the choice of the name of the field situates the field among other disciplines, creates the expectations set for it, guides the areas in which it can be applied, and, much in a Kuhnian manner, affects the choice of problems for the field.[567] Certainly names like *knowledge processing science*, *computer engineering*, *informatics*, and *computer science* arouse different connotations.

There was a group of proponents of the algorithmic, rationalistic research tradition, and Edsger W. Dijkstra was among them. In the October 1972 issue of CACM, Dijkstra wrote,

---

561 See e.g. the English version of Cormen et al., 1998:p.*xv*–"Laboratory for Computer Science". In the Russian translation (Cormen et al., 2001:p.15) it is "Лаборатория информатики" (Laboratory of informatics).

562 Dr. Alexander Kolesnikov (Aug. 5th 2004, oral communication); Ms. Evgenia Chernenko (Aug 2nd 2004, oral communication).

563 Dodig-Crnkovic, 2002

564 Naur, 1966

565 Dijkstra, 1987

566 Brooks, 1996

567 cf. Brooks, 1996

> [1972] *We must not forget that it is not our* [computing scientists'] *business to make programs; it is our business to design classes of computations that will display a desired behavior.*[568]

Dijkstra's viewpoint is a well-founded and original one[569], and top-down approaches similar to Dijkstra's have been promoted by, for instance, Niklaus Wirth and Peter Naur[570]. In 1972, proofs of program correctness were usually written *a posteriori*, that is, after the program had been written[571]. Instead of *a posteriori* proofs, Dijkstra suggested that one should start designing classes of computations by formally (mathematically) specifying what the program should do, and then gradually *deriving* the actual program from these specifications. This resembles the mathematical technique of proof by construction. Since derivations are mathematical transformations between two systems, if the rules for transformations are correct, the resulting program is necessarily correct (bug-free[572]). From a mathematical point of view, Dijkstra's approach is durable, but it never gained much support outside minor coteries in academic circles, and it was later rebutted by James H. Fetzer, who noted that verification can only establish correctness between two formal models—program and formal description—but programs as causal phenomena are not appropriate subjects for deductive verification[573]. Brian Cantwell Smith noted that although one can prove the correctness of the relationship between the program and the model, it certainly does not say anything about how well a program does what it was intended to do[574].

Legitimating Number Crunching as a Science

If physics is considered to be the king and mathematics the queen of the sciences[575], then logic is probably the father of the king. In this picture, computer science can hardly be said to belong to the royal family. In the 1970s, many mathematicians

568 Dijkstra, 1972

569 See, for instance, an early writing by Dijkstra in Dijkstra, 1968b.

570 Wirth, 1971; Naur, 1969; Naur, 1972

571 Dijkstra, 1975

572 Donald Knuth often has been said to have written a memo that ended with a sentence "*Beware of bugs in the above code; I have only proved it correct, not tried it.*". Knuth dates this memo to March 29th, 1977. This seemingly innocent quotation does have a good point–human errors can occur in any part of the program-construction process. Proving a program correct does not mean that the program corresponds to the proof, that is, that the translation from proof to program was done correctly.

573 Fetzer, 1988; see also an overview of the formal verification debate in Colburn, 2000:139-152 passim.

574 Smith, 1996 in Kling, 1996:pp.810-825 (found also in Johnson & Nissenbaum, 1995:pp.456-479).

575 Easton, 2006

who saw *infinity* as a prerequisite for mathematical depth, saw computers only as number crunchers, or perhaps as tools for numerical analysis[576]. Hence, the best legitimation of the field of computing as a science would have been to gain the recognition of the mathematical community. Donald Knuth, who is not only a computer scientist but also a recognized mathematician, saw the theory of computing as a subject worthy of study as such. In the April 1974 issue of *American Mathematical Monthly* he wrote,

> [1974] *Like mathematics, computer science will be somewhat different from the other sciences, in that it deals with* [hu]*man-made laws which can be proved, instead of natural laws which are never known with certainty*. [...] *The difference* [between mathematics and computer science] *is in the subject matter and approach—mathematics dealing with more or less with theorems, infinite processes, static relationships and computer science dealing more or less with algorithms, finitary constructions, dynamic relationships*.[577]

This extract of Knuth's article has two parts: In the first part Knuth noted the similarities between computer science and mathematics. In the second part he sketched the differences between the two. Alas, Knuth's ontological position is not clear in this article. However, the term "deals with man-made laws" implies that he took a position of *ontological subjectivity*. That is, the term can be understood as taking an attitude that the laws in computer science exist because there are intelligent beings that make them exist. Note, however, that Knuth's wording does not rule out ontological objectivity in the sense that *human-made* can refer to a symbol-theory system that describes universal facts that exist without people. In other words, Knuth's statement may mean that the laws of computer science are windows to *noumena*[578]. (Note that Knuth uses the term *laws* instead of constructions, theories, or hypotheses.)

Knuth seems to have argued for *epistemological objectivity* in the sense that laws "can be proved". That laws can be proved true indicates that Knuth took there to be a universal *logic*, following which, any independent thinker would proceed from the same axioms and premises to the same conclusions. Underlying Knuth's comparison

576 Dijkstra, 1987

577 Knuth, 1974

578 According to Immanuel Kant noumenon is *thing-in-itself*, independent of the subject, and noumena are opposed to *phenomena* that refer to the world as experienced by people. Note that in Kant's philosophy, noumena and phenomena are separate: One cannot expect to achieve certainty about noumena by observing phenomena.

of computer science with empirical sciences there is an assumption that is indicative of extreme epistemological objectivity: Even natural sciences are uncertain compared to computer science. From this epistemological point of view, the laws of computer science are facts that researchers have constructed to perfection within the bounds that the researchers have set—hence the factuality of the laws of computer science compared to the imperfect interpretations of the unbounded external world made by natural scientists. Knuth's epistemological position can be understood differently if one allows axioms, rules, and logic to be matters of taste (chances are that Knuth does not consider them to be matters of taste). It is not certain, however, what are *laws* in computer science.

In the discussion about the explanations of stability[579] (p.127ff. of this thesis), I argued that computer scientists are much more responsible for the level of complexity of their own discipline than natural scientists. Earlier design choices in control structures, architectures, languages, techniques, data structures, syntax, semantics, and so forth, define the starting points for future challenges. A corollary of my argument is that if computer science is seen as a stable science, it is stable only because computer scientists have built it that way. Stability or instability is not an inherent quality of computer science, but a result of external aspects that have shaped (human-made) computer science. In this sense, Knuth's position seems consistent with my position. If Knuth believed that stability and instability are inherent qualities of computer science, he might not have characterized the science as being human-made. Of course there is always the possibility that Knuth took the *stability* as a result of the inherent qualities of computer science, but took the *instability* as a result of the limited understanding of the inherent structures of computation.

The second part of Knuth's text deals with differences between mathematics and computer science. Knuth distinguished computer science from mathematics by examining their subject matters and approaches. Computer science deals with finite constructions that are characterized by dynamic relationships, and uses algorithms to deal with these dynamic relationships. Knuth listed three characteristics of computer science that differentiate it from mathematics. First, *finiteness* (of space, but not of time[580]) is a prerequisite of realizability. That is to say, infinitely large constructions cannot be realized with computational instruments—but infinitely long computations

579 Hacking, 1999:pp.84-92.

can be. Second, *dynamic relationships* are a prerequisite for modeling the dynamic world (unless a stable "grand unified theory" can be formed). Third, *algorithms* shift the focus from static models towards processes or automation. These three characteristics of computer science boil down to *realizability*, or whether a given task can be computed with some sort of mechanism.

It seems that in Knuth's computer science, facts are ontologically subjective—the facts that computer science deals with exist because there are thinking creatures that make them exist. Yet it seems that Knuth took these facts to be epistemologically objective in the sense that they are not matters of preferences, evaluations, or moral attitudes[581].

Only two months after Knuth's article was published, Dijkstra also wrote, in the same journal, about the differences and similarities between mathematics and programming; Dijkstra called programming "an activity of mathematical nature"[582]. Dijkstra pointed out three characteristics of the mathematics curriculum that are possibly detrimental to the cognitive skills needed by programmers[583]:

(1) a standard collection of concepts *vs.* concept-creating skills;

> [1974] *In the standard mathematical curriculum the student becomes familiar (sometimes even very familiar!) with a standard collection of mathematical concepts, he* [sic] *is less trained in introducing new concepts himself.*

(2) learning standard notation *vs.* ad-hoc notation (note that *ad hoc* is considered positive in this sense, unlike the falsificationist sense);

> [1974] *In the standard mathematical curriculum the student becomes familiar (sometimes even very familiar!) with a standard set of notational techniques, he* [sic] *is less trained in inventing his own notation when the need arises.*

580 This argument holds if matter is considered finite. Maximum storage space usage cannot exceed the limits of the physical machine with which the computation is realized. Infinitely long computations can be set up, but they most probably do not live up to the expectation and run infinitely. Consider, for instance, the suggested half-life of proton and the predicted decay of matter to iron.

581 cf. Searle, 1996:p.1.

582 Dijkstra, 1974

583 Dijkstra, 1974

(3) and shallow hierarchy *vs.* deep hierarchy;

> [1974] *In the standard mathematical curriculum the student often only sees problems so "small" that they are dealt with at a single semantic level. As a result many students see mathematics rather as the art of organizing symbols on their piece of paper than as an art of organizing their thoughts.*

Dijkstra's first observation was that programmers are expected to be able to express themselves in both natural language *and* in formal systems—which, together, lead to skills in concept creation. Whereas Pólya wrote that when solving a problem, mathematicians may need to formulate new questions about the problem, Dijkstra wrote that in the process of problem-solving in computing, programmers often come up with intermediate concepts and may need to construct their own formalisms[584]. This may at first give the impression of problem solving as a class C-O-C problems (closed starting points, open technique, closed goals), or what Kuhn called "puzzles" in normal science. Dijkstra held that although concept creation can be found in mathematics, it is especially characteristic of computer science. My interpretation is that concept creation in computing is in effect the continuous (re-)creation of conceptual schemes.

Whereas Knuth noted that computer science deals with finitary constructions and dynamic relationships[585], Dijkstra's second comment seems to imply that programmers (dynamically) re-create, or at least re-model, these relationships and concepts as they work. Dijkstra wrote that programmers often need to manipulate a given formal syntax in order to formulate a theory to justify their algorithm—which leads to the ability to invent one's own formalisms.

In connection with his third comment, Dijkstra noted that programmers have to be able to move between time grains the size of nanoseconds (one clock cycle[586]) to time grains the size of hours (whole computations), and all the levels between. The ratio between these time grains can easily be 10^{10}—being able to move and operate between this many orders of magnitude helps in mastering complexity. My interpretation is that when programmers approach problems through computational tools, they are not only (re-)defining relationships between a number of semantic levels

584 Dijkstra, 1974; Pólya, 1957:pp.210-211.

585 Knuth, 1974

586 A clock cycle is the time period at which a computer performs one basic operation such as addition, or transferring a value from one register (temporary data storage) to another.

within computing, but also between the semantics of computing and the obscure semantics of the world that the computers are a part of. That is, they are semantically *relating* computers with their surroundings.

Yet the strongest argument of Dijkstra relates to theory. Dijkstra wrote,

> [1974] *In other words, given the problem, the programmer has to develop (and formulate!) the theory necessary to justify his* [sic] *algorithm. In the course of this work he will often be forced to invent his own formalism.*[587]

That is, Dijkstra argued that often programming is, in effect, a formulation of theories. Of course, Dijkstra's approach to programming was different than many other approaches at the time; it was "developing program and proof hand-in-hand". Taking this approach into account, computer science, as seen by Dijkstra and Knuth, still seems like an extraordinarily dynamic field. If programmers truly are constantly reshaping the *conceptual framework* (constructions) of computer science, if they are looking for new ways of *applying* computing (relationships), and if they are often developing *new theories* (tools) for the field, dynamism is not only an integral part of the *constructions* of computer science but also an integral part of the autopoïetic *construction* of computer science. Dynamism is not an integral part of *relationships* in computations but it is also an integral part of *relating* computer science to its surroundings.

An interesting question follows from Knuth's and Dijkstra's depictions of computer science: *If* computer science (*1*) deals with algorithms, finitary constructions, dynamic relationships; *and if* it (*2*) deals with these facts by means of concept-creation, ad-hoc formalisms, symbol systems, and theory-reconstruction; (*3*) can computer science explain itself? In other words, can computer science as a theory-methodology explain computer science as a socially constructed phenomenon? Are social phenomena reducible to sorts of computation? Some researchers do indeed present computational "algorithmized" approaches to sociocultural phenomena[588].

The Tug of War Between the Theoretical and Practical

Dijkstra's and Knuth's writings were addressed to the mathematical community, which at the time often did not recognize the disciplinary identity of computer sci-

587 Dijkstra, 1974
588 Brent et al., 2000; Gabora, 1995; Easton, 2006

ence. Elsewhere, the pressures of the business world were building up on computer science. At the same time as some leading computer scientists worked to gain the recognition of the mathematical community, proponents of the practical, "real-life" aspects of computer science worked to close the gap between the theoretical and practical sides of computer science. They claimed that computer science in the academic sense had detached from the real world, and that the computer science of the time had very little to do with computers[589]. Even more, some questioned the theories of computer science. Abraham Kandel wrote in the June 1972 issue of CACM,

> [1972] *Industry gets graduates from computer science departments with a bag full of the latest technical jargon but no depth of understanding the real computer systems and no concept of the problems they will be asked to solve.*
>
> [...] *It is quite obvious that there is no effective theory of computer science as such. In fact, there are no effective models of computers.*[590]

Naturally, *effective* is an ambiguous concept (see the discussion on pages 111ff. of this thesis), and without stating the criteria for an effective theory of computer science, opinions such as this have only discourse value. Be that as it may, comments such as the one above became increasingly common. The demands of adding practical training and lab work to the university curriculum led to an increased interest in software-oriented or commercially oriented computer science programs[591], which began to shape what computer science was in practice. That is, the needs of the software industry guided the interests of researchers and teachers, which manifested in what was taught in universities. George Glaser, the president of AFIPS (American Federation of Information Processing Societies, Inc.) addressed the 10th annual meeting and conference of the Inter-University Communications Council (Educom), stating,

> [1974] *A formal education in computer science is not an adequate—not even appropriate—background for those who must design and install large-scale computer systems in business environment.* [...] *The educational system is providing nicely for a body of competent computer researchers and*

589 Kandel, 1972
590 Kandel, 1972
591 Pitts and Bateman, 1974

> *teachers but has done little to provide for the needs of those who must apply computer technology, particularly in a business environment.*[592]

Since computer systems by the 1970s had grown—and kept on growing—in size, they were increasingly unmanageable without new approaches[593]. This was due to the fact that large computational systems are unities of structured elements, irreducible to the sum of component elements and structure[594]. (Because the complexity in a system arises not from the parts of the system, but from the semantical and functional connections between these parts, a system is more complex than the sum of its parts.) The resulting problem was called the *software crisis*[595]. Academic computer science of the time was accused of being unable to respond to the software crisis—for instance, Michael J. Spier wrote in the ACM SIGOPS Operating Systems Review,

> [1974] [...] *our current computer science, as it applies to operating systems, does not provide us the necessary foundation in terms of which we would understand and control programs of significant complexity.* [...] *The applicability* [of formal computer science disciplines] *to the definition and global design of operating systems is close to nil.*[596]

This ineptitude led to a lack of university graduates who were able to "*work effectively in all phases of the development of large software systems*"[597]—the interests and expertise of industry did not meet the interests and expertise of academia. This limitary problem concerned both industry and academia because neither was able to maintain their ever-growing computing systems. As a response to the growing complexity of computer systems, around the mid-1970s the *systems approach* (systems analysis or systems engineering, which had began emerging as a field around 1950s) began to gain popularity among computer scientists. Although there had been studies of complex systems, such as Herbert A. Simon's *The Sciences of the Artificial*[598], there was a lack of rigorous methodologies for mastering complexity.

---

592 Glaser, 1974
593 Sommerville, 1982:p.*v*; Campbell-Kelly & Aspray, 2004:pp.176-180.
594 Spier, 1974
595 Campbell-Kelly & Aspray, 2004:pp.176-180.
596 Spier, 1974
597 Egan, 1976
598 Simon, 1981 (orig. 1969)

Software Engineering

Systems engineering had started developing when new tools and machines had become so complex that it was no longer possible for a single individual to design them[599]. In systems engineering; the life cycle process, high complexity, and the application of system engineering techniques are recognized throughout the project life cycle[600]. In computer science, the systems approach relies on scientific principles of systemic synthesis rather than on algorithmic solutions[601]. A more specific term *software engineering* was first introduced in 1968 at a conference held to discuss the software crisis[602]. Although software engineering can be considered to be a sub-area of systems engineering in general, it has also contributed to the development of systems engineering.

Note that systems engineering was a response to the problems that arose when the complexity of systems exceeded the skills of a single person. I argue that if one wishes to explain the ontological, epistemological, or methodological assumptions that the designs of a system may incorporate, the emergence of systems engineering signifies a shift from explaining individuals to explaining groups. That is, when one wants to explicate the intentions behind building a complex system, collective intentions and perhaps multiple intentions need to be catered for. The intentions and motivations for constructing the system may be heterogeneous and conflicting.

Ian Sommerville's characterization of software engineering, in his 1982 book *Software Engineering*, reveals the crux of the heated 1970s debate. The debate was about whether software engineering should be considered to be a branch of computer science:

> [1982] *Software engineering is a practical subject. It is concerned with building usable systems economically and, to this end, utilises appropriate, rather than fashionable, techniques. The software engineer should be conservative—he* [sic] *cannot afford to experiment with each and every new technique put forward by research scientists. His principal responsibility is to produce a working system to specification, on time and within budget, and the use of untested methods might compromise this intention. On the*

599 Sage, 1992:p.6.
600 Shemer, 1987
601 Egan, 1976
602 Naur & Randell, 1969

other hand, he or she should not ignore new developments nor should they be rejected simply because they are new.[603]

Sommerville's quote helps to explain why software engineering was rejected by theoretical computer scientists, programmers, and hardware specialists. Sommerville's quote begins with a notion that software engineering is a practical subject that contrasts with the theoretical, mathematically oriented aspects of computing. The need to achieve practical goals probably made it harder for software engineers to gain the acceptance of mathematically oriented theoreticians. According to Sommerville's definition, software engineering also concerns human issues, namely usability—which juxtaposes software engineering with the (practical) electronic engineering and programming aspects of computer science. Sommerville also emphasized the economic motivation, which distances software engineering even further from the mathematical sciences, towards business goals.

Sommerville continued with a relativist argument about appropriate techniques. Substantially, *anything* (that gets the job done) *goes*. However, Sommerville's notion of conservativeness and the disassociation from untested methods differentiates the techniques of software engineering from the techniques of (progressive) *research* (which is the focus of Feyerabend, Kuhn, and Popper). The key words in Sommerville's definition of software engineering are *production*, *operationality*, *time frame*, and *budget*—not, for instance, *theory-creation* or *fact-finding*. The emphasis of Sommerville's definition suggests that a software engineer should be a *bricoleur*[604], an opportunist who has a toolbox full of different tools (methods) that help him or her to accommodate to a variety of situations[605]. Although the *bricoleur*-concept has been lauded by some qualitative research authorities, such as Denzin and Lincoln[606], I doubt that the theoretical computer scientists of the 1970s would have appreciated the bricolage approach to research.

If software engineering is a practical, opportunistic, business-oriented, and close-to-human work (exclusive of the research aspect) as it is described as Sommerville;

603 Sommerville, 1982:pp.2-3.

604 The concept of *researcher-as-bricoleur* is from Denzin and Lincoln, whose description of a competent researcher is very similar to Feyerabend's (compare Denzin and Lincoln, 1994:pp.2-3 with Horgan, 1996:p.52). The concept of *bricolage* is originally from Claude Lévi-Strauss, who used it in a somewhat derogatory sense (Lévi-Strauss, 1966:pp.16-17).

605 Horgan, 1996:p.52.

606 Denzin and Lincoln, 1994:pp.2-3.

then it is easy to see why the computer scientists of the era wanted to dissociate from software engineering. After all, in the early history of computing machinery, computer science had been considered to be a second-class science, mainly because of its practical bent and the impression of computing as a service operation[607]. The computer scientists at the time might have been afraid that acknowledging software engineering as an integral part of computer science could have undermined whatever status computer science had achieved as a scientific discipline.

Sommerville's characterization is certainly not representative of the era's software engineering in general. There was, indeed, no consensus about the definition of software engineering at the time. Nonetheless, Sommerville's definition offers a good example of the different levels of confrontation between what was at the time considered to be computer science and what was considered to be software engineering. This confrontation was sometimes heated and it did not die out easily; for instance, in 1989 Dijkstra wrote that software engineering, "The Doomed Discipline", had accepted as its charter, "how to program if you cannot"[608].

The Focus Turns to Programming

In the previous sections I noted that in the early days of computing, programming was not considered worthy of the university stamp—neither by mathematicians[609] nor by computer scientists[610]. But before the beginning of the 1980s, programming had become established as an integral part of computing as a discipline. Consider the following definition of computer science from a B.Sc program in computer science, introduced by Khalil and Levy in ACM SIGCSE Bulletin;

> [1978] *Our (first order) definition is that computer science is the study of the theory and practice of programming computers. This differs from the most widely used definition by emphasizing programming as the central notion and algorithms as a main theoretical notion supporting programming.*[611]

Although Khalil and Levy are mathematically oriented computer scientists, their definition directly contradicts Knuth's opinion that algorithms are "*the central core*

607 Aspray, 2000
608 Dijkstra, 1989
609 Aspray, 2000
610 Atchison et al., 1968
611 Khalil and Levy, 1978, underlining added.

of the subject and the common denominator which underlies and unifies the different branches"[612].

Khalil and Levy's foregrounding of programming derives, first, from their idea that programming is to computer science as the laboratory is to the physical sciences and, second, from the context of their definition—in their article Khalil and Levy introduced a graduate program in computer science. Very much contrary to Dijkstra's opinion, they believed that one cannot conceive of an understanding of computer science without having experience in programming[613]. Khalil and Levy's position clearly reflects an empiricist *a posteriori* tradition rather than a rationalist *a priori* tradition and it focuses on *techniques and instruments* rather than on *discovering and proving laws* (or designing classes of computations[614]). Khalil and Levy's focus on programming may come from the needs of software engineering during and after the software crisis: Khalil and Levy wrote that the graduates of their program are expected to have an "acceptable" level of *professional expertise*[615]. Professional expertise in Khalil and Levy's definition is mainly programming expertise.

In 1968, the ACM published a recommendation for a four-year program in computer science, a report which later came to be known as *Curriculum '68* (the report in which the curriculum was published is known as the *'68 report*)[616]. The curriculum guidelines encouraged university computer science departments to drop electronics and hardware courses in favor of mathematics and algorithms courses[617]. It has been argued that Curriculum '68 served as a "fundamental source document for the establishment of computer science education in the United States"[618]. However, it has been also claimed that the Curriculum '68 report was of little interest to employers and business practitioners, particularly when compared to alternative curricula advanced by the IEEE or the DPMA[619]. Critics such as Raymond Wishner and Richard Hamming wanted to see topics that are more practical than theoretical included in Curriculum '68—they wrote in CACM and JACM,

612 Knuth, 1974. Remember Knuth is a mathematically-oriented computer scientist like Khalil and Levy.
613 Khalil and Levy, 1978
614 Dijkstra, 1972
615 Khalil and Levy, 1978
616 Atchison et al., 1968
617 Ensmenger, 2001
618 Austing et al., 1977
619 Ensmenger, 2001; Wishner, 1968; Hamming, 1969

> [1968] [...] *Good education should not be solely directed towards academicians whose only economic justification is to teach in order to turn out recursively new generations of academicians.*[620]
>
> [1969] *Were I setting up a computer science program, I would give relatively more emphasis to laboratory work than does Curriculum '68, and in particular I would require every computer science major, undergraduate or graduate, to take a laboratory course in which he* [sic] *designs, builds, debugs, and documents a reasonably sized program.*[621]

Both Wishner and Hamming criticized the lack of emphasis on practical issues of computing. Wishner argued that the Curriculum '68 addresses the needs of physical scientists and engineers, but that it does not address the needs of business-systems designers and information technologists. Even though the ACM did recognize the growing importance of meeting business needs to the future of computing, the emphasis of the ACM was on research and education[622]. It was argued, for instance, that the ACM wanted to see *fundamental* research in the field of data processing before the ACM would recognize business data processing as a topic of research[623].

About ten years after the ACM had published Curriculum '68, the ACM Curriculum Committee on Computer Science published an update to Curriculum '68: *Curriculum '78: Recommendations for the Undergraduate Program in Computer Science*[624] (the report in which the curriculum was published is known as the *'78 report*). During the ten years between 1968 and 1978, the definition of the field had changed from a mathematically oriented definition to a more diverse definition. During that decade there were major advances in the theory of computation, algorithm analysis, and in the principles and theories for the design and verification of algorithms and programs. At the turn of the 1980s many computer scientists, like Anthony Ralston and Mary Shaw, believed that "*there is nothing laughable about calling computer science a science* [anymore]"[625].

The '68 report and '78 report seem to have both normative and descriptive characteristics in the sense that they recommend what a computer scientist should know by

620 Wishner, 1968
621 Hamming, 1969
622 Ensmenger, 2001
623 Postley, 1960
624 Austing et al., 1979. Though the publication is from March 1979, the report is named Curriculum '78.
625 Ralston and Shaw, 1980

listing what computer scientists do. However, Goldweber et al. argued that soon after the '78 report, the curriculum recommendations became descriptive (Goldweber et al. called them "reactive")[626]. Similar to the '68 report, the '78 report defines the discipline by listing the topics included in the discipline. The '78 report, however, does not define the discipline as strictly as the '68 report, because the authors of the '78 report recognized that the report "*is a set of guidelines, prepared by a group of individuals working in a committee mode*"[627]. In the report, the committee remarked that they did not expect the report to satisfy everyone or intend it to be appropriate for all institutions.

The '78 report is not just *any* curriculum proposition—it was directed at the whole academic field of computing. The '78 report was the effort of a large number of recognized individuals and institutions[628], and, as such, it had certain authority. However, the authors of the '78 report acknowledged that they had to leave a lot to interpretation, that there was a degree of subjectivity in the committee decisions, and that different educational institutions had different aims. Whereas the aim of the '68 committee was to specify a number of course combinations that would entitle a student to receive a degree in computer science[629], the objective of the '78 report committee was to "stimulate computer science educators to think about their programs"[630]. The difference between the motivations and the content of the '68 report and the '78 report is a sign of the dispersion of the discipline—it was no longer possible to define what a computer scientist actually does.

In both the '68 report and the '78 report, computer science is divided into subareas. The '68 report divides computer science into three subareas;

(a) *information structures and processes*,

(b) *information processing systems*, and

(c) *methodologies*.

The '78 report report divides computer science into four subareas;

1) *programming topics*,

626 Goldweber et al., 1997
627 Austing et al., 1979
628 Austing et al., 1979
629 Atchison et al., 1968
630 Austing et al., 1979

2) *software organization*,

3) *hardware organization*, and

4) *data structures and file processing*.

Note the lack of a distinct subarea, *theoretical foundations*, that many computer scientists might consider important. In the '68 report, models of computation is a subtopic of its own, but not in the '78 report. Theoretical topics, such as grammars, automata, and complexity, in the '78 report are scattered amongst other topics. Other subsequent curricula, which have been published in about ten-year intervals, follow the same convention (see, for instance, the nine subject areas of Curriculum '91[631]). My interpretation is that curriculum developers have considered theoretical topics so central to computing that they need to pervade the whole curriculum.

What is also noticeable in the '78 report, compared to the '68 report, is the emphasis on hands-on work. The authors of the '78 report wrote, "*throughout the presentation of the elementary level material, programming projects should be assigned*"[632]. The emphasis on programming is even more visible in the description of *philosophy of the discipline* in the '78 report;

> [1978] *A specific course on structured programming or on programming style, is not intended at the elementary level. The topics are of such importance that they should be considered a common thread throughout the entire curriculum and, as such, should be totally integrated into the curriculum. They provide a philosophy of the discipline, which pervades all of the course work.*[633]

Over the course of ten years, the definition of computer science in the ACM curriculum turned from a theoretical, mathematically based discipline that studies information structures into a programming and applications-centered discipline. Although the topics in the '68 report and '78 report are quite similar, the focus had definitely shifted between the '68 report and the '78 report. Even the "philosophy of the discipline" had changed from information structures into structured programming and programming style. It may be a coincidence that the computer science curriculum, including the philosophy of the discipline, shifted towards the needs of the

631 Tucker et al., 1991
632 Austing et al., 1979
633 Austing et al., 1979

software industry during the years of software crisis. However, because the '78 report is more of a descriptive than a normative account of the computing field, it surely seems that the curriculum committee was just trying to keep pace with the changes in computing practices[634].

Separation from Mathematics

Whereas the critics of the '68 report criticized the '68 report for being too academic, too theoretical, too narrow, and too impractical, the critics of '78 report criticized the '78 report for lacking mathematics and for implicitly stating that "*computer science = programming*"[635]. The difference between the emphasis on mathematics in the two reports is indeed noticeable. Whereas the authors of the '68 report stood firmly behind the mathematical viewpoint of computing, the authors of the '78 report did not see mathematics as the cornerstone of computer science. The following two quotes from '68 report and '78 *preliminary* report exemplify the shift of focus well:

> [1968] *The committee feels that an academic program in computer science must be well based in mathematics since computer science draws so heavily upon mathematical ideas and methods.*[636]

> [1977] [...] *no mathematical background beyond the ability to perform simple algebraic manipulation is a prerequisite to an understanding of the topics* [...] *As was mentioned in the section on the core curriculum, mathematics is not required as a prerequisite for any of that material.*[637]

However, the advocates of mathematically based computer science[638] succeeded to change the above mentioned part of the '78 preliminary report, and the final '78 report included a more conventional wording:

> [1978] *An understanding of and the capability to use a number of mathematical concepts and techniques are vitally important for a computer scientist.*[639]

634 Although they interpret the '78 report as a normative curriculum, Goldweber et al. noted that the '78 report was a reaction towards the rapidly changing field of computing (Goldweber et al., 1997).
635 Ralston and Shaw, 1980
636 Atchison et al., 1968
637 Austing et al., 1977b
638 See Ralston and Shaw, 1980; Davis, 1977.
639 Austing et al., 1979

The leap from a science that is "*well based in mathematics*" to a science where "*no mathematical background beyond the ability to perform simple algebraic manipulation is needed*" would have marked a complete detachment from the mathematical history of computing. The final wording was much more conventional, mentioning the *vital importance* of a number of mathematical concepts and techniques. As computer science began to achieve a a disciplinary identity, the institutional ties between mathematics and computer science weakened steadily. In the August 1981 issue of *American Mathematical Monthly*, Anthony Ralston explained the reason for the divergence of the two disciplines during the 1970s, with four arguments[640].

First, the importance of some of the traditional mathematical areas, such as numerical analysis, decreased in computer science. (Note that the history of numerical analysis is hundreds of years old and that in numerical analysis computers are mostly tools and not a topic of research as such.)

Second, Ralston argued that the difficulties that computer science had faced in being recognized as a separate discipline from mathematics urged many computer scientists to fight for the formation of departments of computer science separate from departments of mathematics. Insofar as Ralston's second argument is correct, the separation between mathematics and computer science was partly a result of professional pride.

Third, Ralston noted that from the early 1970s, the composition of computer science faculties begun to shift from predominantly mathematicians to predominantly computer scientists. This shift seems quite natural. As the first PhD-granting department of computer science was founded 1962 and the first PhD was awarded in 1966, during the early 1970s there could not have been many computer science PhDs around to fill the faculty positions. As the number of PhDs in computer science grew, it seems logical that the departments of computer science were increasingly able to employ computer scientists instead of mathematicians.

Fourth, Ralston argued that people at mathematics departments were not very hospitable to the ideas and techniques of computer science. Ralston criticized the "computer science = programming"-outlook, and spoke up for mathematics-based computer science. Ralston wrote with Mary Shaw, "*inevitably, for any science or any*

640 Ralston, 1981

engineering discipline, the fundamental principles and theories can only be understood through the medium of mathematics"[641].

In short, Ralston gave four reasons for the field of computer science's divergence from the field of mathematics: Disciplinary changes, an insecure disciplinary identity, a growing number of people in the field of computing, and disciplinary disagreements. Only the first one of Ralston's four arguments actually concerns the academic and intellectual aspects of computer science. The other three arguments are perhaps better explained by psychological, sociological, or anthropological aspects. For instance, Ralston's second argument can be explained as a resistance towards an old adversary; Ralston's third argument can be explained as a growth of the number of computer scientists and their placing in working life; and Ralston's fourth argument can be explained as a difference between the cultures of abstractly oriented mathematicians and practically oriented computer scientists.

Throughout the 1970s, descriptions and definitions of computer science increasingly came to include practical issues. Computer science, as such, was claimed to be a legitimized, mature discipline[642], and the legitimization pressures moved increasingly to subareas of computer science (such as software engineering[643]). Yet, during the years between the early 1960s and the early 1970s the ones who worked with computers were mostly professionals in computer programming. Jonathan Grudin wrote that through the 60s and mid-70s, improving *usability* still meant improving *programmer efficiency*[644]. But during the 1970s, a significant change in computing took place. This change, which had a significant impact on both the form and the content of the field, was due to changes in the user base: The computer broke out of the laboratory; it became available to Western office workers and to the general public.

641 Ralston and Shaw, 1980
642 Ralston and Shaw, 1980
643 See, e.g., Pour et al., 2000; Holloway, 1995.
644 Grudin, 1990

Emerging Interdisciplinarity

Life was simple before World War II. After that, we had systems.[645]

In This Section:

- ✔ What factors have led to the growing interdisciplinarity of computing as a discipline?
- ✔ What is the status of the von Neumann Architecture and the Turing Machine?
- ✔ Where does the complexity of computer systems come from?
- ✔ How can one cope with complexity?
- ✔ To whom are computer scientists accountable?

In the beginning of the 1970s, computer programmers remained the principal users of computers[646]. *Human*-computer interaction was not yet a research field of its own, but there was a significant amount of research on the psychology of computer programming and *programmer*-computer interaction[647]. Between 1965 and 1975 integrated circuit electronics[648] reduced the cost of computer power by a factor of a hundred[649]. The reduced costs led to the computer breaking out of the laboratory, which changed the user base dramatically. The new users of computing technology were no longer committed to the technology per se.

Paul Ceruzzi noted that it is not clear, however, to what extent personal computing was simply the result of a natural outcome of advances in semiconductor technology or whether it was the result of a conscious effort to effect a social transformation of computing[650]. The development of personal computing is too recent a phenomenon to get a proper perspective of[651]. The development of personal computing is far from being over, and the long-term consequences of personal computing are still speculative. Because this thesis focuses on the development of computing and not on the consequences of it, in this section I discuss the transformation of computing during the shift in the user base, rather than the consequences of it.

645 Attributed to Grace Hopper in Schieber, 1987.
646 Grudin, 1990
647 Baecker et al., 1995:p.41.
648 See Figure 18 on page 234 about the "third generation" of computers.
649 Campbell-Kelly & Aspray, 2004:p.198.
650 Ceruzzi, 1999
651 Campbell-Kelly & Aspray, 2004:p.207.

Shifts in the User Base, Status, and Content of Computing

As the ration of programmers to other computer users became increasingly smaller, the psychology of programming became less central to the studies of interaction between humans and computers[652]. The dramatic shift in the user base during the 1970s had an equally dramatic effect on the academic discipline of computing. Whole fields of study such as HCI (or CHI)[653], (management) information systems[654], operating systems[655], and networks[656] were born and the field of computing was again stretched in a number of directions. This era shaped computing into a truly multi- or interdisciplinary field.

The media had, since the birth of computers, portrayed computers to the general public with phrases such as "the robot Einstein" and the "tireless ally of science"[657]. In 1959 some visionaries had already understood that computers are not just tools to replace human computers, but that they offer some unprecedented opportunities—Herb Grosch wrote in the Datamation,

> [1959] *The dream is wider than to produce a slick matrix inversion routine or to simulate a* [Burroughs] *205* [computer] *on a* [Burroughs] *220. It's to remodel the whole world of the future with a tool unequaled for challenge since the invention of the alphabet, to amplify human intelligence by organizing and rationalizing the staggering flow of information that is the nervous system of society*.[658]

Yet, it still took some fifteen to twenty years before researchers at large stopped regarding computers as merely calculators. Towards the end of the 1970s an understanding that computer science is indeed an interdisciplinary science, and not easily definable, emerged. In the Second International Conference on Software Engineering, Peter Wegner wrote a progressive characterization of computing;

652 Baecker et al., 1995:p.41.
653 Baecker et al., 1995:pp.38-43.
654 Baskerville et al., 2000:p.63; Alavi & Carlson, 1992.
655 Silberschatz et al. noted that operating systems have a longer history (Silberschatz et al., 2002:p.11), but Denning argued that in the early 1970s, operating systems as a subfield of computer science made the transition from a poorly understood set of techniques to a well-understood set of core principles (Denning, 1985).
656 Denning, 1985
657 Bowles, 1996
658 Grosch, 1959

> [1976] *Computer science is in part a scientific discipline concerned with the empirical study of a class of phenomena*[659]*, in part a mathematical discipline concerned with the formal properties of certain classes of abstract structures, and in part a technological discipline concerned with the cost-effective design and construction of commercially and socially valuable products.*[660]

Peter Wegner's description in the passage above does not emphasize only one viewpoint of computing. Wegner drew content from connections with empirical, mathematical, and engineering traditions. He extended the "phenomena" of computer science to include "*other* [hu]*man-made entities and concepts which owe their existence to the development of computers*"[661], which could be interpreted as an open invitation for interdisciplinary studies. Under a somewhat loose interpretation of Wegner's description, studies of individual phenomena, such as net addiction[662]; interpersonal phenomena, such as net date and virtual rape[663]; social phenomena, such as virtual communities[664]; or perhaps even abstract phenomena, such as information society[665] and collective intelligence[666] would all belong to the field of computer science. Loosely read, Wegner's description of computer science would be equal to the classic Newell et al.'s 1967 description of computer science as the "*phenomena surrounding computers*"[667].

However, because neither Wegner nor Newell et al. could have predicted the phenomenal changes in computing and its uses, it is perhaps best not to celebrate their statements as wide-open or all-embracing normative accounts of computer science. Wegner, for one, interpreted Newell et al.'s "phenomena" as algorithms, programs, programming languages, and such[668]. Sociocultural phenomena are not explicitly or implicitly included in either definition. Nevertheless, although Wegner's definition was probably not meant as broadly as a modern-day reader might interpret it, it still

659 The "phenomena" of computer science include digital computers, programming languages, algorithms, programs, and other human-made entities and concepts which owe their existence to the development of computers (Wegner, 1976).
660 Wegner, 1976–this definition will be compared later with one by Denning et al., 1989. Underlining added.
661 Wegner, 1976
662 Young, 1998
663 Civin, 2000:pp.93-95.
664 See, e.g., Rheingold, 2003.
665 Castells, 1996; Castells, 1997; Castells, 1998
666 See, e.g., Lévy, 1997.
667 Newell et al., 1967
668 Wegner, 1976

explicitly stresses an interplay of empirical research, formal theories, and design and implementation, all of which are considered cornerstones of computing nowadays.

One of the founding fathers of artificial intelligence, Allen Newell, argued later that information processing will change *all disciplines*, not only computer science[669]. In 1985 Newell claimed that scientists should be prepared for "*some radical, and perhaps surprising, transformations of the disciplinary structure of science (technology included) as information processing pervades it*". Newell's argument was that scientists in the future would no longer do *object-level* research such as observing, experimenting, theorizing, testing, and archiving but they would do *meta-level* research. In order to rise to the meta-level, scientists would need to understand activities of science so that they could make systems that can automatically do the trivial object-level research. In a sense, Newell's prediction has come true in computational sciences.

Societal Conscience Awakens

As the sophistication of computing technology increased, so did the anxiety about the effects of computing technology on society. In 1969, Richard Hamming noted in his Turing Award lecture,

> [1969] *We know that in this modern, complex world we must turn out people who can play responsible major roles in our changing society, or else we must acknowledge that we have failed in our duty as teachers and leaders in this exciting, important field—computer science.*[670]

The changes in attitude were soon reflected in computer science education. Already in 1972, when computers were not yet available to the general public[671], Horowitz, Lee, and Shaw proposed a *Computers and Society* course for computer scientists[672]. They concluded that if research is done without concern for its influences *on all aspects of society*, that research can become a destructive rather than constructive mechanism. Horowitz, Lee, and Shaw gave examples of technologies that have had both positive and negative effects, such as the automobile, nuclear power, and pesti-

669 Attributed to Newell in Bobrow & Hayes, 1985.

670 Hamming, 1969

671 Paul Ceruzzi claimed that personal computing (the definition of which includes both technology and culture) was invented in 1974 (Ceruzzi, 1999:p.72).

672 Horowitz et al., 1972

cides[673]. The content of their proposed course is broad and also very similar to the issues that are troubling today (including political, economic, cultural, social, and moral issues).

The emerging trend of examining the relationship between computers and society was evident in Khalil and Levy's curriculum of computer science[674]. They wrote, "*In the spirit of our times, an awareness of the social issues and controversies is also important for the well-educated person*". For instance, starting as early as the 1970s, *CACM* had had extensive debates over the social responsibilities of computing professionals[675]. The status of social, philosophical, and ethical considerations in curricula changed during the years between 1968 and 1978. Whereas the '68 report states that "[social] *issues are not the exclusive or even the major responsibility of computer science*", the '78 report regards social, philosophical, and ethical issues "*of such importance to computer scientist that they must permeate the instruction at this level.*"[676]. Note, however, that those social, philosophical, and ethical issues concern only the societal effects of introducing computing technology and not sociologically or philosophically oriented meta-research of computer science.

The Limits of the Model Set by Turing and von Neumann

Social concerns were not the only concerns that the increasing speed of computers brought about. In 1981, John Backus, who had been the head developer of FORTRAN starting from 1954, expressed his pessimism about the programming tools of the 1980s[677]. Backus stated that while it was perhaps natural and inevitable that languages like FORTRAN should have developed out of the concept of the von Neumann computer as they did, the fact that such languages have dominated computer scientists' minds ever since is unfortunate. In Backus' opinion, people have come to regard the DO, FOR, and WHILE statements, and the like, as powerful tools, whereas they are, in fact, weak palliatives that are necessary to make the primitive von Neumann style of programming viable at all—Backus wrote in the book *History of Programming Languages*,

673 Horowitz et al., 1972
674 Khalil and Levy, 1978
675 See, for instance, the issues of *CACM,* July 1973 (a large debate on ethical code), and May 1974 (a large debate on social and political questions).
676 Atchison et al., 1968; Austing et al., 1979
677 Backus, 1981:p.43.

> [1981] [The] *"von Neumann languages"* [such as ALGOL and Backus' own creation FORTRAN] *create enormous, unnecessary roadblocks in thinking about programs and in creating higher level combining forms required in a really powerful programming methodology*. [...] *It is unfortunate because their long-standing familiarity will make it hard for us to understand and adopt new programming styles which one day will offer far greater intellectual and computational power.*[678]

In the light of new or experimental programming paradigms; such as object-oriented programming, message-passing programming, reflective programming, and empirical modeling; Backus seems to have been correct. Backus argued that once accustomed to one programming paradigm, it may not be very easy to switch over to another. It must be noted, though, that von Neumann's decisions were tied to the state-of-the-art of von Neumann's time. Von Neumann *did* recognize the advantages of parallel computation, but architectural design of the time was incapable of delivering a parallel system, and techniques of multiprogramming had not yet been developed[679].

When Donald Knuth was working on what were to become the most applauded works in computer science, he faced a choice that dealt with Backus' concern: Knuth needed to choose a programming language for the examples in his book *The Art of Computer Programming*. Knuth made an original choice; the reference language in his book series is a symbolic machine language called "MIX"[680]. Knuth stated that in his book series, the choice of language was the hardest decision to make. He gave six reasons for choosing the assembly language MIX[681].

First, Knuth argued that with a machine-oriented language a programmer will not be influenced by the design decisions of *any particular language*. Second, the programs in the book are rather *compact*, and as such, expressible in assembly language. Third, high-level languages are inadequate for discussing important *low-level details*. Fourth, Knuth argued that those who are more than casually interested in computers, should be schooled in machine language, since it is a *fundamental* part of a

678 Backus, 1981:p.43.
679 Lee, 1996
680 Knuth, 1968
681 See Knuth, 1968, or Knuth, 1997:pp.*viii-ix*. These reasons change between editions of Knuth's book. In the 2nd edition, Knuth noted that algebraic languages are more suited to numerical problems than to the nonnumerical problems–in the 3rd edition this is not mentioned.

computer. Fifth, some machine language would be necessary anyway as *output of the software programs* in many examples[682]. Sixth, new algebraic languages go in and out of fashion, but Knuth wanted to emphasize concepts that are *timeless*.

Knuth's first, third, and sixth arguments are apparently an aim at *universality*; however, the MIX symbolic assembly language is universal only to sequential von Neumann-architecture designs (and not universal for all automatic computing, cf. e.g., ZISC architecture[683]). Because experimental or unimagined future technologies may be incommensurable with von Neumann-architecture, MIX cannot be taken as universal in any true sense of the word[684]. For instance, John Backus called von Neumann-architecture primitive and defective[685]. Although Knuth's algorithm examples are not influenced by any particular high-level programming language, they are definitely influenced by one particular model of computation and a particular assembly language. Note an interesting point: Knuth argued that using assembly language is "much closer to reality"[686] than using any high level language. He also noted that assembly language is a fundamental part of a computer, and that assembly language concepts are timeless[687]. The emphasis that Knuth gave to assembly language operations suggests that Knuth took them as the non-divisible basic building blocks of computer programming (or *atomic operations*—from Greek $\alpha\tau o\mu o\nu$, indivisible).

It would be naïve to think that von Neumann-architecture and Turing-computability would, in someways, be the last word for machine computability, and I am not certain that many computer scientists would even claim that they are. Copeland and Proudfoot wrote that the aim of Turing's 1936 paper[688] was not to explain the limits of machine computation, but to specify the simplest machine that can perform any calculation that can be performed by a human mathematician who has unlimited time, and who works with paper and pencil in accordance with some "rule-of-

682 The *emulator*, *trace*, and *monitor routines* discussed in Chapter 1 of the first volume in Knuth's book series (Knuth, 1997) deal with machine language. Chapter 12 (Programming Language Translation) in one of the forthcoming volumes of Knuth's book series will probably deal with machine language input and output.

683 Zero Instruction Set Computer (ZISC) imitates a neural network and it does not have a separation between CPU and data. See Madani et al., 2001 for a brief explanation to ZISC-036 processor.

684 Ultimately *universal* means something that is non-spatial, non-temporal, and absolutely epistemologically objective.

685 Backus, 1978. In this Turing Award lecture, Backus introduced the term "*von Neumann bottleneck*".

686 Knuth, 1997:p.*ix*.

687 Knuth, 1997:p.*ix*.

688 Turing, 1936

thumb" or rote method[689]. Turing himself went on to investigate the idea of machines, which he called *o*-machines (oracle machines), that can calculate mathematical tasks that the Universal Turing Machine (UTM) cannot[690]. Wegner and Goldin argued that later, Turing considered also *c*-machines (choice machines), which added interactive choice to computation as well as *u*-machines (unorganized machines), which he planned for modeling the brain[691]. Piccinini wrote that according to Turing, there is no upper bound to the number of mathematical truths provable by intelligent human beings, because they can invent new rules and methods of proof[692]. Unlike the output of a machine, for which the rules have to be known, the output of a human mathematician is not a computable sequence[693]. Following Turing's *o*-machines a number of *hypercomputational* models[694] have been introduced in fields such as neural computing[695] and analog computation[696] (yet there is a debate over whether hypercomputation is indeed possible).

Richard Hamming questioned how long the Turing Machine model (or the von Neumann-model) will last in the area of algorithms, since, from Hamming's point of view, the Turing Machine model no longer modeled modern hardware well[697]. Hamming noted a number of notions in classical algorithm books that are out of date with modern hardware. For example, modern chips do not have "halt" instruction and their cache memories and pipelines can greatly alter the running-time estimates of algorithms, but the cache memories and pipelines are not under the control of the programmer. Whereas Knuth wrote in the *Art of Computer Programming* that assembly concepts are timeless, in the later editions he also wrote that it must be admitted that MIX is now quite obsolete and that MIX will be replaced with MMIX in the future editions of the book[698]. Assembly languages *do* change as machine architectures

689 Copeland & Proudfoot, 2000. Note that although an *algorithm* needs to halt at some point of its execution (Knuth, 1997:pp.5-7), in order to be *computable* in the sense of the word used by Turing (Turing, 1936), a computable process does not need to halt. Note also that unlimited time also assumes an unlimited life span for the human mathematician.

690 Turing, 1939

691 Wegner and Goldin, 2003

692 Piccinini, 2003

693 Piccinini, 2003

694 B. Jack Copeland called computation of functions or numbers that cannot be computed in the sense that Turing (Turing, 1936) means it, *hypercomputation* (Copeland, 2002). Copeland gave a number of examples of models of hypercomputation. See also Wegner and Goldin, 2003.

695 Siegelmann, 2003

696 MacLennan, 2003

697 See Hamming, 1997.

698 Knuth, 1997:pp.*ix,*124.

change, and essentially, nothing in the history of computing indicates that either languages or architectures are timeless.

I take it that von Neumann-architecture has gained enough technological momentum so that it is, despite all of its limits, nowadays largely taken as a paradigm, as an unquestioned foundation, for successful computation. There are, however, serious attempts to break the barriers set by von Neumann-architecture and Turing-computability. Because of the special character of computer science as a human-made endeavor, these attempts are not responses to anomalies in the paradigm (which is based on von Neumann-architecture and Turing-computability), but calculated attempts to break the limits of the paradigm[699]. In a sense, Backus' critique of the von Neumann-architecture[700] and Knuth's choice of assembly language[701] point towards the awkwardness of the concept of *programming paradigms*—programming paradigms are not paradigms in the Kuhnian sense because new programming paradigms do not render old programming paradigms obsolete. Calling approaches to programming *paradigms* is confusing and ungrounded; for instance, Knuth's choice, assembly language, can co-exist with whatever approach Backus had in mind.

Despite the fundamental importance of theoretical issues, when computers increasingly spread to different disciplines and areas of work, understanding the fundamentals of computation may *not* be a primary concern for most working people. Rather, high-level and heavily context-dependent *abstractional* languages will be required in order to empower workers (who use computers merely as tools) to program or reprogram them. Knuth's book series is, in the end, meant for people who want to understand computing and computers and not for those who use them as tools.

699 See Copeland, 2002; Copeland & Sylvan, 1999.
700 Backus, 1978
701 Knuth, 1968

The Complexity[702] of Computer Systems

In 1969, Herbert A. Simon wrote in his book *The Sciences of the Artificial*,

> [1969] *An ant, viewed as a behaving system, is quite simple. The apparent complexity of its behavior over time is largely a reflection of the complexity of the environment in which it finds itself.*[703]

A computer, vis-à-vis an ant, is an extremely simple system, or at least its functioning is reducible to a small number of extremely simple parts that function in a simple and straightforward manner. The entities on each level of abstraction in computer science (at large) work in a trivial manner. The apparent complexity of computing and the computer is largely due to the complexity of the semantic interconnections *between* abstraction levels in computing. In the following two subsections, I clarify this argument. Note that on the following pages, Figures 22-24 are simplifications of reality, and they are only meant to be suggestive sketches about the complexity of computer systems—the reality of computer systems is even more complex. After I have briefly discussed a number of abstraction levels in computer science, I examine the possible sources of complexity between the highest and lowest levels of abstraction.

First, the operation of *logic gates* is simple and straightforward. However, when large numbers of logic gates are cascaded to implement *logic elements* such as adders, multiplexers, and multipliers (see Figure 22), there are different implementation choices for each, some of them simpler, some more complex[704].

---

702 *Complexity* in the field of computing is an ambiguous term. For instance, *computational complexity* refers, generally, to how much time and space a certain computation requires and *hierarchical complexity* refers often to the number, variety, and interrelations of a system's elements (see Rescher, 1998:p.1; Simon, 1981). In addition, in *holistic systems* complexity refers to the unpredictability of a system due to emergent features of the system's parts, the large number of interactions in the system, and the autonomous independence of the behavior of the parts (see Rescher, 1998:p.2). Here the term *complexity* refers to a hierarchical complexity of a system and to its unpredictability over time. A large number of different measures of complexity (about 45) can be found in Horgan, 1996:p.303. See also a taxonomy of modes of complexity in Rescher, 1998:p.9. What is more, sociologist John Urry noted that metaphors, theories, and concepts of complexity are also moving from natural sciences to the field of sociology (Urry, 2004).

703 Simon, 1981:p.64 (orig. 1969). Although Simon elides an exact definition of complexity (Simon, 1981:p.195), his term *hierarchical complexity* is used in a way that it refers to a system's complexity, both in organizational and in operational terms.

704 Hennessy & Patterson, 1996:pp."A-38"-"A-60".

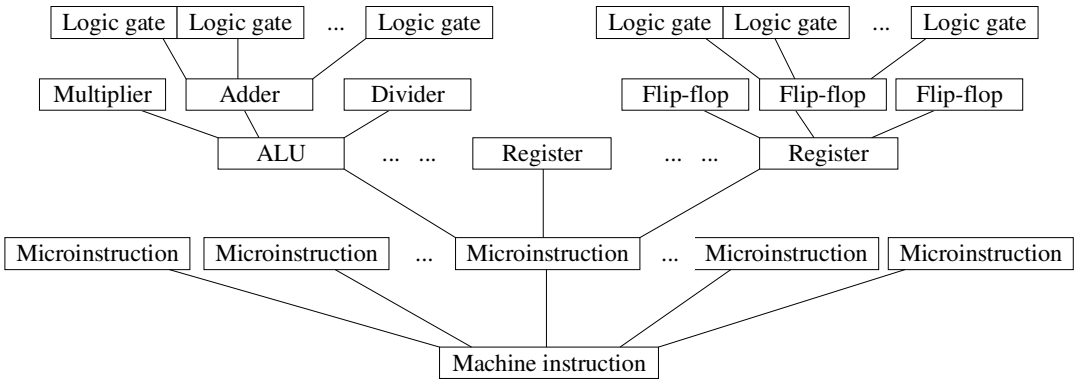


Figure 22: Abstraction Layers at the Machine Level

Second, the operation of each of the *logic elements* is simple, but when they are coupled together to form parts of an *execution unit*—parts such as ALUs (Arithmetic Logic Units), registers, and FPUs (Floating Point Units)—the connections between logic elements and an execution unit become numerous and complex.

Third, the operation of the parts of an *execution unit* (ALUs, registers, and such) is simple and straightforward, but the number of micro-operations (each of which in, e.g., CISC (Complex Instruction Set Computer) architecture, utilize parts of the execution unit) per *microinstruction* is often large. What makes things more complex in modern machines is that they often utilize superscalar processors (which have a number of ALUs, FPUs, etc.), and that they are sped up by pipelining.

Fourth, the number of *microinstructions* per *machine instruction* is also large in typical CISC machines. That is, a single machine instruction (macroinstruction) usually results in a number of microinstruction calls[705]. Again, things are made more complex by the fact that some modern processors are internally RISC (Reduced Instruction Set Computer), but nevertheless emulate a CISC architecture.

Fifth, *machine instructions*, as such, are simple and unambiguous. However, the number of machine instructions that *statements*, *expressions*, and the like in high-level programming languages produce is large (see Figure 23) and their connection with statements is complex, especially in code that is optimized in some way. Note that in reality this picture is further complicated by the fact that the *operating system* usually works as an additional layer between machine instructions and high-level language[706].

705 Clements, 2000:p.232.
706 Clements, 2000:p.213.

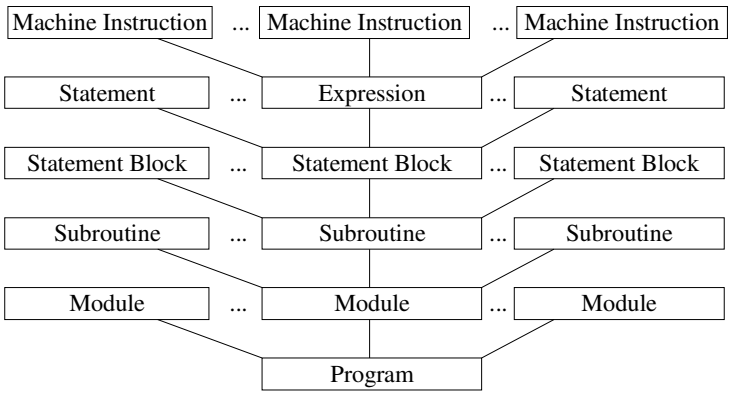


Figure 23: Abstraction Layers on a Language Level

Sixth, *statements* and *expressions* in imperative programming languages like FORTRAN are simple and unambiguous as such, but *statement blocks*, including loops, conditional branches, and the like, are more complex and prone to errors.

Seventh, in procedural programming, *statement blocks* form *functions* or *subroutines*. Although the contents of statement blocks can be syntactically correct, complexity arises from their use. Incorrect use of statement blocks—a lack, misplacement, or misuse of an initialization loop; common variables or other interconnections between different blocks; redundant blocks; and other problems with statement blocks—can cause confusion and errors.

Eighth, *functions* and *subroutines* usually form *modules*, libraries, or other encapsulated entities. Although every single subroutine can be correct, their incorrect use or incompatibilities between subroutines (e.g., errors with semantics, boundaries, error-checking, unit disparities, or functioning) are the usual causes of errors. Again, the errors are not due to the complexity of the subroutines (they are actually simple entities with given input, output, boundaries, and other restrictions), but the errors are due to their complex interrelations when put together.

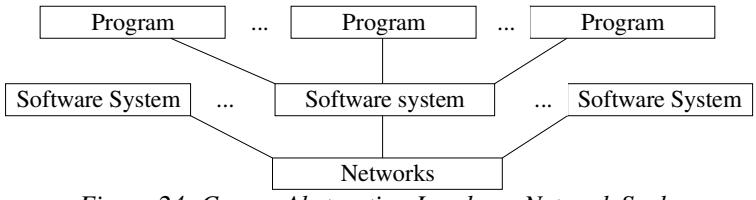


Figure 24: Coarse Abstraction Levels on Network Scale

Ninth, *modules*, *libraries*, and the like, constitute *computer programs.* Although a module may work perfectly in one program, it may not do so in another. Two mod-

ules may have incompatibilities due to their side-effects, and these incompatibilities may not show any signs before the two modules are used together. Furthermore, the underlying mechanisms in modules may have limitations that only materialize in complex programs. For instance, Frederick P. Brooks, Jr., wrote in his book *The Mythical Man-Month* about the difficulties that arise in large, complex programs and complex systems[707].

Tenth, full operating systems and other *software systems* (see Figure 24) include a number of more or less interdependent *computer programs*. For instance, the router at my home, the piece of hardware that forwards data packets between my computers and the Internet, runs 56 programs (or tasks) simultaneously. Although a single program may be simple and unambiguous, large collections of co-operating and concurrent programs exhibit vast complexity. For instance, Michael J. Spier visualized the operating systems of the early 1970s as "a messy ball of spaghetti"[708]. Note that there is a number of abstraction layers, firstly, between programs, and secondly, between programs and the operating system, but, for simplicity's sake, they are not included in Figure 24.

Eleventh, there are *networks* of computers, like the Internet, where each computer can run different sets of *software systems*, yet all the connected computers need to be interoperable to some degree. There is actually a large number of abstraction layers between a software system and a network of computers[709], but these layers are also omitted from Figure 24 for the sake of brevity.

Although many complexities are excluded from this description of computer systems —such as complexities that arise from data structures, real-time systems, interfaces with an external environment, intermediate abstraction layers, data semantics, shared objects, and so forth—the underlying complexity in this description is enormous. Yet, the functional and structural definitions of the objects on every single level of abstraction are simple. The examples above show that the complexity of computer systems comes from the semantic connections between abstraction levels. Further-

707 Brooks, 1975

708 Spier, 1974

709 For instance, the Open Systems Interconnection Reference Model, OSI model, has seven functional layers–physical, data link, network, transport, session, presentation, and application layers (see ISO standard 7498-1, available at `www.iso.org`).

more, the structure of this complexity is not an inherent structure of the world, but it is a human construction—this is clarified in the following subsection.

The Sources of Complexity

There are a number of very practical questions that this complexity creates. For instance, the term *implementation level* belongs to common computer science language. Yet one might ask, "What exactly *is* the implementation level?". Between the top and bottom levels in Figures 22-24 (logic gates and computer networks) there are a number of intermediate abstractions, so it is not clear where exactly the *abstraction boundaries* should be drawn. (Brian Cantwell Smith interpreted these issues as ontological questions[710].) If one considers the microinstruction level, especially in modern hardware, it is even difficult to say what is *hardware* and what is *software*[711].

The interconnectedness of each of the abstraction levels makes design difficult. For instance, in a system with four semantic levels *w-x—y-z*, often when the semantic gap between levels *x* and *y* is narrowed, the gap between levels *w* and *x*, as well as *y* and *z*, may widen (*w—x-y—z*). It is a difficult question whether a wide semantic gap, that is, complex semantic interconnections between two levels, should be preferred to a large number of semantic levels.

In Searle's ontology[712], the chemical and physical behavior of the substances that *logic gates* are made of are brute facts. Once manufactured, the existence and properties of logic gates are independent of humans—they are ontologically objective facts. It is a debatable matter whether axioms in Boolean logic (on which logic gates are founded upon) are ontologically objective facts or not. The ontological status of axioms is irrelevant in the argument concerning the social construction of computer systems anyway. That is because when logic gates are combined to create complex logic elements such as multipliers, the decisions of *how to combine them* are made by a number of arguments other than brute facts, logic, or physics, as discussed in the following paragraph. Note that even the binary system is not chosen for com-

710 Smith, 1998:p.27.
711 James H. Moor noted the problem of *hardware-software*-division in Moor, 1978.
712 Searle, 1996:p.7.

puters because of its superiority over other numeral systems, but because of the ease and low cost of manufacturing binary elements[713].

When logic gates are coupled to make *logic elements*, there is a number of possible implementations, but none of them is optimal on all terms. Multipliers, for one, can be implemented in a number of ways, including the Wallace tree, Booth's algorithm, look-up tables, and so forth[714]. The implementation of a multiplier is a trade-off between different kinds of aspects, and, as a result, some machines form products by using simple shift-and-add operations, whereas some expensive, high-speed computers use very large logic arrays involving hundreds of gates[715]. Different kinds of needs result in different implementations. Adders, too, can be implemented by combining logic gates in different ways, and all combinations have different benefits[716]. The number of possible designs of an adder is, theoretically, infinite.

Alan Clements wrote that the design criteria of logic circuits are basically speed, the number of interconnections (which influences the amount of wiring), and the number of packages (which influences the number of external pins)[717]. He further claimed that that the design of logic circuits in reality is often affected by factors "other than these" (the cost of silicon die, its testing, and its packaging are such factors[718]). Some designs also consume more power than others (e.g., draining the battery and producing heat), which is a crucial issue in some, but not in all, products. As an example, Hennessy and Patterson[719] compared three chips that all have been designed to meet the same IEEE specifications. They noted that even though the constraints have been the same, the designers have ended with completely different implementations because of different design choices and different trade-offs.

There are a variety of reasons for the different design choices on the level of *execution units*. In addition to the above mentioned economical and manufacturing aspects, the choices that designers face are dependent on functional requirements, such as the level of superscalarity and the number and depth of pipelines. The combination of adder/subtractors, and/ors, multiplexers, and other units that constitute a

713 Clements, 2000:p.151.
714 Clements, 2000:pp.194-200.
715 Clements, 2000:p.194.
716 See Swartzlander, 2004 in Tucker, 2004.
717 Clements, 2000:p.32.
718 Hennessy & Patterson, 1996:p.10.
719 Hennessy & Patterson, 1996:pp."A-61"-"A-63".

functional ALU is large in even the simplest ALUs[720]. The more logic elements there are in an execution unit, the more complexity there is between the seemingly simple levels of execution units and the logic elements. The complexity often results in flaws. For instance, a number of bugs in Intel's x86-series has been reported[721]. It should be noted that the problems are not due to the complexity of any of the elements on each of these abstraction levels, but due to the large number and complexity of the connections between abstraction levels. It is practically impossible to provide a proof of the correctness of a computer[722].

The Semantic Gap

In addition to economical, manufacturing, and functional reasons, there are also historical reasons for the different design approaches: Hennessy and Patterson wrote that the birth of CISC architecture dates back to the late 1960s and 1970s, when most system programs were still written in assembly language[723]. At the time, researchers had begun to understand that von Neumann machines do not adequately provide for the constructs that occur in common programming languages[724]. The difference between the concepts of high-level languages and the concepts of *machine language* was called *the semantic gap*. For instance, loops in high-level languages have to be implemented as conditional jumps in machine language, because there is no loop construct in machine language. Glenford Myers argued that the large semantic gap contributes to software unreliability, performance problems, excessive program size, compiler complexity, and distortions of the programming languages[725].

This semantic gap between the machine language and the high-level language corresponds to what I referred to above as *complexity of the semantic interconnections between abstraction levels in computing*. Hennessy and Patterson were explicit about the source of complexity in compiling high-level language into machine language: "*The complexity of a compiler does not come from translating simple statements such as A=B+C.* [...] *Complexity arises because programs are large and*

720 See Leeser, 2004 in Tucker, 2004.

721 For instance, Dr. Dobb's Microprocessor Resources: `http://www.x86.org/secrets/intelsecrets.htm` (accessed September 27th, 2006). Dr. Dobb's Journal is one of the earliest magazines for computer hobbyists, now published largely via the Internet.

722 cf. Kidder, 1981:p.184.

723 Hennessy & Patterson, 1996:p.114.

724 Kavipurapu & Frailey, 1979

725 Myers, 1978

globally complex in their interactions"[726]. The complexity of writing a correct compiler, Hennessy and Patterson argued, is a major limitation in the amount of code optimization that can be done. They advised that designers of computer architecture should understand how compilers are written, and designers of compilers should understand architectures thoroughly[727].

It seems that the more natural a language is for humans, the harder it is to translate to machine language. Symbolic assembly language can be translated directly to machine language, but it is not very easy to read. Assembly language is especially unsuitable for large programs. Procedural languages like C are usually not very complex languages to compile, but their abstraction level is not very high, and hence they are not at their best in very large programs. Some object-oriented languages such as Java and Eiffel are complex languages to compile, but they are flexible regarding their automatic memory management, they are agnostic about many data types, they have large ready-made libraries, and they are well-suited for large, dynamic projects.

In some machines there is a large semantic gap in how machine language is translated to the actual operations of the execution unit(s). Between the machine language level and the execution units level is the *microinstruction* level. Johan Stevenson and Andrew Tanenbaum wrote that decisions around choosing the instruction set are *highly intuitive*, and the choice depends on the skill and experience of the designers[728]. In other words, the choices between varieties of RISC and CISC architectures as well as which instructions to include in the chosen architecture, are intuitive choices that designers make. The computer designers in the 1970s aimed at reducing the semantic gap between high-level languages and machine languages by increasing the complexity between machine language and execution units (CISC architecture also reduces the number of memory accesses). That is, whereas in the "old design" the compiler translated high-level statements into a large number of simple machine language instructions, most of which had direct counterparts on the execution unit level, in the "new design" the compiler translated high-level statements into a small number of complex machine language instructions, which run a

726 Hennessy & Patterson, 1996:p.95.
727 Hennessy & Patterson, 1996:pp.89-91.
728 Stevenson & Tanenbaum, 1979

large number of microinstructions on the execution unit level. Overall complexity was not reduced—only the location of complexity was changed.

Digital design (i.e., design at the machine level) is always a delicate balance between economics (e.g., the costs of the design effort), functionality (e.g., the complexity and efficiency), architectural choices, power consumption and heat dissipation, designers' preferences, standards, and so forth. The large number of competing designs implies that if there is an ultimate architecture and implementation, it has not yet been found.

Complexity on the Language Level

The complexity onthe language level is enormous. The semantic gap between machine instructions and basic *statements and expressions* was already discussed, so only the complexity between statements (and expressions, which together are the basic building blocks of an imperative programming language) and complete functional programs is examined here. Frederick Brooks wrote in *The Mythical Man-Month* that conceptual integrity is *the* most important consideration in software system design[729]. However, the conceptual integrity of a programming language does not guarantee the conceptual integrity of anything produced using that language, because programmers can always produce a conceptual mess out of conceptual integrity. (Moreover, in 1931 Kurt Gödel had already shown that any consistent formal system, no matter how conceptually coherent, necessarily contains some propositions that cannot be proven or disproven[730].)

Statements and expressions in a high-level language are simple and clear. Also, the purposes of statement blocks can be stated simply and clearly. However, combining statements and expressions together to form *statement blocks* always leads to a semantic gap between these two semantic levels. A number of semantical associations need to be drawn between statements, expressions, variables, and the like, to implement the desired functioning of a statement block. Consider, for instance, the following piece of code from Kernighan and Ritchie's definitive C-language hand-

729 Brooks, 1975:p.42.
730 Gödel, 1931

book[731] (the following code—a string copy routine—can be considered to be a part of a statement block or all of it):

```
while(*p++=*q++);
```

In this syntactically- and semantically-correct code snippet, the number of semantic associations is about as large as the actual number of characters in the code. By *semantic association*, I refer to the different semantics, or meanings, that a syntactical element, such as the assignment operator =, can have, depending on the context. For instance, the assignment operator = is used here in two meanings. It assigns a value and it returns a value. Firstly, the operator = denotes a semantic association (an assignment) between two variables `p` and `q`; that is, the operator = assigns the value of `p` to be the same as the value of `q`. The `*` used before the pointer variable alters the semantics of assignment so that the assignment is not done to the pointer variables, but that the block of memory that `p` refers to is altered so that it becomes identical with the block of memory that `q` refers to. Secondly, the operator = also denotes a semantic association (return value) between the caller (the context; in this case, the caller is the `while` statement) and the value that has been assigned.

The increment operators `++` in the code above are used in a postfix position in relation to the variables `p` and `q`. That is, in each case the incrementation is done after the variable has been noted. Furthermore, the increment operator assigns the pointer variable to refer to the next element in the array, so the absolute incrementation of the pointer variable depends on the data type of the variable. The order of association matters in a number of places: Unary operators like `*` and `++` associate from right to left[732], assignment expressions are noted after unary operators, and postfix `++` increments the value of the preceding variable *after* the value of the variable is noted.

The statement after `while` is an empty operation, that is, a no-op or null statement. The expression that `while` evaluates as its exit condition is the return value of the assignment expression =, and in C language the semantical rule concerning character data type in a Boolean expression is that anything different from `'\0'` is interpreted as true, and only `'\0'` is considered false.

731 Kernighan & Ritchie, 1988:p.106. This is the ANSI C string copy routine.
732 Kernighan & Ritchie, 1988:p.95.

All the statements and expressions in the statement block (code snippet) above are simple and clear. Also, the purpose of the statement block is simple: Copy the string `q` to `p`, including the terminator character `'\0'`. The complexity in this code comes from the number of semantical connections between the variables, operators, expressions, and statements. Knowing the language syntax is hardly enough to cope with language idioms such as the above-mentioned code snippet. Kernighan and Ritchie noted that convenience with code like that comes with experience[733]. Note that there are differences in the semantics of statement blocks between languages; for instance, there are different conventions about whether a statement block defines a variable scope or not, in different languages.

While statement blocks can be abstracted to be clear and precise, the *subroutines* that they form are rarely so. For instance, in a software engineering handbook *Code Complete*, Steve McConnell[734] argued that the areas that most often cause problems are the subroutine's interface, its scope, declaring and initializing data, its control structures, and so forth. Although there can be a clear and precise understanding of the definition of the subroutine, and although there can be a clear and precise understanding of each of the statement blocks, the semantical mesh between a subroutine and its constituents causes problems.

It seems that complexity tends to increase towards the higher-level concepts. Be that as it may, the discussion above should be enough to show that the locations of complexity are thoroughly human choices. It cannot be shown that complexity could be avoided, but it is certain that the sources of complexity on a number of semantic levels come from human decisions. Edsger W. Dijkstra wrote that computer scientists cannot know what level of simplicity can actually be obtained, but that it is certain that the central challenge of computing, "*how not to make a mess of it*", has not been met[735]. Especially it should come clear from this characterization of abstraction levels that the complexity of the semantic network (not a hierarchy as one might expect) between a software system and a logic gate is dizzying. In the following subsection, a number of viewpoints of complexity in computer science are discussed.

733 Kernighan & Ritchie, 1988:p.105.
734 McConnell, 1993:p.67.
735 Dijkstra, 1999; Dijkstra, 2001

Mastering Complexity

One of the pioneers of artificial intelligence (AI), Marvin Minsky, saw computer science as the science of mastering semantical properties of super- and subclasses, or different sizes of aggregates, or connections between complexes and entities[736]. In this sense, Minsky leaned towards the inherent-structurist side rather than the nominalist side—Minsky wrote in the book *The Computer Age: A Twenty-Year View*,

> [1979] *In many ways, the modern theory of computation is the long-awaited science of the relations between parts and wholes; that is, of the ways in which local properties of things and processes interact to create global structures and behaviors.*[737]

Prima facie this argument seems to include the structurist assumption that there is an inherent hierarchy in the world, and that it is a hierarchy that a discipline such as computer science can find. Herbert Simon has argued that when dealing with hierarchical abstraction levels, subparts belonging to larger aggregates also interact in an aggregate fashion[738]. That is, when one is dealing with abstractions, one does not need to consider the interactions of single subparts because the interactions of large, abstracted entities are those that matter. However, in computer science it is hard to maintain any strict definition of the granularity of aggregate entities. For instance, it is in principle possible to construct a program-level entity (a computer program) from microinstruction-level entities in an architecture where there is no difference between machine instructions and microinstructions.

James Moor even argued that the hardware-software distinction is a subjective and a pragmatic distinction[739]. For instance, for the user of a microwave oven, the whole thing is hardware. But for the engineer of a microwave oven there is often software and hardware. For the systems programmer, circuitry is hardware, but a circuit designer can see microprograms as software. A graphics programmer may not even known which parts of his program are going to be hardware-accelerated and which run on software.

---

736 Minsky, 1979–however, Denning et al. claimed that the same statement also applies to physics, mathematics, and philosophy (Denning et al., 1989).

737 Minsky, 1979

738 Simon, 1981:p.218.

739 Moor, 1978

The abstraction levels in Figures 22-24 are artificial and they can be changed, skipped, removed, or additional levels can be introduced. In fact, all these are done quite often. For instance, (1) the abstraction levels between machine instructions and execution units are vague in modern machines; (2) many high-level programming languages allow in-line assembly instructions; (3) the operating system creates a level between high-level language and assembly language; and (4) superscalarity, pipelining, multiple logic elements, and multiple execution units make processor abstraction levels vague. But in the end, the structurist-nominalist debate cannot be proven on either side, and at the moment it seems that there is no ultimately correct answer for the question if the hierarchical abstractions of computer science are a *consequence* of an inherent hierarchy of the world, or if hierarchical abstractions are a *tool* for understanding an unorganized world.

Although Minsky's quotation[740] in the beginning of this subsection seems to position Minsky on the structurist side of the debate, that interpretation is undermined by Minksy's rejection of the central role of mathematical logic in the representation of knowledge. In fact, he denied the idea of universality, claiming that it is impossible to find a fixed, universal logic applicable to all kinds of knowledge[741]. Many such attempts have been made, but Minsky's outlook of incommensurability puts a high price on fixed logic: All one's beliefs and knowledge must be made self-consistent, and this is "*essentially impossible to achieve*"[742]. Therefore Minsky emphasized the exploitation of certain, already-learned skills in building new knowledge in an interdisciplinary manner. Minsky's expectations of the future of computer science were quite high. He wrote,

> [1979] *Like mathematics,* [computer science] *forces itself on other areas, yet it has a life of its own. In my view, computer science is an almost entirely new subject, which may grow as large as physics and mathematics combined.*[743]

If computer science is a truly interdisciplinary subject that has a distinct disciplinary identity that is not fully dependent on other subjects, researchers of today have a golden opportunity to document the birth of a discipline. This, of course, calls for

740 Minsky, 1979
741 Minsky, 1979
742 Minsky, 1979
743 Minsky, 1979

investigating the processes that give birth to the discipline, but the methodological base of computer science seems inadequate for such an investigation. In Section 4.2 I analyze some complementary, qualitative research approaches to meta-research on computer science and present examples of existing qualitative meta-research on computer science.

Artificial Intelligence

The inclusion of artificial intelligence (AI) in computer science brings new disciplinary viewpoints into computer science. For example, psychology, linguistics[744], and philosophy[745] are intertwined with computing in AI. The roots of AI are in (1) the conceptual development in the nineteenth century that brought a *representational model of thinking*, (2) the progression in *logic* from symbolization to Boolean logic to predicate calculus, and (3) the development of *computational instruments* through Babbage's mechanical tools to the modern computer[746]. Although AI has a long history, Vernon Pratt argued that there is no continuum, plan, or story in this history, but a number of projects, each different; some of them have been successful, some have been abortive, all for different reasons[747]. In retrospect, Brooks has criticized the attention that the field of AI got during 1970s and 1980s, arguing that too large a fraction of the national public investment in computer science research went to AI, compared to other promising opportunities[748]. And in Brooks' opinion, more serious than the wasted dollars, was "*the diversion of the very best computer science minds and* [...] *academic laboratories*."[749]

Although theories, concepts, and methodologies from psychology, linguistics, and philosophy are used in AI, AI is not considered to be a part of psychology, linguistics, or philosophy. Those disciplines are tools that are used in research on AI. In the early years of computer science, it was argued that because computer science utilizes theories, concepts, and methodologies from mathematics, it should also be considered to be a part of mathematics. My interpretation is that the fact that studies of AI may also contribute to the disciplines of psychology, linguistics, or philosophy is

744 Charniak and McDermott, 1985:pp.87-167 (Chapter 3: "Vision") and pp.169-254 (Chapter 4: "Parsing Language") make the connections clear.
745 Minsky, 1979
746 Pratt, 1987:p.2.
747 Pratt, 1987:pp.1-7.
748 Brooks, 1996
749 Brooks, 1996

not substantive in the identity of the field of AI—AI can have a unique disciplinary identity regardless of its intellectual origins and where its results are applicable. The relationship of disciplines and auxiliary disciplines is discussed further in Section 4.3.

How to Bridge the Semantic Gaps?

Edsger Dijkstra's article in the June 1974 issue of *American Mathematical Monthly*[750] may have contributed to how Minsky regarded computer science[751]; Dijkstra wrote that as a consequence of the hierarchical nature of computer systems, programmers gain agility with which they switch back and forth between various semantic levels, between global and local considerations, and between macroscopic and microscopic concerns. Dijkstra regarded this agility as being an extraordinary ability among scientists. Dijkstra's position was further elaborated in a polemical article published in the summer 1987 issue of *Abacus*[752], where Dijkstra made a case about the uniqueness of computing science. Dijkstra considered the complexity in computing to exist not only in the semantical domain but also in the ratios between the time grains of phenomena in computing—he wrote in *Abacus*,

> [1987] *The ratio between an hour (for the whole computation) and several hundred nanoseconds (for an individual instruction) is 10^{10}, a ratio that nowhere else has to be bridged by a single science, discipline, or technology.*[753]

There is substantial semantical complexity in domains of computing, and the complexity can be argued to be thoroughly human-made. In a similar manner, if there is any substantial semantical complexity in the domain of time, then it is thoroughly human-made.

Dijkstra's use of both *orders of time*[754] and *semantic levels*[755] as indicators of the complexity of the domain of computing should be noted. It seems that orders of

750 Dijkstra, 1974
751 Minsky, 1979
752 Dijkstra, 1987
753 Dijkstra, 1987. Actually, photographs of objects of interest in physics range in size from 10^{-16} to 10^{25} meters. "*A ratio of 10^{41} is a very large difference in scale, even for a computer scientist; moreover, physicists do not rule out the study of even larger, or even smaller, objects*" (Stewart, 1995). However, Dijkstra might have meant that computer scientists may deal with very large ratios within a single project.
754 Dijkstra, 1987
755 Dijkstra, 1974

time and semantical complexity are not separate, but interdependent to some degree. Reducing the number of orders of time that the execution of a computable task takes is sometimes done by rearranging and redesigning abstraction levels and connections between and within them—for instance, by redesigning languages, compilers, and their connections with underlying architecture.

Firstly, rearranging and redesigning abstraction levels to be better suited for human reading and for quick comprehension tends to increase the amount of time that it takes to execute a program that solves a computational task (time can be measured in, for instance, clock cycles). Secondly, reducing the time that it takes to execute a program that solves a computational task is often done by creating or modifying the algorithm central to the task, and there is often a trade-off between simplicity and speed—simpler solutions are often slow, whereas faster solutions are often complex.

For an extreme example of the first of the two arguments above, take the assembly language and Java language. Assembly language generates fast code but it can be cumbersome to use, whereas Java generates slower code but allows one to produce highly functional programs much faster. For an example of the second of the two arguments above, take sorting algorithms. It is not by coincidence that the first sorting algorithm that students are taught is usually the bubble sort algorithm. The bubble sort is simple, although its time complexity is not the lowest possible. The time complexity of the quicksort is lower, but its functioning is arguably more complex to understand than that of bubble sort. Creating a new algorithm to solve a task means assigning new semantics to variables, data, and operations, and often new variables and operations are needed. Note, however, that semantical simplicity need not always mean better or worse time complexity in comparison to semantical complexity.

In the 1970s and 1980s Dijkstra claimed that computer scientists switch agilely between global and local considerations and between macroscopic and microscopic concerns. A later article by Dijkstra implies that by the year 2001 he did not have high hopes for computer scientists' agility any more. Dijkstra wrote that computer scientists should be deeply ashamed about the bug-ridden software of today[756]. It seems that computer scientists are particularly good at mastering complexity in strictly bounded realms (microcosmoses), such as the distinct levels of Figures 22-

756 Dijkstra, 2001

24, but as bugs in nearly all software and especially software systems show, computer scientists invariably fail at coalescing a number of microcosmoses into coherent systems (macrocosmoses). Complexity of a computational system arises, at least, from errors caused by the depth of semantic gaps, from errors caused by the number of semantic levels, and from errors caused by the number of entities on each semantical level.

Computer scientists can successfully construct technological microcosmoses which are based on exactitude, and manage the complexity within them. The rules of microcosmoses can be made very exact and unambiguous. However, when computer science moves outward to study even larger and more unpredictable entities such as computer networks, exactness decreases. The unpredictability in very large systems does not come only from bugs in computer hardware and software, but also from the real world's power outages, electrical interference, faulty machinery and cabling, intentional actors (humans), and even malicious code and malicious intentional actors[757]. There is a large amount of inexactness in the real world.

Coming of Age

In April of 1990, the Computer Science and Technology Board (CSTB) of the National Research Council of the U.S. formed a committee to assess the scope and direction of computer science and technology[758]. The project was motivated by the observation that the intellectual focus of computer science (and engineering) was changing significantly, and so was the environment where computing technology was being applied. The committee noted that the increasing utility of computing in all aspects of society was creating demands for computing technology that would be more powerful and easier to use[759]. The committee characterized the changing field with three judgments, expressing a broad view of computer science and engineering:

> [1992] *Computer science and engineering is coming of age.* [...] *It has established a unique paradigm of scientific inquiry that is applicable to a wide variety of problems.* [...Thus,] *intellectually substantive and challenging CS&E problems can and do arise in the context of problem domains outside CS&E per se.*

757 Note that no amount of source-level verification or scrutiny can protect a programmer from using untrusted code (Thompson, 1984). For instance, Trojan horses (a sort of malicious code) can be inserted on any level of software–they can lay in a compiler, an assembler, a loader, or even hardware microcode (Thompson, 1984).

758 Hartmanis et al., 1992

759 Hartmanis et al., 1992

> [...] *The traditional separation of basic research, applied research, and development is dubious.* [...] *Distinctions between basic and applied research are especially artificial, since both call for the exercise of the same scientific and engineering judgment, creativity, skill, and talent.*
>
> [...] *The growing ubiquity of computing within society places a premium on the largest possible diffusion of CS&E expertise to all endeavors in a society whose computing applications stress the existing state of the art.*[760]

It should be noted that the committee's report (hereafter referred to as the *Hartmanis Report*) explains the future directions of *national funding*, which was also obvious in the recommendations that the committee set—funding was directed to research areas that best supported national endeavors. For example, the development of high performance computing and communications programs was a top funding priority because (1) it was seen to be essential to the nation's future economic strength and competitiveness, (2) it was considered to be among the grand challenges of sciences, and (3) interdisciplinary and applications-oriented computer science and engineering (CS&E) research was on the rise[761].

The report was criticized right after its publication—Three months after the publication of the report, Eric A. Weiss commented on the report, writing, "*We may confidently expect that* [the report] *will have the same minimal effect of its predecessors*"[762]. However, the Hartmanis report put forward some views of computing that although not exactly new, were something that had not really been written in an official report or published in a widely read and oft-quoted journal (*CACM*). The Hartmanis report made some bold suggestions, which I discuss in the following paragraphs.

The reference to *computer science and engineering* (CS&E) instead of *computer science* (CS) is a statement in its own right. Grouping computer science and engineering together implies that computer science and computer-related engineering are not separate disciplines. This interpretation is congruent with the argument in Hartmanis report that distinctions between basic and applied research are artificial.

760 Hartmanis et al., 1992
761 Hartmanis et al., 1992
762 Weiss, 1993

The committee noted the obvious: Some intellectually challenging computational and engineering problems arise in problem domains outside computer science and engineering. Rather than deepening the theoretical and experimental science base of computer science and engineering, the committee's overall judgment was that "*more benefit is likely to accrue to the field and the nation if the broadening course is taken*"[763]. The committee noted that relatively few researchers were devoted to *interdisciplinary* and *applications-oriented* work and that relatively many were devoted to investigating problems at the *core of computer science and engineering*. However, the committee believed that the former, interdisciplinary and applications-oriented work was likely to have a more significant impact than the latter, basic research.

Accordingly, the committee decided to recommend broadening the field (CS&E). Especially the committee urged *academic* computer science and engineering to (1) increase interdisciplinary interchange; (2) support computing in areas of economic, commercial, and social significance; (3) abandon artificial distinctions among basic research, applied research, and development; and (4) enhance the cross-fertilization of ideas in CS&E between theoretical underpinnings and experimental experience.[764]

John R. Rice noted that the Hartmanis report signified a change in the funding of the discipline[765]. Earlier computer scientists were able to just get the money, irrespective of societal, economic, national, environmental, or extra-disciplinary consequences. If the recommendations in the Hartmanis report were followed, the field would have to earn its support by making the case that computer science research will have significant societal benefits. Peter Wegner criticized the principle that research should be judged by its practical impact, when he argued that researchers might lose their motivation if the aims of science were externally imposed[766]. This discussion comes back to the two arguments discussed in Chapter Two.

On one hand, from the Kuhnian point of view, proper science must be able to set its own standards for recruiting scientists and evaluating their work[767]. The viewpoint of sociologist C. Wright Mills was consistent with the viewpoint of Kuhn; Mills

763 Hartmanis et al., 1992
764 Hartmanis et al., 1992
765 Rice, 1993
766 Wegner, 1993
767 Fuller, 2003:pp.45-46.

noted that if science is not autonomous, it cannot be a publicly responsible enterprise[768]. On the other hand, Popper and Feyerabend wrote that science cannot work like the Mafia, a royal dynasty, or a religious order[769]. My position is that if computer science gets funding for its functions from society, it must be responsible to society and there must be control mechanisms that are external to computer science.

The committee further noted that the *core* of CS&E is highly dynamic, as a result of rapid changes in the field. The committee's image of computer science and engineering was that of (1) a collection of vaguely separated fields (2) connected by an unstable core.

However, when the committee stated that CS&E can be applied in a variety of fields of study, the committee *did not* state that (1) if computer science is applied as a tool for disciplines such as physics or social sciences, then those disciplines would become subsumed under computer science or that (2) if computer science is applied as a tool for disciplines such as physics or social sciences, then those disciplines would subsume computer science. Physicists still study physical phenomena, using computer models as tools, and social scientists still study human phenomena, employing computers as tools. This discussion is taken under consideration in Section 4.3.

768 Mills, 1959:p.106.
769 Feyerabend, 1970; Fuller, 2003:p.46.

Recent Definitions

The question 'What can be automated?' is one of the most inspiring philosophical and practical questions of contemporary civilization.[770]

In This Section:

- ✔ What is the fundamental question being asked in the discipline of computing?
- ✔ What are the main research topics in the discipline of computing?
- ✔ What are the recent debates about computing about?
- ✔ Can computing be separated from societal issues?

In the beginning of the 1980s computers had spread to many homes in the Western countries. Interface design specialists were turning their focus towards understanding the context of use, the goals and activities of end users[771], and the group interactions of end users[772]. Since the 1990s there has been a clear shift towards human-centered computing[773]. Respectively, the attention devoted to the social implications of computing has continued to increase[774]. A new understanding of the importance of application areas to the development of computing has also emerged[775].

It is not certain where the frontier incidents of today's computer science take place: perhaps around open-source and open-content[776], the cultural roots of computer science[777], or around interactive computation and the limits of computation[778]. Currently the debate about the possible extensions and definitions of computer science is still lively[779]. In this section I analyze the past president of ACM Peter J. Denning's "*fundamental question underlying all of computing*" and stretch Denning's fundamental question into a more apt form for today's computing; discuss recent discus-

770 Forsythe, 1969
771 Cockton, 2004
772 Grudin, 1990
773 See, e.g., Shneiderman, 2002; Weiser & Brown, 1997; Negroponte, 1995; Dertouzos, 2001; Cockton, 2004.
774 Although there have been debates about the impact of computer uses on different aspects of society throughout the history of modern computing (for a good survey see Martin, 1993). Academic debate dates back long too: The ACM SIGCAS (Special Interest Group on Computers and Society) newsletter has been published since 1970.
775 Hopcroft, 1987
776 Samoladas et al., 2004; Cusumano, 2004; Johnson, 2005; Stallman, 2005; Bowyer, 2006
777 Schreiber, 2005; Tedre et al., 2006
778 Kugel, 2005; Wegner & Goldin, 2006
779 Crowcroft, 2005; Denning, 2005; Klawe & Shneiderman, 2005; Guntheroth, 2006; Poon, 2006; Argamon & Olsen, 2006

sions about the identity of computer science; analyze Frederick P. Brooks Jr.'s concerns about computing; and ask whether debates about the name of the field should be passé by now.

What Can Be Automated?[780]

In his article in the January-February 1985 issue of *American Scientist*, Peter Denning defined computer science as "*the body of knowledge dealing with the design, analysis, implementation, efficiency, and application of processes that transform information*"[781]. The aspects that Denning listed were not new, but by noting that computer science does not deal with calculation only, Denning arrived at what he called the fundamental question underlying all of computer science, "*What can be automated?*"[782]. (Later Denning et al. changed the question to"*What can be (effectively) automated?*"[783].) This question has been often argued to define the topic area of computer science[784], but in my opinion, regarding this question as *the* fundamental question of computer science is an oversimplification. In this and the following subsections, I criticize Denning's "fundamental question underlying all of computer science" from modern computer science points of view and I present a "fundamental question" that is more in line with the topic area of today's computer science than Denning's original question.

In addition to his *fundamental question* (singular) underlying all of computer science, Denning listed eleven topic areas of computer science and outlined the *fundamental questions* (plural) asked in each topic area. Denning had 50 questions altogether.

The fundamental questions in most of Denning's topic areas of computer science are not all related to the fundamental singular question "*What can be (effectively) automated?*". In fact, in 19 out of the 50 fundamental questions that Denning mentioned,

780 The title "What Can Be Automated?" is borrowed from the title of the 1980 report of the NSF Computer Science and Engineering Research Study (COSERS), *What Can Be Automated?* (Arden, 1980).

781 Denning, 1985

782 Denning attributed the question to the COSERS report (Arden, 1980). The COSERS report correctly attributes the question to Forsythe (Arden, 1980, preface).

783 Although Denning noted the demand for effectiveness already in 1985 (Denning, 1985), he did not include the word *effective* in the "fundamental question" at the time. The word *effective* was added to the "fundamental question" in Denning et al., 1989.

784 See the entry "Computer Programming and Computer Science" (Knuth, 1992) in the *Academic Press Dictionary of Science and Technology* (Morris, 1992).

the question deals with *how* instead of *what*[785]. Denning's 50 fundamental questions include questions such as "*How can large databases be protected from inconsistencies generated by simultaneous access* [...]*?*", "*How can the fact that a system is made of components be hidden from users who do not wish to see that level of detail?*", and "*What basic models of intelligence are there and how do we build machines that simulate them?*".

Answers to questions that ask *what* are different from answers to questions that ask *how*. The question "*What can be automated?*" divides all possible processes into those processes that can be automated and those processes that cannot be automated. Conversely, there may be many correct answers to "*How can process p be automated?*". For instance, process *p* could be automated by using either approach *a*, approach *b*, or approach *c*. Approaches *a*, *b*, and *c* can all be optimal in their use of space, time, and other resources, or they can all be non-optimal in different ways.

The difference between questions that ask *what* and questions that ask *how* can be seen in the formulation of the questions. The question "*What can be automated?*" seems like a generic question that concerns all phenomena. The question does not specify the object (e.g., "*Which processes (things, tasks, etc.) can be automated?*"). On the contrary, the questions that ask how things can be automated without an exception specify the object (e.g., "*How can one automate p?*" or "*How can one automate p*(*f*(*x*))*?*"[786]).

The question "*What can be automated?*" is a theoretician's question, and the theoretician should not be concerned about the specific technology that might be used to automate process *p*, be the technology electronics, optics, pneumatics, or magic[787]. The question "*How can process p be automated?*" is a practitioner's question, and the practitioner needs to make a number of design choices concerning the implementation of the technology that automates process *p*.

I suggested earlier the *underrepresentation problem*[788] (*i.e.*, "Given a number of models that all successfully model and predict different aspects of a phenomenon, but that all are flawed in some way(s), how does one determine which model to

785 The number of Denning's questions (excluding subquestions) is 50, and the word *how* appears in 19 questions.

786 *p* denotes a single process, and *p*(*f*(*x*)) denotes a class of processes that share the same essential properties *f*(*x*), which render the class of processes to be possible to be automated.

787 This is how Edsger W. Dijkstra characterized the theoretician's position (Dijkstra, 1986).

788 See Section 2.1, page 60 and Section 2.3, page 155.

use?"), which I argued to be more suitable for computer science than the underdetermination problem suggested by Willard v.O. Quine[789]. The underrepresentation problem is especially valid in the practitioner's problem; "*How can process p be automated?*". In the previous section I showed that there is not always a single optimal way to implement a technology that automates process *p*. Respectively, the underrepresentation problem for a practitioner reads, "*Given a number of different implementations that all automate the process p but that all have non-optimal aspects, how does one determine which implementation is best?*".

Similar to the underdetermination problem, there is no solution for the underrepresentation problem. If none of the implementations that automate process *p* are optimal in all aspects, the choice of implementation depends on the practitioner's opinion, experience, and proficiency. The practitioner has to weigh the relative significances of non-optimal and optimal aspects of different implementations and then make a decision about the implementation. Note, however, that although notions of significance are ontologically subjective, they can still be epistemologically objective. Although notions of significance do not exist without people ascribing degrees of significance to phenomena, people in a given community can feel the same about the significances of aspects of phenomena.

Because the answer to the question "*Can process p be automated?*" is a singular yes/no answer, and because the number of processes in the world is essentially infinite, theoreticians often approach the question "*What can be automated?*" by asking, "*Why can some processes be automated?*" or "*What kinds of properties are typical of processes that can be automated?*". For the practitioner, the singular question "*Can process p be automated?*" is important: If the answer is yes, the practitioner asks, "*How can process p be automated?*", and perhaps even, "*Which implementation automates process p best?*" (In the latter question the practitioner may have to make subjective choices because of the underrepresentation problem). The practitioner, too, can ask questions such as "*What kinds of problems do certain implementations automate best?*". In Figure 25, processes 1 to 5 are singular instances in an infinitely large set of processes. Except for some special cases, a theoretician is not interested in studying an infinite number of single cases, because it would take infinitely long to do so. Conversely, the practitioner usually works with single cases.

789 See Quine, 1980:pp.37-41.

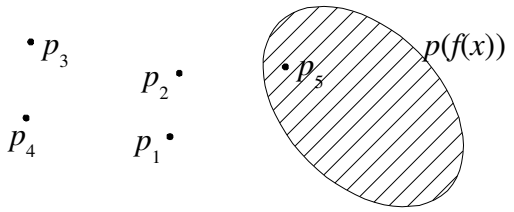


Figure 25: Singular Processes $p_1..p_5$ and a Class $p(f(x))$ of Processes

The area marked by $p(f(x))$ in Figure 25 is a class of processes that share the same essential properties $f(x)$. These properties either make the processes in $p(f(x))$ possible to be automated or impossible to be automated. Theoreticians in the field of computing are generally interested in finding classes of processes that can be automated; the ultimate goal of a theoretician is to find the essential property that would divide all processes into those processes that can be automated and those processes that cannot be automated. For instance, the Church-Turing Thesis was earlier considered to divide all processes into those processes that can be automated and those processes that cannot be automated[790]. One of a theoretician's tasks is to provide proofs that classes of processes belong either to those that can be automated or to those that cannot be automated. When a theoretician devises a proof about a class of processes, the theoretician inevitably faces the question "*Why can this class of processes be automated?*".

The Denning Report

In the 1980s, computer science had been diversifying rapidly, and the old debate about the essence of computer science still continued. In the spring of 1985, the ACM and the IEEE Computer Society formed a task force to describe the intellectual substance of the field in a "new and compelling way" (hereafter referred to as *Denning's task force*)[791]. Denning's task force finished their report (hereafter referred to as the *Denning Report*) in 1989, defining *computing as a discipline*[792] as;

> [1989] *The systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implement-*

790 See Hopcroft, 1987. Nowadays there have been doubts if the limits of what the Turing machine can compute are also the limits of what can be mechanically computed in general (Copeland, 1997; Copeland & Sylvan, 1999; Copeland & Proudfoot, 2000).

791 Denning et al., 1989

792 "Discipline of computing" here includes "*all of computer science and engineering*" (Denning et al., 1989).

ation, and application. The fundamental question underlying all computing is, "What can be (effectively) automated?" [793]

Denning's task force noted that in the *discipline of computing* (computer science and computer engineering), science and engineering cannot be separated because of the fundamental emphasis on *efficiency*. The definition of Denning's task force has been widely cited and it can be considered to be a milestone in the field of computing. In addition to the short definition above, the task force characterized computing as a discipline as the intersection of three major research paradigms: *theory*, *abstraction (modeling)*, and *design*. The Denning Report includes the following characterizations of each major research paradigm:

- *Theory* is the bedrock of the *mathematical sciences*: Applied mathematicians share the notion that science advances only on a foundation of sound mathematics.

- *Abstraction* (*modeling*) is the bedrock of the *natural sciences*: Scientists share the notion that scientific progress is achieved primarily by formulating hypotheses and systematically following the modeling process to verify and validate them.

- *Design* is the bedrock of *engineering*: Engineers share the notion that progress is achieved primarily by posing problems and systematically following the design process to construct systems that solve them[794].

In the Denning Report it is argued that many debates about whether computer science belongs more to one of the above mentioned paradigms than to others are implicitly based on an assumption that one of the three is more fundamental than the others. Denning et al. claimed that a closer examination of computing reveals that the three paradigms are so intricately intertwined that it is irrational to say that any one is fundamental.

For example, instances of theory appear at every stage of abstraction and design, instances of modeling at every stage of theory and design, and instances of design at every stage of theory and abstraction[795]. The task force noted that although the three

793 Denning et al., 1989 (short definition)

794 Denning et al., 1989. Note that design is the central issue when dealing with *artifacts*, or objects created by people, as opposed to those occurring naturally (Freeman and Hart, 2004).

795 Denning et al., 1989

paradigms are inseparable, they are distinct from one another because they represent separate areas of competence. Although design is sometimes considered to be a non-academic subject, it has been argued that without the *science of design* many of the opportunities of computing would be thrown away[796].

It should be noted that a similar division of computer science into *theory*, *modeling*, and *design* had been suggested in a number of places before the Denning Report. One such division is found in Purdue University's first computing curriculum in 1962—computer science in Purdue was divided into *theory*, *numerical analysis*, and *systems*[797]. These correspond roughly to theory, modeling, and design. More interestingly, another definition of computer science, strikingly similar to the one in the Denning Report, was presented in 1976 by Peter Wegner in the second IEEE conference on software engineering[798].

Wegner described computer science as being a part mathematical, part scientific, and part technological discipline. Wegner wrote that the mathematical part was "*concerned with the formal properties of certain classes of abstract structures*", that the scientific part was "*concerned with the empirical study of a class of phenomena*", and that the technological (engineering) part was "*concerned with the cost-effective design and construction of commercially and socially valuable products*". The only part that differs between Wegner's definition and the Denning Report is the engineering part. Whereas Denning's task force emphasized *processes*, Wegner underscored *goal-orientation*. He wrote,

> [1976] *Research in engineering is directed towards the efficient accomplishment of specific tasks and towards the development of tools that will enable classes of tasks to be accomplishment more efficiently.*[799]

Yet Wegner also mentioned the *process-nature* of the engineering part. He divided the engineering part of computer science into two parts; *practical engineering* and *research-based engineering*. Wegner wrote,

> [1976] [The problem-solving paradigm of the practicing engineer] *generally involves a sequence of systematic selection of design decisions which progressively narrow down alternative options for accomplishing the task until*

796 Freeman and Hart, 2004
797 Rice and Rosen, 2004
798 Wegner, 1976
799 Wegner, 1976

> *a unique realization of the task is determined.* [The research engineer] *may use the paradigms of mathematics and physics in the development of tools for the practicing engineer, but is much more concerned with the practical implications of his research than the empirical scientist or mathematician.*[800]

Although it is clear that Denning's task force was not the first to divide the discipline of computing into theory, modeling, and design, Denning's task force was apparently not aware that that type of a division had been proposed many times before. (They did not cite or acknowledge Wegner's conference paper[801].)

Ed Lee, in the 34th IEEE Computer Society International Conference, San Francisco, presented a definition of computing that was much different from Denning's definition, yet Lee's and Denning et al's definitions were both presented in 1989. Ed Lee wrote,

> [1989] *Computer science is the field of human endeavor that includes the study, the design and the use of machine based data processing and control systems to enhance peoples' ability to study information, perform work or explore reality.*[802]

In Lee's opinion, the focus of *human-made* computer science is *the human*. He wrote, "*Neither a computer nor the teaching of computer science has any value or meaning outside of its impact on people*"[803]. Lee's definition of computer science places the *human*, not the *computer* at the core of computer science. In other words, computer science has no intrinsic value, but it only acquires value through its effects on people or society. This view could be called *human-centered computing*[804].

There are a variety of terms that are similar to human-centered computing; for instance, Shneiderman's *new computing*, Mahmood's *end-user computing*, and Johnson's *user-centered technology*[805]. All those terms have their strengths but also have their weaknesses: Shneiderman's term, *new computing*, emphasizes a shift from "old

800 Wegner, 1976

801 Note that the conference proceedings where Wegner's paper was published also included papers by two out of seven members of the task force–Peter J. Denning and David Gries; that is; Proceedings of the 2nd International Conference on Software Engineering, 1976, San Francisco, California, United States, October 13 - 15, 1976. Denning's paper *Sacrificing the calf of flexibility on the altar of reliability* appears on pages 384-386, and Gries' paper *An illustration of current ideas on the derivation of correctness proofs and correct programs* appears on page 200.

802 Lee, 1989

803 Lee, 1989

804 The term also appears in the name of the book by Harris et al., 2003.

805 See Shneiderman, 2002; Mahmood, 2002; Johnson, 1998, respectively.

computing" (which is about what computers could do) to "new computing" (which is about what users can do), but, although the content of Shneiderman's *new computing* is rich and human-centered, the term *new computing* itself is vague (today's new computing is tomorrow's old computing). Mahmood's term, *end-user computing*, and Johnson's term, *user-centered technology*, both focus on the human aspects of computing, but *user* in both Mahmood's and Johnson's terms implies that a human is *active* in his or her interaction with computing technologies.

Quite the contrary, a human is often *not active* in his or her interaction with computing technologies. I have noted earlier[806] that there was a large technological shift in the period between the early 1990s and mid-2000s: The increased affordability, miniaturization, integration, and interoperability of information and communication technology took computing machinery from the desktop to the pocket, from cable-bound to wireless, from rare to ubiquitous, and from shared to private. In addition, during the period between the early 1990s and mid-2000s the amount of technology increased, its forms diversified, and information and communication technology gradually became an integral and commonplace part of many people's lives in industrial countries. Computing technologies have pervaded everyday life; people use computing technology when they turn off their alarm in the morning, heat their breakfast, drive their cars to work, collaborate with their colleagues, call their friends, and entertain themselves[807]. But computing technologies are also often active when people do not even know they are there: when switching lights on and off, regulating room temperatures, monitoring traffic, and so forth.

Although the ubiquity and the number of technologies have increased, the need for people to be active users of computing technology has not increased to the same extent, because new computing technologies often work without people having to be aware of them (*ubiquitous computing*[808]). Because very often people are in contact with computing technology without knowing it, and because the verb *use* seems to connote some kind of conscious intention to use technology, I prefer the term *hu-*

806 As a co-author in Kamppuri et al., 2006; As a co-author in Kamppuri et al., 2006b.
807 Kamppuri et al., 2006
808 Weiser, 1993

man-centered computing to the terms *user-centered computing* and *end-user computing*. I agree, however, that the choice of the term is debatable[809].

Although it can be argued that the birth of human-centered computing happened in the 1980s when the focus of interface design begun to turn towards *end users*[810]; it can also be argued that the shift from machine-centered computing to human-centered computing is still largely incomplete. Note that neither the theoretician's questions "*What can be effectively automated?*" and "*Why can class p(f(x)) of processes be automated?*" nor the practitioner's questions "*Can process p be automated?*" and "*How can process p be automated?*" include, explicitly or implicitly, any questions about why processes should be automated at all. The theoretician's and practitioner's questions belong clearly to machine-centered ("old"[811]) computing.

Questions such as "*Should process p be automated or not?*", "*Why should process p be automated?*", "*When should process p be automated and when not?*", and "*What individual or societal consequences does automating process p have?*" impugn whether some specific processes should be automated[812]. These questions belong clearly to human-centered ("new"[813]) computing. I noted earlier in this thesis that, in the late 1950s, C. Wright Mills warned his readers about the division between the general public, who do not understand technology, and technologists, who do not understand anything other than technology[814]. Fortunately, Mills' division is unlikely to exist as sharply as Mills described it; it is more likely that each individual also understands technology to some degree, and each individual understands also something else to some degree. Be that as it may, the creators of technology must bear some responsibility for the uses of their technology.

In 1966 C.P. Snow wrote that people with a variety of abilities, not only technologically oriented abilities, must study, control, and humanize the effects of the computer revolution[815]. Snow wrote that unlike our ancestors, who could not foresee the ef-

809 In the ACM publication *interactions* [sic] Donald Norman made a case against the whole concept of *human-centered design* (Norman, 2005).

810 Baecker et al., 1995:41; Grudin, 1990; Shneiderman, 2002:p.11.

811 Shneiderman, 2002

812 For instance, Kimmo Raatikainen proposed that questions "What should we allow to be automated" and "What should we automate" belong also to the questions of computer science (Raatikainen, 1992).

813 Shneiderman, 2002

814 Mills, 1959:p.175.

815 Snow, 1966

fects of the first industrial revolution, there is no excuse for the people of today to not control the computer revolution. However, the chances are that the methodological, conceptual, and theoretical framework of computer science, as described in, for instance, the Denning Report[816], turns out to be insufficient to deal with the changes brought about by the computer revolution. The chances are also that the framework of computer science turns out to be insufficient for selecting, recording, understanding, explaining, analyzing, or predicting phenomena in the field of human affairs.

The questions of machine-centered computing are *descriptive questions*, questions about "what is", whereas many of the questions of human-centered computing are *normative questions*, questions about "what ought to be". David Hume argued that answers to descriptive questions cannot be derived from answers to normative questions and vice versa[817]. *If* the field of computing is shifting towards human-centered computing as it has been described[818], and *if* Mills' and Snow's warnings about the separation of technologists and general public is true, *then* the professionals of computing face new questions to which the theoretical-methodological-conceptual toolbox of computer science may offer little help. The field has been amended before: For instance, the *ACM Code of Ethics and Professional Conduct*[819] deals with the ethical responsibilities of computing professionals. That is, the practitioners of the field of computing are already expected to be knowledgeable about some ethical questions.

One might argue that contemplating on ethical issues does not belong to computer scientists but to philosophers. This argument has a number of problems because computer science is not detached from society. The products of computer science are used in society and its scientific activities are made possible by society. Receiving funding from external sources entails ethical questions such as the motives of the funders, scientists' obligations to funders, and the economic pressure to demonstrate progress. Effects on society entail ethical questions such as who is affected by technology, how they are affected, and if such effects are desirable. The role of laypersons in decision-making, that is, who should make decisions about technology, is

816 Denning et al., 1989
817 Hume, 1739:Book III, pp.507-521.
818 See, e.g., Shneiderman, 2002; Grudin, 1990; Weiser & Brown, 1997; Negroponte, 1995; Dertouzos, 2001.
819 `http://www.acm.org/constitution/code.html` (accessed September 27th, 2006)

also an important issue. Computer scientists, who know the possibilities, limitations, and side-effects of computing technology better than anyone else, have expert knowledge that philosophers do not have. Although it might be beneficial for computer scientists to understand societal and philosophical issues, it might also be beneficial for people from the humanities and social sciences to understand technological issues.

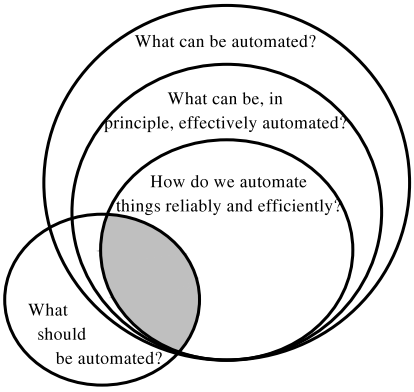


Figure 26: Four Fundamental Questions in Human-Centered Computing

The theoretician's questions, the practitioner's question, and the human-centered question are portrayed in Figure 26.[820] All those questions are important in themselves and it is important to conduct research in non-intersecting areas, too. However, whereas in machine-centered computing the production of effective and reliable computing and communications machinery takes place in the area delineated by the question "*How do we automate things reliably and efficiently?*", in human-centered computing the responsible production of useful computing and communications machinery takes place in the intersection marked with light gray.

In contrast to Denning's fundamental question underlying all of computer science, "*What can be (effectively) automated?*", the modern version of the fundamental question underlying all of computer science should be, "*How can one effectively automate processes that can be automated and that should be automated?*". This question includes the practitioner's question, "*How can process p be automated?*", the theoretician's question, "*What can be effectively automated?*", and the human-centered question "*What should be automated?*". All three aspects are necessary, but answers to them they lie in different domains of knowledge. "*What can be auto-*

820 Figure 26 is from Tedre, Matti (2006) *What Should be Automated–The Question Underlying Human-Centered Computing*. To be presented at the HCM2006 International Workshop on Human-Centered Multimedia, October 27, 2006, Santa Barbara, USA.

mated?" belongs to the domain of theoretically oriented computer science. "*What should be automated?*" belongs to the domains of the humanities and social sciences. "*How can one effectively automate ...?*" belongs to the domain of engineering-oriented computer science.

A New Era of Growth

By the late 1980s, the discussion about the disciplinary identity of computer science had still not yet ceased. In 1987, Edsger Dijkstra wrote that the "incoherent bunch of disciplines" that began computer science, hardly appealed to the intellectually-discerning palate of mathematicians. What Dijkstra called *computing science* deals with what is common to the use of any computer in any application: Dijkstra wrote in *Abacus*,

> [1987] *The computing scientist could not care less about the specific technology that might be used to realize machines, be it electronics, optics, pneumatics, or magic. At the same stroke, computing science separated itself from all the specific problems of embedding computers meaningfully in some segment of some society.*[821]

In the late 1980s, Dijkstra's view that computing science should be completely detached from its surroundings did not draw much support. Perhaps the reason is that also those theoreticians to whom Dijkstra referred to as *computing scientists* were content being compartmentalized within the discipline of computer science. Nonetheless, much of Dijkstra's critique is still valid. For instance, Dijkstra made a point that *iterative design* is the paradigm for the pragmatist who is stuck in a vicious circle of testing-debugging, because testing can only reveal the existence of bugs; not their non-existence[822].

Dijkstra argued that *computing scientists* did not face the problem of iterative design because computing scientists' programs have been rigorously proven correct[823]. Note, however, that I noted earlier in this thesis that human errors can occur in any part of the program-construction process (see page 278 of this thesis, footnote 572). Proving a program correct does not mean that the program corresponds to the proof; that is, that the translation from proof to program was done correctly.

821 Dijkstra, 1987
822 Dijkstra, 1974
823 Dijkstra, 1987

Dijkstra's argument for developing proof and program hand in hand can be questioned from a surprising direction—cryptanalysis (the following argument is a concrete example of James H. Fetzer's conceptual criticism of program verification[824]). It has been discovered that many cryptographic algorithms can be broken by observing the *cryptosystem*, that is, by observing the physical tool that implements the algorithm. For instance, *timing attacks* are based on the fact that given enough information about the timing of operations during the running of an algorithm, it is often trivial to break the cipher[825]. The weakness of the cryptosystem does not come from the algorithm per se but from its implementation[826]. No matter how rigorously a "computing scientist" would translate the proof to program, the program can fail its purpose—not because of flaws in any part of the algorithm or the translation process, but because computers, unlike algorithms, are not abstract objects[827]. It seems that contrary to what Dijkstra argued, there are cases when the computer scientist *must* know the machine details.

In other words, Dijkstra wanted to distance the computing scientist from the real world—he wanted computing scientists to work in the abstract world of computation. Dijkstra criticized software engineers for making a mess out of programming. In Dijkstra's ideal world, there are no bugs, because programs are proven correct, and there are no timing attacks, because formulæ are not timed. It seems unfair that Dijkstra criticized other practitioners of making a mess of a job Dijkstra would not be ready to undertake. By not acknowledging real world needs, Dijkstra only foisted the problems upon others. The moment one wants to utilize computing scientists' programs, someone needs to convert the program to some programming language (which immediately risks bugs in the program), and take the program into the unpredictable, uncontrollable, uncertain, inexact, analogous, and ambiguous world.

Even though Dijkstra's critique is valid to some extent, it has had minimal impact on the practice of computing. One of the reasons is the limited utility of program verification in real-world systems, as noted by Brian Cantwell Smith and James H. Fetzer[828]. The field of computing was already, at the time of Dijkstra's writing, much

824 Fetzer, 1988
825 Kelsey et al., 2000
826 See Bar-El et al., 2004.
827 This remark is based on an oral communication by mr. Olli Vertanen (February 17th 2006).
828 Fetzer, 1988; Smith, 1996

broader than Dijkstra's vision of the field. And contrary to Dijkstra's intention to define the field of computing narrowly, there was pressure to broaden it further, as John E. Hopcroft stated in his 1986 Turing Award lecture[829].

Hopcroft was concerned about the scientific base of computer science falling behind the technological base. He wrote,

> [1987] *Much of the credit for the emergence of computer science as a discipline rests with the dedication and commitment of a relatively small number of researchers who had a vision of the potential of computing and the perseverance to make this vision a reality.*[830]

Hopcroft recognized that there is a dualistic relationship between computer science and its surroundings. He wrote,

> [1987] *Today, there are signs that computer science is turning to applications areas. As it contributes its models, tools, and techniques to these new fields, they in turn will contribute new ideas and methodologies that will greatly enrich and expand the scope of computer science.*[831]

Hopcroft regarded *application areas* as the key to a new era of growth in computer science. His vision was that ideas and methodologies from disciplines other than computer science could contribute to computer science. That is, *extending the discipline of computing with perspectives from other disciplines would be beneficial to computing*—Hopcroft expected it to lead to a new era of growth in computer science.

In 1987, new uses of computers had already had a major impact on society and on the way people think and live. Hopcroft stated that he regarded computer science as the means to take people to a higher plane of knowledge about the world, especially in understanding different intellectual processes. Hence, Hopcroft urged his fellow computer scientists to "*formulate a new vision, to shape the goals for the next generation of researchers*"[832]. Formulating a new vision and shaping new goals was, in Hopcroft's opinion, the *responsibility* of his contemporaries.

---

829 Hopcroft, 1987
830 Hopcroft, 1987
831 Hopcroft, 1987
832 Hopcroft, 1987

The only thing required from computer scientists, according to Hopcroft, was personal commitment. He wrote, "*We must also commit ourselves to the future of computer science before fully discerning its shape.*"[833] Even though Hopcroft wrote that computers have penetrated almost every aspect of life, including medicine, education, and economics, and even though Hopcroft predicted that computers will alter people's lives even more dramatically in the future, Hopcroft's "commitment" did not include responsibility or accountability for any of these changes. Hopcroft called for a national commitment to computer science, including universities and policymakers, but nowhere in Hopcroft's vision did he call for responsibility from the side of computer scientists.

What's In a Name? (Part I)

Although the debates over the disciplinary identity of computing have become infrequent, they still surface every now and then. In the December 1995 issue of *IEEE Computer*, George McKee wrote that *science* (in *computer science*) refers to "*the set of intellectual and social activities devoted to the generation of new knowledge about the universe*"[834]. (Note that McKee did not further discuss the concept of *science*—cf. page 31 of this thesis.) McKee argued that *computer science* should be replaced with a term that does not include *science*, such as *computics*;

> [1995] *The fundamental issue is about intellectual honesty and the self-respect it engenders. If computists are just acting like scientists and not actually doing science, they shouldn't use the word* [science] *to describe their discipline.*[835]

McKee noted that mathematicians have acknowledged the nonscientific nature of mathematics by choosing the name of the field to end with "-ics". Considering the naming of other established fields of scientific inquiry, I find McKee's critique confused: Very few natural sciences include the term *science* in their name[836], but a number of non-natural sciences such as social science, political science, and cognitive science do. In addition, there is no prototypical similarity between disciplines that end with "-ics". Physics deals with natural laws and ontologically objective phenomena, but linguistics and economics deal with constructed realities and ontolo-

833 Hopcroft, 1987
834 McKee, 1995
835 McKee, 1995
836 Environmental science and neuroscience are among the few exceptions.

gically subjective phenomena. A better critique of the *science* part of *computer science* than McKee's was given by Frederick P. Brooks Jr. in his ACM Allen Newell Award lecture. In his lecture, Brooks argued that a folk adage of the academic profession says, "*Anything which has to call itself a science, isn't.*"[837] By Brooks' criterion, physics, chemistry, history, and anthropology *may* be sciences; political science, social science, and computer science definitely are not.

Brooks made a distinction between a scientist and an engineer: The scientist builds in order to study, and the engineer studies in order to build. He wrote that unlike the disciplines in the natural sciences, computer science is a synthetic, engineering discipline. Hence, Brooks wrote,

> [1996] *If our discipline has been misnamed, so what? Surely computer science is a harmless conceit. What's in a name? Much. Our self-misnaming hastens various unhappy trends.*[838]

The first unhappy trend that Brooks mentioned is that the self-misnaming of computer science implies that computer scientists have to accept a perceived pecking order that respects natural scientists highly and engineers less so. Brooks believed that computer scientists seek to appropriate the natural science station for themselves.

As I see it, the "pecking order" is currently implied in academical terminology at large, not only in the naming of computer science. *Pure science* sounds as it were free of the impurities that *applied science* contains. However, although the common connotations of the term *pure science* may mix descriptive aspects (where the term refers to a set of activities or ways of conducting research) and normative aspects (where the term implies that pure science is superior to applied science), the names do not seem to be the crux of the matter. I do not see how the pecking order that Brooks mentioned would cease to exist, no matter how the field were renamed or which station computer scientists sought to appropriate. There is something deeper than mere naming in the tug of war between the pure and applied sciences—after all, as Knuth noted, disciplines such as mathematics and chemistry are far from what their etymology indicates[839].

837 Brooks, 1996
838 Brooks, 1996, emphasis in original underlined.
839 Knuth, 1985

The second unhappy trend that Brooks mentioned comes from the argument that in the sciences the discovery of facts and laws is taken as an end itself. A *new* fact or a *new* law is an accomplishment in science. Brooks wrote that if computer scientists regard themselves as scientists, then computer scientists will regard the invention and publication of endless varieties of computers, algorithms, and languages as an end. However, it is not certain if this unhappy trend has yet come to pass. On one hand, one might count, for instance, the number of conference paper titles advertising *novelty* as the paper's main contribution and draw conclusions about the pervasiveness of that unhappy trend. On the other hand, Brooks' view of "*the computer scientist as a toolsmith*"[840] seems an equally appropriate description of the field today. The trends towards human-centered[841] or value-centered[842] computing as well as the emergence of new topic areas between computer science and many other disciplines (such as computational physics and computational biology) are driving a type of computer science in which computers are seen as tools, not as proper ends per se.

The third unhappy trend that Brooks mentioned is that computer scientists tend to forget the users and their real problems. He wrote that computer scientists tend to climb into ivory towers to dissect tractable abstractions of once-real problems in esoteric vocabularies. The tractable abstractions may have left the essence of the real problem behind. Brooks wrote that the surfacing of esoteric vocabularies is a sign of this trend.

Brooks is not alone in his concern about the distance between theoretical scientists' work and practical problems[843]. Brooks also argued that increasingly esoteric vocabularies pose a problem for computing, and that those vocabularies are symbols of the distance between the domain of computing and other domains of life. I interpret the appropriation of established philosophical terminology, such as *ontology* and *agency*[844] in nonphilosophical topics, as an embodiment of this unhappy trend.

840 The title of Brooks' article is *The Computer Scientist as Toolsmith* (Brooks, 1996).

841 See, e.g., Shneiderman, 2002; Weiser & Brown, 1997; Negroponte, 1995; Dertouzos, 2001.

842 Gilbert Cockton wrote that *value*; be it political, personal, organizational, cultural, experiential, spiritual, or economic; must drive the field of human-computer interaction (Cockton, 2004).

843 cf., e.g., Knuth, 1991

844 See, e.g., Castel, 2002; Sánchez et al., 1994.

Donald Knuth's article in the November 1991 issue of *Theoretical Computer Science* is in line with Brooks' concern about the distance between scientists' work and everyday problems[845]. Knuth wrote that theory and practice in computer science are more intimately connected than in any other discipline—"*They live together and support each other*". In his article, Knuth criticized "pure" science:

> [1991] [...] *there was a time when the only applied mathematics you could find in* [Journal of Pure and Applied Mathematics] *consisted of applications to pure mathematics itself! When theory becomes inbred—when it has grown several generations away from its roots, until it has completely lost its touch with the real world—it becomes sterile.*[846]

Knuth advised those computer professionals who spend most of their time on theory to start turning some of their attention to practical things. He wrote to them, "*It will improve your theories*". Similarly, Knuth advised those computer professionals who spend most their time with practice to start turning some of their attention to theoretical things—"*It will improve your practice*".[847]

The discussion about the importance and inseparability of theory and practice is a sign that the distinction between theoreticians and practitioners is becoming more subtle. In addition, Denning et al.'s definition of computing as a discipline[848], introduced earlier in this thesis, consisted of three different but separate aspects of computing: theory, modeling, and design. Although practice can be understood as encompassing both modeling and design, I argue that there is still another aspect that should be recognized as an inseparable part of computing as a profession—the social-ethical aspect.

In a society where computing technology is a catalyst of change, computer scientists; who understand the restrictions, capabilities, prospects, and threats of computing technology better than anyone else; must also take responsibility for the technology they create. Although computing technology is not the *cause* of social change, it makes new social, organizational, and economic changes possible, and the kinds of changes it makes possible are influenced by the forms of technology[849].

---

845 Knuth, 1991
846 Knuth, 1991
847 Knuth, 1991
848 Denning et al., 1989
849 See, e.g., Castells, 2000; Castells, 2001:pp.36-61; Florida, 2003:p.26; Fiske, 1994.

One might argue that computer science and its products are value-free, and that it is not in the computer scientist's power to regulate what other people do with computing technology. Even if computer science *were* value-free, the flaw in Kuhn's *normal science* that Fuller, Popper and Feyerabend pointed out applies well to the aforementioned argument[850].

If computer scientists would want to avoid the flaw in normal science that Fuller, Popper, and Feyerabend described, computer science should have the same safeguards that modern democracies have—safeguards which regularly force politicians to be accountable to people other than themselves[851]. My question is, paraphrasing Feyerabend[852], "If computer scientists are only accountable to themselves, then what essentially makes computer science different from primitive social formations such as the Mafia, a royal dynasty, or a religious order?". Fuller noted that an élitist vision of untouchable science should have no place in today's world, where the costs and benefits of science loom as large as the costs and benefits of any other public initiative. Taking into account the multi-billion dollar investments in computer science by governments, institutions, and organizations, and taking into account the powerful impacts that computer science and technology enable on individuals and societies, computer scientists cannot think of their enterprise as a private playpen.

Knuth wrote that those computer professionals who focus on theory should pay attention to practical things. He continued that those computer professionals who work with practical things should pay attention to theoretical things.[853] It has been noted many times that another important area that computer professionals should focus on, is their awareness of their surroundings, that is, how their work can be used and misused in society[854]. The ethical code of the ACM is directed towards those issues.

The fourth trend that Brooks mentioned is that if computer scientists honor the more mathematical and abstract parts of computer science more, and the practical parts

850 Feyerabend, 1970; see Fuller, 2003:p.46. See also Section 2.1, page 84.
851 cf. Fuller, 2003:p.46.
852 cf. Feyerabend, 1970; cf. Fuller, 2003:p.46.
853 Knuth, 1991
854 See Kling, 1980 for examples.

less, young and brilliant minds will be “misdirected” away from challenging and important practical problems[855].

In light of the history of computing machinery, and insofar as practical matters are important at all, Brooks' notion is warranted. It has been argued that one of the reasons why pre-war British scientists working in the field of computing had less success than their American counterparts was that the British scientific community resisted and ignored practical research[856]. In the U.S., where the engineer was the hero of the new century[857], applied (engineering) science flourished and resulted in advances in computing machinery, but in Britain, where the theoretical sciences were revered, all the students who got scholarships went to study theoretical subjects[858]. My interpretation of Brooks’ fear is that if his dystopia were to come true on a large scale, the field of computing could stagnate. (There are, however, alternative fixes to the problems in computer science. For instance, Dijkstra wrote that computing professionals should seek to fight the shoddy quality of software by concentrating on formal, theoretical methods[859].)

In summary, Brooks argued that there are four unhappy trends that computer science is following: (1) accepting a pecking order where theory is respected more than practice; (2) regarding the invention and publication of endless varieties of computers, algorithms, and languages as an end; (3) forgetting the users and their real problems; and (4) directing young and brilliant minds towards theoretical subjects.

To resist these four unhappy trends, Brooks suggested a *driving-problem approach*, which relies on working on the problems of another discipline. Brooks argued that working on a field different than computer science helps the computer scientist for five main reasons[860]. First, it aims the computer scientist at relevant problems, not just at exercises or at toy-scale problems. Second, it keeps the computer scientist honest about success and failure, so that he or she does not fool himself or herself easily. Third, it makes the computer scientist face the *whole* problem, not just the easy or mathematical parts. Fourth, facing the whole problem forces the computer

855 Brooks, 1996
856 Bowles, 1996
857 Bowles, 1996; Kevles, 1987:p.293.
858 Bowles, 1996
859 Dijkstra, 1989; Dijkstra, 1999
860 Brooks, 1996

scientist to learn or develop new computer science, often in areas that would have never otherwise been addressed. That is, working on a field other than computer science also benefits computer science. Brooks' fifth reason is that working in a field other than computer science is *just plain fun*. Brooks' text is a strong argument for engineering-oriented computer science, which is an enterprise where the value of products is judged according to their utility or usefulness.

A New Species Among The Sciences

The 1993 Turing Award winner, Juris Hartmanis, took an opposite view to the view of Brooks. Whereas Brooks wrote that computer science is not a proper science, Hartmanis argued that computer science differs so fundamentally from the other sciences that it has to be viewed as *a new species among the sciences*, especially because it deals with human-made phenomena that are explored by human-made paradigms and methods[861]. Hartmanis wrote that computer science is laying the foundations and developing the research paradigms and scientific methods for the exploration of the world of information and intellectual processes that are not directly governed by physical laws.

The difference between Brooks and Hartmanis' views seem to be mainly due to their different views about the definition of *science*. If one thinks about science as Brooks did—as a "branch of study concerned with the observation and classification of facts, especially with the establishment and quantitative formulation of verifiable general laws"[862], then many areas of computer science are not science proper. Hartmanis' view of science was different. Hartmanis noted that computer science is not about verification, but that theory and experimentation in computer science are focused "more on the *how* than the *what*"[863]. He believed that the results of theoretical computer science are judged, for instance, by the insights they reveal about various models of computing, and that the results of experimentation are judged by demonstrations that show the possibility or feasibility of doing things that were earlier thought to be impossible or unfeasible.

From the viewpoint of the philosophy of science, Brooks' view of science *as verification* was flawed. Early in the 20th century, Popper refuted the notion of fact veri-

861 Hartmanis, 1994
862 Brooks, 1996
863 Hartmanis, 1994

fication in science. But Hartmanis' view of science has also faced criticism. While Hartmanis regarded computing as a new kind of a science, N.F. Stewart responded to Hartmanis by writing that computer scientists should strive to make computer science similar to the natural sciences[864]. According to Stewart, the traditions of computing inhibit its development into a proper science. Michael C. Loui, in response to Hartmanis, noted that it would be more appropriate to call computer science *a new species of engineering*[865]. In addition to Hartmanis, Loui, and Stewart, Marvin Minsky had also noted the difficulties of actually defining computer science:

> [1979] *Computer science has such intimate relations with so many other subjects that it is hard to see it as a thing in itself.*[866]

The views of Hartmanis and Minsky raise a question: How can these distinguished Turing Award-winning computer scientists see their science in completely disparate ways? It might be simply because both of them, and many other opposing views too, are right. Gal-Ezer and Harel noted that computer science is definitely a new and important science (Hartmanis' view), but its relationships with other fields like mathematics, physics, electrical engineering, and life sciences such as brain research and human genome research are also very significant (Minsky's view)[867]. Finally, it should be noted that without some serious qualifications, arguments like Stewart's, that computer scientists should aspire to shape computer science to be more like natural sciences, are inherently flawed insofar as computer science is not a natural science—that is, insofar as computer science does not deal with naturally occurring phenomena[868].

Computational Science

One of the newer topics that brings together theory and practice in computing is *computational science*. In 1993 computational science was an emerging topic whose place in the scientific world was still unclear. Ten years afterwards, in 2003, the topic was included in the list of *core technologies* of computing[869]. According to D.E.

---

864 Stewart, 1995
865 Loui, 1995
866 Minsky, 1979
867 Gal-Ezer and Harel, 1998
868 For instance, Donald Knuth, argues that computer science is an unnatural science, which deals with artificial things (Knuth, 2001:p.167).
869 Denning, 2003 (See page 69 of this thesis).

Stevenson, the "Clemson View" (named after Stevenson's university) of computational science is as follows:

> [1993] *Computational science is an emerging discipline characterized by the use of computers to provide detailed insight into the behavior of complex physical systems*. [...] *The proper subject of computational science is proper modeling and correct computation.*[870]

According to Stevenson, computer science students can easily see computer science devoid of meaning and programming devoid of empirical import[871]. He argued that elementary algorithm books present algorithms for minimum spanning trees but do not explain what minimum spanning trees are used for. Problem solving in computer science, according to Stevenson, teaches *problem solving devoid of problems*. On the contrary, Stevenson argued that *computational science* addresses problems that have important implications for humankind.

There is indeed a number of high-technology, high-publicity achievements that can be attributed to computational science, such as the Human Genome Project, SETI@home, and numerical weather prediction[872]. These projects and fields are not considered to be computer science, but none of them would be viable without computer science and computing technology. It has to be noted that the relationship between computational science and computer science is not a one-way relationship. That is, although computational science benefits from computer science, in the aforementioned three topics in computational science the topic areas have spurred new technologies, new approaches to automation, and new computational strategies —all of which benefit computer science[873].

In 1993 Stevenson wrote that computational science is an interdisciplinary undertaking that could use computer science as an active partner—but that could quickly develop computer support without computer science. However, it seems that thirteen years after Stevenson's argument, computational science is very tightly coupled with computer science. The topics in three major journals in computational science—*Scientific Computing World*, *IEEE Computing in Science & Engineering*, and *SIAM*

870 Stevenson, 1993

871 Stevenson, 1993

872 See Collins et al., 2003 about the Human Genome Project; Shirts & Pande, 2000 and Anderson et al., 2002 about the SETI@Home Project; and Gneiting & Raftery, 2005 about numerical weather prediction.

873 Collins et al., 2003; Shirts & Pande, 2000; Anderson et al., 2002; Gneiting & Raftery, 2005

Journal on Scientific Computing—deal with a wide variety of topics, most of which are closely related to computer science.

What's in a Name? (Part II)

Even very recently computer science has been defined by listing its constituents. The following quote is from a computer science textbook by Glenn Brookshear:

> [2003] *Computer science is the discipline that seeks to build a scientific foundation for such topics as computer design, computer programming, information processing, algorithmic solutions of problems, and the algorithmic process itself.*[874]

There seems to be as many listings as there are computer scientists. Brookshear's description of computer science suffers from two common problems that are almost contradictory. Firstly, Brookshear attempted to narrow the focus of computer science by mentioning some of its topics, but mentioning some topics inevitably gives less emphasis to other topics that other computer scientists might consider the most important in computer science. Secondly, Brookshear left the list open for arbitrary additions by using the phrase "*such topics as*". An additional problem with using lists of topics as a definition of a discipline is that it is assumed that all the topics are definable themselves. Because of these problems, Brookshear's definition of computer science has little informational value. It is close to impossible to characterize the whole academic field of computing by making a list of topics with which all researchers would unanimously agree. In this sense, Denning's interrogative approach to defining the discipline of computing ("*What can be effectively automated?*") offers a better overview of computing.

In his letter to the editor in the March 1988 issue of *CACM*, Peter Kugel hinted back to the 1967 definition of computer science by Newell et al.[875], when Kugel asked, "*Lots of people drive cars but that does not justify an 'automotive science'. Do computers justify a 'computer science'?*"[876]. Among other questions, Kugel asked whether computer science should be redefined, and wrote:

> [1988] *Perhaps our* [computer scientists'] *role is like that of logic in the medieval curriculum that was supposed to sharpen the mind. Perhaps we de-*

874 Brookshear, 2003:p.1.
875 Newell et al., 1967
876 Kugel, 1988

> *serve a place in the curriculum only to remind students that, as William James wrote in 1899, "laboratory work and shop work ... give honesty; for when you express yourself by making things, and not by using words, it becomes impossible to dissimulate your vagueness or ignorance by ambiguity".*[877]

The second part of Kugel's quote raises the same concern that Richard W. Hamming wrote about in 1969—that computer science without concrete applications would become idle speculation, "hardly different from that of the notorious Scholastics of the Middle Ages"[878]. Although Kugel's tone implies pessimism about the role of computer science as merely a tool to sharpen the mind, philosopher of computing Brian Cantwell Smith saw the same phenomenon in an optimistic light. In *On the Origin of Objects*, Smith wrote that computation is not a topic of a single discipline, "*Computation is not a subject matter*"[879]. Smith's argument is that as computing becomes an integral part of the humanity's body of knowledge, it would be a mistake to think that anthropologists, sociologists, journalists, educators, etc., would be just *users* of computation[880]. On the contrary, Smith argued that they participate in the invention of computing—creating user interfaces, proposing architectures, rewriting the rules, and so forth. Incidentally, in a sense, Smith answered to Kugel's question (see the following question by Kugel and a notion by Smith—the original texts are not connected):

> [1988] *Lots of people drive cars but that does not justify an 'automotive science'. Do computers justify a 'computer science'?*[881]

> [1998] *Computers turn out in the end to be rather like cars: objects of inestimable social and political and economic and personal importance, but not the focus of enduring scientific or intellectual inquiry.*[882]

In the March 1985 issue of *American Mathematical Monthly*, Donald Knuth expressed his opinion about the name of the field of computing:

> [1985] *I suppose the name of our discipline isn't of vital importance, since we will go on doing what we are doing no matter what it is called; after all,*

877 Kugel, 1988
878 Hamming, 1969
879 Smith, 1998:p.73.
880 Smith, 1998:p.360.
881 Kugel, 1988
882 Smith, 1998:p.74.

other disciplines like Mathematics and Chemistry are no longer related very strongly to the etymology of their names.[883]

In 1996, Frederick P. Brooks, Jr., asked rhetorically "*What's in a name? Much*"[884]. Knuth's answer seems to be, "not much". The discussion about the name of the field may have been vital when it was necessary to distinguish computer science from engineering and mathematics. After all, when the discipline of computing was young, the organization of universities as well as the different granting foundations and institutions probably did require computing to have a disciplinary identity[885]. Also the public image of computing as a discipline, including its name, may have had an effect on the early development of computing[886], but there is little point in discussing the name for the field of computing anymore. As George Forsythe noted, in a purely intellectual sense such jurisdictional questions are sterile and a waste of time[887].

883 Knuth, 1985
884 Brooks, 1996
885 Forsythe, 1968
886 Hamming, 1969
887 Forsythe, 1968:p.455.

Research in the Discipline of Computing

To understand information processes, computer scientists must observe phenomena, formulate explanations, and test them. This is *the scientific method.*[888]

In This Section:

- ✔ What are the research approaches taken in computer science?
- ✔ What kinds of meta-research are there in the field of computing?
- ✔ What is the methodology of computer science?
- ✔ Can anarchism in science be justified?

It has been argued that computer scientists publish relatively few papers with experimentally validated results[889]. In addition, it has been argued that research reports in the discipline of computing rarely include an explanation of the research approach in the abstract, key word, or research report itself[890], which makes it difficult to analyze how computer scientists arrived at their results.

Methodology in Computing Curricula

Robert Glass argued that the typical computing researcher draws his or her research skills from (*1*) a background of mentoring—master-apprentice relationships with senior professors in a PhD program—and from (*2*) patterning activities—examining the writings of successful prior researchers[891]. Kuhn would probably have regarded this kind of transfer of knowledge as an excellent case of "paradigm as exemplar". In the Kuhnian view, exemplars are sets of *concrete* puzzle-solutions that are used as models or examples, and that replace the explicit rules of normal science[892]. However, although learning from exemplars might work for some computer scientists, it is not certain if exemplars can give the typical computer scientist the methodological knowledge that choosing and using methods and techniques requires.

888 Tichy, 1998. Emphasis in original.
889 Tichy et al., 1995
890 Vessey et al., 2002
891 Glass, 1995
892 Kuhn, 1996:pp.187-191. See also page 67 of this thesis.

Research *methodology*[893] courses in typical computer science curriculum are rare; for instance, the official ACM/IEEE curriculum recommendations (CC2001[894]) do not include a course on methodology. Yet, CC2001 has a number of courses that deal with *techniques* or *methods*; such as courses in *formal methods*, *proof techniques*, *algorithmic strategies*, and *methods and tools of analysis in social and professional issues*. It is briefly mentioned in CC2001 that students ought to have an understanding of the *scientific method*[895] and that topics concerning methods should be found throughout the curriculum. But the scientific method is just a broad set of principles, not an actual *method of inquiry*. The scientific method does not deal with issues such as what to measure, how to measure it, or the validity and reliability of measurement.

More discussion on methods in CC2001 can be found in, for instance, SE6 (validation and verification), DS3 (proof techniques), PF2 (problem-solving strategies), AL1 (algorithmic analysis and empirical measurement), OS11 (evaluation models), HC1 (hypotheses, experimental results, correlations, sciences of psychological and social interaction), HC3 (usability tests, interviews, surveys, and experiments), and HC4 (task analysis). But the only course on *methodology* that is suggested in CC2001 is *Programming Methodology*[896]. The course does not actually address methodological issues broadly. Introducing methods (techniques, procedures, or tools of inquiry) without methodology (principles and foundations of methods) is shallow at best, misguiding at worst. It is dubious if one can use a tool without knowing its limitations, pitfalls, methodological and epistemological linkages, and theoretical burdens.

Because there is ambiguity about the research approaches used in computer science, in this section I chart the research approaches and methods in the field of computing. I also analyze some of the meta-research in the field of computing.

893 See Section 1.3, page 13 of this thesis for a definition of the terms *method*, *methodology*, and *research approach*.
894 Denning et al., 2001
895 See Section 9.1.2 of Denning et al., 2001.
896 Topics in *SP3 Methods and tools of analysis* are: "Making and evaluating ethical arguments; Identifying and evaluating ethical choices; Understanding the social context of design; and Identifying assumptions and values" (Denning et al., 2001:p.143).

Gordana Dodig-Crnkovic has summarized research in computing using four categories: *modeling*, *theory*, *experimentation*, and *simulation*[897]. Dodig-Crnkovic's categories are very similar to Denning et al.'s categories: *theory*, *modeling*, and *design*[898]. In Dodig-Crnkovic's view, *modeling* is common to the three other categories (theory, experimentation, and simulation) because in all sciences a phenomenon of interest must be simplified before it can be studied. That is, researchers cannot record and analyze all the variables in a study; they have to exclude most. In fact, it cannot even be said that researchers *exclude* variables, but that they *include* some variables and that all others are excluded. This is due to the fact that the number of superfluous variables, such as the color of the nearest hat, is infinite.

In Dodig-Crnkovic's description, *theoretical research* is done following the classical logico-mathematical tradition—building theories of logical systems with stringent definitions of objects (axioms) and operations (rules) for deriving and proving theorems. In Dodig-Crnkovic's description, *experimentation* is done following the scientific method, that is, observing phenomena, formulating explanations and theories, and testing those explanations and theories (this corresponds to Denning et al.'s *abstraction*, i.e., *modeling*). In Dodig-Crnkovic's description, *simulation* is done, for instance, in visualization, computer-based modeling, and numerical analysis, and simulation refers mainly to computational science. Denning et al.'s *design* seems like a broader category than Dodig-Crnkovic's *simulation*. Whereas Dodig-Crnkovic underplayed the role of design in computer science, Denning et al. emphasized the engineering method—that is, according to Denning et al., the cycle that consists of defining requirements, defining specifications, designing and implementing, and testing.

In a recent analysis of research in the field of computing Glass, Ramesh, and Vessey divided computing into three subfields: *computer science*, *software engineering*, and *information systems*[899]. Glass et al.'s report was based on 1485 articles from a set of recognized journals from each of the three subfields between the years 1995-1999.

897 Dodig-Crnkovic, 2002

898 Denning et al., 1989

899 Glass, Ramesh, and Vessey's article was published in a number of journals. See, for instance, Glass et al., 2004; Ramesh et al., 2004.

Their sample consisted of 628 articles from computer science, 369 articles from software engineering, and 488 articles from information systems.

Glass et al. found that in computer science and software engineering the *research approaches* were mainly formulative—that is, formulating a process, method, algorithm, concept, model, or framework. In information systems the research approaches were mainly evaluative—that is, conducting deductive, interpretive, or other kinds of evaluations (see Figure 27)[900].

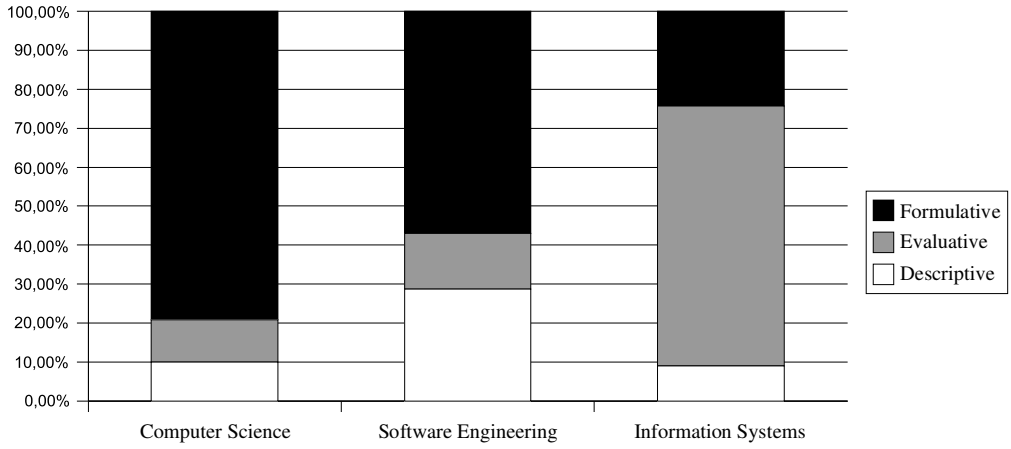


Figure 27: Research Approaches in Computing Disciplines

Glass et al. also found that the preferred *research method* in computer science is mathematical analysis[901], in software engineering the preferred methods are conceptual analysis and concept implementation, and in information systems a variety of methods can be found (see Figure 28)[902].

The rich variety of topics in the subfields of computing is matched by the rich variety of research approaches and methods used. Especially in the subfield of information systems, researchers utilize a wide variety of methods. That is, there is no dominant method or approach in the subfield of information systems.

900 Glass et al. do not present figures such as Figure 27 and Figure 28; figures presented here are aggregates of Glass et al.'s findings.

901 Ramesh et al., 2004 use the term *conceptual analysis based on mathematical techniques.*

902 Note that in Figure 28 *laboratory experiment* means experiments with human participants. Note also that Figure 28 does not include all the methods that Glass et al. identified. Only the methods that have been used in more than 5% of the articles in any of the three subfields are included. For instance, *data analysis* was used in only 0.2% of articles in computer science, in 2.2% of the articles in software engineering, but in 5.3% of the articles in information systems; therefore *data analysis* is included in Figure 28. In absolute numbers this means that in order to be included in Figure 28, a research method should have been used in more than 31 (out of 628) articles on computer science, *or* in more than 18 (out of 369) articles on software engineering, *or* in more than 24 (out of 488) articles on information systems.

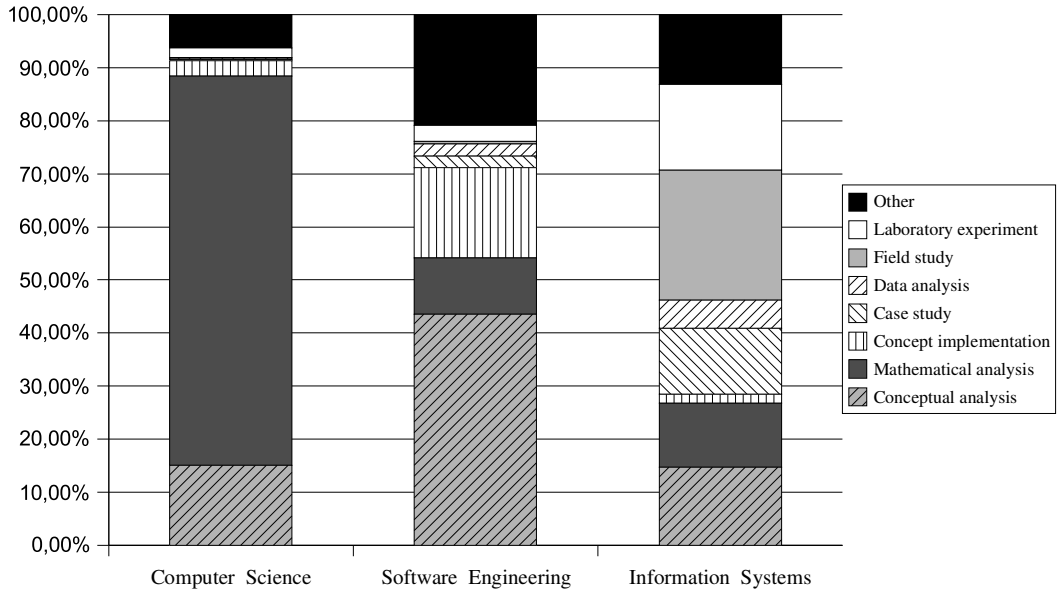


Figure 28: Research Methods in Computing Disciplines

When Glass et al. looked at the *reference disciplines* used in computing subfields, they found software engineering to be 98.1% self-referential; that is, they found that researchers in software engineering referred primarily to other research in software engineering. References outside the field of software engineering were rare. They also found that in computer science 89.3% of the references were self-referential and that 8.6% of the references were to mathematics. The reference disciplines in the subfield of information systems were found to be diverse. In information systems only 27.2% of the references were to other studies in information systems. The rest of the references were to management (18.0%), economics (11.1%), cognitive psychology (10.7%), social and behavioral science (9.0%), management science (6.6%), and others.

However extensive Glass et al.'s study may be, it may not adequately describe what actually happens in computer science research. The granularity of the method categories in Glass et al.'s study is coarse (e.g., there is actually a large number of methods that belong to the broader category *field study*). Glass et al.'s classification scheme was not developed during their study, but it was borrowed from another study[903]. Also, the choice of mainstream journals may have biased the sample of articles towards mainstream research so that alternative methods may have gotten lesser

903 The scheme is borrowed from Morrison & George, 1995.

attention. This is not to criticize Glass et al.'s study, but to note that in addition to the typical methods in computing that Glass et al.'s research approach highlighted, a number of other methods can easily be found. Even mainstream journals in computer science have proposed, for instance, a *hermeneutic* approach and an *action research* method[904]. Indeed, when one starts to dig deeper into the subdisciplines of computing, it turns out that computer scientists are quite flexible in their research approaches.

Glass et al. grouped research approaches into *descriptive*, *evaluative*, and *formulative* approaches. There are also different classification schemes for research approaches in the field of computing, such as a classification into (1) *formal theory*, *design and modeling*, *empirical work*, and *hypothesis testing*[905]; a classification into (2) *observational methods*, *historical methods*, and *controlled methods*[906], a classification into (3) *quantitative methods*, *qualitative methods*, and *benchmarking*[907]; and a classification into (4) *scientific methods*, *engineering methods*, *empirical methods*, and *analytical methods*[908]. Although some of these classes are named *methods*, they are so broad that they actually describe sets of methods or research approaches.

If one considers specific methods instead of research approaches, there is a rich variety of methods in the field of computing. In the theoretically oriented subfield of computer science, the methods include *mathematical proofs*, *simulations*, and *proofs of concept by implementation*[909]. In the subfield of software engineering, "methods" have been reported to include *project monitoring*, *quantitative and qualitative case studies*, *assertion*, *field studies*, *analysis of legacy data*, *lessons-learned* (post-project meta-research); *static analysis*, *quantitative and qualitative experiments*, *replicated experiments*, *synthetic environment experiments*, *dynamic analysis*, *quantitative and qualitative surveys*, *screening*, *effects analysis*, *benchmarking*, *simulation*[910], and *participatory design*[911].

904 West, 1997; Avison et al., 1999; Baskerville, 1999. Published in *Communications of the ACM* and *Communications of AIS*.
905 Tichy et al., 1995
906 Zelkowitz & Wallace, 1997
907 Kitchenham, 1996
908 Glass, 1995
909 Ramesh et al., 2004
910 Combined from Zelkowitz & Wallace, 1997 and Kitchenham, 1996.
911 Muller et al., 1993

Glass et al. found that information systems is the subfield of computing that has the most diversity in terms of research methods and reference disciplines. Other meta-researchers in computer science have found the same: Research methods in the subfield of information systems have been reported to include many of the methods used in theoretically oriented computer science and software engineering, plus, "methods" such as, *laboratory experiments* with human subjects, *field experiments*, *case studies*, *surveys*, *ex post descriptions*[912]; *field studies*[913]; *conceptual studies*, *reviews/tutorials*[914]; *forecasting*, *game/role playing*, *subjective/argumentative* methods, *descriptive/interpretive* methods[915]; *interviews*, *qualitative content analysis*, *ethnography/hermeneutics*, *grounded theory*, *critical theory*, *consultancy*[916]; *futures studies*, *questionnaires*[917], and *organizational ethnography*[918].

Note that some of the methods listed here are broad categories, some narrow and specific, but their sheer number and variety clearly demonstrate the multiplicity of methods used in the discipline of computing. Often methods are mixed to offer a wider perspective on a topic[919]. In a sense, some of the research in the field of computing might be characterized as a *bricolage* (see page 188 of this thesis). The number of methods used by computer scientists[920] is large, but as I noted earlier, computer sciencists are not necessarily given a thorough methodological training. It is uncertain if the background of mentoring and patterning activities[921], and the current official computing curriculum[922] can give the *computer-scientist-as-bricoleur* the necessary knowledge to move between research paradigms. Denzin and Lincoln used the term *researcher-as-bricoleur* in a laudatory sense, whereas Lévi-Strauss used the same term in a derogatory sense. Whether *computer-scientist-as-bricoleur* is a laud-

912 *Ex post description* means that interest in reporting the results of the project develops after the project is complete (or is partially complete) (Alavi & Carlson, 1992). Note that Alavi and Carlson did not to use the well-established phrase *ex post facto.*

913 Alavi & Carlson, 1992

914 Lai & Mahapatra, 1997

915 Galliers & Land, 1987

916 Mingers, 2003–see also Mingers, 2001.

917 Choudrie & Dwivedi, 2005

918 Walsham, 1995

919 See Johnson & Onwuegbuzie, 2004 for mixed-methods research.

920 The term *computer scientist* in this context covers the professionals in all disciplines of computing, not only theoretical computer science, software engineering, or information systems.

921 Glass, 1995

922 Denning et al., 2001

atory or a derogatory term depends on how well the *computer-scientist-as-bricoleur* knows and handles the variety of methods.

Eclecticism and Opportunism

Tichy et al. have argued that computer science does not conform to the *falsificationist ideals* of (experimentalist) science[923]. Computer science does not seem to conform to the *Kuhnian ideal* either, as it is very difficult to find an overarching paradigm in computer science, and because alternating between research paradigms without conflicts is difficult[924], or even impossible. Based on the methodological nonconformity of computer scientists, computer scientists' work may conform best to Feyerabend's theory of science. In Feyerabend's characterization of science, scientists have certain ideas that work until a new situation turns up, and then they try something else[925]. In Feyerabend's account, scientists are *opportunists* above all else.

I noted earlier that there are arguments that (1) the computer science curriculum does not include rigorous studies in computer science research methods; (2) computer scientists learn their research skills from examining earlier research (mostly in computer science); (3) research papers in computer science rarely include a methodology section; and (4) computer scientists utilize a large variety of methods. Insofar as these arguments are correct, they imply that computer science is a methodologically eclectic discipline.

Methodological opportunism and eclecticism (which also usually entails epistemological eclecticism) is not necessarily a negative approach to research. As long as researchers are knowledgeable about the methodological and epistemological frameworks in which they operate, they can disregard epistemology and make informed crossings of methodological boundaries. For instance, if a researcher studies how people use a particular software system, mixing a statistical study of task completion times with user interviews is perfectly appropriate (and often even advisable) as long as the researcher knows the frameworks in which each of the methods of inquiry operate. Conscious breaches of methodological and epistemological norms can be characterized as methodological and epistemological anarchism. Research that does

923 Tichy et al., 1995; Tichy, 1998
924 Denzin and Lincoln, 1994:pp.2-4.
925 Feyerabend in an interview in 1992, reported in Horgan, 1996:p.52.

not conform to *any* established frameworks can be good and fruitful too, but researchers entering such enterprises must be especially knowledgeable of their anarchism, and especially capable of proposing their arguments based on the arguments' credibility[926]. *Informed anarchism* is one thing, ignorant anarchism another.

Meta-Research in Computer Science

All of the aforementioned methods in the various fields of computing are applied to the *subjects* of computer science. Meta-research in computer science (excluding software engineering, parts of which can be considered to be meta-research) employs a much narrower set of methodologies. Meta-research in computer science has been done mostly using content analysis (or through the classification study, which is a subcategory of content analysis)—for instance, the studies by Alavi and Carlson, Tichy et al., Vessey et al., Palvia et al., and Glass et al.[927] used varieties of content analysis. The articles in *IEEE Annals of the History of Computing* and most books on the history of computing employ historical research methods, mostly primary- and secondary-source, descriptive, chronological research.

There is an abundance of analytical arguments about the *status quo* of computer science; that is, they are not based on empirical findings, but on the credibility of the analysis and argument. Most of the discussion about computer science as a discipline quoted in this and previous chapters are based on the analytical tradition. The philosophy of computing is also largely based on the analytical tradition (not in the sense of *analytical philosophy* or *mathematical analysis*).

Some authors, such as Robert L. Glass and Tichy et al.[928], who have conducted meta-research in the field of computing have argued that the analytical research tradition is "seriously flawed" and "alarming" because the analytical research tradition does not necessitate the use of empirical studies to empirically validate hypotheses, models, or designs. However, the meta-research done by Glass, as well as that by Tichy et al., are flawed in the very same manner that they criticize. Glass and Tichy et al. were of the opinion that there is too little empirically validated research in computer science, but neither Glass nor Tichy et al. have empirically validated that empirically

926 For instance, Galileo Galilei was such a researcher. Feyerabend argued that Galileo used not only brilliant argumentation, but also *propaganda* and *psychological tricks* (Feyerabend, 1993:p.65).

927 Alavi & Carlson, 1992; Tichy et al., 1995; Vessey et al., 2002; Palvia et al., 2003; Glass et al., 2004

928 Glass, 1995; Tichy et al., 1995

validated research is beneficial for computer science. In the following, I discuss Tichy et al.'s article more thoroughly.

Tichy et al. (*i*) surveyed 400 research articles in the field of computing and classified the research approaches taken in the articles; (*ii*) surveyed and classified the research approaches taken in two non-computing journals; (*iii*) and evaluated the quality of the articles—Tichy et al. "*used the fraction of space each* [design and modeling] *article devotes to evaluation as an indicator of quality*"[929].

However, although Tichy et al. embraced and called for empirical studies, they did not empirically validate three essential assumptions underlying their argument: (1) that the quality of an article and space devoted to empirical validation in the article are correlates; (2) that empirical, quantitative validation is a sine qua non of first-class science; and (3) that computer science and natural sciences are commensurable. That is, in their critique of computer science, Tichy et al. did not establish why the standards of computer science should be the same as in the natural sciences. Even if they had established that computer science and natural sciences are commensurable, they should have established that it is *computer science* that is flawed and not the other sciences.

Tichy et al. (1) wrote that their "collective impressions" support a positive correlation between the quality of experimental evaluation and the amount of space devoted thereto; (2) quoted Donald Knuth and Peter J. Denning to support their view that experimentation should be at the heart of computer science; and (3) referred to a number of informal discussions with a number of computer scientists to point out the feelings that computer scientists have about the standards of computer science.

Tichy et al. conceded that they have not validated their assumption that *quality of experimental evaluation* and the amount of space devoted to the description of experimental evaluation have a positive correlation. However, Tichy et al. have not considered that the *quality of design/modeling research* and the space devoted to empirical validation in a report might not always be correlates. Some of the most influential and epochal design arguments in computer science have been introduced without any empirical tests at all. For instance, Dijkstra's article "GO TO Statement

929Tichy et al., 1995

Considered Harmful"[930] (in which Dijkstra presented structured programming) had a tremendous impact on program design, yet no research was ever performed to measure the value of Dijkstra's argument[931].

Both quotes that Tichy et al. have chosen to support their conviction about the decisiveness of experimentation in computer science seem ill-chosen for Tichy et al.'s purposes. First, they cite Knuth's well-known remark "*Beware of the bugs in the above code; I have only proved it correct, not tried it.*"[932]. Knuth's remark is from a memo with a formal proof of program correctness, but Tichy et al. specifically exclude "*formally tractable propositions, e.g., lemmata and theorems and their proofs*" from their study. Second, they cite Peter J. Denning, but in that particular article Denning made a peculiar argument:

> [1980] *The hypothesis may concern a law of nature—for example, one can test whether a hashing algorithm's average search time is a small constant independent of the table size by measuring a large number of retrievals.*[933]

It is difficult for me to see in which sense the average search time of a hashing algorithm *A* (which is a human construction) implemented on a computer brand *B* (which is a human construction, and where both *A* and *B* rely on the framework of modern computation, which is a human construction), would be a law of nature in the sense that it would tell anything about nature *proper*—that is, about naturally occurring phenomena (see page 120ff. in this thesis). Denning seems to be confused about laws of nature.

Although I share Tichy et al.'s subjective feeling that too little empirical work is being conducted in the field of computing, I must note that Tichy et al. did not empirically validate all their claims. Their arguments are not based on experimentation only, but also on the credibility of their argumentation. I argue that the reason for the lack of empirical validation of their argument is that Tichy et al.'s claims cannot be exhaustively empirically validated. This problem ultimately derives from the fundamental separation of normative and descriptive claims (often called Hume's

930 Dijkstra, 1968

931 Glass, 2005; MacLennan, 1999:pp.126-127.

932 Knuth, Donald E. (1977) *Notes on the van Emde Boas construction of priority deques: An instructive use of recursion*. A memo sent to Peter van Emde Boas, Bob Tarjan, and John Hopcroft, March 29, 1977. Knuth's comment available at Knuth's web page `http://www-cs-faculty.stanford.edu/~knuth/faq.html` (accessed September 27th, 2006).

933 Denning, 1980b

Law[934]). It is impossible to make normative claims about science and rest those claims solely on the reliability and validity of *research findings*. I will elucidate this argument in the following pages.

Even if, in the field of natural sciences, the quality of research would correlate with the amount of empirical verification, and even if the quality of an article and space devoted to empirical validation in that article would correlate, Tichy et al. would still have needed to argue why computer science, a non-natural (unnatural[935]) science, should be judged according to the standards of natural sciences (i.e., they would still have needed to argue why computer science and natural sciences are commensurable). It would be especially difficult to make such argument without deriving "what ought to be" from "what is" (see page 35 of this thesis).

There is a fundamental difference between argumentation in research that makes descriptive claims and argumentation in research that makes normative claims. Based on Hume's Law *only* research that makes *only* descriptive claims can be, even in theory, argued exhaustively with empirical results. If a researcher makes a normative claim, there is no empirical study that can be used to prove the claim, so the researcher has to rely on the credibility of his or her analysis and argument. Normative arguments are value statements, and they cannot be based on empirical results[936].

Normative Arguments Cannot Be Based on Empirical Results

Let me give an example. Suppose that a researcher argues that increasing experimentation would be beneficial to computer science. He or she can, for instance, take two groups of computer scientists and double the amount of experimentation that group 1 does but keep the experimentation that group 2 does the same. The researcher can then measure the outcomes of the two groups and establish, for instance, that the products of group 1 are faster, more reliable, or more usable than the products of group 2. The researcher can make hundreds of similar tests, and become convinced that there is are positive correlations between amount of experimentation and speed, reliability, or usability. After the comparisons, the researcher has established a descriptive claim which states, for instance, that "*The amount of experi-*

934 Hume, 1739:Book III, pp.507-521. See the discussion in Section 2.1, pages 35ff. in this thesis.

935 Knuth, 2001:p.167. See the argumentation in Section 2.2, pages 129ff. in this thesis.

936 It is a debated issue in contemporary philosophy if normative claims can be proven correct at all (Richmond Campbell (2004) in Stanford Encyclopedia of Philosophy: *Moral Epistemology*).

mentation correlates positively with the speed, reliability, and usability of software", and in theory, there is no upper limit of how reliable the claim can be.

However, because the researcher argues that increasing experimentation would be *beneficial* to computer science, he or she has to establish one more claim: That *speed*, *reliability*, or *usability* is beneficial to computer science. Unfortunately to the researcher, there is no empirical experiment that he or she can rely on if he or she wants to establish what is *beneficial*, because statements of beneficialness are value statements.

There are, however, some senses in which one could make the claims above. Firstly, if the researcher stated that *computer science is a science that examines how to efficiently automate tasks*, then it indeed would be a fact that a speed increase is beneficial to computer science (and respectively, increased experimentation would be beneficial to computer science), but the fact would be true only by definition, not true by any objective sense. Changing the definition of computer science would change the truth of the claim. Additionally, the statement would not help in making claims about usability of reliability.

Secondly, if the researcher conducted a poll where 100% of interviewees said that success of computer science is measured by how well computer scientists manage to make things usable, then it indeed would be a fact that according to the people included in the poll, improved usability is beneficial to computer science (and respectively, increased experimentation is beneficial to computer science), but the fact would be true by consensus, not true by any objective sense. Another poll with different people or at a different time could lead to different results. Strictly speaking, attributes of technology are value-free (see pages 140ff. of this thesis), and also in practice, for instance, *speed* can lead to benevolent as well as malicious results. Take, for instance, the efficiency of a sorting algorithm and the efficiency of a network virus.

Normative claims inevitably include a hidden or explicit value premise, such as "increased speed is desirable", or "ease of use is desirable", and therefore they are assessed according to their plausibility, credibility, and desirability. Credibility, of course, can be increased with experimentation, but ultimately it has to be based on human judgment. Much of the meta-research in computer science is only descript-

ive, that is, the researchers have refrained from making claims about whether the *status quo* of research in computing is desirable or not. I do not have an objection to meta-research with normative claims either, as long as the value statements they contain are explicit and not hidden. Making value statements explicit is a difficult task for a researcher though; in practice one has to accept many normative claims in research articles by inferring what the article authors' valuations were.

Tichy's later argument for increasing experimentation in computer science, published in the June 1998 issue of *IEEE Computer*[937], faces the same difficulties as Tichy et al.'s earlier argument. Tichy assumed that (*1*) the primary subject of inquiry in computer science is information; and (*2*) although information is neither energy nor matter, it should be studied following the traditional scientific method. Objections to these assumptions are easy to formulate. For instance, if the primary focus of computer science is defined as something other than information, Tichy's argument faces difficulties. Similarly, if information is irreducible, emergent, or epistemologically subjective (always interpreted), Tichy's argument faces difficulties. Tichy assumed falsificationist ideals and a reductionist view of information, but if one adopts a different philosophy of science or a different assumption about the essence of information (or if one questions the definition of the term *information*, which is not very difficult because the term *information* is notoriously ambiguous), one can equally successfully argue for, for instance, a hermeneutic research tradition instead of the scientific method.

937Tichy, 1998

Section Overview

Although I am not looking for particular *trends* in the development of computing as a discipline, and although I have not done a formal analysis of the sample articles, looking at how discussions about the discipline have developed reveals a number of overarching motifs that may partly explain the development of the discipline. I have identified three particularly lucid endeavors.

The first motif in my sample of articles is *external detachment*: In these articles the authors have argued that computing is a new discipline instead of being a part of some other discipline. Their prime motivation is descriptive—to argue for a recognition of computing as an autonomous discipline based on the uniqueness of computer science, and thus to argue for a redrawing of disciplinary boundaries. In these articles the authors have usually contrasted computing with mathematics or engineering, or discussed science in general terms[938].

The second motif is *internal collectiveness*: In these articles the authors have tried to formulate an overarching understanding of computing as a discipline. There are discussions about the academic credentials of the discipline, as well as discussions about who is considered to be a computing professional. There are both descriptive and normative characteristics in these discussions. On one hand, some authors have tried to develop a collective picture of computing as a discipline by describing what computer scientists *actually do*[939]. On the other hand, some authors have tried to lead computer scientists by arguing for what computer scientists *should do*. Normative accounts have been approached basically from two directions: Some authors have weighed aspects of computing and then argued for the practical importance of some aspects (such as programming) over others, whereas other authors have argued for an intellectual or conceptual supremacy of one viewpoint of computing (such as formal proofs of correctness) over others[940].

The third motif is *internal expansion*: In these articles the authors have argued for an extension of the discipline to topics that have not been previously considered to be

938 See, e.g., Newell et al., 1967; Forsythe, 1968; Hamming, 1969; Knuth, 1974; Dijkstra, 1974.

939 See, e.g., Forsythe, 1967; Atchison et al., 1968; Austing et al., 1979; Denning et al., 1989; Hartmanis et al., 1992; Brookshear, 2003:p.1.

940 See, e.g., Hamming, 1969; Minsky, 1970; Naur, 1966; Dijkstra, 1972; Khalil and Levy, 1978; Lee, 1989.

part of the discipline. There are both descriptive and normative motivations. Firstly, some authors have argued that people in the field of computing are actually doing a certain topic already, and that the topic should therefore be formally identified as a subfield of computing (such was the case with, e.g., networks and operating systems[941]). Secondly, some authors have argued that it is important to steer some effort to some new topics that have traditionally not been a part of the discipline (such was the case with, e.g., HCI and software engineering[942]).

It should be noted that the separation of the above-mentioned endeavors into normative and descriptive is somewhat rough; some descriptions of computing could with ease be read as guidelines to computing *proper*. The same problem of ambiguity between normative and descriptive accounts of science was visible in Kuhn's *normal science*: Philosophers of science criticized Kuhn for ambiguity about the nature of his account of science[943]. (Note that Kuhn's account was not criticized as much for making normative claims as for not making normativity clear.)

Kuhn indeed conceded that his work includes both descriptive and normative sides, but argued that if one has a theory of how science works, that theory must have some implications for how scientists should work[944]. Note that the key word in Kuhn's reply is *some*: The less emphasis the word *some* is given, the stronger becomes the objection that if the norms of science are drawn *only* by efficiency, then no practices are ruled out—not even the most unethical ones. But if computer scientists have to abide by ethics, then a normative account of computer science cannot include unethical practices. That is, then a normative account of computer science cannot be based on scientific arguments only. In effect, insofar as computer scientists are members of society and bound by the norms of society, scientific practice and societal norms cannot be separated—and insofar as scientists are moral people, scientific practice cannot be separated from ethical questions.

941 See, e.g., Glass, 2005 for an overview of software history; Rosen, 1972 for the development of operating systems between 1965 and 1975; and Spier, 1974 for an early explanation of why operating systems deserve a subdiscipline of its own.

942 See, e.g., Naur & Randell, 1969; Minsky, 1979; Simon, 1981; Hopcroft, 1987; Stevenson, 1993; Sommerville, 1982:pp.2-3; Baskerville et al., 2000:p.63. See also Randell, 1979 for a historical account of the emergence of software engineering and Baecker et al., 1995 for a history of HCI.

943 Feyerabend, 1970

944 Kuhn, 1970

There is significant chronological overlap between the three motifs mentioned above —*external detachment*, *internal collectiveness*, and *internal expansion*. The detachment of computing from the disciplines that gave birth to computing began in the 1950s, and, as computing as a discipline gradually gained legitimacy, significant debates about the disciplinary autonomy of computing ceased after the 1970s. Expansion has been a characteristic of the field of computing throughout the existence of the field; for instance, there are topics of computing such as artificial intelligence, human-computer interaction, and software engineering that are considered to belong to the field of computing today, but that were once rejected. Computing as a discipline has also been overshadowed by an identity crisis throughout its history; every decade the boundaries of the discipline have been redrawn, and every decade the debates about the disciplinary identity of computing have taken new forms.

There have been different categorizations of different aspects of computing. One of the most oft-quoted, but perhaps the most vague, is the dichotomy between *science* and *art*[945]. This dichotomy emphasizes the logico-rational side of computing and the creative crafting side of computing. Variations of the same juxtaposition can be seen in the division between abstract and pragmatic[946], theory and practice[947], form and content[948], academy and industry[949], computing and computer[950], algorithms and programs[951], science and technology[952], and so forth[953].

The characterization of computing through its scientific and practical aspects unnecessarily excludes important characteristics of computing. An alternative categorization of the different aspects of computing is the tripartite of the field into its *mathematical*, *scientific*, and *technological* aspects[954]. This division is also characterized as *theory*, *modeling*, and *design*[955], as well as *theory*, *numerical analysis*, and *sys-*

945 Forsythe, 1967
946 Forsythe, 1968
947 Knuth, 1991; Minsky, 1970
948 Minsky, 1970
949 Kandel, 1972; Egan, 1976
950 Dijkstra, 1987; Dijkstra, 1972; Brooks, 1996
951 Khalil and Levy, 1978
952 Hopcroft, 1987
953 One can easily come up with dozens of similar juxtapositions, e.g., universal vs. particular; scholarly vs. professional; abstract vs. concrete; global vs. local; objective vs. subjective; formal vs. intuitive; form vs. function; general vs. particular; pure vs. applied, etc.
954 Wegner, 1976
955 Denning et al., 1989

tems[956]. I argued that in addition to the *theory*, *modeling*, and *design* aspects of computing, an *ethical* component is necessary, because the decisions that computer scientists make are not separate from societal concerns—if simply for the fact that neither computer scientists nor products of computer science are outside of society.

There is a broad variety of definitions of computer science which include an equally broad variety variety of *research subjects* of computer science. The subject of computer science has been argued to be, for instance, *information*[957], *computers*[958], *algorithms*[959], *information structures and processes*[960], *computing machines* (actual or potential)[961], *the phenomena surrounding computers*[962], *information processing systems*[963], *the nature of data and use of data*[964], *classes of computations*[965], *computer programming*[966], *complexity*[967], *automation*[968], *users*[969], and *models*[970].

In a similar vein, computer science as an activity has been argued to include, for instance, *representing*, *processing*[971], *designing*[972], *mastering complexity*[973], *formulating*[974], *programming*[975], *doing empirical studies*[976], and *modeling*[977]. Computer science has been conceptualized as mathematics, as engineering and design, as an art, as a science, as a social science, and as an interdisciplinary endeavor[978].

This gamut of topics and approaches does not fit under a single epistemological or methodological system. To cope with this variety, computer scientists have em-

956 Rice and Rosen, 2004
957 Forsythe, 1967; Finerman, 1970
958 Forsythe, 1968; Hamming, 1969; Brooks, 1996
959 Knuth, 1974; Denning et al., 1989
960 Atchison et al., 1968
961 Finerman, 1970
962 Newell et al., 1967
963 Atchison et al., 1968
964 Naur, 1966; Lee, 1989
965 Dijkstra, 1972
966 Khalil and Levy, 1978
967 Dijkstra, 1999; Dijkstra, 2001; Simon, 1981; Minsky, 1979
968 Denning et al., 1989
969 Lee, 1989; Shneiderman, 2002; Mahmood, 2002; Johnson, 1998
970 Stevenson, 1993; Denning et al., 1989
971 Forsythe, 1967
972 Dijkstra, 1972; Forsythe, 1967; Denning et al., 1989
973 Dijkstra, 1974
974 Dijkstra, 1974
975 Khalil and Levy, 1978
976 Wegner, 1976
977 Denning et al., 1989; Stevenson, 1993
978 Goldweber et al., 1997

ployed methods from a diversity of fields. In this sense, computer scientists are jacks-of-all-trades. However, it is not certain if the resulting computer science is valid in any of the fields that are utilized in computer science. This may be due to the fact that computer science curricula frequently lack the sine qua non of *researcher-as-bricoleur*: a broad education in methodologies.

It seems that situating computing in the *scientific*, *technological*, *economical*, *institutional*, *cultural*, *philosophical*, *personal/individual*, and *political* context opens new windows to *why* computing has developed as it has. Even if one knew the current state of computing field inside out, it would be difficult to claim to understand computing if one did not know the circumstances in which computing had developed. There is a difference between knowing *what is*, and knowing *why it is*, and social studies of computer science is perhaps needed to explain the *whys* of computer science.

For instance, one could determine that FORTRAN is the dominant language in scientific computations, but without knowing the history of FORTRAN one does not why does it dominate (it is certainly not for the technical merits of FORTRAN). One could speak with confidence about a division of the discipline of computing into theory, modeling, and design, but without knowing the history of the discipline one does understand why the tensions between these three parts exist. In a similar vein, it is a matter of disciplinary self-understanding to be aware of the background of the software crisis and its suggested solutions, the sociocultural history of the stored-program-paradigm, the philosophical criticism of iterative design, the recurring polemics about the term *computer science*, and so forth. Those topics fall within the area of social studies of computer science. In the following chapter I characterize and analyze my vision of social studies of computer science.

4.Social Studies of Computer Science

> Our objective in this book is to state precisely and clearly where and why sociological analysis is necessary in the understanding of scientific knowledge. Our main method is to present historical case studies. We then show how sociological analysis applies in these cases, and how it is an essential complement to even the most insightful interpretations derived from other perspectives.
>
> Barnes, Bloor, & Henry (1996)
> *Scientific Knowledge: A Sociological Analysis*

Social studies of computer science, as an extension of computer science, refers to a multidisciplinary, multi-method approach to computer science, but in this thesis some specific reference disciplines, in relation to social studies of computer science, are foregrounded: *sociology*, *history*, *anthropology*, and *philosophy*. Using these four disciplines as prototypical labels for social studies of computer science does not mean that social studies of computer science should be limited to these four disciplines. Social studies of computer science could be done from the point of view of many other disciplines, too, such as psychology, communication studies, or economics. Foregrounding sociology, history, anthropology, and philosophy is, however, usual in social and cultural studies of science[1].

In the previous chapter I raised doubts whether the methodological toolbox of computer science would be adequate for selecting, recording, understanding, explaining, analyzing, or predicting human phenomena. In this chapter I focus on some qualitative methods that are common in the social sciences and humanities (albeit there is a lot of quantitative research in the social sciences and humanities, too[2]). I take it that computer scientists are more familiar with quantitative than qualitative research methods, and in this chapter I focus on the less familiar, qualitative methods. It must be noted that social studies of computer science should be open for alternative and complementary research approaches.

Opening the door for various kinds of research has its flip side, too. Norman Denzin and Yvonna Lincoln argued that it is very difficult to move from paradigm to another paradigm within one study, because scientific paradigms are overarching philosophical systems[3]. Yet they also noted that there is fundamental difference between *paradigms* and *perspectives* in research. They wrote that one can more easily move

1 Pickering, 1995:pp.215-217.
2 Bernard, 1995
3 Denzin and Lincoln, 1994:p.2

from perspective to perspective within the same study because *perspective* is a looser concept than *paradigm*. Denzin and Lincoln continued that the use of multiple methods, multiple empirical materials, multiple perspectives, or multiple observers to understand a phenomenon—sometimes called *triangulation*, and lately *crystallization*—is often used to add rigor, breadth, or depth to investigation[4]. Limiting the discussion in this chapter to a few qualitative research methods is not a position towards either quantitative or mixed-method research, but a practical choice given the target audience of this thesis, computer scientists. The research approaches and viewpoints presented in this chapter are chosen because they produce different kinds of knowledge than the common theoretical-modeling-scientific toolbox of computer science does.

The first aim of this chapter is to form an overall picture of the academic location for fields such as sociology, history, anthropology, and philosophy as extensions of computer science. The second aim of this chapter is to establish the intellectual contribution of social studies of computer science. Having these two aims in mind, in this chapter a descriptive account of computer science is formed first (Section 4.1), followed by an articulation of how computing could benefit from sociological, historical, anthropological, and philosophical perspectives (Section 4.2). Then I argue why research done from these perspectives belongs to computer science and not to sociology, history, anthropology, or philosophy (Section 4.3), and finally I discuss some implications of this study (Section 4.4).

4 Denzin and Lincoln, 1994:p.2. See Stake, 1994, for discussion about triangulation in case studies; see Johnson & Onwuegbuzie, 2004, for a discussion about arguments for and against mixed method research.

4.1.Computer Science: an Efficient Anarchy

By claiming that they can contribute to software engineering, the soft scientists make themselves even more ridiculous. (Not less dangerous, alas!)[5]

In This Section:

- ✔ A characterization of *computer science* for the rest of the chapter.
- ✔ What is the theoretical-conceptual framework of computer science?
- ✔ Is methodological and epistemological anarchism detrimental to computer science?

In the beginning of Chapter Two I noted that there are multiple interpretations of the term *science*. Also, *computer science* can be understood in a number of ways. For instance, it can be understood as specific *classes of activities*, such as modeling, creating theoretical explanations, developing, automating, or designing[6]. It can be understood as a *way of thinking*[7] in which an individual is able to switch between abstraction levels and simultaneously consider microscopic and macroscopic concerns[8]. It can also be understood as an *umbrella term* for a large variety of topics, such as robotics, e-commerce, visualization, and data mining[9]. Computer science can be understood as a rigid *institution* (e.g., "academic computer science"), but computer science can also be understood as having a temporal dimension, thus forming a *historical continuum*: "the 45-year path of computer science". It can be understood as broadly as *studies of phenomena surrounding computers*[10] or as narrowly as *computer science = programming*[11]. Computer science can be interpreted as the intersection or union of subjective and objective, or heuristic and formalistic issues: *the art and science of processing information*[12]. Computer science can be also understood as a profession[13]. Many of these interpretations of computer science can have both normative and descriptive meanings, and

5 Dijkstra, 1975b
6 Denning et al., 1989
7 Arora & Chazelle, 2005
8 Dijkstra, 1987; Dijkstra, 1974; Knuth, 1974
9 Zadeh, 1968; Denning, 2003
10 Newell et al., 1967
11 See, e.g., Denning, 2004, Denning et al., 1989, Ralston, 1981, Ralston and Shaw, 1980.
12 Forsythe, 1967
13 See Computing Sciences Accreditation Board's web page for *Computer Science: The Profession*: `http://www.csab.org/comp_sci_profession.html` (accessed September 27th, 2006)

they can be combined, pooled, and re-organized to produce an even greater variety of interpretations of computer science.

Although the purpose of this thesis is not to offer yet another definition of computer science, for clarity's sake it is necessary to characterize how the term computer science is used in this chapter. In this and following sections, the term *computer science* is used as a loose collection of topics and concepts regarding studies of *computers and phenomena surrounding computers*, that is, studies that aim at contributing to knowledge about automatic computation or at refining computational tools, theories, concepts, or processes. In this sense, computer science can include topics and concepts from fields such as hardware design, human-computer interaction, information systems, computer architectures, multimedia design, and programming languages.

Because of the vagueness of the term *computer science*, some qualifiers of the term are used hereafter when necessary. Computer science *qua activity* refers to the explicit and implicit practices, methods, uses of tools, meaning negotiations, or other modi operandi that are a part of the activities of computer science. Computer science *qua knowledge* refers to the implicit or explicit conceptual and theoretical frameworks, methodologies, epistemological and ontological presuppositions, shared knowledge, tacit knowledge, or other types of information that are a part of the knowledge of computer science available in the public domain.

I do not wish to set a temporal limit on the use of the term *computer science* in this context—therefore, the abacus, systems of numeration, and other old inventions, as well as the tacit or explicit theories of how they work, can be understood as the constituents and products of computer science. Insofar as it can be said that scholars in Ancient Greece worked on biology and physics, computer science can be said to have a long history, too. For instance, ethnocomputational[14] concepts and mechanisms such as the Antikythera mechanism (65 B.C.), Inca Quipus (as far as 3000 B.C.), Bamana sand divination, many African fractal concepts, and, certainly, the ancient concept of the algorithm[15] can be considered to belong to computer science. However, my discussion focuses on the past 60 or 70 years of computer science.

14 A term from Tedre et al., 2006.

15 See Solla Price, 1959; Ascher & Ascher, 1981; Eglash, 1997; Eglash, 1999; Zemanek, 1979, respectively.

There are parts of computer science that are considerably stable in the sense that they have served as an uncontested basis for research for long periods of time. Of these considerably stable parts of computer science, perhaps the one that is most well-known and the most associated with computer science is the stored-program paradigm, which was formed during the early 20th century, and was epitomized in the *First Draft of a Report on the EDVAC* and the construction of *BINAC* and *EDSAC*[16]. Although many constituents of modern computer science—such as algorithms, the universal Turing Machine, and the stored-program-paradigm—date back a long time, the field at large has changed radically between the 1940s and 2000s.

Although there are considerably stable parts of computer science, knowledge about computing, activities of computer scientists, and computer science as a discipline have changed significantly after the conception of digital electronic computing. For instance, researchers of computational complexity have constantly redrawn the practical boundaries of computing, practices of program and model construction have changed, and concerns about efficiency (and usability) have led to new fields such as software engineering and human-computer interaction[17]. Computer science is today considered to include fields such as software engineering, the social implications of computing, knowledge representation and reasoning, data modeling, network security, and human-computer interaction[18]. Those fields have sometimes arisen from the intersection of computer science and other disciplines, sometimes from specialized topics in computer science, but they had not been research fields of their own before the emergence of digital computing.

Research in Computer Science

I shall not repeat the discussion about the different designations of either the *computer* or the *science* part of the term *computer science*. Both parts are disputed. Surely, the term *science* in *computer science* is not a technical notion that would define the content or form of the discipline, but computer *science* certainly is a popular notion and a commonly used term. The inclusion of *science* to describe the discipline was done either implicitly or explicitly over the course of time, and the frequency of discussion about the term *computer science* (see Chapter 3.4, passim) sug-

16 Neumann, 1945; Worsley, 1950; Mauchly, 1979
17 Grudin, 1990
18 Denning et al., 2001:p.17.

gests that the inclusion of *science* to describe computer science was chosen deliberately.

The number of research topics in computer science has increased since the official establishment of the discipline, and the topics have also diversified. The 25-item list of computer science topics in 1968 (Figure 21, page 269) consists of mathematical and engineering topics, but the 30-item list of computer science topics in 2003 (Figure 7, page 69) consists of a variety of topics such as *e-commerce*, *workflow*, *human-computer interaction*, and *computational science* that have arisen from the cross-section of computer science and other fields such as business, psychology, cognitive science, sociology, and natural sciences. The perspectives on what kinds of research are considered to be computer science have changed over the 35 years between 1968 and 2003, and even more over the 60 year-history of electronic, digital computing. Computer scientists of the 1950s might not have considered intellectual property issues, software project management, graphics and visual computing, or data modeling as computer science at all. For instance, as late as 1968, computer scientists were hesitant to include even such technical topics as *programming* under the umbrella of computer science[19].

A comparison of the official curricula recommendations between 1968 and 1996 suggests that the *research approaches* have also diversified after 1968. Curriculum '68 is strictly computer-centered and mathematical, whereas, around the year 1996, interdisciplinary and multidisciplinary trends are writ large on curricula[20]. The number of utilized and approved research approaches has increased simultaneously with the increase of the number of topics of computer science.

Scientific Statements in Computer Science

There is uncertainty about what are *scientific statements* in computer science. Because of the insurmountable ambivalence of the term *facts* (discussed in Chapter Two), I use a more relaxed term, *scientific statements*, instead. In this section, the term *scientific statements* in computer science refers to statements that are meant to be either (1) a basis for future computer science; (2) alternatives to, refutations of, or criticisms of earlier scientific results, or (3) information for extra-scientific stake-

19 Atchison et al., 1968

20 See Goldweber et al., 1997, who discuss the development of computing curricula, including, for instance, Atchison et al., 1968; Austing et al., 1979; Tucker et al., 1991; and Denning et al., 1989.

holders such as educators, policy-makers, and the general public. The notion of *scientific statements* is not a technical but an intuitive one, and as such, it is, in the end, subjectively interpreted.

During the first half of the history of digital electronic computing, 1945-1975, when computer science was considered to be a mathematical and engineering discipline, the natural scientific community at large was geared towards proving or falsifying theorems[21]. I have shown earlier that, in practice, computer scientists at the time did not work like falsificationists ought to have worked. It has been argued that computer scientists today do not conform to the falsificationist ideals either[22].

In 1968 Dijkstra made a normative statement that "*the GO TO statement should be abolished from all 'higher level' programming languages*" based on his subjective observation that GO TO statements increase code entropy[23]. Although no empirical research was ever performed to measure the truth value of Dijkstra's hypothesis of the problematic nature of GOTOS, computer scientists gradually shifted towards less GO TOS in programming[24]. Dijkstra's statement was not founded on empirical results or formal proof but on his experience. The persuasiveness of Dijkstra's statement originates from somewhere else than its scientific merits—perhaps it originates from the intuitiveness of his statement and from his colorful and persuasive language evident in, for instance, phrases such as "*the use of the GO TO statement has such disastrous effects*", "*as wise programmers aware of our limitations*", "[the] *GO TO statement* [...] *is just too primitive*", and "*an invitation to make a mess of one's program*".

If Dijkstra's statement "*GO TOS are harmful*" can be considered to be a scientific statement (instead of a pseudo-scientific statement) in falsificationist terms, the conditions under which the statement is falsified must be clear[25]. For instance, it could be asked if Dijkstra's statement is falsified if one can find one case in which a GO TO decreases code entropy. Knuth showed that it is possible to program in a structured manner using GO TOS[26], yet computer scientists still have not considered Dijkstra's statement to have been falsified (or, alternatively, GOTOless programming is popular

21 Chalmers, 1976:pp.1-4,59-61.
22 Tichy et al., 1995; Tichy, 1998
23 Dijkstra, 1968
24 Glass, 2005; MacLennan, 1999:pp.126-127.
25 Popper, 1959:pp.65-66.
26 Knuth, 1974b

despite the refutation of Dijkstra's statement). In falsificationist terms, if one cannot divide findings into those that permit Dijkstra's statement and those that falsify Dijkstra's statement, Dijkstra's statement is not falsifiable, and as such, it should be considered to be pseudo-science. (Indeed it is a difficult task to state clearly when Dijkstra's statement would be refuted.)

On one hand, it might be difficult to consider Dijkstra's statement to be a scientific statement because it has not been empirically tested. On the other hand, much research and practice in computer science has been built on the assumption that GO TOS are harmful for the clarity of code. One way to take Dijkstra's statement is as a *folk theorem*, a hypothetical statement or theorem, which is widely held but has not been not empirically tested. However, although it has been argued that there are many folk theorems in computer science[27], their place in the conceptual and theoretical framework of computer science is dubious.

Social Constructionism in Computer Science

In the wake of social constructionism in the early 1960s, the factuality of positivist statements about science ("facts") as well as the mechanisms of falsificationism increasingly came to be questioned. Kuhn's work[28] questioned the foundations of scientists' work in the modeling and design fields of computer science, and Lakatos' work[29] questioned the foundations of mathematical proofs, which are typical of theoretical computer science. Both Kuhn's and Lakatos' arguments have been employed in computer science. For example, in his 1978 Turing Award speech *The Paradigms of Programming*, Robert W. Floyd characterized changes in programming with the Kuhnian term *paradigm* (in my opinion, incorrectly[30]). In the May 1979 issue of *CACM*, De Millo et al. wrote the following about social processes and proofs of theorems and programs: "*Contrary to what its name 'proof' suggests, a proof is only one step in the direction of confidence*"[31]. DeMillo et al. argued that,

27 Harel, 1980; Denning, 1980

28 Kuhn, 1996 (orig. 1962)

29 Lakatos, 1976; Lakatos, 1970

30 Floyd, 1979. Floyd apparently used the term *paradigm* in the meaning of an *exemplar* (discussed in Kuhn, 1996:187-191). This meaning of paradigm is indeed more suitable for *programming paradigms* than the more common understanding of a paradigm as a *disciplinary matrix*. However, Floyd did not mention any *anomalies* that would have ever triggered a paradigm shift in programming. My interpretation is that Floyd used the term *paradigm* in a different way than Kuhn did.

31 De Millo et al., 1979

for instance, proofs of program correctness are social constructs, and referred to Lakatos' work.

In 1980, in two separate issues of *CACM*, David Harel and Peter Denning discussed a number of folk theorems or folk myths in computer science, theorems which are simple, intuitive, widely believed, of obscure origin—and some of which are false[32]. These are, Denning wrote, usually referred to as "well-known" theorems. Today there are even a number of rules-of-thumb called "laws" in computer science—take for instance, Moore's Law, Rock's Law, Machrone's Law, Metcalfe's Law, and Wirth's Law[33]. It is strange that there do not seem to be many objections to the lax use of the term *law* in the field of computing. Although one might argue that all computer scientists know that those rules-of-thumb are not laws proper, negligent use of the term *law* blurs its meaning.

The unclarity about the nature of research results in computer is also visible in how computer science is done. It is difficult to single out a theoretical framework for computer science; should any of the dominant frameworks of science be adopted as a normative framework for computer science *proper*, some established topics from the descriptive framework of computer science would be excluded. The fact that no single methodological system is applicable to the methods of inquiry in computer science means that computer science is methodologically disunited. The fact that there is no consensus about the nature (epistemological status) of research results in computer science means that computer science is epistemologically disunited. If different epistemological and methodological stances towards phenomena in computer science are kept apart, computer science can be characterized as a *multidisciplinary enterprise*. If different epistemological and methodological stances towards phenomena in computer science are blended, computer science can be characterized as an *interdisciplinary enterprise*.

Anarchism in Computer Science

Multidisciplinary approaches to science employ research approaches from a number of disciplines to offer a variety of perspectives of the same phenomena. In multidisciplinary approaches, the different research approaches are utilized without

32 Harel, 1980; Denning, 1980
33 Ross, 2004

modification. *Interdisciplinary approaches* blend two or more disciplines together to form a new field or approach. Sciences that can be characterized as interdisciplinary include, for instance, biocomputing (computer science, biology, and biotechnology)[34] and quantum computing (modern physics, computer science, and material science)[35].

Computer science can be characterized as an interdisciplinary science to some degree: Computer science originates from numerous disciplines, yet it offers unique systems of explanation, such as computational models and algorithms. Computer science is not an *antidisciplinary synthesis*, though. Characteristically, antidisciplinary syntheses eliminate the friction between and around the different research approaches[36]. Quite the contrary to antidisciplinarity, and in addition to interdisciplinarity, computer science bears some characteristics of a multidisciplinary science: A number of incompatible research approaches are used in different branches of computer science. The boundaries around disciplines that have come together to form computer science still exist within computer science, and the boundaries distinguish labels such as *theoretical computer science*, *human-computer interaction*, and *computer architectures*.

There is no stern watchdog Computer Science to enforce methodological regimentation and to rule out non-legitimate, ill-suited, or inappropriate methods, tools, or approaches. It is, in fact, difficult to imagine a united methodology or epistemology for computer science, due to the vast number of different sorts of concepts computer scientists have to cope with—computer scientists have to work with, for instance, logic elements, software systems, network protocols, and human-computer interfaces. The results and statements in the field of logic circuit design are different from the results and statements in, for instance, software engineering, information retrieval, and robotics. The multiplicity of epistemological and methodological views render computer science a methodologically and epistemologically eclectic discipline. That is, there is no methodology or epistemology that would be commonly agreed upon as being superior to other methodological or epistemological views.

34 Thacker, 2004:p.99.

35 Berman et al., 1998:p.1.

36 See Pickering, 1995:p.216 for *antidisciplinary new synthesis*. Note that antidisciplinary does not mean *anti-science*. There is a number of other approaches related to inter- and multidisciplinarity, such as *crossdisciplinary* and *transdisciplinary* approaches, but the differences between the different multiperspectival approaches are not the focus of this thesis.

Despite the methodological and epistemological multiplicity in computer science, the unwritten science policy or attitude in computer science cannot be said to be laissez-faire, non-interference, or pluralism. There can actually be strict rules *within* branches of computer science (for instance, theoretical computer science relies strictly on positivist methodology), representatives of different branches may consider other branches as less worthy[37], and proponents of some approaches to scientific inquiry may impose their approaches on others[38]. However, although individual computer scientists may be strict about their views of science, computer scientists as a professional group cannot be said to profess positivism, constructionism, postpositivism, objective or subjective Bayesianism, or such. If there is a term to describe computer scientists as a professional group, it is opportunism.

In their education, computer scientists get very little or no methodological training, but they usually learn through mentoring and exemplars. A conscious methodological non-regimentation and a disregard of frequent calls for more methodological rigor render computer science an epistemologically and methodologically anarchist discipline. Note that neither those sociologists of scientific knowledge who argue for rather anti-realist views of scientific knowledge nor those philosophers of science who advocate anarchistic science, claim that all science is equally important[39]. However, not all sociologists of knowledge nor all philosophers of science believe that there is a single correct approach to intellectual inquiry.

The Outcomes of Anarchism

During the past 60 years, research in computer science has helped the creation of a number of new research fields, spurred research in other disciplines, deepened knowledge about automatic computing, and advanced computing technologies. My reading of the history of computer science is that computer science has been efficient[40] *because* of anarchism, not *despite* it. Anarchism had already been woven into computer science (from different disciplinary threads) from the beginning of early electronic computing. The birth of the stored-program-paradigm was already a result of a successful combination of a number of epistemologically and methodologic-

37 Glass et al., 2004

38 Tichy, 1998

39 See Hacking, 1999:p.65; See Horgan, 1996:pp.52-54.

40 *Efficient* as a term refers to a high ratio of output to input. It should not be confused with *effective*, which refers to having an intended or expected effect (AHD, 2004).

ally incompatible disciplines. Over the past 60 years, computer science has been influenced by a large and eclectic bunch of disciplines. Had computer scientists adhered to a single normative theory of science, it would have been impossible to utilize the theories, concepts, models, or methods of incompatible disciplines. For instance, Goldweber et al. argued that anthropology, applied psychology, computer science, cultural studies, economics, ergonomics, ethics, history, linguistics, management, mathematics, philology, philosophy, semiology, sociology, and politics have been relevant to the development of computing[41]. Goldweber et al. argued that one cannot do justice to this diversity by applying a single disciplinary perspective.

In addition, without anarchism in computer science, the innofusion of many innovations in computer science could have been much slower (and some innovations might have not been introduced at all). For instance, the quick innofusion of innovations such as the stored-program-paradigm, high-level programming languages, and structured programming can be attributed to technoscientific anarchism—that is, to the fact that those innovations did not need to undergo rigorous empirical testing before earning an official stamp of approval. Instead, those innovations are products of (and still subject to) the mangle of practice.

Anarchism can also be seen in that many innovations in computer science have been contrived despite the lack of support by the establishment (sometimes even despite strong opposition from the establishment). Two especially lucid examples arise from the narrative in Chapter Three: Firstly, the academic establishment was strongly opposed to the construction of ENIAC, and secondly, high-level programming languages were developed despite the resistance of the computer science community. It is not, however, my purpose to argue whether or not anarchism *ought to be* a tenet of computer science, but just to note that it *seems* to be the primus motor of change in computer science.

The functioning of Kuhnian normal science is well-known. The logic of scientific discovery, confirmation, and justification in falsificationism is well-documented. Insofar as computer science is not a strictly regimented discipline but an eclectic, opportunistic, and anarchistic discipline and insofar as knowing the modi operandi of a discipline is considered to be important, the workings of computer science deserve to

41 Goldweber et al., 1997

be researched in their own right. In the following section I describe some viewpoints that social studies of computer science can offer and what kinds of knowledge those viewpoints produce.

4.2.Approaches to Social Studies of Computer Science

It is difficult to conceive how the behavioral and social sciences could be simultaneously trivial, useless, unscientific, and threatening.[42]

In This Section:

- ✔ What kinds of new viewpoints can social studies of computer science contribute to our understanding of computer science?
- ✔ What do all the *ethno*-approaches have to do with computer science?
- ✔ Are all the viewpoints in social studies of computer science actually new to computer science?
- ✔ What is the point of case studies?
- ✔ How does one give proofs of correctness in social studies of computer science?

Anthony Ralston attributed four reasons to computer science's diverging from mathematics: disciplinary changes, an insecure disciplinary identity, a growing number of people in the field of computing, and disciplinary disagreements[43]. Only one of Ralston's reasons, disciplinary changes, can be explained in solely (contemporary) technical and computational terms. I argued earlier that the other three reasons are perhaps better explained in other terms, such as in psychological, sociological, or anthropological terms (p. 295).

In Chapter Three I presented a number of historical accounts of computer science that suggest that modern computer science was born in the 1940s as a result of a number of organizations, a number of top people, many coincidences, a variety of disciplines, an uncommon political situation, a certain culture, unusually liberal funding, and a number of technical and scientific breakthroughs. If it is true that one cannot understand even a single, static affair in a society without understanding history[44], one certainly cannot understand a multifaceted and dynamic phenomenon such as computer science without the use of historical materials. Modern computer science has been surrounded and shaped by a vastly complex conjunction of affairs since the 1940s.

Due to their rich and colorful history, computer science and computer technologies include plenty of phenomena, the form and functioning of which cannot be explained in terms internal to those phenomena. For instance, one cannot explain the

42 Smelser, 1988:p.14.
43 See Ralston, 1981.
44 Mills, 1959:p.149.

design and the (non-)diffusion of any programming language by referring solely to computer science qua knowledge. It has been argued that understanding the design and diffusion of any programming language requires understanding its history and the original motivations which provided the impetus for its development in the first place[45].

So it is implausible that one could understand the current state, a static snapshot, of computer science qua knowledge without understanding its history. It is *nearly* tautological that one cannot understand *why* computer science qua knowledge is what it is without understanding its history. Insofar as *history* is understood as a *human history*, then history is a cultural and societal (and economic, political, ideological, etc.) history. And insofar as computer science is a product of an array of sociocultural forces, any portrayal of computer science is a historically, culturally, and societally specific image. Finally, it *is* tautological that computer science qua *human activity* always happens in some philosophical, historical, and sociocultural framework. (I certainly do not mean that computer science that is situated in a historical, cultural, and societal framework could not be objective. *Objectivity* can be defined in a number of ways that permit comparisons of socially constructed knowledge[46].)

In addition, I must make a side note that one can legitimately adopt the positivist viewpoint and argue that sciences are free of any historical, cultural, and societal influences (although the credibility of such an argument may not be straightforwardly established). However, it is equally legitimate to adopt a constructionist viewpoint and argue that understanding computer science requires understanding its historical, cultural, and societal context.

The importance of historical, cultural, and societal self-understanding of computer science are explicitly noted in both the Computing Curricula 1991 and 2001:

> [1991, 2001] *Undergraduates also need to understand the basic cultural, social, legal, and ethical issues inherent in the discipline of computing. They should understand where the discipline has been, where it is, and where it is heading. They should also understand their individual roles in this process, as well as appreciate the philosophical questions, technical*

45 cf., e.g., Denning, 2003; Rosenblatt, 1984

46 For instance, John Searle wrote, "[T]*he contrast between epistemic objectivity and epistemic subjectivity is a matter of degree*." (Searle, 1996:p.8).

problems, and aesthetic values that play an important part in the development of the discipline.[47]

The courses that concern *social and professional issues in computing* in the Computing Curricula 2001 are various: the courses cover the history of computing (SP1), the social context of computing (SP2), methods and tools of analysis (SP3), professional and ethical responsibilities (SP4), risks and liabilities (SP5), intellectual property issues (SP6), privacy and civil liberties (SP7), computer crime (SP8), economic issues in computing (SP9), and philosophical frameworks (SP10)[48].

The Contribution of Social Studies of Computer Science

Similar to other kinds of intellectual inquiry, social studies of computer science also works within a certain conceptual and theoretical framework, and entails a number of assumptions. In this thesis, those assumptions are, in broad terms, those entailed in the constructionist, contingent, non-relativist, and nominalist viewpoints to science. My perspective of social studies of computer science operates within that framework of assumptions. In other words, social studies of computer science, as understood in this thesis, entails the assumptions that much of people's knowledge is constructed (rather than absolute), that the history and development of current computer science is one out of an infinite number of possible routes (rather than an inevitable course), that there is a world of ontologically and epistemologically objective things (rather than only subjective statements about the world), and that many of the observed hierarchies and structures in computer science are constructed in order to give structure to the discipline (rather than being a result of an inherently structured world).

From a narrow point of view, social studies of computer science should have a place within computer science if social studies of computer science can contribute to knowledge about the *subjects of computer science* (see the characterization of computer science on page 374). From a broad point of view, if one considers disciplinary self-understanding to be a part of a mature discipline[49], then one should also acknowledge research that can contribute to the meta-theories, meta-knowledge, onto-

47 Tucker et al., 1991:p.73; Denning et al., 2001:p.141.

48 Denning et al., 2001:pp.141-146.

49 Barry Barnes, David Bloor, and John Henry have defended this argument well in Barnes et al., 1996:pp.*iix-xii*. They wrote, "We see the sociology of scientific knowledge as part of the project of science itself, an attempt to understand science in the idiom of science." (p.*iix*).

logy, epistemology, and methodology of a particular discipline. For example, De Millo et al.'s research on theory-formation in computer science is a contribution to the meta-theories of computer science[50], Harel's research on theorems that are untested yet widely held contributes to the meta-knowledge of computer science[51], Brian Cantwell Smith's *On the Origin of Objects* is a study of the ontology and the epistemology of computer science[52], Kidder and Suchman have contributed to the understanding of how computer scientists actually work[53], and there are numerous examples of research on the methodology of computer science[54]. Other aspects of a broad interpretation of computer science can be considered to be, for example, sociocultural impacts of computing and computing ethics. The left-hand oval (with the dotted line) in Figure 29 portrays the narrow interpretation of computer science and the right-hand oval (with the dotted line) in Figure 29 portrays the broad interpretation of computer science (MO in Figure 29 stands for modi operandi, that is, techniques or methods of working).

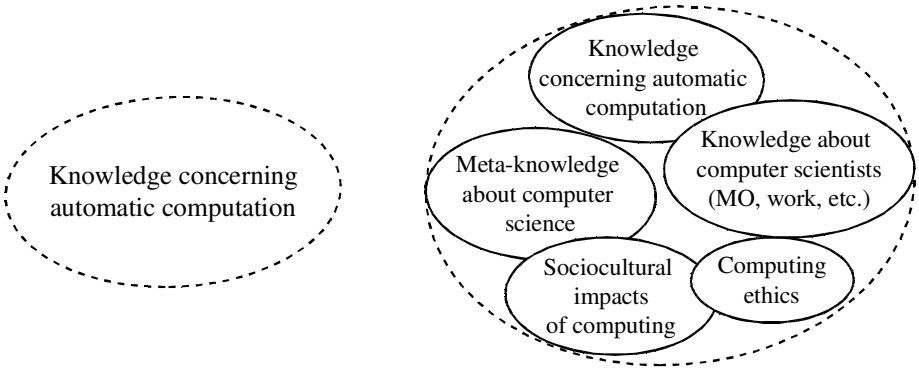


Figure 29: Narrow and Broad Interpretations of Computer Science

The status of social studies of computer science as a part of the discipline of computer science depends on one's view of computer science. If one were to agree that disciplinary self-understanding is a part of a scientific discipline, it follows that if it can be shown that social studies of computer science contributes either to knowledge concerning automatic computation or to meta-knowledge about computer science, then the social studies of computer science should have a place within computer science. Naturally, one need not consider just *any* type of research that could *possibly*

50 De Millo et al., 1979
51 Harel, 1980; see also Denning, 1980; Ross, 2004.
52 Smith, 1998
53 e.g., Suchman, 1987; Kidder, 1981
54 Tichy et al., 1995; Glass, 1995; Alavi & Carlson, 1992; Vessey et al., 2002; Palvia et al., 2003 Glass et al., 2004

contribute to computer science as being computer science. Ultimately the community of duly recognized computer scientists judge what they will accept as being computer science proper. Researchers who consider their research to be computer science can only be persuasive about the positive contribution of their research to computer science and hope that other computer scientists think so, too.

I will portray the intellectual contribution of social studies of computer science to the broader field of computer science by referring to some aspects of research that social studies of computer science entails. I discuss (1) three different sources of information, (2) a linkage to the sociohistorical context, (3) ethnomethodology, (4) ethnographic methods, (5) a non-generalizing focus on cases, and (6) measures of interpretive research. Those six research aspects are not a central part of traditional computer science, albeit they are not completely excluded from computer scientists' toolboxes. I connect my discussion with examples of existing studies of computer science in order to show that computer science qua knowledge has benefited when researchers have incorporated those aspects of research in their inquiry.

It is not necessary for my argument to discuss research methods or research approaches in a very detailed manner. It is, in fact, difficult to say anything very conclusive about qualitative methods without going into too much detail. As an example of broadness vs. detail, consider *The Handbook of Qualitative Research*, edited by Norman Denzin and Yvonna Lincoln[55]. The second edition of the book has 643 pages and 36 relatively broad chapters, resulting as a handbook of non-specific guidelines (e.g., Chapter 15: *Ethnography and Participant Observation*). The third edition of the book has 1210 pages and 45 specific chapters, resulting in an encyclopedia of narrowly applicable, specialized methods (e.g., Chapter 21: *Critical Ethnography as Street Performance*).

I do not discuss paradigm choices, methodological issues, or research design here, either. They are not of primary importance in charting the possibilities that social studies of computer science could provide. Of course, in actual social studies of computer science, research design and paradigm commitment (or non-commitment) have to be catered for and made clear, because they set the basis for the strategies of inquiry, form a framework for the research, outline the scope of the research, and

55 Denzin and Lincoln, 1994; Denzin & Lincoln, 2005

bound the applicability of the research results. Although discussing methods without explicitly discussing methodology is somewhat risky, I have in this thesis discussed philosophical aspects of research widely enough so that the basic methodological grounds are in place for the discussion in the rest of this section.

Social Studies of Computing

The constructionist paradigm is not unknown in the discipline of computing. Constructionist views of knowledge can be found in computer science topics that are in close contact with the social sciences, humanities, or education field—for instance, the first of the following two quotations is from the field of computer science education and the second is from the history of computing.

> *I think it is important to remember that definitions are arbitrarily created by human beings.* [...] *It is important to remember that people who saw the computer as a new and novel experience created the definitions of computer science that exist today.*[56]

> *Yet the ideas behind computers are so common and so accepted that we tend to forget the human qualities that produced them.*[57]

Some eminent computer scientists have also stressed the ontologically subjective nature of computing. Peter Naur's book *Computing: A Human Activity*[58] is comprised of a large number of Naur's articles and essays that emphasize the notion that the field of computing is both a humanly constructed and constructive endeavor. That is, computing is "*a human activity involving certain human purposes and intents, certain human insights, and certain* [hu]*man-made tools and techniques*", but the field of computing is also a "*field that is remarkable for its feats of design and constructions of active devices of hitherto unheard of complexity and effectiveness.*"[59] In Naur's opinion, careful analysis of the issues of computing lead, to some extent, to a concern for human issues, whether they are more appropriately denoted as psychological or sociological issues. Also, Dirk Siefkes has investigated the influences that the social and cultural backgrounds have had on computing machinery and computer science[60].

56 McGuffee, 2000
57 Grier, 2002
58 Naur, 1992
59 Naur, 1992:pp.*xiii*, *xiv*.
60 Siefkes, 1997 in Freksa et al., 1997.

Rob Kling (1944-2003) was probably the first to use the term *social studies of computing*[61]. Kling's research was both on the impact of computers on society and on the impact of society on computing and information systems. The same term has been used, for instance, in Philip E. Agre's studies of technological discourse[62], and in the book *Virtual Society? Technology, Cyberbole, Reality*, edited by Steve Woolgar[63]. In the field of science and technology studies there is a wide variety of sociologically, historically, anthropologically, and philosophically oriented research about the different aspects of computing[64]. Often such research studies go under an umbrella term, such as STS or SSK.

There is a slight difference between the terms *social studies of computer science* and *social studies of computing*. One can interpret the former as denoting an emphasis on the academic discipline *computer science*, and the latter as denoting a broad focus on all computing. Because this thesis concerns an extension of computing *as a discipline*, I use the term *social studies of computer science*.

Three Sources of Information

The disciplines that have been referred to throughout this thesis—sociology, history, anthropology, and philosophy—can each contribute a unique viewpoint to social studies of computer science. The research approaches that are either explicitly discussed in this section or implicit in the sources of this section can be categorized, according to their source of information, into

(1) those that study *phenomena in situ*, or *what people do* (for instance, an ethnographic observation of practices of computer science in a real setting, or observational field studies at locations that play a part in the innofusion of computer systems—locations such as academic institutions, computer manufacturers, professional institutions, government offices, homes);

61 Kling, 1980; see also Wellman & Hiltz, 2004.

62 Agre, 1995

63 Woolgar, 2002. The rubric *social studies of computing* has also been used by, for instance, Dr. Ron Eglash from Rensselaer Polytechnic Institute, Dr. Philip Brey from the University of Twente, and Dr. Anne Fitzpatrick from Virginia Polytechnic Institute and State University.

64 Such as Viller & Sommerville, 1999; Crabtree et al., 2000; Hartswood et al., 2002; Suchman, 1987; Godin, 1997; Olazaran, 1996; MacKenzie, 1993; Bowker, 1993; Forsythe, 1993; Kidder, 1981.

(2) those that study *reports of phenomena*, or *what people say* (for instance, interviews with people in the computing field or discourse analysis of debates about computing); and

(3) those that study *mute evidence* like written texts and artifacts, the creators of which are not alive or cannot be interviewed (for instance, studies of historical records, old or new computational instruments, or statistics).

The boundaries of these three categories (research of reports, research in situ, and research of mute evidence) are admittedly vague. However, it does not really matter if all studies that can be considered to be social studies of computer science do not fit well under any single category because these categories are not presented in order to *define* social studies of computer science, but in order to present some practicable conceptual categories for types of social studies of computer science. It is easy to find studies that have aspects belonging to each of these three categories. Take sociologist Manuel Castells' commended trilogy on the information society, for instance[65]. Castells' research includes statistical and other data on social, technological, and economic developments in different parts of the world[66]. Drawing on that data, Castells made a number of arguments about the information society, and his analysis has been acknowledged by a large number of authors[67].

Castells mainly does research on *reports*; he does research on the contemporary commentaries and arguments of other philosophers, sociologists, policy-makers, and so forth. It can be argued that Castells' research also has some characteristics of research on *mute evidence*; because many artifacts and structures of information society that Castells described date back a long time and their creators are dead or unknown. Because Castells is an active and dynamic part of a number of cultures that he describes, one could argue that Castells' research also has some characteristics of research *in situ*. His research is not, however, explicitly based on *in situ* observations, and the research is not research *in situ* in the primary meaning of the term. It is difficult, and perhaps not even desirable, for a researcher who studies a phenomenon to remain a complete outsider to the phenomenon—yet being an insider to a phenomenon does not render one's research *in situ*.

65 Castells, 1996; Castells, 1997; Castells, 1998
66 Saukko, 2005 in Denzin & Lincoln, 2005.
67 See Saukko, 2005 for references.

This thesis can be interpreted to be research on *reports* because much of this thesis, especially Section 3.4, deals with near-contemporaneous commentaries about the field. That is, the source material in this thesis consists largely of recent statements about computer science, which were articulated by contemporary computer scientists, which were addressed to other computer scientists, and the authors of which are still alive. This thesis, especially Section 3.3, also has characteristics of studies of *mute evidence* because some of the sources in this thesis are archive materials that do not allow for interaction or commentary.

The Sociohistorical Context of Computer Science

No matter in what terms the shaping of computer science is presented, if computer scientists wish to retrospectively understand the reasons why computer science and computing have shaped as they have, the methods of those computer scientists must include historical methods. This is because computer science and computing are always situated in some sociohistorical context. It should be noted that historians often use terms such as *military historian* or *social historian* to denote their theoretical emphasis[68]—in a similar vein, *historian of computer science* is used in order to clarify the specific focus of the historical inquiry.

A historical study of computer science needs to link aspects of computing with changes of some kind[69], be they social, cultural, algorithmic, technological, or conceptual changes. If a researcher merely reports on some static aspects of computing in the past without a discussion of *why* those aspects are as they are or *how* those aspects have become to be, the study might best be called *archeology* rather than history[70].

One just cannot take historical accounts and use them as uncontested facts[71]. To understand historical documents or other historical information, the historian of computer science must have a *point of view*; interpretation is not possible without a point

68 cf, e.g., Tuchman, 2004 in Lewis-Beck et al., 2004.

69 cf. Lemon, 2003:pp.294-295.

70 *Computer archeology* might be a fruitful idea; there are, in the spirit of Michel Foucault's *The Archæology of Knowledge* (i.e., a method of historical analysis freed from the anthropological theme), studies in *media archeology*, such as: Zielinski, Siegfried (2006) *Deep Time of the Media. Toward an Archaeology of Hearing and Seeing by Technical Means.* MIT Press, Cambridge, MA, as well as Huhtamo, Erkki; Parikka, Jussi; Sihvonen, Tanja (eds.) (*forthcoming, 2007*) *Archæologies of Media.*

71 Tuchman, 1994

of view[72]. (Perhaps surprisingly, Popper made the same point explicit in natural sciences. He wrote that natural scientists cannot collect data without having a point of view, "*A science needs points of view, and theoretical problems*"[73].) Although the ideal of history-writing in the early twentieth century was to be true to the scientific method[74], modern history is chiefly interpretive[75].

Narrative as a form of history-writing is especially dependent on happenings[76]. For instance, it is not very revealing about the history of programming languages to write, "In the early 1950s machine language programming was popular and in the early 1960s FORTRAN was a popular programming language". A narrative becomes meaningful only when it links happenings. For instance, the historian may expound on why there was a shift from machine language programming to high-level programming, who began the shift, what factors contributed to the shift, and how the shift *actually* happened—the shift certainly did not happen overnight. An analytic history of computer science should not only explicate, for instance, what is "a (statistically) typical 1950s computer scientist" (e.g., a young male, in his 30s, with a background of electrical engineering or mathematics), but also explain the reasons why typical computer scientists of the 1950s shared those characteristics and how those characteristics affected computer science qua knowledge[77].

A historian of computer science has to be familiar with the main lines of his or her topic, and from this familiarity arises the point of view according to which the researcher chooses, reads, and analyzes the historical material[78]. Computer science and electronic computing are old enough that a historical study of computer science can include both (1) secondary sources, such as works of communications specialists, literary critics, or historians; reference guides; references of good monographs, or citation indexes, as well as (2) primary sources, found in, for instance, archives, statistics, censuses, letters, diaries, newspapers, or popular literature[79]. Whereas the authors of primary sources are always eyewitnesses to the phenomena they report,

72 Tuchman, 1994
73 Popper, 1959:p.88.
74 Hofstadter, 1968; Gottschalk, 1950:pp.230-231.
75 Hofstadter, 1968
76 Lemon, 2003:pp.298-301.
77 See Lemon, 2003:pp.295-297 for analytical and descriptive histories. Note that this thesis clearly is not an analytical history of computer science.
78 Tuchman, 1994
79 Tuchman, 1994

the authors of secondary sources have not been *first-hand* witnesses of the phenomena they write about[80]. Note that in historical studies the researcher is never expected to first gather the facts, and then arrive to a theory. In fact, *any historical question implies a theory*, for without a theory, the researcher is haphazardly collecting a conglomeration of shapeless facts[81].

The importance of the historical research about computers has been acknowledged[82], and the history of computer science (or computing) as a research field is well-established. The works on the history of computing include monographs as well as journal articles. A prime example of a history of computer science journal is the *IEEE Annals of the History of Computing*, which has been published quarterly since 1979. The monographs include, for instance, general histories of computers[83], topic-specific books[84], works on the history of modern computers[85], and works on the general history of computation[86]. Many of these historical accounts offer sociohistorical interpretations of the discipline of computing.

Historical research on computer science, such as the kind of research published in *the Annals*, includes not only the analysis of texts, but also the analysis of other mute evidence, such as devices, parts of devices, blueprints, diagrams, components, and other material traces. The problem of interpretation of mute evidence is that there may no longer be anybody alive to articulate the intentions behind the creation of the material[87]. This problem of interpretation and credence is, however, not unique to historical research. It has been argued that in all types of interactive research the analyst has to decide whether or not to take commentary at face value and how to evaluate spoken or unspoken responses[88]. In fact, social scientists have increasingly used historical methods and historians have increasingly used social sciences' methods[89].

80 Gottschalk, 1950:p.53.
81 Tuchman, 2004
82 See, e.g., Zhang & Howland, 2005; Lee, 1996; Lee, 1996b.
83 Campbell-Kelly & Aspray, 2004
84 Sammet, 1969
85 Flamm, 1988
86 Williams, 1985
87 Hodder, 1994
88 Hodder, 1994
89 Tuchman, 2004; Hofstadter, 1968 in Lipset & Hofstadter, 1968.

If there is no way to gather indigenous commentary, material artifacts pose special problems for historians of computer science[90]. A similar problem is discussed earlier in this thesis; it is noted that it is impossible to understand an unknown artifact with certainty without knowing the intentions of the creators of the artifact (see p.131 of this thesis). Historians have argued that there is no "original" or "true" meaning of an artifact outside a specific sociohistorical context[91]. Von Neumann's *First Draft of a report on the EDVAC*[92] is an example of mute evidence that needs to be situated in its sociohistorical context in order to be understood. It is difficult to capture von Neumann's intentions, motivations, and meanings, and his metaphor transfer from neuropsychology to computing technology (p.213 of this thesis) blurs even the *technological* parallels between von Neumann's draft and the language of today's computer science. Artifacts are always produced under certain material conditions embedded within social and ideological systems[93], and the EDVAC plans were produced in an especially rare social, political, cultural, and economic situation.

A narrative history of computer science portrays a *living computer science* instead of a gallery of snapshot images. Many "milestone" concepts and events in computer science have in reality been far from discrete steps—the milestone concepts and events have often been multifaceted issues, and they have formed as a result of controversies, debates, and power struggles. Besides just representing important "lessons learned", these dynamic controversies are important because everything that is considered to belong to the core knowledge of computer science is traceable to a number of controversies or discussions[94]. Looking backward to discover parallels and analogies to modern technology can provide the basis for developing the standards by which the viability and potential for a current or proposed activity is judged[95].

Although this thesis does not fully fulfill the criteria of a historical study of computer science, Section 3.3 of this thesis is based on an analysis of primary and secondary sources on the history of computing. Note, however, the lack of a historical framework, which would have connected the narrative in Section 3.3 to history at

90 cf. Hodder, 1994
91 cf. Hodder, 1994; Hodder's argument is about *texts*.
92 Neumann, 1945
93 Hodder, 1994
94 Robert L. Glass argued for the importance of knowing the "milestones" in a manner similar to this (Glass, 2005).
95 Lee, 1996

large and which would have been necessary for a historical study. The interpretation of the sources in Section 3.3 is not done with a historical framework, but with the conceptual framework presented in Chapter Two. Whereas a postmodernist historian might have read the source material as texts that purposively or inadvertently took sides in some sort of struggles for power[96], in this thesis the goal is much more modest—to merely treat the source material as accounts that more or less capture the essence of a specific phenomenon, at a specific time, at a specific place.

Ethnomethodology

Even if there were a methodology *proper* of computer science, that methodology may not correspond to how computer scientists *actually* investigate computing, to how they give structure and meaning to computing, or to how they sustain and manage that knowledge. That is, if there were rigorous, official set(s) of methods of computer science, the practices of computer scientists still might not match those official set(s) of methods. How people give structure and meaning to knowledge and how they sustain and manage that knowledge are the focus of *ethnomethodology*[97].

Similar to *methodology*, *ethnomethodology* is not a method in any straightforward sense[98]; it is more of a study of specific actions, "people's methods", which constitute the social activities of a group of people[99]. An argument could be made about the relationship between ethnomethodology and well-formulated systems of inquiry. Namely, if there were a well-formulated and extensive methodology of computer science, it could be argued that ethnomethodologists could not say much about computer science. It could be argued that ethnomethodologists studying computer science would find the same phenomena that had been written down already.

However, ethnomethodology has been successfully used in studies of how scientists of different disciplines, including mathematics and natural sciences, create and maintain knowledge[100]. Although scientific methods can be highly technical, they are

96 Tuchman, 2004. See also Lemon, 2003:pp.370-389 for a philosophical account of the problems of postmodernism in history-writing.

97 Holstein & Gubrium, 1994; Denzin and Lincoln, 1994:p.204; The term *ethnomethodology* originates from Harold Garfinkel's book *Studies in Ethnomethodology* (Garfinkel, 1967).

98 The term *methodology* is tricky: it can refer to (1) a set of principles and assumptions that underlie a set of methods; but also to (2) a study or theoretical analysis of such working methods (AHD, 2004). Unfortunately, both meanings are necessary in this subsection. The plural *methodologies* refers to various *sets of methods.*

99 Lynch, 2004; Lynch, 1996

100 Clayman, 2001

specialized instances of the much broader social phenomenon: Scientific methods are instructions that enable members to reproduce a community's practices[101]. When ethnomethodologists study natural sciences, they delve into the unexplicated, obscure foundations and features of practices that are not mentioned in the formal methodological prescriptions or reports[102]. It must be noted that ethnomethodology is not a search for subjective meaning—ethnomethodological descriptions rarely take the form of first-person experiential reflections[103]. More often, such descriptions are written in the third person, without privileging a "private" or individual vantage point[104].

Even if there were a rigorous and well-formulated methodology of computer science, the only case when ethnomethodologists could not say anything new about the methods and practices of computer science is that case in which all the practices of creating, maintaining, using, abusing, proving, refuting, negotiating, accommodating, appropriating, or contextualizing knowledge would already be explicit. However, if there is no well-formulated, rigorous, and all-extensive methodology of computer science; or even if there is a *formal* methodology of computer science that is not always followed to the letter, ethnomethodology delivers an especially attractive promise: That of explicating the *actual* ways of constructing and managing knowledge in computer science. This could be called, for instance, the in situ methodology of computer science or the tacit methodology of computer science.

In other words, ethnomethodological approaches in social studies of computer science may benefit computer science (both as an activity and as a body of knowledge) to the extent that they can expose how the philosophical, theoretical, conceptual, and methodological frameworks of computer science are created, maintained, and managed. For instance, ethnomethodological studies may be revealing about the manners in which new innovations are conceptualized by groups of computer scientists and other stakeholders; the processes through which conceptual consensus is achieved; how epistemologically subjective results in computer science are communicated, confirmed, adopted, objectified, and institutionalized into epistemologically objective facts; how knowledge is transmitted; how computer science qua know-

101 Lynch, 2004
102 Clayman, 2001; Lynch, 2004
103 Lynch, 2004
104 Lynch, 2004

ledge gives meaning to computer science qua activity and the results of computer science; how computer science qua activity generates computer science qua knowledge; and how both intra-scientific and extra-scientific contradictions are dealt with.

The practical value of ethnomethodology has been recognized in software engineering, human-computer interaction, and other kinds of research on the relationship of work and computers, and there is quite much ethnomethodological research in those fields[105]. There are also reports on how epistemologically subjective proofs are created and transformed into epistemologically objective facts in computer science[106]. Such reports shed light on the very foundations of knowledge creation in the computing disciplines. (Note that a nominalist or an anti-realist position to scientific knowledge is not a sine qua non of social studies of science. Social studies of science may well build on, for instance, Searlean ontology[107].)

In addition to the aforementioned studies in which ethnomethodology has been used to study the users of computational systems, there is also research on the methods and practices used in computer science. There is research on, for instance, how rhetorics in discourse have influenced technological decisions[108]; how contingent social elements affect the closure of scientific debates[109]; how some mathematical parts of computer science are negotiated, rather than deduced[110]; what kinds of rhetorical strategies have been used in arguing for the universality of computing technology[111]; and how knowledge engineers' epistemological stances are reflected in artificial intelligence technology[112].

105 Viller & Sommerville, 1999; Crabtree et al., 2000; Clayman, 2001; Hartswood et al., 2002; Lucy A. Suchman expresses her ethnomethodological viewpoint in her book *Plans and Situated Actions* (Suchman, 1987:pp.49-50). The whole issue of *European Journal of Information Systems* 13(3) (Sept. 2004) was dedicated to interpretive approaches to information systems and computing, e.g., ethnography, ethnomethodology, phenomenology, "technomethodology", and hermeneutics.

106 See Richard De Millo et al.'s theoretical review *Social Processes and Proofs of Theorems and Programs* (De Millo et al., 1979).

107 See Bloor, 1996. For Searle's ontology see Searle, 1996:p.7.

108 Godin, 1997, examines the rhetorics surrounding the innofusion of a health technology.

109 Olazaran, 1996, examines how Minsky's and Papert's proofs and arguments were interpreted as showing that neural nets were not a fruitful approach to artificial intelligence.

110 MacKenzie, 1993, examines how the IEEE standard for floating-point arithmetic arose as a result of negotiation.

111 Bowker, 1993, explains the arguments that the practitioners of cybernetics used to varying degrees to argue that they were producing a new, universal science.

112 Forsythe, 1993, draws on ethnographic material about knowledge engineers' work, shows that building a knowledge-based system necessarily involves interpretation and selection, and suggests that knowledge engineers should be trained in qualitative social science.

The knowledge construction processes in computer science have been mostly examined using analytical methods[113]. Ethnomethodological research at large is often ethnographic and pays especially close attention to the interactional, discursive aspects of the study setting[114]. For instance, whereas in traditional ethnographic research on computer science one might assume that the language of computer science is a neutral conduit for description, in ethnomethodologically oriented research on computer science, descriptions, accounts, or reports should be treated not merely as being *about* some social world as much as being *constitutive* of that world[115].

Ethnomethodological investigations can be conducted with a variety of methods. If the organization of social interaction is the focus of an ethnomethodological investigation, ethnomethodology is often coupled with conversation analysis[116]. Ethnomethodological studies usually require extensive participant observation in specialized work settings[117]. Ethnomethodological studies include both *methodological investigations* (systematic reflections about the methods and efforts to design methods) as well as *methodic practices* (in the sense of practices performed in accordance with some methodological design)[118].

In summary: although it cannot be said that there is an ethnomethodological tradition in the field of computer science, ethnomethodology is not unknown to computer scientists, either. In the computer science literature there are studies in which ethnomethodology has been explicitly utilized as well as studies that can be characterized as ethnomethodology. The majority of the ethnomethodological studies in computer science literature report on the users of information technologies and are aimed at informing, for instance, system designers, interface experts, and software engineers. Also present are ethnomethodological investigations in which the practices and behaviors of computer scientists are studied, yet those studies are more commonly aimed at informing sociologists than they are aimed at informing computer scientists. As far as understanding how computer science *actually* works and how computer scientists actually work can benefit computer science, ethnomethodological research on computer scientists' work can inform computer science.

113 De Millo et al., 1979; Crabtree, 2004; Hartswood et al., 2002
114 Holstein & Gubrium, 1994
115 Holstein & Gubrium, 1994
116 Holstein & Gubrium, 1994; Lynch, 2004
117 Lynch, 2004
118 Lynch, 2004

Ethnographic Methods

Although ethnographic methods can be used in social studies of computer science to gather knowledge about the social world of computer science (in, e.g., ethnomethodological research), they can also be utilized to benefit computer science qua knowledge. In this subsection I discuss what ethnographic methods can bring to computer science and in what kinds of studies of computer science ethnographic methods are already being used.

I noted earlier in this thesis that systems engineering was developed as a response to the problems that arose when the complexity of systems became too great for one person to cope with. Even though systems engineering can be considered to be a response to a problem in management rather than a response to a problem in computer science, *software engineering* was developed specifically to sort out the software crisis. Software engineering focuses on problems unique to computer science.

The complexity of new projects and systems necessitates broad approaches to understanding system development. Suppose that one wants to explain the ontological, epistemological, methodological, or material assumptions, decisions, foci, or compromises that system design may incorporate. It is not enough to study individual actors and their surroundings because systems are no longer designed or managed by individuals; studying groups is necessary. That is, when explicating the design decisions behind a complex system, collective or multiple perspectives need to be accounted for. Ethnographic methods offer researchers of computer science a unique way of understanding the processes and dynamics behind, for instance, computer architecture design. Instead of historical studies, which are conducted in retrospect, ethnographic methods are studies of the present—studies of computer science in the making.

It must be noted that the term *ethnography* has been used in a large variety of meanings. One characterization of that ethnography is the "*art and science of describing a group or culture*"[119]. The data of ethnography are derived from the direct observation of behavior in particular groups[120]. As a verb, *doing ethnography* merely means

119 Fetterman, 2004. Note that the original meaning of *ethnography* is the book-length record of anthropologist's observations and analysis about his or her involvement in a community (Agar, 2001).

120 cf. Conklin,1968 in Sills, 1968.

the collection of data that describe (some parts) of a culture[121]. Michael H. Agar noted that roughly speaking, a researcher using the scientific method seeks universal laws, emphasizes control of the research process, preserves the initial assumptions throughout the study, relies on linear models, and represents data with numbers[122]. Agar continued that by contrast, again roughly speaking, a researcher using ethnography seeks local particulars, emphasizes adaptability in the course of study, develops new concepts over the course of the study, relies on systemic and processual models, and represents data more often with words than with numbers[123]. The promise of ethnography in social studies of computer science lies in the extent to which ethnography succeeds in eliciting the perspectives and realities of computer scientists[124]. That is, the promise lies in the extent to which ethnography can explain how the activities of computer scientists create the body of knowledge of computer science.

It is a common misunderstanding that all ethnography is qualitative research, and this misunderstanding probably arises from rough characterizations of ethnography, such as the ones above. There are a variety of ethnographic methods, but *usually* they share the same features: (1) *exploring phenomena* rather than testing hypotheses; (2) emphasizing *unstructured data* instead of analytic categories; (3) focusing on *cases* in detail instead of large populations, and (4) *explicitly interpreting* the meanings and functions of human actions[125]. However, even those ethnographic methods that rely on unstructured data instead of static categories or rely on a hermeneutic approach cannot avoid the inscription error that is involved in all human inquiry[126]. Because ethnographers cannot report all of their sense experiences, they have to make choices about aspects of the phenomena to report, and thus, in these reports an inscription error is inevitable.

Traditional ethnographic methods usually incorporate participant observation, in which the ethnographers are expected to live in some specific society or with some specific group for an extended period of time (it has been argued that the ideal is about 2 years), actively participate in the daily life of its members, and carefully ob-

121 See, e.g., Bernard, 1995:p.16; Agar, 2001 in Smelser & Baltes, 2001; Conklin,1968.
122 Agar, 2001
123 Agar, 2001
124 Fetterman, 2004 argued that this the aim of ethnographers is to elicit the insider's or *emic* perspective or reality.
125 Atkinson & Hammersley, 1994
126 See Smith, 1998:pp.50-52 for *inscription error*.

serve all aspects of their life as a way of obtaining material for their study[127]. Ethnographic methods in social studies of computer science may reveal some aspects in the practices of computer scientists that have direct consequences on computer science qua knowledge. Lately ethnographers have begun to engage critically with their own participation within the (auto)ethnographic frame[128], which marks a shift from participant observation to the observation of participation[129].

Tracy Kidder's *The Soul of a New Machine*[130] is an example of an ethnographic-type participant observation in the field of computing. Kidder observed a group of engineers at Data General from 1978 to 1980—the whole period of a design, implementation, testing, and release of a new 32-bit minicomputer (which became the Data General Eclipse MV/8000). In his book, Kidder described the company work environment and the machine, concentrating on not only technological decisions, but also on things such as the engineers' emotions, the birth of innovations, bottom-up management, the dedication and motivations of the engineers, the pressures caused by tight schedules, disappointments, and engineering artistry. Kidder discussed how architectural design is *actually* done, the challenge of designing a new 32-bit architecture while maintaining downward compatibility to legacy architecture, decisions concerning microcode, instruction set, registers, diagnostics, input/output, types of components used, and so forth. A competent computer scientist can get acquainted with the architecture of Data General Eclipse MV/8000 computer by studying its blueprints and specifications. Kidder's book offers some viewpoints on *why* the architecture is what it is.

Ethnographic methods aim at describing and interpreting social phenomena such as ways of working, group relationships, communication, metaphors, and tropes[131]. Ethnography (as well as participant observation) is a uniquely humanistic, interpretive approach[132]. Perhaps because of this uniqueness, ethnographic methods are sometimes contested, even in qualitative research[133]. Be that as it may, since ethnographic methods emphasize understanding phenomena in their rich sociohistorical

127 Tedlock, 2005; Atkinson & Hammersley, 1994; Bernard, 1995:p.78.
128 Tedlock, 2005
129 Denzin & Lincoln, 2005:p.380.
130 Kidder, 1981
131 Atkinson & Hammersley, 1994
132 Atkinson & Hammersley, 1994
133 Denzin and Lincoln, 1994:p.203.

contexts, ethnographic methods can be utilized in order to examine, for instance, patterns of production of scientific results, innovation, and standards in computer science; it can be utilized to study mechanisms of technological production, design, adoption, rejection, diffusion, non-diffusion; and so forth. In fact, scientists today have a unique opportunity to examine and document the early formation of the discipline —modern computer science is no older than 60 years, and some authors have argued that many parts of computer science, such as information systems and software engineering, are, in Kuhn's terms, still at the pre-paradigm stage of scientific development[134].

Focus on Cases

Many studies in computer science aim at generalizability; computer scientists often generalize to the populations from which their data are sampled. That is, computer scientists often argue that their results are applicable to all similar data. In research where generalizations are made, the significance of single cases is often downplayed. In those studies in social studies of computer science that aim at contributing to computer science qua knowledge by explaining how and why computer science has taken its current form, single cases are important. Single cases, such as Donald McKenzie's case of the negotiation of floating-point arithmetic[135], are important because they can offer information about the *how*s and *why*s of technoscience (yet single cases can also contribute to generic theories about technoscience).

The term *case study* can be understood *as a method*, but here it is understood *as the focus* of a specific study. When *case study* is understood as an indicator of the focus of the study, case studies can be quantitative or qualitative, although many studies that are labeled as case studies are qualitative[136]. The driving question behind case studies is, "What can be learned from the single case?"[137]. However, it is typical of case studies that the researcher is ultimately interested in a process, or in a population of cases[138].

Of the different varieties of the case study, instrumental and collective case studies are beneficial for social studies of computer science. *Instrumental* case studies are

134 Wernick & Hall, 2004
135 MacKenzie, 1993
136 Stake, 1994
137 Stake, 1994
138 Denzin and Lincoln, 1994:p.203; Denzin & Lincoln, 2005:p.380.

conducted because the researcher believes that a particular case may provide insight into an issue, theory, concept, technology, or such[139]. *Collective* case studies are instrumental case studies extended to a several cases[140].

Many philosophers of science use instrumental, historical case studies in their argumentation. For instance, Kuhn's *The Structure of Scientific Revolutions* goes into the particular details of the histories of oxygen, X-rays, and the Copernican revolution; Lakatos' book *Proofs and Refutations* focuses on the development of a single formula (Euler's formula), and Feyerabend's *Against Method* details Galileo Galilei's work[141]. These philosophers of science engaged in analytical discourse about what their cases reveal about science. In the Section 3.3 of this thesis I discuss a number of cases that I take to be particularly revealing about the development of the discipline of computing.

The book *Funding a Revolution: Government Support for Computing Research*[142] includes five case studies in computer science: *relational databases*, *the Internet and the World Wide Web*, *theoretical computer science*, *artificial intelligence*, and *virtual reality*. These five case studies take the form of a historical narrative. Although the book is a report of a study that aimed at understanding some economic questions about computing and communications, the five case studies also reveal much about the development of technologies. The authors rationalized their choice of a historical narrative by arguing that a historical narrative allows analogies to be drawn between events that occurred decades apart and that a historical narrative can accommodate complexity more easily than can a tightly-structured analytical essay. The authors also wrote that the case studies in the report "present finely nuanced accounts that convey the ambiguities and contradictions common to real-life experiences"[143].

Evaluation of Studies of Social Reality

The crux of qualitative research is not numbers and proofs. Unlike the subjects of mathematics and logic, the subjects of disciplines such as sociology, history, anthropology, and philosophy are often not well-structured, logical, coherent, or well-

139 cf. Stake, 1994
140 Stake, 1994
141 Kuhn, 1996 (orig. 1962); Lakatos, 1976; Feyerabend, 1993 (orig. 1975).
142 NSR Computer Science and Telecommunications Board, 1999; chaired by Thomas Hughes.
143 NSR Computer Science and Telecommunications Board, 1999:p.3.

defined. The concept of culture, for instance, is troublesome—some anthropologists have even argued that because of the vagueness of that concept, it should be discarded[144]. Hypotheses about "the laws of social reality" are rare, although computer scientists have presented discrete models to describe, for instance, culture and society[145]. As long as there is no axiomatic system of social reality, it is not possible to prove hypotheses about social reality correct. Of course, researchers can (and do) make models of aspects of social reality, and these models may be quite accurate in predicting social events, yet nothing in the success of a model proves the model correct (although its success may prove it very useful).

For an example of the difficulties of evaluating social sciences—difficulties that mathematical disciplines do not have—take Manuel Castells' research on the information society[146]. Although Castells' trilogy has been widely lauded, it has also been criticized. It has been argued that there is a certain blind spot, typical of realism, in Castells' work[147]. Namely, Paula Saukko noted that many realists believe that an analysis of, for example, statistics, can reveal how the world "really" is to the researcher[148]. She wrote that realists do not generally acknowledge that the categories they use for excavating the "truth" out of the data can be biased by politics, culture, organizations, or such. Quite the contrary, even statistical analysis has been argued to be anything but an objective tool; for instance, national traditions and preferences in statistical research affect statistical surveys[149].

If one were to agree that changes in technology follow from choices that mirror the social relations of innofusion, it would be an error to assume that having exposed the choices and their motifs, one could simply deduce the rest of reality from them[150]. Researchers who assume that by unearthing the social, cultural, economic, institutional, personal, and other human variables they can converge on the "true" state of affairs commit the inscription error[151].

144 For discussion see Abu-Lughod, 1991 in Fox, 1991; Brumann, 1999.
145 Gabora, 1995; Brent et al., 2000. Also Harold Kincaid argued that there are indeed laws in social sciences (Kincaid, 2004).
146 Castells, 1996; Castells, 1997; Castells, 1998
147 Saukko, 2005
148 Saukko, 2005
149 Desrosières, 1996 in Hantrais & Mangen, 1996.
150 Noble, 1999
151 Smith, 1998:pp.50-53.

Inscription error, in this context, refers to the phenomenon where a researcher first creates a model or a tool, such as a classification system or conceptual framework, and then reads results gathered with this model or tool as if the results were independent empirical discoveries. Models and tools such as classification systems, conceptual frameworks, data structures, or computational models are influenced by the researcher's existing knowledge about the domain, as well as by the epistemological and methodological commitments of the researcher. In the field of statistics, differences in classification systems and their changes over time are actually seen today as phenomena that deserve to be examined in their own right[152].

Measures of Research

There are a variety of measures of research quality in the humanities and social sciences, and it has been argued that there is a particularly marked lack of agreement in academia about the appropriate basis for "confirmation procedures" in research[153]. I present here a number of measures of research quality that might be less known than validity and reliability, yet which should be considered in social studies of computer science. The concepts presented here are controversial, and the aim of this discussion is not to participate in the debate about qualitative research criteria but to introduce alternative criteria of research to the readers whose background is in quantitative research.

Ian Hodder argued that the "twin struts of confirmation" are *coherence* and *correspondence*[154]. (Confirmation here does not refer to a proof, but to increasing the credibility of an argument.) *Internal coherence* is the degree to which the parts of an argument do not contradict each other and to which the conclusions follow from the premises[155]. Internal coherence is to some extent, a subjective measure, because what is considered to be a sound argument may differ between disciplines and between individuals. That is, what is considered to be a sound argument in the field of logic may differ from what is considered to be a sound argument in anthropology. Ultimately, the audience interprets and judges every argument in interpretive research.

152 Desrosières, 1996
153 See Hodder, 1994.
154 Hodder, 1994
155 Hodder, 1994

External coherence refers to the degree to which the interpretation of material fits theories, models, or interpretations accepted within and outside the discipline[156]. As there is no grand theory of everything (yet)[157] (nor a grand theory of social reality), only very trivial arguments can be externally fully coherent. Note the difference to mathematics: It is common that in mathematics either the conclusions follow from the premises or they do not—there is no "degree of coherence". (However, it has been argued that the quality of a mathematical proof is also a subjective measure rather than an objective, external, or universal measure[158].)

Correspondence between data and theory is an essential part of a coherent argument. The notion of correspondence between theory and data does not imply the absolute objectivity and independence of theory and data, but it rather embeds the fit of data and theory within coherence[159]. The data are made to cohere by being linked within theoretical arguments[160]. Similarly, the coherence of the arguments is supported by the fit to data. The more robust fit there is between data and theory, the better correspondence they can be said to have.

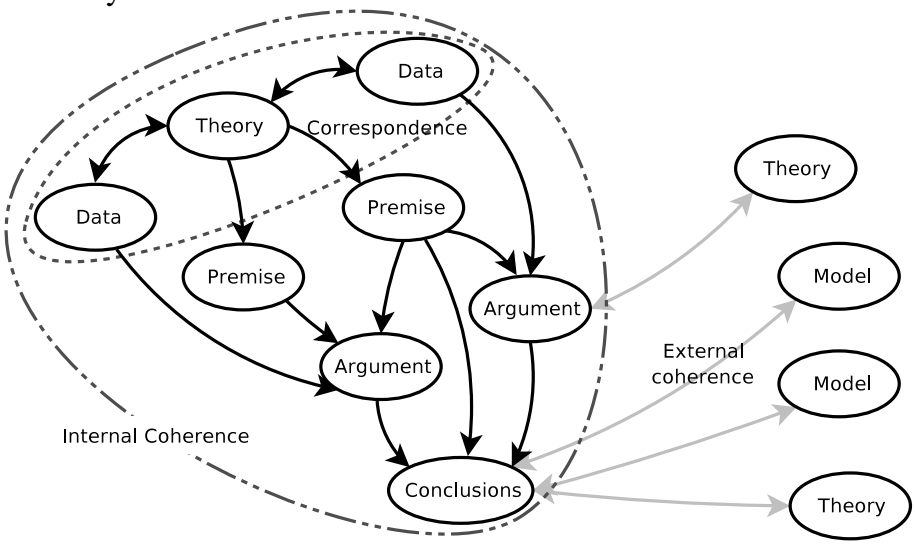


Figure 30: Correspondence and Coherence in Research

The concepts of correspondence and coherence are portrayed in Figure 30. Correspondence expresses how well sets of data cohere within the selected theoretical

156 Hodder, 1994

157 It has been argued that authors such as Stephen Wolfram (Wolfram, 2002) and Andrew Pickering (Pickering, 1995) have both, in their own ways, aimed at a "Theory of Everything" (Gingras, 1997; see also the popular article in *The Economist*, May 30th 2002, US edition, pp.79-80: "*The Emperor's New Theory*".)

158 Lakatos, 1976; De Millo et al., 1979

159 Hodder, 1994

160 Hodder, 1994

framework in the study, internal coherence expresses how well arguments and conclusions follow from the data and theory, and external coherence expresses how well the conclusions and arguments resonate with other pieces of research and theory.

In addition to the traditional measures of validity; such as instrument validity, face validity, data validity, and criterion validity[161]; there are a variety of less frequent measures of validity that should be considered in social studies of computer science, such as those mentioned by Paula Saukko: *contextual validity*, *dialogic validity*, and *self-reflexive validity*[162]. According to Saukko, contextual validity refers to the thoroughness and defensibility of the analyses of social, historical, political, or economic processes and structures. She continued that dialogic validity refers to the extent to which research is able to expose tacit, experienced, emotional, and embodied knowledge and understanding[163]. Saukko wrote that self-reflexive validity is based on the critical reflection of how social discourses shape or mediate people's self-experiences and the experiences of their environment. She noted that self-reflexivity refers to the extent to which the researcher is aware of the discourses that guide the research analysis itself.

Although contextual, dialogic, and self-reflexive validity are subjective measures, it must not be forgotten that validity is *never* an objective measure[164]. H. Russell Bernard wrote that ultimately, the validity of a concept depends on two things: the *utility* of the device that measures it, and the *collective judgment* of the scientific community[165]. Even the validity of physical concepts, such as *force*, is ultimately subjective. One can hit golf balls with different velocities to demonstrate the concept of force. It appears that the concept of force is quite valid—the concept can be utilized to predict the acceleration and the trajectory of the golf balls quite accurately. It appears that the concept is reliable, too—the concept of force can be utilized in a large number of different tests, and few or no anomalies are encountered in those tests. The scientific community has a high degree of agreement that the concept of force portrays and predicts changes of momentum well. However, much in a Popperian manner, it cannot be said that the validity of the concept of force as a cause of accel-

161 Bernard, 1995:pp.38-42.
162 Saukko, 2005
163 Saukko, 2005
164 Bernard, 1995:p.43.
165 Bernard, 1995:p.43.

eration has been proven à la mathematics—beyond any doubt—but only that it has withstood a rich variety of tests without contradicting results, and that therefore, the concept of force as a cause of acceleration is valid to very high degree.

Ian Hodder[166] attributed three other criteria to the success of research: *fruitfulness* (how many new directions, new lines of inquiry, new perspectives are opened up), *reproducibility* (the extent to which other people, perhaps with different perspectives, come to the same results), and *intersubjective agreement* (on a science that balances between a number of disciplines, the adequacy of the results to these disciplines).

Research that attacks an obstacle that hinders progress in a number of topics[167] often turns out to be especially fruitful. That is, there are certain obstacles that, after they are overcome, allow a number of topics to be pursued. There is a large number of examples of especially fruitful research projects in computer science that have solved some sorts of reverse salients and that have, consequentially, opened new topics. Take, for instance, the stored-program concept in the 1940s, compilers in the 1950s, packet-switching networks in the 1960s, public-key cryptography in the 1970s, the graphical interface in the 1980s, the world wide web at the turn of the 1990s, and decentralized file-sharing in the 2000s. There are proofs of concept in, for instance, DNA computing and quantum computing, and a practicable implementation of either one would yield new opportunities. Research on DNA computing and quantum computing may or may not turn out to be very fruitful—but taking into account the history of computing it is also likely that new research openings come from surprising directions.

I have now discussed a number of research aspects that might turn out to be useful in social studies of computer science, yet that do not necessarily belong to the traditional computer scientists' toolbox. These aspects of research are an example of the new viewpoints that a humanities and social sciences-based approaches can bring (and have brought) into understanding computer science and computing. Of course no two studies are the same, and the utilized research aspects follow from the research questions in each study, the pragmatic and theoretical interests in each study, and the

166 Hodder, 1994

167 Obstacles that hinder progress in a number of topics have been called “reverse salients” (MacKenzie and Wajcman, 1999 :pp.11-12).

theoretical and conceptual frameworks in each study. The modern computer scientist-as-a-bricoleur ought to be cognizant of different research approaches; the computer scientist working with social studies of computer science needs a full toolbox and the knowledge of how to use those tools appropriately.

4.3.What Makes a Study Computer Science?

In spite of the numerical analysts' claims for the fundamental importance of [mathematics]*, a surprising amount of computer science activity requires comparatively little of it.*[168]

In This Section:

- ✔ What is the relationship between science and auxiliary science?
- ✔ What kinds of research can be considered to be *computer science*?
- ✔ What kinds of methodological and epistemological commitments must computer science make?

Even if one acknowledged that understanding computer science qua knowledge requires understanding its history and its sociocultural surroundings, and even if one agreed that social studies of computer science is a valid topic, one could argue that research in social studies of computer science belongs to the domains of, say, sociology, history, anthropology, and philosophy. In this section I consider different ways to interpret the home field of interdisciplinary studies. I analyze four ways to formulate a statement about the relationship between a main discipline and an auxiliary discipline: (1) the auxiliary discipline remains independent of the main discipline; (2) the auxiliary discipline is subsumed by the main discipline; (3) the auxiliary discipline subsumes the main discipline; and (4) the auxiliary discipline and the main discipline create a new, independent discipline. I refer to these as *statements 1-4*. I also expound on my argument why social studies of computer science belongs to the field of computer science.

What Is the Relationship Between a Science And Its Tool?

In this section, by *discipline* I refer to an established branch of knowledge that is significantly different from other branches of knowledge and that has a clearly defined subject area, such as sociology, philosophy, physics, mathematics, or anthropology. In this section, by *field of study* or *field* I refer to a branch of knowledge that is more specific than a discipline, but which also has its own defining characteristics. Disciplines usually incorporate a number of fields of study; for instance, endocrinology, microbiology, and physiology are all fields of the discipline of biology. In this section, by *research topic* I refer to a specific topic within a field of study; for instance, logic programming is a topic of the field of programming, which belongs to the dis-

168 Hamming, 1969

cipline of computer science. It has to be noted that many boundaries between disciplines, fields, and topics are somewhat arbitrary and blurry, and therefore the terms rely, to some degree, on intuition.

By *main discipline* I refer to the discipline that traditionally has been connected with a specific subject. For instance, the main discipline that studies quarks is physics, the main discipline that studies human interactions is sociology, and the main discipline that studies automatic computation is computer science. By *auxiliary discipline* I refer to the discipline that provides additional tools for research in a main discipline. For instance, physics and social sciences can use tools provided by computer science, and computer science can use tools provided by mathematics or social sciences. Note that *auxiliary discipline* does not mean that the discipline would not be worthy of academic studies per se, but that its tools and theories are used in other disciplines too. Mathematics, for instance, is an auxiliary discipline for many other fields.

I use *computational sciences* as an example for my argument. There is a field called *computational physics*. In computational physics, computers are used to numerically model and simulate physical processes[169]. Without computers, computational physics would not exist, but computational physics is still a field of *physics*, not a field of computer science. Surely, some computer scientists also refine the tools of computer science so that they are better suited for computational physics, but computational physics is still a field of physics.

Of the four statements presented above (see p.411 of this thesis), neither Statement 2 (i.e., the auxiliary discipline is subsumed by the main discipline) nor Statement 3 (i.e., the auxiliary discipline subsumes the main discipline) are all-extensive descriptions of the current practice, or desirable normative accounts of practice. Statement 2 is not a correct descriptive account because although computer science uses mathematics as a tool, mathematics is not considered to be a part of computer science. Statement 3 is not a correct descriptive account because although physics uses computer science as a tool, physics is not considered to be a part of computer science. If statements 2 and 3 are taken as normative accounts of science, then borrowing tools between disciplines would result in merging disciplines. Statements 2 and 3 are de-

169 Denning, 1991

sirable normative accounts only if one wishes there to be a small number of extensive *ur*-disciplines that do not share anything with other *ur*-disciplines.

Statement 4 (i.e., the auxiliary discipline and the main discipline create a new, independent discipline) is a valid descriptive account to some extent. The contact points of two disciplines often bring about new research topics or fields, such as computational geography, mathematical sociology, and physical chemistry. However, it seems that usually those contact points indeed bring about research *topics* or *fields*, but not *disciplines*. (Note, however, that also computer science was born from a number of fields, and it is now argued to have a disciplinary identity.) Statement 4 is controversial as a normative account because if borrowing tools from other disciplines would breed new disciplines, the number of disciplines would grow to be enormous.

Statement 1 (i.e., the auxiliary discipline remains independent of the main discipline) is not an all-extensive descriptive account or a flawless normative account either, but Statement 1 seems to describe the current interplay of disciplines well. Following Statement 1 would keep the boundaries and relationships of different kinds of scientific inquiry relatively stable. Given that one does not want scientific inquiry to collapse into a small number of *ur*-disciplines or want scientific inquiry to disperse into a very large number of microdisciplines, Statement 1 seems to also be a good normative account.

Table 3 presents my interpretation of the relationship between auxiliary and main disciplines when an auxiliary discipline is considered to be independent of a main discipline (statement 1). The rows in Table 3 represent fields or research topics. The first column specifies the research subject in each field, the second column specifies who defines the foci of the field, the third column specifies who provides tools for research in the field, the fourth column specifies who explains the results in the field, and the fifth column specifies which discipline benefits from the research.

The field of computational physics (quantum mechanics) is presented on the first data row. In this branch of computational physics, the foci of the research on *quarks* is defined by *physicists*, not computer scientists. *Computer science* provides the tools for studying quarks, but the results are explained by *physicists*, not computer

scientists. Physics benefits the most from computational physics, but the collaboration may also benefit computer science.

Table 3: Examples of Auxiliary and Main Disciplines

Subject	***Who defines foci***	***Who provides tools***	***Who explains results***	***Beneficiary***
Quarks	Physicists	Computer scientists	Physicists	Physics
Culture	Anthropologists	Computer scientists	Anthropologists	Anthropology
Complexity	Computer scientists	Mathematicians	Computer scientists	Computer science
Programs	Computer scientists	Anthropologists	Computer scientists	Computer science

Similarly, on the second data row, when anthropologists conduct a cultural domain analysis with a computer, the subject is culture, and the foci are defined by anthropologists. Although computer science provides the tools for anthropologists, such as the ANTHROPAC program, the results are explained by anthropologists, and anthropology qua knowledge benefits the most.

On the third data row of Table 3, when computer scientists study, for instance, computational complexity, the foci are defined by computer scientists. Although computer scientists use the tools borrowed from mathematics, computer scientists explain the results and computer science qua knowledge is the main beneficiary of the collaboration.

In the same manner, on the fourth data row, when computer scientists study programs, the foci are defined by computer scientists. Although computer scientists can use tools borrowed from anthropology, computer scientists explain the results and are the main beneficiaries of the collaboration[170].

Note that there is research which contributes to theories *of* computer science and research which contributes to theories *about* computer science (theories about theories can be called meta-theories, yet meta-theories are not the kind of research I mean here). The research which contributes to theories *of* computer science consists of, for example, studies which describe, analyze, revise, or refute the subjects of computer science (e.g., algorithms, interfaces, or architectures). The research which contributes to theories *about* computer science consists of, for example, studies which

170 For instance, the studies on programming cultures by Rajlich et al. might have benefited from anthropological methods (see Rajlich et al., 2001; Wilde et al., 2001). Studies in computer science that have benefited from anthropological methods include, for instance, Lucy A. Suchman's research (Suchman, 1987).

describe how computer scientists work and studies which describe how computer scientists should work if they want their science to flourish.

To Which Field Does Social Studies of Computer Science Belong?

It would not be especially unorthodox to argue that *If research aims at honing theories, practices, models, philosophy, or other aspects of computing, then the research is computer science.* The crux of this argument is that it does not commit research to any methodology, conceptual framework, or theoretical framework. It represents an epistemological anarchist view, which is arguably a characteristic of computer science already: It does not matter what methods and standpoints a computer scientist uses for studying computer science as long as the research is aimed at the benefit of computer science. If the aim of the research is to benefit mainly some other discipline, then the research may be classified as belonging to some other discipline, not computer science. Granted, there is a number of difficulties with this argument. For instance, it is often difficult to say what kinds of research benefit a discipline; often the short-term beneficiary of a study is different from the long-term beneficiary of that study and often fruitful research has multiple direct or indirect beneficiaries. This debate drifts back to the philosophy of science, and I must concede that there is no criteria according to which one can tell *with certainty* if a research benefits mainly computer science or some other discipline. Ultimately, the scientific community always decides what is computer science and what is not.

Some might consider it unorthodox to argue that it would be irresponsible to deny *any* (ethical) methodology at *any* point of time because people do not know which methodologies may reveal unexpected features of computing. Certainly, sometimes, in some settings, some methodologies are more productive than others, but I argue that there is no single methodology that is more productive than all the others, at all settings, at all times[171]. (And I would not dare to say that a certain methodology would not be useful in any setting ever.) I do not go to the extent of saying that society should *support* all kinds of methodologies in all settings—I am sure that some methodologies are ineffective in some settings, and I am sure that the cost-benefit ra-

171 It can be debated if the systematic refusal of methodology can be considered to be methodology itself. Regardless, it seems implausible that the systematic refusal of methodology would be beneficial at all settings, at all times.

tios of some methodologies in some settings can be poor. But the cost-effectiveness of methodologies in general is not within the scope of this thesis.

The aim of social studies of computer science is to enhance or enrich the understanding of *computer science qua knowledge* by using the methodological toolboxes of, for instance, sociology, history, anthropology, and philosophy. Social studies of computer science can also aim at refining the tools of computer science, *computer science qua activity.* Social studies of computer science aims to benefit the discipline of *computing*, offers new viewpoints to *computing*, and enlarges knowledge on *computing*. Social studies of computer science does not aim at refining the understanding or tools of, say, sociology, history, anthropology, or philosophy, but utilize their tools of investigation. The methodologies of sociology, history, anthropology, and philosophy are added to the toolbox of computer science in order to have a better repertoire of conceptual, theoretical, and methodological frameworks for understanding computer science.

Note that if researchers study, for instance, the subculture of software engineers, the mental models of programmers, the communication patterns of HCI specialists, or the metaphysical assumptions of theoretical computer scientists, and if their research is intended to add to the theories, models, concepts, or such of *social sciences*, then the research belongs to the field of social sciences and not to computer science. However, if researchers study programmers' cultures, and if their research is intended to benefit computer science, then the research belongs to computer science. An example of the latter is Václav Rajlich et al.'s study, which shows that effective comprehension of software requires viewing legacy programs not simply as products of inefficiency and stupidity, but as artifacts of the circumstances in which they were developed[172].

The theoretical soundness of arguments about which kinds of studies belong to computer science is one bone of contention—it is another controversy whether social studies of computer science can ever *actually* offer any new viewpoints on computer science. Based on Section 4.2, I take it that there is already a body of research that can be taken as social studies of computer science and that has contributed to com-

172 Rajlich et al., 2001

puter science qua knowledge. The status of such studies as a legitimate branch of computer science has not been established, though.

I argued in Section 3.4 that in the debates about the essence of computer science there have been *internally expansive* motifs—that is, arguments for an extension of computer science to include topics that have not been previously considered to be a part of the discipline (see p.366 in this thesis). I also argued that those motifs come in two forms: descriptive and normative.

The descriptive, expansive arguments are similar to my argument that (*1*) computer science is already being studied successfully from the perspectives of sociology, history, anthropology, and philosophy, and (*2*) that social studies of computer science should be considered to be a part of the discipline because it is already an implicit part of the discipline. The normative, expansive arguments are similar to my argument that (*1*) restricting scientific inquiry in any way is detrimental to science (although scientists still must operate ethically) and, (*2*) therefore, researchers cannot exclude any tools of inquiry. That is, sociological, historical, anthropological, philosophical, and all other kinds of research that are aimed at benefiting computer science should be a part of computer science (however computer science is defined).

4.4.Discussion

We need a hermeneutic computer science.[173]

In This Section:

- ✔ Is it possible to have one account of computer science?
- ✔ Is it desirable to have one account of computer science?
- ✔ How is computer science in practice done?
- ✔ What is the contribution of social studies of computer science to computer science qua knowledge?
- ✔ What disciplinary implications does social studies of computer science have?

I have now presented an argument about the opportunistic or anarchistic nature of computer science, suggested some complementary approaches to scientific inquiry in computer science, and rationalized the interdisciplinary and disciplinary connections of social studies of computer science. In this section I analyze the making of normative and descriptive statements about computer science, present a characterization of computer science in the mangle[174], argue that many aspects of computer science that arise from the mangling character of computer science can be captured with social studies of computer science, and propose an addition to the ACM Computing Classification System: K.9 (*Social Studies of Computer Science*).

In terms of normative and descriptive accounts of computer science, there are two basic questions that need to be asked and there are two basic subjects of those questions. The questions are the normative and descriptive questions: "Is it *desirable* to have *x*?" and "Is it *possible* to have *x*?"[175]. In the same context, the subjects *x* of those two questions can be "*normative accounts of computer science*" or "*descriptive accounts of computer science*". From these premises one can derive four questions. In the following two subsections I ask two of those four questions—those that are relevant to this thesis. I ask if it is desirable to have a single normative account of computer science, and I ask if it is possible to have a single descriptive account of computer science.

173 West, 1997

174 Andrew Pickering's term (Pickering, 1995).

175 In *Consolations for the Specialist* (Feyerabend, 1970) Feyerabend raised two questions about the Popperian doctrine; he asked if it is *possible* to have science as Popper portrayed it, and if it is *desirable* to have science as Popper portrayed it.

On Normative Accounts of Computer Science

My normative question is, "Is it *desirable* to have one normative account of computer science?". The debate between "Popperians" and "Kuhnians" has been discussed in this thesis already, so I will get straight to the point. The "Feyerabendian" argument is that forcing any single ontological, epistemological, or methodological account of scientific inquiry is forcing a dogma onto thinking. It is a forceful occupation of intellectual territory; it is nothing short of shackling innovation, inference, argumentation, and perhaps all thinking, within the limits of one ontology, epistemology, or methodology.

It might be a matter of some concern that a dogmatic view of computer science contradicts what some people may hold as the ideals of science, such as scientists' freedom to set their own goals, to pursue their inquiries in a field of their choice, or to choose whatever approaches they feel are necessary for their work[176]. It is definitely an important matter that a dogmatic view of science requires instant clarity of research—that is, research results must be explainable in terms of preordained, select theoretical and conceptual frameworks.

There cannot be radically new innovations (*radical* in the sense that those innovations would shake the foundations of computer science) if new ideas are bound to the explanatory power of computer science as it exists at a given time. If one celebrates the ultimate supremacy of the current theoretical framework of computer science, then the limits of computer science cannot be probed from incommensurable points of view, and a scientific revolution in computer science is not possible (because revolutions, at least Kuhnian ones, are transitions between incommensurable frameworks).

From an idealistic point of view, the only normative account of computer science that does not restrict progress in any way is the anarchistic philosophy of science. But at the same time it should also be conceded that the phrase "does not restrict progress" is not equal to the phrase "promotes progress". Although the anarchistic account of science allows for novel initiatives to be created, some other accounts of science, although more restrictive, may better further progress towards some goals. Note, however, that following the anarchistic philosophy of science does not forbid

176 See Subramanyam, 1981:p.50 for a discussion on *scientific freedoms.*

scientists from even the strictest methodological regimentation if the scientists feel that regimentation is necessary. If scientists do not wish to close any doors for progress, scientists can choose to be opportunists and use whatever ethical approaches they find necessary to achieve their goals.

Despite the difficulty of making grand normative statements about science, there is one normative element that I believe the majority of computer scientists, regardless of their field of research, would say *should* belong to the normative account of computer science: *the ethics of research*. Nihilists debarred, computer scientists, as active members of society, must adhere to the norms and ethics of their society. Regardless of their ontological, epistemological, or methodological standpoints, all researchers must work ethically. The problems with the justification of certain ethical prescriptions in research are the same as the problems that are studied in the field of ethics.

Although all normative claims (with the exception of renouncing all normative claims) restrict intellectual inquiry, in practice computer scientists frequently give various descriptions of the world as well as "practical" prescriptions of how computer science should be done. Although normative questions are cleanly severed from descriptive questions by Hume's Guillotine (see page 35 of this thesis), in computer science normative and descriptive statements intertwine. In the following subsections I discuss the anatomy of descriptive statements in computer science and the ways in which descriptions gain "practical" normative value.

On Descriptive Accounts in Computer Science

My descriptive question—the one I mentioned in the beginning of this section—is, "Is it *possible* to have a single descriptive account of computer science?". The difficulty of answering my descriptive question comes from the vagueness of the object of the question—computer science. As I have shown earlier, especially in sections 3.3 and 3.4, it is hard to conceive of an account of computer science which would not downplay research in any subfield of computing, but which would still be an informative account of computer science.

Depending on the reading of the term *computer science*, one can arrive at different ends regarding my question above. If computer science is defined as a mathematical science, a variety of descriptive accounts of computer science can be argued for—

that is, there can be meaningful discussion about the ontology, epistemology, and methodology of computer science. But if the subjects of computer science are multiform (formulæ, machines, programs, people, usability, and so forth) it is difficult to come up with an overarching ontological, epistemological, and methodological outlook that would describe computer science correctly[177]. The number of disparate academic disciplines today is a living example that the world (physical and social) does not seem to surrender to narrowly focused inquiries.

Mathematical and computational models are precise and unambiguous, yet they are confined to the abstract world of mathematics and they fail to capture the richness of reality. Narratives and ethnographies are rich in dimensions and sensitive to detail, yet equivocal and context-dependent. Narratives have little use in deriving formulæ, and formal proofs have little explanatory power regarding usability. It is difficult to see how a single descriptive account of computer science could accommodate software engineering, complexity theory, usability, the psychology of programming, management information systems, virtual reality, and architectural design.

To the question, "Is it possible to have one descriptive account of computer science?", I am compelled to answer that given the diversity of research that goes under the name *computer science*, faithful descriptive accounts of current computer science are either narrow and applicable to only some subfields of computer science[178], or—when the descriptive account concerns computer science at large—so broad that they do not exclude much[179]. It is very difficult to imagine an informative and extensive ontological, epistemological, or methodological account of computer science.

When attempting to describe computer science it will always be easy to make counterarguments and numerous counterexamples; nonetheless, I describe my view of how the growth of knowledge[180] in computer science works in general. In the next

177 I have shown earlier in this thesis that even in technically-oriented branches, computer scientists do not work as, for instance, falsificationists ought to work. That is, computer scientists *do not* formulate hypotheses and then try to falsify them. Kuhn's philosophy of science is not much help in describing computer science either, because Kuhn's philosophy is a generic account of science, and it does not offer much help in making a distinction between what could be considered "paradigmatic computer science" and "non-paradigmatic computer science".

178 e.g., Dijkstra, 1974

179 e.g., Newell et al., 1967

180 I do not take positions on the determinism or progress implied in the term *growth of knowledge*, but I use the term because it is an established term. Alternative terms that have different emphases are, for instance, *(re)construction of knowledge*, *the mangle*, and *(re)shaping of science.*

subsections, based on sections 3.3 and 3.4, I discuss the formation of "practical" normative statements in computer science.

"Practical" Normative Statements in Computer Science

As I have already noted, the broadness of computer science makes it especially difficult to describe how computer science is actually done, or to describe how computer science should be done—the same accounts of computer science should cover inquiries about nature as well as inquiries about people and groups. However, because computer scientists frequently do make "practical" normative statements (or heuristics, or rules-of-thumb, or general rules), a practical computer scientist might ask, "on what grounds, and by which processes, are normative statements about computer science made and followed?".

I argued in Section 3.4 (page 363ff.) that normative arguments about computer science cannot be based on empirical evidence only. Because normative arguments inevitably include a hidden or explicit value premise, they must ultimately be assessed according to their desirability, plausibility, and credibility. I noted that the credibility of normative arguments can be increased with experimentation, forceful argumentation, and cogent analysis, but that ultimately the acceptance of normative arguments is based on human judgment.

The line between scientific, descriptive statements and normative statements in computer science is often vague. Scientific statements and findings about how "things are" are often also meant as normative statements about how computer scientists should do their work if they want their work to flourish. Take, for instance, a hypothetical (prototypical) study where researchers reported that a certain disposition of menu items in a piece of software led to an average 15% increase in the productivity of people using that particular software[181]. If those researchers were to argue that because of this increase in productivity a certain disposition of menu items should be favored, these researchers would be making a normative statement from a descriptive statement. "Casual" or "implied" normative statements based on scientific findings, such as the previous example, are commonplace in computer science. One might interpret this characteristic of computer science as giving computer science an engineering flavor; heuristics are, according to Billy Vaughn Koen, the cornerstone

181 Studies similar to this are numerous. Take, for instance, those by Arnold, 1989 and Somberg, 1987.

of the engineering method[182]. Follow-up questions immediately arise: "How do heuristics in computer science form?", "How do individual heuristics in computer science become a part of the body of knowledge of computer science?", and "By which rules does one compare and choose between competing heuristics?".

The Mangle in Computer Science

My interpretation of the mechanisms of the growth of knowledge in computer science, based on the sources in sections 3.3 and 3.4, is that scientific statements about the subjects of computer science, computer science qua knowledge, gain their credibility through demonstrations, publicity, and debate. Those statements gain their credibility via conjectures, modifications, proofs, refutations, critique, and adjustments. Instead of following any given set of rules of computer science *proper*, computer scientists present theories, demonstrate techniques, or implement mechanisms, and then submit those theories, techniques, or mechanisms for public criticism.

Through a continuous mangle of practice[183]—corrections to theories, honing of techniques, improvements of mechanisms, negotiations between stakeholders, debates, power struggles, and constant criticism—some theories and techniques gradually become a part of the relatively stable core[184] of computer science. That is, theories and techniques gradually increase their epistemological objectivity[185]. Similarly, the knowledge that researchers gain when they construct and hone instruments is gradually crystallized into heuristics and theories about instruments, which, in part, gradually become a part of the relatively stable core of computer science. This relatively stable core can be considered to be a sort of Kuhnian set of exemplars that guide the work of computer scientists. In this sense it could be said that over the course of time scientific statements gain normative value.

It should be noted that many normative statements about computer science are not falsifiable. For instance, the statement that GOTOs increase code entropy was never formulated in a falsifiable form—that is, in a form that would have created a non-empty class of potential falsifiers of that statement[186]. The stored-program concept was never formulated as a falsifiable hypothesis. An argument that "computer sci-

182 Koen, 1987; Koen, 2003
183 Pickering's term (Pickering, 1995).
184 Lakatos' term (Lakatos, 1970).
185 In Searle's terms (Searle, 1996).
186 In Popper's terms (Popper, 1959:pp.65-66).

ence (as a body of knowledge or as a class of practices) can benefit from sociological, historical, anthropological, and philosophical viewpoints" is not falsifiable either. That is, there are no potential falsifiers of the argument. Nevertheless, the lack of falsifiability has not stopped computer scientists before, and I see no reason that it should stop computer scientists in the future. In this sense, however, it might be more appropriate to talk about *heuristics* instead of *scientific statements* in computer science. (However, the term *heuristic* has not been firmly established in computer science language, perhaps because heuristics are more of a feature of engineering than a feature of science.)

Three Positions on Public Debate

If public debate is considered to be a part of the formation of computer science, it should be asked, "Against which conceptual and theoretical framework are the theories, techniques, or theories about instruments in computer science criticized?"[187]. In the absence of a generic conceptual and theoretical framework for intellectual achievement (i.e., a Grand United Theory or Theory of Everything), theories, techniques, heuristics, and theories about instruments always rely more on one conceptual/theoretical framework than others. For instance, it could be argued that theories about public-key cryptography rely largely on a logico-mathematical framework and computational complexity; interface design techniques rely largely on frameworks of design, psychology, ergonomics, and cognitive science; and architectural design relies largely on frameworks of physics, electronics, and logic design. Regarding the framework to be used for evaluating theories, techniques, or theories about instruments, one could argue for at least three different positions: I call these positions the *absolutist position*, the *monodisciplinary position*, and the *intersubjectivist position*.

In the *absolutist position* the authors of the theory, technique, or theory of an instrument define the framework in which their theory, technique, or instrument should be criticized. However, adopting the absolutist position risks leading to a science in which every research result could be insulated from even the most trivial of criticism by purposefully choosing the frameworks within which criticism is applicable.

187 In *the mangle* theory, Pickering identifies the parts of the mangle to be theories, instruments, and the theories about how the instruments work (Pickering, 1995).

In the *monodisciplinary position*, the frameworks on which a theory, a technique, or a theory of an instrument relies are also the frameworks on which any critique should rely. This position is a fertile breeding ground for chauvinistic science (see page 113 of this thesis); that is, this position allows extending the rules and explanations from one "protodiscipline" to other disciplines, while simultaneously keeping the protodiscipline impervious to any outside influences.

It first looks like intellectual overkill to take the *intersubjectivist position* that every theory, technique, or theory of an instrument could be criticized from any conceptual/theoretical point of view. The possible discomfort with the intersubjectivist position is due to the fact that in the intersubjectivist position the number of possible frameworks for criticism is infinite.

Theoretically speaking, if intersubjective agreement (which refers to the adequacy of research results to different disciplines, see p.409) is regarded as one measure of research, the intersubjectivist position is a natural position. The higher the intersubjective disagreement about a given theory, technique, or theory of an instrument, the lower the credibility of that theory, technique, or theory of an instrument. For instance, if a certain technique for user interface design is credibly criticized from a mathematical or a technical point of view, the intersubjective agreement about that technique decreases. The more criticism that a certain technique for user interface design faces, the less credible it becomes.

Practically speaking, adopting the intersubjectivist position might not be much different from adopting the monodisciplinary position because in practice different conceptual and theoretical frameworks are incommensurable to some degree. For instance, sociological concepts and theories are not very efficient for criticizing integrals, and mathematical proofs are of little use for criticizing claims about human-computer interaction.

I consider the greatest flaw of the intersubjectivist position to be relativism about criticism. Whereas in the absolutist position there is the danger of *excessive detachment* of theories, techniques, or theories of instruments, in the intersubjectivist position there is the danger of *excessive relativism* of theories, techniques, or theories of instruments. In the absolutist position the value of theories, techniques, and theories of instruments can be detached from any external references, but in the intersubject-

ivist position theories, techniques, and theories of instruments gain their value from external references. In the absolutist position the boundaries of criticism can be set so strictly that there is no room for any criticism, and in the intersubjectivist position, when all criticism should be taken at face value, any critique can be adequate and to-the-point from some perspective.

I agree that neither shutting out criticism, taking the chauvinistic attitude towards criticism, nor becoming overwhelmed by criticism holds a normatively superior station over the other two positions. Descriptively speaking, computer scientists do not seem to be united behind any of the three positions mentioned above (absolutist, monodisciplinary, or intersubjectivist positions). Practical computer science is a mixture of receptivity and tenacity; computer scientists can at the same time draw on a number of disciplines and be stubborn in the face of multidisciplinary opposition. Take, for instance, Harvard's Howard Aiken, who led a multidisciplinary laboratory, yet who stubbornly resisted development steps in physics and computer science (see p.225 of this thesis) and Moore School's Eckert and Mauchly, whose multidisciplinary team opposed the scientific establishment of the time and built the ENIAC (see p.204 of this thesis).

In light of the sources in sections 3.3 and 3.4, I am compelled to say that instead of any of the three positions above, the mangle of practice (a cycle of conjectures, demonstrations, and implementations, as well as criticism, corrections, refutations, negotiations, and improvements) maintains a continuous redefinition of the descriptive and normative frameworks of computer science. There is no fixed conceptual or theoretical framework of computer science—whenever it is written down, the next moment it has already been altered. There are no fixed normative frameworks in computer science either—what is considered to be desirable in computer science today may not be that tomorrow. Some modern philosophers of science and sociologists of science argue that the lack of fixed frameworks is neither uncommon nor undesirable in any science[188].

188 Pickering, 1995; Feyerabend, 1993

The Mangle in Practice

In terms of computer science, Pickering's description of the "mangle of practice"[189] does not rule out the epistemological objectivity of scientific statements. Although all computer scientists have their own motifs, goals, visions, beliefs, and agenda, the computer science community and the "resistances" of the physical (and, indeed, computational) world constantly force a revaluation of the status of theories, techniques, and theories of instruments[190]. In the mangle, those theories, techniques, and theories of instruments that are thickly woven into the fabric of other theories, techniques, and theories of instruments; that are socially stabilized; and that have a robust fit with the physical and computational world; have a strong degree of epistemological objectivity[191].

I argued earlier (p.379) that (1) the fact that no single methodological system is applicable to the methods of inquiry in computer science means that computer science is methodologically disunited, and that (2) the fact that there is no consensus about the nature (the epistemological status) of research results in computer science means that computer science is epistemologically disunited. Methodological and epistemological disunity need not always be detrimental to a discipline, though. Although a methodological and epistemological disunity may cause conflicts within a discipline, as well as an untenable mingling of research paradigms, it also enables multi-method validation and connections to a wide variety of disciplines. Methodological and epistemological isolationism could inhibit alternative hypotheses and discourage complementary support that multi-method cross-checking of research results can offer.

Note that the intersubjectivist position is valuable only if intersubjective agreement is considered to affect the acceptance of a theory, technique, or theory of an instrument. If the value of theories, techniques, and theories about instruments is considered to be an intrinsic or field-specific feature, then the intersubjectivist position is meaningless. In Pickering's description of how science works, it is evident that intersubjective agreement affects the innofusion of theories, techniques, or theories of

189 Pickering, 1995

190 See the sections on objectivity, historicity, and relativity in Pickering, 1995:pp.194-212.

191 I use Searle's meaning of the term *epistemological* objectivity here (Searle, 1996:p.8). Note, however, that Pickering and Searle, whose books were published around the same time, do not refer to each other (Pickering, 1995; Searle, 1996).

instruments. In addition, in the mangle the success of theories, techniques, and theories of instruments also depends on their fit to non-scientific and physical realities.

One can take the mangle theory to the extreme and argue that there are really no scientific and extra-scientific arguments, but that everything affects everything. Fortunately one does not need to adopt such an extreme view to argue that social studies of computer science can capture something essential about computer science. If one agrees that the acceptance of scientific statements is not solely based on the credibility of the statements themselves, but on their fit to the existing framework of science, society, and the physical world, one can argue that social studies of computer science can capture something unique about the mangle of computer science.

Social studies of computer science can reveal how computer science is created and maintained. Social studies of computer science can explain how the statements in computer science are externalized, objectified, internalized, and reified[192]; that is, it can explain the processes through which many things that computer scientists produce are afterwards perceived as something other than human products. Social studies of computer science can explicate implicit assumptions, shared attitudes, and tacit knowledge. Social studies of computer science produces unique meta-knowledge of computer science. Meta-knowledge (knowledge about knowledge) is an important aspect of understanding computer science, because it can offer insight into even the most insightful theories of computer science.

The Disciplinary Implications of Social Studies of Computer Science

Social studies of computer science can be considered to be studies that situate and investigate computer science in its scientific, technological, social, historical, cultural, linguistic, political, economic, institutional, personal/individual, and other socially constructed frameworks. Social studies of computer science offers a view of computer science (qua knowledge and qua activity) in which more than merely the technological aspects of computer science are explicitly considered to be influential. As an umbrella term for different kinds of studies, social studies of computer science is methodologically, epistemologically, and ontologically uncommitted, but specific social studies of computer science can be committed to specific research philosophies.

192 Berger and Luckmann's terms (Berger & Luckmann, 1966).

Because of the multidisciplinarity of social studies of computer science, social studies of computer science can be considered to be a branch of computer science, sociology, history, anthropology, and philosophy, depending on the focus, angle, and aims of each study. Those kinds of investigations of social studies of computer science whose subject is meta-knowledge about the theories, practices, models, philosophy, or other aspects of computing can easily be considered to be a part of computer science and not a part of, say, sociology. It is much harder to categorize those kinds of social studies of computer science investigations that have a humanistic subject—subjects such as computer scientists' culture, the information distribution channels in computer science, or the social processes of proofs and theorems. The community of computer scientists is ultimately the judge of what kinds of research are included under the umbrella of computer science. However, situating social studies of computer science under clearly defined categories may not always be practical or reasonable. After all, those kinds of research that are most fruitful—those that benefit the largest number of disciplines—are the most valuable from the intersubjective point of view.

Social studies of computer science investigations might create more questions than answers. Keeping the doors open for sociological, historical, anthropological, or philosophical investigations of computer science may not increase the probability of triggering radically new ideas in computer science much. Instead of being a source of a revolutionary flux of radically new ideas, acknowledging social studies of computer science is rather a matter of disciplinary honesty and self-understanding (which is by no means a more modest undertaking than being a source of new ideas).

Social studies of computer science might not provide many answers to the current questions of computer science, such as “Can process *p* be automated?”, “What is the most efficient implementation for automating process *p*?”, or “What kind of interface for instrument *i* creates the least cognitive overhead?”. The logico-mathematical, modeling-based, and design-oriented branches of computer science can deliver answers to those questions.

Researchers of social studies of computer science focus on those aspects of computer science that are beyond the reach of computational theories, practices of design, or computational modeling. Researchers of social studies of computer science pose

questions such as “Why is the form and function of implementation *n* as it is?”, “Which aspects of computer science are contingent and why?”, “What reasons are there behind the popularity or unpopularity of certain programming languages?”, and “Why is the stored-program paradigm dominant?”. These questions cannot be answered from solely logico-mathematical, design, or modeling points of view. The answers to these questions require, for instance, sociological, historical, anthropological, and philosophical approaches.

Research that can be considered to be social studies of computer science is nowadays conducted by, for example, computer scientists, researchers of science and technology studies, interface designers, management information systems specialists, and historians of computing. The knowledge produced by social studies of computer science does not necessarily clash with the knowledge produced by other, more technologically- or mathematically-inspired branches of computer science. Rather than clashing with existing knowledge, social studies of computer science offers alternative viewpoints on the topics of computer science; it offers meta-knowledge about computer science. Social studies of computer science can be considered to be a complementary part of the multidimensional enterprise that computer science has been since the birth of modern electronic computing.

It has to be acknowledged, though, that the aims of social studies of computer science are not solely descriptive. In the sense that social studies of computer science offers alternative explanations of concepts, theories, instruments, techniques, methods, or designs of computer science, social studies of computer science can have normative aims too. In this sense, social studies of computer science can indeed be understood as questioning the legitimacy and foundations of some aspects of computer science. Questioning and contradicting is, of course, an essential condition for the growth of knowledge.

Those studies that I consider to be examples of social studies of computer science (Section 4.2), are consistent with current trends in science and technology studies in the sense that those studies refrain from making arguments about sciences at large. Instead of making arguments about science at large, current research in social studies of computer science has focused on issues of computer science as a unique branch of science. Unfortunately, because fields such as sociology of scientific knowledge

(SSK) and STS are relatively young fields, it is too early to say if research in those fields has affected science or the way science is conducted.

An Essential Part of Mature Computer Science

It has been argued that disciplinary self-understanding is an essential part of a scientific discipline[193]. In a similar manner, social studies of computer science, as portrayed in this thesis, is an essential part of the disciplinary self-understanding of computer science. It produces meta-knowledge about computer science. It helps computer scientists to understand the degree to which aspects of computer science rely on brute facts and the degree to which aspects of computer science are social constructs. It allows (but does not force on anyone) a realist philosophy, while it maintains that statements about reality are always socially constructed. It offers alternative angles on computer science qua knowledge and qua activity. It allows one to be critical about technological determinism, milestones, and machinery, and it helps one to understand the human-made character of theories and technologies. Social studies of computer science may not offer insight into the brute functioning of computing technology or into the logico-mathematical structures of the formal, deducted theories, but it does offer insight into *why* and *how* the technological-theoretical frameworks of computing have been constructed the way they have.

In Section 4.1 I noted that I use the term *computer science* here as a loose collection of concepts regarding studies of the computers and the phenomena surrounding computers; studies which aim at contributing to knowledge about automatic computation or at refining computational tools, theories, concepts, or processes (p.374). In Section 4.2 I expanded on how disciplinary self-understanding belongs to the project of computer science as a discipline (p.386). Social studies of computer science can respond to both characterizations of computer science (see Figure 29 on p.387 of this thesis).

Social studies of computer science can aim at contributing to knowledge about automatic computation by exposing essential human-constructed aspects of computer science. For instance, Harel's and Denning's arguments questioned a number of widely held “facts” of computer science[194]. Social studies of computer science can

193 Barnes et al., 1996:pp.*iix-xii*.
194 Harel, 1980; Denning, 1980

also aim at contributing to meta-knowledge about computer science by, for instance, offering knowledge about the social and psychological mechanisms that construct and maintain the tools, theories, concepts, and processes of computer science. For instance, Kidder's and Suchman's ethnographic studies reveal patterns of activities that are central to the technological and scientific production of computer science[195].

There are different kinds of social studies of computer science in the different branches of computer science, but it is difficult to find a suitable category for social studies of computer science in the ACM CCS (Computing Classification System[196]) —suitable in the sense that the human or sociocultural perspective would be explicit and it would be clear that those studies produce meta-knowledge about computer science. In the absence of an ACM CCS category that would underline the unique character of social studies of computer science, many articles that are, in essence, social studies of computer science are categorized into the branches of computing that those articles seem to benefit best[197]. In this sense, it has already been recognized that social studies of computing can benefit computing qua knowledge. Sometimes, however, social studies of computer science are classified into general categories[198].

The ACM category that best denotes a sociocultural or human perspective is K (*Computing Milieux*)[199], and many social studies of computer science are indeed tagged K.*. The subcategories of K, however, currently consist mainly of topics whose main concern is the *societal impacts of computing*, not the *impact of society on computing* or the *social construction of computing* (see Table 4).

195 Kidder, 1981; Suchman, 1987

196 See `http://www.acm.org/class/1998/` (accessed September 27th, 2006).

197 Avison et al.'s article *Action Research* (Avison et al., 1999) is categorized H.1.1 (Systems and Information Theory / *Information Theory*), K.6.1 (Project and People Management / *Systems Development*). Hirschheim and Klein's article *Four Paradigms of Information Systems Development* (Hirschheim & Klein, 1989) is categorized K.6.1 (Project and People Management / *Systems Development*), H.4 (Information Systems Applications), and K.4 (Computers and Society). Tedre et al.'s *Ethnocomputing: ICT in Cultural and Social Context* (Tedre et al., 2006) is categorized K.6.1 (Project and People Management), H.1.1 (Systems and Information Theory), and K.4 (Computers and Society).

198 For instance, Dave West's article *Hermeneutic Computer Science* (West, 1997) is in category A (*General Literature*).

199 The categories A-J in ACM Computing Classification System [1998] are (A) General Literature; (B) Hardware; (C) Computer Systems Organization; (D) Software; (E) Data; (F) Theory of Computation; (G) Mathematics of Computing; (H) Information Systems; (I) Computing Methodologies; and (J) Computer Applications.

Table 4: Top Two K. Sublevels in ACM CCS [1998]*[200]

K.1 THE COMPUTER INDUSTRY Markets Standards Statistics Suppliers	**K.5 LEGAL ASPECTS OF COMPUTING** K.5.0 General K.5.1 Hardware/Software Protection K.5.2 Governmental Issues K.5.m Miscellaneous
K.2 HISTORY OF COMPUTING Hardware People Software Systems Theory	**K.6 MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS** K.6.0 General K.6.1 Project and People Management K.6.2 Installation Management K.6.3 Software Management (D.2.9) K.6.4 System Management K.6.5 Security and Protection (D.4.6, K.4.2) K.6.m Miscellaneous
K.3 COMPUTERS AND EDUCATION K.3.0 General K.3.1 Computer Uses in Education K.3.2 Computer and Information Science Education K.3.m Miscellaneous	**K.7 THE COMPUTING PROFESSION** K.7.0 General K.7.1 Occupations K.7.2 Organizations K.7.3 Testing, Certification, and Licensing K.7.4 Professional Ethics (K.4) K.7.m Miscellaneous
K.4 COMPUTERS AND SOCIETY K.4.0 General K.4.1 Public Policy Issues K.4.2 Social Issues K.4.3 Organizational Impacts K.4.4 Electronic Commerce (J.1) K.4.m Miscellaneous	**K.8 PERSONAL COMPUTING** Games [*] K.8.0 General K.8.1 Application Packages K.8.2 Hardware K.8.3 Management/Maintenance K.8.m Miscellaneous

Because there are a good variety of studies which borrow tools from the social sciences and humanities, which produce unique socioculturally sensible information about the knowledge as well as the construction of knowledge in computer science, and which clearly belong to computer science, there is a need for category K.9 (*Social Studies of Computer Science*) in the ACM classification system.

Tagging a publication K.9 would indicate (1) the choice of a sociological, historical, anthropological, philosophical, or other social sciences- or humanities-based perspective, and (2) the production of meta-knowledge about computer science qua knowledge and qua activity. In a manner similar to sociology, history, anthropology, and philosophy, the methods of social studies of computer science can be qualitative, quantitative, or mixed-methods. Since ACM-published articles are often giv-

200 Source `http://www.acm.org/class/1998/` (accessed September 27th, 2006). Republished here with permission of the ACM (permission granted for digital and printed publications).

en more than one category tag, tagging a publication K.9 does not exclude the article from any other categories.

In fact, in the sense I described in Section 4.3, some of the current K.* classes (see Table 4) are less about the discipline of computer science than my proposed K.9 (*Social Studies of Computer Science*). That is to say, in Section 4.3, I argued that if research aims at honing theories, practices, models, philosophy, or other aspects of computing, then the research is computer science (page 415 of this thesis). Many of the K.* categories are geared towards understanding something other than aspects of computing—take, for instance, K.5 (*Legal Aspects of Computing*), K.4.1 (*Public Policy Issues*), and K.6.1 (*Project and People Management*). If one agrees that those categories—the aim of which is not exclusively to understand automatic computation or computer science but rather to understand the impact of computing technologies on society—belong to computer science, it should be easy to agree that K.9 (*Social Studies of Computer Science*), the aim of which is to understand computer science, is at least an equally legitimate part of computer science.

5. Conclusion

This research stems from the oft-stated need to explore the connections between technology, academia, institutions, the sciences, social milieux, human practices, economic concerns, agenda, ideologies, cultures, politics, arts, and the other technological, theoretical, and human aspects of the world. The rationale of this thesis is simple: If perspectives from disciplines such as sociology, history, anthropology, and philosophy can positively contribute to the body of knowledge, meta-knowledge, or practices of computer science, then one can reasonably argue for an extension of computer science with those perspectives.

In this chapter I restate the findings and insights of my research. I have made my arguments in Chapters Two, Three, and Four, and there are neither references nor any new arguments in this chapter. I finish this chapter by evaluating how well my research findings respond to my research questions and by considering research questions that arise from this research.

I begin by outlining the conceptual framework of this thesis (which is reported in Chapter Two). I continue by presenting my analysis of the development of computing and computer science in light of my conceptual framework (this analysis is described in Chapter Three). From there I continue to analyze two particularly clear models that can explain technical and scientific change in computer science, to analyze the workings of computer science, and to describe the anatomy of proofs and assertions in computer science (that analysis is described in Chapter Three and Chapter Four, passim). hyppy

Following that, I discuss the "fundamental question underlying all of computing" (which is discussed in detail in Section 3.4). Then I present my argument about eclecticism, opportunism, and anarchism in computer science (the argument is presented in more detail in Section 4.1). After that, I briefly summarize the implications of my philosophical position on this research. I continue by describing the place and role of computer science in society (summarized from Chapters Two, Three, and Four), outlining social studies of computer science (described in Sections 4.2 and 4.3), and proposing the implications of this thesis for computer science (summarized from Chapter Four). I finish this chapter with an evaluation of this thesis.

The Philosophy of Computer Science

The philosophy of computer science is in disarray. Despite some attempts to develop the philosophy *of* computer science à la the philosophy *of* physics and the philosophy *of* mathematics, there is no common understanding of the content, aim, focus, or even topic of the philosophy of computer science. It is disputable whether computer scientists can be said to adhere to the precepts of any single major philosophy of science. Many branches of computing lack uniform laws, design principles, well-established and well-grounded guidelines, or other aspects that a scientific paradigm should have. Because there is no clear philosophical framework for computer science research, each researcher has to either adopt one from other sciences or construct his or her own from scratch. In an interdisciplinary science like computer science, constructing a holistic philosophical framework might be a difficult task, though. Researchers may need to adopt different kinds of conceptual, theoretical, methodological, and philosophical frameworks for each research study. In this sense, the term philosophy of computer science might perhaps be better construed as an umbrella term for a number of eclectic frameworks, concepts, and theories, than construed as a holistic, all-inclusive single framework.

It seems that trying to find a middle-way between different kinds of extreme accounts of science and technology leads to yet new extremes. All extreme positions, such as naïve positivism and relativism, face difficulties, and none of them are absolutely superior over the others. Only a variety of pragmatic accounts prevail. Hence, in this thesis I do not argue for a theoretical framework, but for a conceptual framework in which I bring together a number of theories of science and technology that resonate well with the historical development and the working mechanisms of computing and computer science.

My conceptual framework includes concepts from a number of major philosophies of science, yet I do not tie my research to any one of them specifically. For instance, in the context of computer scientists' work I use Popper's and Feyerabend's terminology, in describing the development of computer science I use Kuhn's terminology, and in describing the anatomy of proofs in computer science, I use Lakatos' terminology. The epistemological and ontological aspects of my conceptual framework are built on John Searle's realist ontology, which also is successful at explaining socially

constructed reality. In Searle's ontology, there are facts that do not depend on any attitudes people may have about them. However, people can attach particular meanings to objects or facts—or create new facts altogether. Those facts exist only when people, who bear those facts, exist, and they cease to exist when people, the fact-bearers, cease to exist.

On one hand, acknowledging *brute facts* enables computer scientists to be true to the working mechanisms of the physical world, which is important because physical reality determines some characteristics and limits of automatic computation. On the other hand, the recognition of *institutional facts* allows for explanations of socially constructed things that would not exist without people, such as money, labels, names, and theories. Institutional facts are important because names, labels, theories, arguments, and other constructed facts constitute a part of the reality with which computer scientists work.

The chemical and physical behavior of the substances that computers are made of does not depend on humans. Everything else in the domain of computers is socially constructed. For instance, the arrangement of logic gates, the radix of numeral systems, the design choices of execution units, abstraction layers, and hierarchies are intuitive choices that computer designers make. Designers make their choices based on economics, functionality, architectural choices, power consumption, heat dissipation, standards, design preferences, and so forth. Although computer designs are completely human-made, one cannot tell for certain if the hierarchical structures of computer systems are a *consequence* of an inherent hierarchy of the world, or if they are a *tool* for understanding an inherently unorganized world.

Algorithms and programs, however, do not have a clear place in Searle's ontology. The seemingly innocent claim that algorithms are ontologically objective—that they exist independently of people—is, in fact, a very controversial claim. If one claims that all computer programs are ontologically objective, then one claims, in effect, that Microsoft Word exists independently of humans. In the natural science sense, it would sound more odd to say that mathematical and logical truths exist regardless of people than to say that they exist only with people. Namely, to argue that mathematical and logical truths exist regardless of people would be to argue that something

immaterial can exist and that people would somehow have access to those immaterial things.

Even if it could be agreed upon that algorithms, logic, and mathematical objects do not exist regardless of humans or other intelligent beings conceptualizing them, algorithms, logic, and mathematical objects could still be *epistemologically objective*. An algorithm *A* that is able to perform function *f*, performs function *f* independently of anybody's attitudes or feelings towards the algorithm. It is an objectively ascertainable fact that algorithm *A* performs function *f*. Rather than arguing for the universality and non-temporality of algorithms, my position is that algorithms are constructions, which cannot be considered to be correct or incorrect in any absolute sense, but which are useful in a limited context. Algorithms are correct or incorrect only within a certain logico-mathematical-technological framework, but within that framework their correctness or incorrectness is an objectively ascertainable fact.

Contingencies Surrounding Computing

Historians of computing have argued that the circumstances in which the ENIAC and EDVAC were developed were brought together by the political situation, the war effort, advancements in science and engineering, new innovations in instrumentation, interdisciplinarity, influential individuals, coincidences, a disregard for costs, and a number of other sociocultural factors. In a similar manner, the founding of the office computer business in Britain was a result of a number of sociocultural, economic, and technological factors mixed with extraordinary foresight, or, simply, contingency.

Also, the development of high-level programming languages has been attributed to a number of influential sociocultural factors. Organizations, institutions, sponsors, and profit-making companies played a major part in the early development of programming languages. The purposes, problem domains, and functional requirements of languages influenced language design. And also individuals—their backgrounds, motivations, and mental models—were influential in how programming languages developed. It has also been argued that culture affects, for instance, the amount of funding a field gets, the valuation of theory and practice, the foci of research, the popularity of technological disciplines, and public support.

The contingency thesis states that the current state of affairs in technoscience could have developed through a very different course than it did, and that a successful alternative to current technoscience does not need to have any commonalities with current technoscience. Adopting the contingency thesis brings with it the possibility for alternative paths in computing: Had the people at Moore School been, say, Vannevar Bush, Konrad Zuse, and Alan Turing instead of Eckert, Mauchly, von Neumann, and Goldstine, computer technology might have developed in a very different way; had some of the circumstances of the British computing scene been different, British office computing might have developed in a different way; had the starting points for the development of high-level programming been different, the means of commanding the computer might be very different from what it is today.

There are no means for proving the contingency thesis right or wrong. Because researchers of today can only speculate on alternative routes that computing might have taken, one can only argue for the contingent character of the current situation. I argue that the computing of today is just one possible path of an infinite number of paths—there is no *necessary* path for technological development.

The Stored-Program Paradigm

If one were to examine computing from the Kuhnian point of view, it would be hard to find scientific crises or accumulating anomalies in the history of electronic computing. Even the most decisive changes in the field of computing—the shift from mechanical to electronic computation and the conception of the stored-program computer—were not results of anomalies. The conception of the stored-program computer was not a *refutation of* anything, but a *shift to* a technical and theoretical paradigm—to the *stored-program paradigm*. The stored-program paradigm, however, entails only a technical model and a theoretical framework. It does not dictate forms of inference, logic of justification, modes of argumentation, practices of research, conventions for settling scientific disputes, or other aspects of a scientific paradigm. Regarding inference, logic, argumentation, or other kinds of conventions and practices, computer scientists rely on various scientific paradigms.

The stored-program concept marked the birth of a theoretical and technological paradigm in computing. The blueprints for EDVAC introduced a number of concepts that lacked counterparts in the technoscience of the time (e.g., *memory addresses* and

registers). Because those concepts were incommensurable with the science of the time, there could not be a continuation (or explanation, or reduction) between old and new concepts. The concept *register*, for instance, did not have a counterpart in the pre-stored-program world. Post-stored-program computing was incommensurable with pre-stored-program computing.

Currently, von Neumann-architecture has gained enough momentum that it is, despite all of its limitations, largely taken as an unquestioned foundation for successful automatic computation. Nonetheless, there are serious attempts to break the barriers of von Neumann-architecture and Turing-computability. Those attempts are not responses to anomalies in the von Neumann-architecture and Turing-computability, but attempts to break the limits set by them. The Kuhnian concept of paradigms, albeit oft-utilized, may not be fully applicable to technological change.

Technological Momentum

If one is concerned about the effect of technology on society, or on people, one is a technological determinist to some degree. That is, if and only if one believes that technology can have an effect on society, one can be worried about the sociocultural effects of technological change. Technological determinism comes in a variety of types and in spectra of degrees, but the crux of technological determinism does not seem to be about technology as much as it is about a technofetish in which human progress is measured by the sophistication of technological artifacts. But technological abundance is a poor measure of human or cultural progress—for instance, democracy, equality, and human rights are not results of technological development.

My conceptual framework includes Thomas Hughes' brand of technological determinism, called *technological momentum*, in which both social constructionism and technological determinism are characteristics of the development of technological systems. Young technological systems exhibit characteristics of social construction, but the more widespread and more established technological systems become, the more characteristics of technological determinism they exhibit.

Hughes' temporal axis in technological determinism gives epistemological objectivity an interesting twist: The recognition of newly-born innovations is an epistemologically subjective matter, but usually the older and more prevalent innovations become, the more epistemological objectivity they gain. The increase in epistemolo-

gical objectivity is closely linked with those innovations becoming more rigid, less responsive to outside influences, and thus more deterministic by nature.

The non-discrete nature and temporal dimension of epistemological objectivity are important in this thesis for a number of reasons. Firstly, they are of use in proportionating and locating computing with other, usually older disciplines. Secondly, they help to understand change and stability in computing. Thirdly, they help to understand computing as a cause and as an effect—technologies are shaped by society but they can also be used to shape society.

Technological Momentum in Computer Science

The growth of technological momentum is evident in a number of my examples of the history of electronic computing. In those examples, the birth of innovations has taken place due to a contingent convergence of various sociocultural and technoscientific factors, but in the course of time the innovations have grown rigid and become institutionalized. In the following text I present three particularly lucid examples of the growth of technological momentum.

Firstly, in the early days of electronic digital computing there was great uncertainty about research directions, and most computing machines of the time differed from each other in their architecture, design, constraints, and working principles. Over the course of time knowledge about the directions of computing accumulated, and researchers increasingly followed traditional paths which others had tread. This is traditionally called the growth of knowledge, yet it is also characteristic of the growth of technological momentum.

Secondly, previous design choices and compatibility issues did not hinder early computer designers so they were able to start with a tabula rasa. But, over time, computing systems became more complex and more interdependent. As the number of computer installations grew, the design decisions of computing had to be increasingly based on compatibility. The first united architecture, the IBM System/360, marked a definite turning point in the change from social constructionism to technological determinism in computing machinery.

Thirdly, the early construction of FORTRAN was directed by sociocultural and personal motivations, as well as economical and institutional considerations. The more

FORTRAN developed and the wider it spread, the more it institutionalized and the more rigid it became. Over time FORTRAN gained momentum that had a tremendous impact on the subsequent development of computing. FORTRAN achieved such an institutional status that it is even today regarded as the lingua França of scientific computing.

As FORTRAN gained technological momentum, it became more a shaper of its environment than shaped by it. FORTRAN was followed by much more elegant programming languages, especially ALGOL. Although ALGOL was supported by a large number of influential stakeholders, it was never able to gain the momentum FORTRAN did, and it never became a similar shaper of computing practice. (ALGOL was a significant conceptual milestone and a shaper of academic computing, though). FORTRAN's colossal technological momentum made it the de facto standard of computing for a long time.

The shift from machine language programming to high-level language programming was not, however, a paradigm shift in the Kuhnian sense. The chaotic state of early programming was restrictive in many senses—it barred non-specialists, separated computer brands, and hindered portability. It is true that this chaos seems like a computing version of Kuhn's *pre-science* state. But although there was clearly a revolution, it was not a Kuhnian revolution: Firstly, higher-level languages did not replace machine languages, but provided a powerful alternative (machine-language programming is still needed); secondly, there were no anomalies that had led to a crisis. The introduction of high-level language programming and the standardization of languages built on earlier work in computing and did not refute or make obsolete any earlier work in computing.

The Mangle of Practice

Technological and intellectual changes in computer science are well explained by Andrew Pickering's *mangle of practice*, which describes the development of science and technology as an ongoing cycle of development and revision of (1) theories and models, (2) the design and theory of instruments and how they work, and (3) the instruments themselves. When a computer scientist works, usually things do not go as planned—the world resists. He or she accommodates to this resistance by revamping some or all parts of the theoretical-technological structure, and tries again. In the end, the computer scientist hopes to get a robust fit between the theoretical and technological elements of a research study. In addition, researchers often need to accom-

modate for sociocultural factors, too—technoscience is not developed in a vacuum but within a dynamic network of societal, economic, cultural, institutional, ideological, political, philosophical, and ethical factors.

The birth of the stored-program paradigm is a prime example of the mangle in computing. The researchers at Moore School of Electrical Engineering had their theories of computation, their prototype machines, and their theories of how their machines should work. They had to deal with numerous dead-ends—sometimes even knowingly; had to devote considerable effort to developing peripheral components, test equipment, and component technologies; had to revise their theories and concepts often; and had to spend a great deal of effort convincing other stakeholders about the value of computing. Through numerous accommodations; such as revisions of theories, modifications of components, and rebuilding of instruments; the researchers at Moore School gradually arrived at the stored-program concept.

The mangle of practice also applies to the mechanisms of the growth of knowledge in computer science. In computer science, scientific statements about the subjects of computer science gain their credibility through demonstrations, publicity, and debate. Those statements gain their credibility via conjectures, modifications, proofs, refutations, critique, and adjustments. Instead of following any given set of rules of computer science *proper*, computer scientists present theories, demonstrate techniques, or implement mechanisms, and then submit those theories, techniques, or descriptions of mechanisms for public criticism.

Through a continuous cycle of corrections to theories, honing of techniques, improvements of mechanisms, negotiations between stakeholders, debates, power struggles, and constant criticism, some theories and techniques gradually become a part of the relatively stable core of computer science. Similarly, the knowledge that researchers gain when they construct and revamp instruments is gradually crystallized into theories about instruments, which, in part, gradually become a part of the relatively stable core of computer science. That is, theories, techniques, and theories about instruments gradually increase their epistemological objectivity. Practical computer science is a mixture of receptivity and tenacity; computer scientists can at the same time draw on a number of disciplines and be stubborn in the face of mul-

tidisciplinary opposition. It should also be noted that the knowledge-cultivating mechanisms can differ between branches of computer science.

I argue that the mangle of practice—a cycle of conjectures, demonstrations, and implementations, as well as criticism, corrections, refutations, negotiations, and improvements—maintains a continuous redefinition of the theoretical-conceptual frameworks of computer science. There are no fixed conceptual, theoretical, or methodological frameworks of computer science. The theoretical-conceptual framework of computer science looks different when viewed from alternative perspectives. There is no normative account of computer science either (with the possible exception of ethical norms)—what is considered to be a good practice today may not be that tomorrow.

It must be noted that the mangle is not a relativist position. The mangle of practice does not rule out the epistemological objectivity of scientific statements. Although all computer scientists have their own motives, goals, visions, beliefs, and agenda, the computer science community and the resistances of the subject area of computer science constantly reappraise the epistemological objectivity of theories, techniques, and theories of instruments. The resistances can arise from unfamiliar aspects of physical reality; from problems and uncertainties with the human-built structures of computation; from inadequate knowledge of the users and the social world; and from a deficient fit between the physical, computational, and social worlds. In the mangle, those theories, techniques, and theories of instruments that are deeply woven into the fabric of other theories, techniques, and theories of instruments; that are socially stabilized; and that have a robust fit with the physical, computational, and human world have strong epistemological objectivity.

Computer Science

There are parts of computer science, such as the stored-program paradigm, that have served as an uncontested basis for research for long periods of time. Nevertheless, the field at large has changed radically between the 1940s and the 2000s. Even before the software crisis in the late 1960s, there was a wide divergence of opinion on what computer science was and what it should become. The software crisis and the emergence of business computing fueled the diversification.

In the early days of modern computing, computer scientists yearned for the recognition of mathematicians and natural scientists. Until the mid-1970s computer science was indeed conceived as a theoretical, mathematically based discipline. During the software crisis, computer science gradually evolved a programming and applications-centered dimension. Throughout the history of computing, shifts in the user base of computing have been paralleled by changes in the academic discipline of computing. Especially in the 1970s a number of fields of study; such as HCI, MIS, operating systems, and networks; emerged, and the 1970s indeed saw the shaping of computing into a truly multidisciplinary or interdisciplinary field.

At present there is a wide array of mutually incompatible viewpoints to what computer science is and what computer scientists do. Differences between the views arise largely from emphasizing some aspects of computing at the expense of some other aspects of computing. The breadth of the field of computer science makes it especially difficult to describe how computer science is *actually* done, or to prescribe how computer science should be done—the same accounts of computer science should cover inquiries about the physical world, about mathematics and logic, as well as about people, groups, and activities. It is difficult to see how a single descriptive or normative account of computer science could accommodate software engineering, complexity theory, usability, the psychology of programming, management information systems, virtual reality, and architectural design.

No single methodological system is applicable to the methods of inquiry in computer science, and there is no consensus about the ontological and epistemological nature of research results in computer science. There is no stern watchdog Computer Science to enforce methodological regimentation and to rule out non-legitimate, ill-suited, or inappropriate methods, tools, or approaches. It is difficult to imagine an overarching philosophy for a science, the topics of which vary from logic to software engineering, from artificial intelligence to complexity theory, and from digital design to human-computer interfaces.

Mathematical and computational models, on one hand, are precise and unambiguous, yet they are confined to an abstract world of mathematics and they fail to capture the richness of reality (physical, social, or even computational). Narratives and ethnographies are rich in dimension and sensitive to detail, yet equivocal and context-de-

pendent. Narratives have little use in deriving formulæ, and formal proofs have little explanatory power regarding usability. The diversity of necessary research approaches renders computer science a methodologically and epistemologically eclectic discipline.

Difficulties in the natural sciences most probably stem from unfamiliar, unrecognized, or unknown complexities in the interdependencies of the physical world. Difficulties in the social sciences most often stem from complexities of the social reality. But when a computer scientist encounters problems, the difficulties stem predominantly from computer scientists' earlier work—computer scientists have brought about the clarity or complexity of their own discipline. Earlier theoretical, conceptual, and design choices in computer science define the basis for future challenges. Computer scientists are much more responsible for the complexity of computer science than physicists, chemists, biologists, sociologists, or anthropologists are responsible for the complexity of their disciplines.

Computer scientists are particularly good at mastering complexity in strictly bounded realms (microcosmoses), but computer scientists invariably fail at coalescing a number of microcosmoses into coherent systems (macrocosmoses). Technological microcosmoses are based on exactitude, predictability, discreteness, and determinism, and complexity is manageable in microcosmoses in which all variables can be controlled. Although computational microcosmoses can be made very exact and unambiguous, when one moves outward to study larger entities such as computer networks, ambiguity and unpredictability increases.

The unpredictability in very large systems does not come only from design flaws on single abstraction layers, but also from the ambiguity of interconnections between semantic levels. In the largest-scale macrocosmoses, unpredictability comes from the world outside the computational boundaries—power outages, electric interference, faulty machinery and cabling, intentional actors (humans), transmission errors, faulty machinery and code, and even “malicious” code and malicious intentional actors. In many branches of computer science, researchers cannot count upon the exactitude, predictability, discreteness, and determinism of computational systems.

Computer science is an extraordinarily dynamic field. The objects that computer scientists study are dynamic, and computer science as a field is extraordinarily dy-

namic. Programmers constantly reshape the conceptual framework (constructions) of computer science, they constantly generate new ways of applying computing (relationships), and they regularly develop new theories, models, and tools for their purposes. In other words, dynamism is an integral part of the *constructions* that computer scientists create, but also an integral part of the autopoïetic *construction* of computer science. Dynamism is an integral part of *relationships* between the objects of computer science, but also an integral part of *relating* computer science to its surroundings.

Because it is very hard to define computer science in any categorical manner, as a very general rule one could argue that if research aims at honing the theories, practices, models, philosophy, or other aspects of automatic computing, then that research is computer science. Because it is impossible to circumscribe the methodology *proper* of computer science, it does not matter what methods and standpoints a computer scientist uses for studying computer science as long as the research is aimed at the benefit of computer science. I argue that it would be irresponsible to deny *any* (ethical) methodology at *any* point of time because people do not know which methodologies may reveal unexpected features of computing. Granted, it is easy to question the characterization of computer science above by asking, for instance, "What exactly is *computing*?", "What if a study benefits many disciplines?", and "How can one tell what contributes to computing and what does not?". Ultimately, the scientific community always decides what is computer science and what is not (yet the community usually does not agree).

Proofs and Assertions

A computer scientist can run an extensive array of tests and become convinced about the statistical correlation between some variables. Descriptive claims, such as "*The amount of experimentation correlates positively with the speed, reliability, and usability of software*", can be based on such tests. However, the argument that experimentation is beneficial for computer science cannot be based on experimentation, because statements of beneficialness are value statements.

Normative statements cannot be based on consensus or definitions (what value statements *can* be based on is indeed a debated issue). Normative statements follow a set of rules different from descriptive statements; for instance, they require establishing

desirability and credibility. Certainly, one can increase the credibility of normative arguments in computer science with broad experimentation, cross-checking of results, elimination of superfluous variables and external inference, valid inference, independent confirmations, and a good fit with existing knowledge, but ultimately the acceptance of normative arguments is based on human judgment.

It is unlikely that outside axiomatic systems, assertions can be *proven* correct. Also, falsificationism fails to capture the essence of scientific justification. Statements such as "GO TOS increase code entropy" and "Good modularization reduces maintenance costs" are examples of statements that are very difficult to put into a falsifiable form, and in a falsifiable form they may not be very useful anymore.

Even though computer science works according to Boolean logic, Boolean logic hardly applies to all aspects of computer science. Unlike Turing-computation, modern computing is not of the form "input-process-output", but a mesh of interrelated, interactive, complex, chaotic systems of actors. The actors in that mesh are computers and other instruments, but the actors are also people and natural phenomena that interact with computers through a variety of interfaces. The causal systems that computers are a part of, and in which computers interact, are non-discrete, non-quantifiable, sometimes intentional, uncertain, dynamic, and exceedingly complex.

Qualitative questions and ambiguities cannot be exorcised from even the most quantitative studies of computer science. Computational instruments and their semantic content are created for a purpose (they exist for some purpose), and they work according to some imperfect, simplified, pruned models of reality. Programmers create static models from a dynamic, complex, and infinite world, and it is impossible to copy-paste aspects of dynamic reality into a static model without losing some qualities of those dynamic aspects. Even mathematical and scientific theories are made for some purposes, they entail hidden assumptions, and their acceptance is a matter of belief.

Theories, models of computation, and tools are not intentional, but theories, models, and tools are constructed by scientists who are intentional. If there are any practical purposes or intentions behind constructing things, then those things cannot be fully understood without understanding the practical purposes or intentions that motivated their creation in the first place. Specifically, one cannot fully understand the func-

tion of the computer without understanding the intentions of its designers. Granted, one can understand the *functioning* of the computer, or how the computer works, without knowing the intentions of the designers of the computer. But one cannot know with certainty the *function* of the computer without knowing the intentions of those who designed it.

What Should Be Automated?

It has been argued that the most fundamental question underlying all of computer science is "*What can be (effectively) automated?*". Taking into account the development of computing towards user-, human-, or value-centered computing, the aforementioned fundamental question is passé; it belongs to the realm of the old, machine-centered computing. I suggest that a more apt fundamental question underlying all of human-centered computing should include the theoretical *what*, the practical *how*, and the ethical *why*. The fundamental question should be, "*How can one effectively automate processes that can be automated and that should be automated?*".

The theoretical, practical, and ethical parts of the fundamental question are equally difficult and equally important. The theoretical problem, "*What can be automated?*", is fundamentally unresolved and fundamentally difficult. In the practical problem, "*How can one automate p*?" the solution is usually underrepresented by models, so the choice of the implementation depends on the practitioner's opinion, experience, and proficiency. The ethical problem, "*What should be automated?*", requires a deep understanding of, for instance, the civic obligations, duties, and liberties; the different conceptions of what is desirable and valuable; the sociocultural differences between laws, norms, and morals; and so forth. Social issues have, ever since the 1970s, been a recognized and important part of computer science education, and social issues even have their own Computing Reviews category (K).

An Efficient Anarchy

Methodological concerns do not play a crucial part in computer scientists' work. Methodological descriptions are often omitted from research reports in computer science. Computer science students usually learn their research skills from mentoring by their professors and from imitating previous research. Typical computer science

curricula do not include courses on research methodology, yet computer scientists utilize a wide array of research methods, and often combine methods in order to gain a wider perspective on the topic. In a sense, many computer scientists might be characterized as *bricoleurs*—as researchers who work between competing paradigms. However, it is not certain if the resulting *bricolage* is valid from the perspective of any of the research approaches entailed in the bricolage.

The community of computer scientists has not in the past conformed, and does not currently conform, to any major philosophies of science. The diversity of research approaches necessary to computer science renders computer science a methodologically and epistemologically eclectic discipline. Based on the methodological and epistemological nonconformity of computer scientists, computer scientists can be characterized as opportunists. The eclecticism and opportunism of computer science, combined with conscious breaches of methodological and epistemological norms, render computer science an anarchistic enterprise.

The *computer scientist-as-a-bricoleur* should be familiar with a number of paradigms, but also be aware that paradigms are more or less incompatible. Scientific paradigms; which are comprised of particular ontological, epistemological, and methodological views; cannot easily be mixed. Although much of the innately interdisciplinary research in computer science can be characterized as *bricolage*, the training of computer scientists does not prepare them for research with multiple paradigms. It is even dubious whether the general training of computer scientists prepares them for research within any single paradigm.

Informed anarchism need not be detrimental to science though. Interdisciplinary work was crucial in the shift from electromechanical computation to electronic computation; interdisciplinarity also spurred new ideas within traditional disciplines. I argue that an eclectic combination of incommensurable crafts and sciences creates an ontological, epistemological, and methodological anarchy, which inhibits dogmatism because no ontology, epistemology, or methodology can claim superiority over others. In an interdisciplinary situation many theoretical or metatheoretical issues, counterarguments, alternatives, or disciplinary controversies have to be elided for practical reasons. Superficial knowledge about powerful ideas enables researchers to utilize concepts or innovations without getting mired in field-specific debates.

But utilizing superficial knowledge also elevates the risk for interdisciplinary researchers to get mired in problems that an expert would avoid, adopting folk theorems as laws, and using research methods in superficial, incorrect, and contradictory ways.

Since 1945 computer scientists have deepened both the theoretical and technical knowledge about computing, and computer science has also worked as a catalyst in the creation of new research fields and spurred research in other disciplines. I argue that computer science has been efficient *because* of anarchism, not *despite* it. Anarchism has been woven tightly into the fabric of computer science from the beginning of early electronic digital computing. For example, the birth of the stored-program-paradigm was a result of a successful combination of a number of epistemologically and methodologically incompatible disciplines such as logic, electrical engineering, mathematics, physics, and radio technology. Since 1945 computer science has been influenced by a large and eclectic bunch of disciplines.

Many innovations in computer science have been spurred despite the lack of support by the academic establishment, and sometimes even despite strong opposition by the establishment. In addition, without anarchism in computer science, the innofusion of many innovations in computer science could have been much slower, and some innovations might have not been introduced at all. For instance, computer scientists widely adopted Dijkstra's the "GO TO statement considered harmful" hypothesis without ever testing it empirically.

The dogmatic demand for reducibility or "instant clarity" in computing is a hindrance for (at least) two reasons: one practical, one ideological. Practically speaking, if new ideas in computing have to be explained in terms of current academic computer science, the power of expression of new concepts is reduced to the power of expression of current academic computer science. In other words, the demand for reducibility sets an upper limit for what can be expressed by any new theory, and that limit is bounded by what can be expressed through the science of that time. Ideologically speaking, the demand for instant clarity elevates academic computer science to the intellectual standard status. If one demands instant clarity, any concept that cannot be clarified by showing a counterpart in current science is automatically considered to be inferior in comparison to the concepts of current science.

The above-mentioned anti-dogmatic arguments against the demand of reducibility are well in line with the fact that a dogmatic view of computer science contradicts a number of ideals of science, such as scientists' freedom to set their own goals, to pursue their inquiries in a field of their choice, or to choose whatever approaches they feel are necessary for their work. A requirement of instant clarity from research—requiring that research results must be explainable in terms of preordained, select theoretical and conceptual frameworks—is clearly untenable. There cannot be radically new innovations if new ideas are bound to the explanatory power of computer science as it exists at a given time.

In some circles there is a high degree of confidence in the explanatory power of computer science when applied to the wide array of fields of scientific and human inquiry. Consequently, there are tendencies to extend computational explanations to different disciplines. Computational models of social reality may be very useful in offering alternative explanations of social interactions. However, an algorithmization of the humanities and social sciences; when combined with a hierarchical view of sciences, a systematic deprecation of interpretive modes of explanation, and an enthroning of logico-mathematical inference; is nothing short of a forceful occupation of intellectual territory. This intellectual takeover has chauvinistic characteristics when it comes coupled with an asymmetric, systematic denial of sociocultural interpretations of computing.

Constructionism and Intersubjectivism

Certainly, my framework is not a structurist or an inevitabilist one, but a constructionist one. In examining the development of computing, I consider the circumstances in which development happens to influence and shape development. The sociocultural, academic, and political environments (and all other environments) of researchers affect the directions and forms that computing and computational theories take. Indeed, these environments make development possible in the first place. There is no "natural direction" towards which the development of computing necessarily must head but the development of computing is subordinate to certain cultural, social, historical, political, ideological, institutional, economic, technological, and scientific milieux.

Constructionism has some ramifications for my thesis. It offers a basis for my analysis of the development of computing. In the constructionist framework, studies of computing from perspectives from disciplines such as sociology, history, anthropology, or philosophy can offer insights into why computing is what it is. But constructionism also confines my research to a certain framework. I conduct my research and make my conclusions within the constructionist framework, and my arguments may not be cogent outside the constructionist framework.

My work also works within the intersubjectivist framework. Theoretically speaking, if the adequacy of research results to different disciplines is regarded as one measure of research, then any critique of research results must be recognized, regardless of the field on which the critique is based. The higher the intersubjective disagreement about a given theory, technique, or theory of an instrument, the lower the credibility of that theory, technique, or theory of an instrument. The intersubjectivist position is not a relativist position; there can still be apposite critique and irrelevant critique, and there can still be well-founded critique and groundless critique.

Practically speaking, it might not make much difference whether a critique of research is freely based on any certain field, intellectual tradition, or school—academic, non-academic, expert, or non-expert alike. In practice, different conceptual and theoretical frameworks are incommensurable to some degree and people in different trades have different concerns. For instance, sociological concepts and theories are not very efficient for evaluating integrals, and mathematical proofs are of little use for evaluating claims about human-computer interaction. Furthermore, sociologists may not usually be interested in evaluating mathematicians' work and vice versa.

Computer Science in Society

Computers and digital communication technologies are an everyday part of Western people's lives, and they weave well into the fabric of modern Western economies, work, socialization, leisure time, governing, media, and many other aspects of life. There is a vast amount of research on how the use of information and communication technologies changes societal phenomena. Today's researchers have few excuses for not controlling the ramifications that using computer technology has on societies.

Although technology per se is value-free, the development and the use of technologies are always processes in which the developers and users of technology have a number of motivations. Unlike technologies, conscious human actions are not value-free. There is a distinction between *technology per se* and the *production or use of technology*. Technologies are unintentional objects and concepts and unintentional objects and concepts do not have morals. Technologies per se are no more good, evil, benevolent, or malicious than they are happy, sad, angry, or meek. This is not to say that the designers of technology are free of responsibility. The designers of technologies have motivations and morals and they must, with the users of technologies, carry the responsibility and the pang of conscience that come from creating technologies that can be used for unethical purposes. Although computer scientists *cannot* be held responsible for the unintended consequences or unforeseen malicious uses of technology, they *must* be held responsible for the foreseeable effects and side-effects of technology, and they *must* be able to justify their choices.

Computer science, as an institution, must be governed by similar safeguards that govern other quarters of modern democracies. Computer science is not separate from society, but it affects society and is affected by society. Computer science cannot be élitist and accountable only to itself; as an institution that is run at society's expense and that is an irrevocable part of society, computer science must also be accountable to society.

If anyone can be accountable for what kinds of technologies are created, ultimately the ones accountable are those who create technology. In a society where computing technology is a catalyst of change, computer scientists—who understand the restrictions, capabilities, prospects, and threats of computing technology better than anyone else—must also take responsibility for the technology they create. However, the chances are that the methodological, conceptual, and theoretical frameworks of computer science turn out to be insufficient to deal with the changes brought about by the computer revolution. The chances are also that the frameworks of computer science turn out to be insufficient for selecting, recording, understanding, explaining, analyzing, or predicting phenomena in the field of human affairs.

Social Studies of Computer Science

Computer science and computing technologies are products of phenomena that cannot be fully explained using only computational theories and concepts. One cannot understand *why* many parts of computer science are what they are without understanding their history because computer science is always produced in some philosophical, historical, and sociocultural framework. Any portrayal of computer science is a historically, culturally, and societally specific image. For instance, one cannot explain the design and diffusion of a programming language by referring to computational theories. Understanding the design and diffusion of concepts and technologies of computer science often requires understanding their history and the original motivations that provided the impetus for their development in the first place.

Historical methods can be utilized in order to retrospectively appreciate the reasons computer science has shaped as it has. Understanding the sociohistorical roots of computer science is important because sociohistorical understanding offers important "lessons learned"; sociohistorical understanding can be used to trace concepts and technologies to their origins in challenges, controversies, and discussions and thus helps one to judge which developments are contingent and which are necessary; it enables one to discover parallels and analogies to modern technology that can be used for reassess current and future developments; it makes it possible to link happenings and assess the reasons for those happenings; and instead of a gallery of "milestones", it portrays a picture of living computer science, with all its nuances, controversies, debates, and power struggles.

Whereas historical methods offer insight into the past of computing, ethnomethodological research offers insight into the present. Regardless of whether one accepts that computer science has rules or not, how computer science is *actually* done may be a different story altogether. Ethnomethodological studies of computer science aim at explicating the actual processes of constructing and managing knowledge in computer science—that is, the in situ or tacit methodology of computer science. Ethnomethodological research delves into the tacit, ambiguous, and complex practices of, for instance, creating, maintaining, using, abusing, proving, refuting, negotiating, accommodating, appropriating, and contextualizing knowledge.

Ethnomethodological approaches can expose how the philosophical, theoretical, conceptual, and methodological frameworks of computer science are created, maintained, and managed. For instance, ethnomethodological studies may reveal the manners in which new innovations are conceptualized by groups of computer scientists and other stakeholders; the processes through which conceptual consensus is achieved; how epistemologically subjective results in computer science are communicated, confirmed, adopted, objectified, and institutionalized into epistemologically objective facts; the knowledge transmission channels between academic and non-academic stakeholders; how computer science qua knowledge gives meaning to computer science qua activity and the results of computer science; how computer science qua activity generates computer science qua knowledge; and how both intra-scientific and extra-scientific contradictions are dealt with. In a word, ethnomethodological studies shed light on the very foundations of knowledge creation and maintenance in the computing disciplines.

After the Second World War systems engineering was developed as a response to the problems that arose when the complexity of systems exceeded the skills of a single person. Social studies of computer science must follow the same development lines. In explaining the ontological, epistemological, or methodological assumptions that the designs of a system may incorporate, social studies of computer science must be able to explain groups, not only individuals. That is, when one wants to explicate the intentions behind building a complex system, collective intentions and perhaps multiple intentions need to be catered for. The intentions and motivations for constructing the system may be heterogeneous and conflicting. Ethnographic methods offer a tool for, for instance, ethnomethodological studies, and it has been used in, for instance, understanding the processes and dynamics behind computer architecture and systems design.

Ethnographic methods are used in describing and interpreting social phenomena—such as ways of working, group relationships, communication, metaphors, and tropes. Researchers doing ethnographic research usually explore phenomena rather than test hypotheses, emphasize unstructured data instead of analytic categories, focus on detailed cases instead of large populations, and explicitly interpret the meanings and functions of human actions. Ethnographic methods can be utilized in order to examine patterns of production of scientific results, innovation, and standards in

computer science in their rich sociohistorical contexts; it can be utilized to study mechanisms of technological production, design, adoption, rejection, diffusion, non-diffusion, and so forth; and it can be used today to examine and document the early formation of a new discipline.

In social studies of computer science, ethnographic methods may help to elicit the perspectives, realities, and group interactions of computer scientists. Those perspectives, realities, and group interactions may reveal some aspects in the practices of computer scientists that have direct consequences on computer science qua knowledge. That is, ethnography can explain how computer scientists and other stakeholders create computer science and computing technology.

The crux of sociological, historical, anthropological, and philosophical research is not numbers and proofs. That is because the subjects of disciplines such as sociology, history, anthropology, and philosophy are often not well-structured, logical, coherent, or well-defined. This is not to say that quantitative methods or mixed-methods would not have their place in social studies of computer science; disciplines such as sociology and anthropology successfully utilize a wide array of quantitative methods too. Quantitative methods and qualitative methods are useful for different purposes. Social studies of computer science can be about exploring and interpreting, but they can also be about explaining and predicting.

Social studies of computer science can aim at revealing the nuances, ambiguities, and contradictions of single cases, but social studies of computer science can equally well aim at generalizing. Single cases can be informative about the *how*s and *why*s of technoscience. It is indeed common in the philosophy of science to draw from instrumental, historical case studies and to engage in analytical discourse about what those cases reveal about science in general.

The Promise of Social Studies of Computer Science

Disciplinary self-understanding is an essential part of a scientific discipline. Social studies of computer science, as portrayed in this thesis, is an essential part of the disciplinary self-understanding of computer science. It produces meta-knowledge about computer science. It helps computer scientists to understand the degree to which aspects of computer science rely on brute facts and the degree to which aspects of computer science rely on social constructs. It allows researchers (but does

not force anyone) to adopt a realist philosophy, while researchers can maintain that statements about reality are always socially constructed. It offers alternative angles on computer science qua knowledge and qua activity. It allows one to be critical about technological determinism, milestones, and machinery, and it helps one to understand the synthetic character of theories and technologies. Social studies of computer science may not help one to understand the raw functioning of technology, but it does allow one to see the big picture—the whys and hows of computer science.

Instead of being a source of radically new ideas about automatic computation, social studies of computer science is a matter of disciplinary honesty and self-understanding. Social studies of computer science might not provide many answers to the technical questions of computer science, such as "Can process *p* be automated?", "What is the most efficient implementation for automating process *p*?", or "What kind of interface for instrument *i* creates the least cognitive overhead?". The logico-mathematical, modeling-based, and design-oriented branches of computer science can deliver answers to those questions. It is not, however, clear if computer science can explain itself. It is not clear if computer science as a theory-methodology can explain computer science as a socially constructed phenomenon.

Social studies of computer science study those aspects of computer science that are beyond the reach of computational theories, design, or modeling. Social studies of computer science pose questions such as "Why is the form and function of *n* as it is?", "Which aspects of computer science are contingent and why?", "Why have some programming languages diffused better than others?", and "What are the reasons behind the dominance of the stored-program paradigm?". Those questions cannot be answered from solely logico-mathematical, design, or modeling points of view. The answers to those questions require, for instance, sociological, historical, anthropological, and philosophical approaches.

There are two essential arguments for the inclusion of social studies of computer science into the official body of knowledge of computer science. The first argument is a practical argument: Computer science is already being studied successfully from the perspectives of sociology, history, anthropology, and philosophy—therefore acknowledging the status of social studies of computer science is just a formal stamp of approval. The second argument is an ideological argument: Restricting possibly

fruitful viewpoints can be detrimental to science, and therefore acknowledging the status of social studies of computer science is an obligation, not a concession.

If one agrees that the acceptance of scientific statements is not solely based on the credibility of the statements themselves, but on their fit to the existing framework of science, society, and the physical world, one can argue that social studies of computer science can capture something unique about the mangle of computer science. That is, social studies of computer science can reveal how computer science is created and maintained. It can explain how the statements in computer scientists are externalized, objectified, internalized, and reified—that is, how computer scientists produce things that are afterwards perceived as something other than human products. It can explicate implicit assumptions, shared attitudes, and tacit knowledge. It can produce unique meta-knowledge of computer science. Meta-knowledge is an important aspect of understanding computer science because it can offer insight into even the most insightful theories of computer science.

Social studies of computer science can be considered to be studies which situate and investigate computer science in its social, historical, cultural, linguistic, political, economic, and other socially constructed frameworks. The aim of social studies of computer science is to offer a view of computer science in which not merely technological and theoretical aspects of computer science, but also sociocultural factors, are explicitly taken into consideration. Although social studies of computer science entails some constructionist features, as an umbrella term for different kinds of studies, social studies of computer science is philosophically uncommitted. Specific pieces of research within social studies of computer science can, of course, be committed to specific research philosophies.

Honing theories, practices, models, philosophy, or other aspects of computing can be done from two angles. First, one can study the *subjects* of computer science and hope to reveal something new about those subjects. Second, one can study *research in computer science* and hope to reveal something in the workings of computer science that reveals either new aspects about the subjects of computer science or about computer science. The former angle is called research and the latter angle might be called meta-research. The former, research, can be, for instance, studies which describe, analyze, revise, or refute the subjects of computer science (e.g., algorithms,

interfaces, or architectures). The latter, meta-research, can be, for instance, studies which describe how computer science researchers work and arguments which describe how computer science researchers should work if they want their science to flourish.

Much of social studies of computer science is meta-research. The aim of social studies of computer science can be to enhance or enrich the understanding of subjects of computer science, but the aim can also be to reveal something about computer science as activity or to refine the frameworks of computer science. The aim of social studies of computer science is *not* refinement of, say, sociological, historical, anthropological, or philosophical theories. Social studies of computer science *need not* clash with existing body of computer science knowledge, but it offers alternative viewpoints to the topics of computer science, portrays reasons and causes in computer science, and builds meta-knowledge about computer science. In this sense, social studies of computer science *may* question the legitimacy and foundations of some aspects of computer science.

It is often difficult to find a suitable category in the ACM Computing Classification System for the existing pieces of research that can be considered to be social studies of computer science. There is no category for studies that contribute to computer science and that entail a sociocultural perspective or produce meta-knowledge about computer science. Because there already are a good variety of studies that borrow tools from the social sciences and humanities, that produce unique socioculturally sensible information about the knowledge as well as the construction of knowledge in computer science, and that clearly belong to computer science, there is a need for category K.9 (*Social Studies of Computer Science*) in the ACM classification system. Tagging a publication K.9 would indicate the choice of a sociological, historical, anthropological, philosophical, or other social sciences or humanities-based perspective and the production of meta-knowledge about computer science. The subcategories K.9.* can denote the types of research more specifically.

Epilogue

My research questions, presented in Chapter One, were

> *Question 1*: Is there a need to broaden computer science with perspectives from disciplines such as sociology, history, anthropology, or philosophy?
>
> *Question 2*: If there is a need to broaden computer science, what kind of arguments support such an extension?
>
> *Question 3*: What consequences may a broad, socioculturally receptive view have on computer science?

My answer to the first question is yes, and I outline the arguments for Question 2 below. Computer science is a broad discipline, and many aspects of computer science are under dispute. I have discussed debates about, for instance, the content, form, methodologies, methods, epistemological status of results, inference patterns, semantics, standards, focus, education, modes of argumentation, logic of justification, and conventions for settling scientific disputes in computer science. I have also discussed the ways in which computer science and technology develop, and argued that a wide array of factors influence technoscientific development in computer science. Those factors include not only technoscientific, but also non-technical and non-scientific factors such as societal, economic, cultural, institutional, ideological, political, philosophical, and ethical factors.

I have argued that neither the form and practices in current computer science nor the elements of development of computer science are fully explainable using the conceptual-theoretical framework of computation and technology. If it is important for computer science that there is an understanding of the ways in which computer scientists create, maintain, and manage knowledge in computing and an understanding of how theories and technologies have developed and develop, then computer scientists need methods that are rooted in disciplines such as sociology, history, anthropology, and philosophy.

As a consequence of acknowledging a socioculturally receptive view; regardless of whether the methods of sociology, history, anthropology, and philosophy are already a part of computer science or not; their role in offering disciplinary insight must be acknowledged (Question 3). This is best done by adding a new category to the ACM

Computing Classification System: K.9 (*Social Studies of Computer Science*), which would be an indication of research that aims at contributing to computer science by producing unique socioculturally sensible information about knowledge and knowledge-construction in computer science.

I have offered clear answers to all my three research questions. Simplified: (1) There is a need to extend computer science, because (2) profound disciplinary self-understanding requires utilizing socioculturally receptive methods of inquiry. Consequently, in order to officially acknowledge the importance of meta-knowledge of computer science (3) social studies of computer science should be acknowledged as a field of study in the ACM Computing Classification System.

My research was theoretical and it followed the tradition of philosophical hermeneutics. I have formed a conceptual framework, traced some contingent elements in the history of a number of innovations and changes in computing, and situated those historical elements in my conceptual framework. The literature in this thesis is typical of science and technology studies, and the majority of my sources deal with central issues, or have been in a central position, in the development of computer science. I have coupled my critical interpretation of the history and development of computing with an analysis of those central issues.

My argument is in line with the current research trends in the field of science and technology studies. The consequences I propose are congruent with the developments in, for instance, the sociology of scientific knowledge, the philosophy of science, the philosophy of technology, and the history of technology. However, during the course of this research topics have arisen that deserve more attention. I outline some of the questions below I believe are most central to my argument.

Firstly, a more thorough review of existing research in social studies of computer science would be useful, and it might lead to a classification or a taxonomy of different kinds of studies of social studies of computer science. It might also be interesting to see how those studies have been classified in the ACM Computing Classification System as it stands.

Secondly, a wider analysis of research approaches that could be utilized in developing the disciplinary identity and self-understanding of computer science would be useful. In this thesis I have taken only a few examples from qualitative approaches,

but the possible contribution and limitations of a wider array of qualitative and quantitative methods, as well as an analysis of mixed-methods studies, would strengthen the prospects of social studies of computer science by building a strong methodological basis.

Thirdly, the philosophy of computer science needs development. Currently the philosophy of computer science is seriously underdeveloped. The philosophy of computer science is *not* the philosophy of artificial intelligence, the philosophy of computing, or the philosophy of science. There is a need for a philosophical framework that focuses specifically on computer science and that can cater to the eclecticism of computer science. There is a need for a philosophy *of* computer science that can accommodate the theoretical, empirical, design-based, and metatheoretical strands of computer science.

References

Abu-Lughod, Lila (1991) Writing Against Culture. In Richard G. Fox (ed.) (1991): "Recapturing Anthropology. Working in the Present": pp. 137-60.

Ackoff, Russell L. (1978) The Art of Problem Solving . John Wiley & Sons, Inc.: New York.

Agar, M.H. (2001) Ethnography. In Smelser & Baltes (eds.) (2001): "International Encyclopedia of the Social & Behavioral Sciences Vol.7": pp. 4857-4862.

Agre, Philip E. (1995) From High Tech to Human Tech: Empowerment, Measurement, and Social Studies of Computing. Computer Supported Cooperative Work (CSCW) 3(2): pp. 167-195.

AHD (2004) The American Heritage Dictionary of English Language Online (2004). Available at http://www.bartleby.com/61

Alavi, Maryam; Carlson, Patricia (1992) A Review of MIS Research and Disciplinary Development. Journal of Management Information Systems 8(4): pp. 45-62.

Alt, Franz L. (1962) Fifteen Years ACM. Communications of the ACM 5(6): pp. 300-307.

Anderson, David P.; Cobb, Jeff; Korpela, Eric; Lebofsky, Matt; Werthimer, Dan (2002) SETI@home: An Experiment in Public-Resource Computing. Communications of the ACM 45(11): pp. 56-61.

Arden, Bruce W. (ed.) (1980) What Can Be Automated?: Computer Science and Engineering Research Study . MIT Press: Cambridge, MA, USA.

Argamon, Shlomo; Olsen, Mark (2006) Toward Meaningful Computing. Communications of the ACM 49(4): pp. 33-35.

Aris, John (2000) Inventing Systems Engineering. IEEE Annals of the History of Computing 22(3): pp. 4-15.

Arnold, Lola M. (1989) Item Selection From Menus: The Influence of Menu Organization, Query Interpretation, and Programming Experience on Selection Strategies. SIGCHI Bulletin 21(1): pp. 81-85.

Arora, Sanjeev; Chazelle, Bernard (2005) Is the Thrill Gone?. Communications of the ACM 48(8): pp. 31-33.

Arthur, W. Brian (1999) Competing Technologies and Economic Prediction. In MacKenzie, Donald; Wajcman, Judy (eds.) (1999): "The Social Shaping of Technology": pp. 106-112.

Ascher, Marcia; Ascher, Robert (1981) Mathematics of the Incas: Code of the Quipu . Dover: Mineola, NY, USA.

Ascher, Marcia (2002) Mathematics Elsewhere: An Exploration of Ideas Across Cultures . Princeton University Press: Princeton, New Jersey, USA.

Asner, Glen R. (2004) The Linear Model, the U.S. Department of Defense, and the Golden Age of Industrial Research. In Grandin et al. (eds.) (2004): "The Science-Industry Nexus: History, Policy, Implications": pp. 3-30.

Aspray, William (2000) Was Early Entry a Competitive Advantage? US Universities That Entered Computing in the 1940s. IEEE Annals of the History of Computing 22(3): pp. 42-87.

Atchison, William F.; Conte, Samuel D.; Hamblen, John W.; Hull, Thomas E.; Keenan, Thomas A.; Kehl, William B.; McCluskey, Edward J.; Navarro, Silvio O.; Rheinboldt, Werner C.; Schweppe, Earl J.; Viavant William; Young, David M. (1968) Curriculum '68, Recommendations for Academic Programs in Computer Science. Communications of the ACM 11(3): pp. 151-197.

Atkinson, Paul; Hammersley, Martyn (1994) Ethnography and Participant Observation. In Denzin, Norman K.; Lincoln, Yvonna S. (eds.) (1994): "Handbook of Qualitative Research": pp. 248-261.

Austing, Richard H.; Barnes, Bruce H.; Engel, Gerald L. (1977) A Survey of the Literature in Computer Science Education Since Curriculum '68. Communications of the ACM 20(1): pp. 13-21.

Austing, Richard H.; Barnes, Bruce H.; Bonnette, Della T.; Engel, Gerald L.; Stokes, Gordon (1977b) Curriculum Recommendations for the Undergraduate Program in Computer Science: A Working Report of the ACM Committee on Curriculum in Computer Science. ACM SIGCSE Bulletin 9(2): pp. 1-16.

Austing, Richard H.; Barnes, Bruce H.; Bonnette, Della T.; Engel, Gerald L.; Stokes, Gordon (eds.) (1979) Curriculum '78: Recommendations for the Undergraduate Program in Computer Science. Communications of the ACM 22(3): pp. 147-166.

Avison, David; Lau, Francis; Myers, Michael; Nielsen, Peter Axel (1999) Action Research. Communications of the ACM 42(1): pp. 94-97.

Backus, J. W.; Bauer, F. L.; Green, J.; Katz, C.; McCarthy, J.; Perlis, A. J.; Rutishauser, H.; Samelson, K.; Vauquois, B.; Wegstein, J. H; van Wijngaarden, A.; Woodger, M.; Nauer, P. (1963) Revised report on the algorithm language ALGOL 60 . Communications of the ACM 6(1): pp. 1-17.

Backus, John W. (1959) The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. (UNESCO) Information Processing: Proceedings of the International Conference on Information Processing. June 15-20, Paris, France: pp. 125-132.

Backus, John (1978) Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. Communications of the ACM 21(8): pp. 613-641.

Backus, John (1981) The History of Fortran I, II, and III. In Wexelblat, 1981: "History of Programming Languages": pp. 25-45.

Baecker, Ronald M.; Grudin, Jonathan; Buxton, William A.S.; Greenberg, Saul (eds.) (1995) Readings in Human-Computer Interaction: Toward the Year 2000 (2nd ed.). Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA.

Bamford, Greg (1993) Popper's Explications of Ad Hocness: Circularity, Empirical Content, and Scientific Practice. British Journal for the Philosophy of Science 44(2): pp. 335-355.

Bar-El, Hagai; Choukri, Hamid; Naccache, David; Tunstall, Michael; Whelan, Claire (2004) The Sorcerer's Apprentice Guide to Fault Attacks. (Proceedings of the) Workshop on Fault Detection and Tolerance in Cryptography. June 30, 2004, Florence, Italy: pp. N/A. Available at Cryptology ePrint Archive: Report 2004/100.

Barnes, Barry; Bloor, David; Henry, John (1996) Scientific Knowledge: A Sociological Analysis . The University of Chicago Press: London, UK.

Baskerville, Richard; Stage, Jan; DeGross, Janice I. (eds.) (2000) Organizational and Social Perspectives on Information Technology . Kluwer Academic Publishers: Norwell, Mass., USA.

Baskerville, Richard L. (1999) Investigating Information Systems with Action Research. Communications of AIS 2(article 19): pp. 2-31.

Bauer, F.L.; Wössner, H. (1972) The "Plankalkül" of Konrad Zuse: A Forerunner of Today's Programming Languages. Communications of the ACM 15(7): pp. 678-685.

Bauer, Walter F.; Juncosa, Mario L.; Perlis, Alan J. (1959) ACM Publication Policies and Plans. Journal of the ACM 6(2): pp. 121-122.

Bemer, Robert W. (1984) Computing Prior to FORTRAN. Annals of the History of Computing 6(1): pp. 16-18.

Ben-Ari, Mordechai (2001) Constructivism in Computer Science Education. Journal of Computers in Mathematics and Science Teaching 20(1): pp. 45-73.

Berger, Peter L.; Luckmann, Thomas (1966) The Social Construction of Reality: A Treatise in the Sociology of Knowledge . Allen Lane: London, UK.

Bergin, Thomas J., Jr.; Gibson, Richard G., Jr. (eds.) (1996) History of Programming Languages II . ACM Press: New York, NY, USA.

Berkeley, George (1971 [1734]) A Treatise Concerning the Principles of Human Knowledge . The Scolar Press ltd.: Menston, Yorkshire, England.

Berman, Gennady P.; Doolen, Gary D.; Mainieri, Ronnie; Tsifrinovich, Vladimir I. (1998) Introduction to Quantum Computers . World Scientific: Singapore.

Bernard, H. Russell (1995) Research Methods in Anthropology: Qualitative and Quantitative Approaches (2nd ed.). AltaMira Press: Oxford, England.

Beynon, Meurig; Russ, Steve (1995 [1994]) Empirical Modelling of Requirements . Warwick University, Department of Computer Science Research Reports CS-RR-277: Warwick, UK.

Bhatt, Chetan (2004) Doing a Dissertation. In Seale, Clive (ed.) (2004): "Researching Society and Culture (2nd ed.)": pp. 410-430.

Bijker, Wiebe E.; Law, John (eds.) (1992) Shaping Technology / Building Society. Studies in Sociotechnical Change . MIT Press: Cambridge, Mass., USA.

Bijker, Wiebe E.; Hughes, Thomas P.; Pinch, Trevor J. (eds.) (1987) The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology . MIT Press: Cambridge, MA, USA.

Bijker, Wiebe E. (1992) The Social Construction of Fluorescent Lighting, or How an Artifact Was Invented in Its Diffusion Stage. In Bijker and Law (eds.) (1992): "Shaping Technology / Building Society: Studies in Sociotechnical Change": pp. 75-101.

Bimber, Bruce (1994) Three Faces of Technological Determinism. In Smith, Merritt Roe and Marx, Leo (eds.) (1994): "Does Technology Drive History? The Dilemma of Technological Determinism": pp. 80-100.

Bloor, David (1971) Two Paradigms For Scientific Knowledge?. Science Studies 1971(1): pp. 101-115.

Bloor, David (1976) Knowledge and Social Imagery . Routledge & Kegan Paul: London.

Bloor, David (1996) Idealism and the Sociology of Knowledge. Social Studies of Science 26(4): pp. 839-856.

Bobrow, Daniel G.; Hayes, Patrick J. (1985) Artificial Intelligence - Where Are We?. Artificial Intelligence 25(1985): pp. 375-415.

Borba, Marcelo C. (1990) Ethnomathematics and Education. For the Learning of Mathematics 10(1): pp. 39-43.

Bouillon, Hardy (1998) Book review: "Gunnar Andersson, Criticism and the History of Science. Kuhn's, Lakatos's, and Feyerabend's Criticisms of Critical Rationalism". Journal for General Philosophy of Science 29(1998): pp. 133-135.

Bowker, Geof (1993) How to be Universal: Some Cybernetic Strategies, 1943-70. Social Studies of Science 23(1): pp. 107-127.

Bowles, Mark D. (1996) U.S. Technological Enthusiasm and the British Technological Skepticism in the Age of the Analog Brain. IEEE Annals of the History of Computing 18(4): pp. 5-15.

Bowyer, Adrian (2006) Who Do You Trust? (Letter to CACM Forum). Communications of the ACM 49(5): pp. 13.

Bremer, Manuel E. (2003) Do Logical Truths Carry Information?. Minds and Machines 13(2003): pp. 567-575.

Brent, Edward; Thompson, Alan; Vale, Whitley (2000) A Computational Approach to Sociological Explanations. Social Science Computer Review 18(2): pp. 223-235.

Brey, Philip (2003) The Social Ontology of Virtual Environments. American Journal of Economics and Sociology 62(1): pp. 269-282.

Bright, Herbert S. (1984) Early FORTRAN User Experience. Annals of the History of Computing 6(1): pp. 28-30.

Brooks, Frederick P., Jr. (1975) The Mythical Man-Month: Essays on Software Engineering . Addison-Wesley: Reading, Mass., USA.

Brooks, Frederick P., Jr. (1996) The Computer Scientist as Toolsmith II. Communications of the ACM 39(3): pp. 61-68.

Brookshear, J. Glenn (2003) Computer Science: an overview (7th edition). Addison-Wesley: New York.

Brumann, Christoph (1999) Writing for Culture: Why a Successful Concept Should Not Be Discarded. Current Anthropology 40(Supplement/February): pp. S1-S27.

Bucchi, Massimiano (2004) Science in Society: An Introduction to Social Studies of Science . Routledge: London and New York.

Bunge, Mario (1979 [2003]) Philosophical Inputs and Outputs of Technology. In Scharff & Dusek (eds.) (2003): "Philosophy of Technology: The Technological Condition": pp. 172-181.

Bunge, Mario (1998) Philosophy of Science (revised ed.) Vol. 2: From Explanation to Justification . Transaction Publishers: New Brunswick .

Burnette, Charles (2004) Forum: Put Cognitive Models in CS and Its Curricula. Communications of the ACM 47(2): pp. 12.

Bush, Vannevar (1945) As We May Think. The Atlantic Monthly July(s.n.): pp. n.a..

"C.J.A." (1967) In Defense of Programmers. Datamation (Letters to the Editor) 13(9): pp. 15.

Campbell-Kelly, Martin; Aspray, William (2004) Computer: A History of the Information Machine (2nd ed.). Westview Press: Oxford, UK.

Campbell-Kelly, Martin; Williams, Michael R. (1985 [1946]) The Moore School Lectures . The MIT Press: Cambridge, Mass., USA.

Carr, John W. Jr. (1957) Inaugural Presidential Address. Journal of the ACM 4(1): pp. 5-7.

Castel, Felipe (2002) Ontological Computing. Communications of the ACM 45(2): pp. 29-30.

Castells, Manuel (1996) The Information Age: Economy, Society and Culture, Volume I: The Rise of the Network Society (2nd ed.). Blackwell Publishing: UK.

Castells, Manuel (1997) The Information Age: Economy, Society and Culture, Volume II: The Power of Identity (2nd ed.). Blackwell Publishing: UK.

Castells, Manuel (1998) The Information Age: Economy, Society and Culture, Volume III: End of Millennium (2nd ed.). Blackwell Publishing: UK.

Castells, Manuel (2000) Materials for an Exploratory Theory of the Network Society. The British Journal of Sociology 51(1): pp. 5-24.

Castells, Manuel (2001) The Internet Galaxy: Reflections on the Internet, Business, and Society . Oxford Press: Oxford, Great Britain.

Ceruzzi, Paul (1997) Crossing the Divide: Architectural Issues and the Emergence of the Stored Program Computer, 1935-1955. IEEE Annals of the History of Computing 19(1): pp. 5-12.

Ceruzzi, Paul (1999 [1996]) Inventing Personal Computer. In MacKenzie, Donald; Wajcman, Judy: "The Social Shaping of Technology": pp. 64-86.

Chalmers, Alan F. (1976 [1999]) What is This Thing Called Science? (3rd. edition). University of Queensland Press: Queensland, Australia.

Charniak, Eugene; McDermott, Drew (1985) Introduction to Artificial Intelligence . Addison-Wesley: Reading, Mass., USA.

Choudrie, Jyoti; Dwivedi, Yogesh Kumar (2005) Investigating the Research Approaches for Examining Technology Adoption Issues. Journal of Research Practice 1(1): pp. Article D1.

Civin, Michael (2000) Male, Female, Email: The Struggle for Relatedness in a Paranoid Society . Other Press: New York, NY, USA.

Clayman, S.E. (2001) Ethnomethodology: General. In Smelser, Neil J.; Baltes, Paul B. (eds.) (2001): "International Encyclopedia of the Social & Behavioral Sciences vol.7": pp. 4865-4870.

Cleland, Carol E. (2001) Recipes, Algorithms, and Programs. Minds and Machines 11(2): pp. 219-237.

Clements, Alan (2000 [1985]) The Principles of Computer Hardware (3rd ed., uncorrected preliminary edition). Oxford University Press: New York, NY, USA.

Cockton, Gilbert (2004) Value-Centred HCI. (ACM) Proceedings of the NordiCHI '04. October 23rd-27th 2004, Tampere, Finland: pp. 149-160.

Cohen, Morris R.; Nagel, Ernest (1934) An Introduction to Logic and Scientific Method . Hartcourt, Brace & World: New York, USA.

Cohen, I. Bernard (1998) Howard Aiken on the Number of Computers Needed for the Nation. IEEE Annals of the History of Computing 20(3): pp. 27-32.

Colburn, Timothy R. (2000) Philosophy and Computer Science . M.E. Sharpe: Armonk, NY, USA.

Collins, Francis S.; Morgan, Michael; Patrinos, Aristides (2003) The Human Genome Project: Lessons from Large-Scale Biology. Science 300(5617): pp. 286-290.

Comrie, Leslie John (1944) Recent Progress in Scientific Computing. Journal of Scientific Instruments 21(8): pp. 129-135.

Conklin, Harold C. (1968) Ethnography. In Sills, David L. (ed.) (1968): "International Encyclopedia of the Social Sciences vol.5": pp. 172-178.

Conte, S.D.; Hamblen, John W.; Kehl, William B.; Navarro, Silvio O.; Rheinboldt, Werner C.; Young, David M.; Atchinson William F. (1965) An Undergraduate Program in Computer Science - Preliminary Recommendations. Communications of the ACM 8(9): pp. 543-552.

Copeland, B. Jack; Proudfoot, Diane (2000) What Turing Did after He Invented the Universal Turing Machine. Journal of Logic, Language, and Information 9(4): pp. 491-509.

Copeland, B. Jack; Sylvan, Richard (1999) Beyond the Universal Turing Machine. Australasian Journal of Philosophy 77(1): pp. 46-67.

Copeland, B. Jack (1997) The Broad Conception of Computation. American Behavioral Scientist 40(6): pp. 690-716.

Copeland, B. Jack (2002) Hypercomputation. Minds and Machines 12(4): pp. 461-502.

Copeland, B. Jack (2004) Unfair to Aiken. IEEE Annals of the History of Computing 26(4): pp. 35-37.

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. (1998) Introduction to Algorithms . The MIT Press: Cambridge, Masschusetts.

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. (2001) Introduction to Algorithms (Russian translation) . MIT Press: Moscow, Russia.

Correll, Quentin (1958) Letters to the Editor. Communications of the ACM 1(7): pp. 2.

Cortada, James W. (1993) Before the Computer: IBM, NCR, Burroughs, and the Industry They Created, 1865-1956 . Princeton University Press: Princeton, N.J., USA. (Cited in Ceruzzi, 1997; original unavailable)

Couvalis, George (1997) The Philosophy of Science: Science and Objectivity . Sage Publications: London, UK.

Crabtree, Andy; Nichols, David M.; O'Brien, Jon; Rouncefield, Mark; Twidale, Michael B. (2000) Ethnomethodologically Informed Ethnography and Information System Design. Journal of the American Society for Information Science 51(7): pp. 666-682.

Crabtree, Andy (2004) Taking Technomethodology Seriously: Hybrid Change in the Ethnomethodology-Design Relationship. European Journal of Information Systems 13(3): pp. 195-209.

Croarken, Mary G. (1992) The Emergence of Computing Science Research and Teaching at Cambridge, 1936-1949. IEEE Annals of the History of Computing 14(4): pp. 10-15.

Croarken, Mary (1993) The Beginnings of the Manchester Computer Phenomenon: People and Influences. IEEE Annals of the History of Computing 15(3): pp. 9-16.

Crowcroft, Jon (2005) On the Nature of Computing. Communications of the ACM 48(2): pp. 19-20.

Cusumano, Michael A. (2004) Reflections on Free and Open Software. Communications of the ACM 47(10): pp. 25-27.

Datamation (1962) Editor's Readout: The Certified Public Programmer. Datamation 8(3): pp. 23-24.

Davis, Ronald L. (1977) Recommended Mathematical Topics for Computer Science Majors. ACM SIGCSE Bulletin 9(3): pp. 51-55.

Day, James (1993) Theoretical Research. In Nickerson, Eileen (ed.) (1993): "The Dissertation Handbook": pp. 76-87.

De Meuter, Wolfgang; Costanza, Pascal; Devos, Martine; Thomas, Dave (2002) Feyerabend: Redefining Computing. Lecture Notes in Computer Science (Springer-Verlag) 2548(Jan 2002): pp. 197-202.

De Millo, Richard A.; Lipton, Richard J.; Perlis, Alan J. (1979) Social Processes and Proofs of Theorems and Programs. Communications of the ACM 22(5): pp. 271-280.

Denning, Peter J. & Metcalfe, Robert M. (eds.) (1997) Beyond Calculation: The Next Fifty Years of Computing . Springer-Verlag: New York, NY, USA.

Denning, Peter J. (Chairman); Comer, Douglas E.; Gries, David; Mulder, Michael C.; Tucker, Allen; Turner, A. Joe; Young, Paul R. (1989) Computing as a Discipline. Communications of the ACM 32(1): pp. 9-23.

Denning, Peter J.; Chang, Carl (chairmen) and CC2001 Task Force (2001) Computing Curricula 2001 . IEEE and ACM. Available on-line at www.computer.org: .

Denning, Peter J. (1980) On Folk Theorems, and Folk Myths. Communications of the ACM 23(9): pp. 493-494.

Denning, Peter J. (1980b) What is Experimental Computer Science?. Communications of the ACM 23(10): pp. 534-544.

Denning, Peter J. (1985) The Science of Computing: What is computer science?. American Scientist 73(Jan.-Feb.): pp. 16-19.

Denning, Peter J. (1991) Computing, Applications, and Computational Science. Communications of the ACM 34(10): pp. 129-131.

Denning, Peter J. (1992) Educating a New Engineer. Communications of the ACM 35(12): pp. 82-97.

Denning, Peter J. (2003) Great Principles of Computing. Communications of the ACM 46(11): pp. 15-20.

Denning, Peter J. (2004) The Field of Programmers Myth. Communications of the ACM 47(7): pp. 15-20.

Denning, Peter (2004b) Forum: Author Responds. Communications of the ACM 47(2): pp. 12-13.

Denning, Peter (2005) Look Beyond Abstraction to Define Computing (Letter to CACM Forum). Communications of the ACM 48(5): pp. 11-12.

Denzin, Norman K.; Lincoln, Yvonna S. (eds.) (2005) The SAGE Handbook of Qualitative Research (3rd ed.). SAGE Publications: London, UK.

Denzin, Norman K.; Lincoln, Yvonna S. (eds.) (1994) Handbook of Qualitative Research . SAGE: London.

Dertouzos, Michael L. (2001 [2002]) The Unfinished Revolution (Perennial paperback ed. 2002). Perennial / HarperCollins: New York, NY, USA.

Desrosières, Alain (1996) Statistical Traditions: an Obstacle to International Comparisons?. In Hantrais, Linda; Mangen, Steen (eds.) (1996): "Cross-national Research Methods in the Social Sciences": pp. 17-27.

Dijkstra, Edsger W. (1968) Go To Statement Considered Harmful. Communications of the ACM 11(3): pp. 147-148.

Dijkstra, Edsger W. (1968b) A Constructive Approach to the Problem of Program Correctness. BIT 8(1968): pp. 174-186.

Dijkstra, Edsger W. (1972) The Humble Programmer. Communications of the ACM 15(10): pp. 859-866.

Dijkstra, Edsger W. (1974) Programming as a Discipline of Mathematical Nature. American Mathematical Monthly 81(June-July): pp. 608-612.

Dijkstra, Edsger W. (1975) Correctness Concerns And, Among Other Things, Why They Are Resented. (ACM) Proceedings of the international conference on Reliable software. April 21 - 23, 1975, Los Angeles, California, USA: pp. 546-550.

Dijkstra, Edsger W. (1975b) How do we tell truths that might hurt?. In Dijkstra, Edsger W. (1982): "Selected Writings on Computing: A Personal Perspective": pp. 129-131.

Dijkstra, Edsger W. (1982) Selected Writings on Computing: A Personal Perspective . Springer-Verlag: Germany.

Dijkstra, Edsger W. (1986) On a Cultural Gap. The Mathematical Intelligencer 8(1): pp. 48-52.

Dijkstra, Edsger W. (1987) Mathematicians and Computing Scientists: The Cultural Gap. Abacus 4(4): pp. 26-31.

Dijkstra, Edsger W. (1989) On the Cruelty of Really Teaching Computer Science. Communications of the ACM 32(12): pp. 1398-1404.

Dijkstra, Edsger W. (1999) Computing Science: Achievements and Challenges. ACM SIGAPP Applied Computing Review 7(2): pp. 2-9.

Dijkstra, Edsger W. (2001) The End of Computing Science?. Communications of the ACM 44(3): pp. 92.

Dodig-Crnkovic, Gordana (2002) Scientific Methods in Computer Science. (Promote IT 2002) Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden. April 22-24, Billingehus, Skövde, Sweden: pp. 1D1.

Dodig-Crnkovic, Gordana (2003) Shifting the Paradigm of Philosophy of Science: Philosophy of Information and a New Renaissance. Minds and Machines 13(2003): pp. 521-536.

Duhem, Pierre (1977 [1914]) The Aim and Structure of Physical Theory (2nd edition, 3rd reprint). Atheneum: New York, USA.

Dumas, Joseph S.; Redish, Janice C. (1999) A Practical Guide to Usability Testing (revised ed.) . Intellect: Portland, OR, USA.

Easton, Thomas A. (2006) Beyond the Algorithmization of the Sciences. Communications of the ACM 49(5): pp. 31-33.

Eaton, Robert J. (2004) The Internet Has Transformed the Economy. In Egendorf (ed.) (2004): "The Information Revolution: Opposing Viewpoints": pp. 38-43.

Eckert, John Presper (1946) A Parallel Channel Computing Machine. In Campbell-Kelly, Martin; Williams, Michael R. (eds.) (1985): "The Moore School Lectures": pp. 527-542.

Edson, Joanne; Greenstadt, John (editor-in-chief); Greenwald, Irwin; Jones, Fletcher R.; Wagner, Frank V. (eds.) (1956) SHARE Reference Manual for the IBM 704 . Loose-leaf copy: IBM.

Egan, L.G. (1976) Closing the "Gap" Between the University and Industry in Computer Science. ACM SIGCSE Bulletin 8(4): pp. 19-25.

Egendorf, Laura K (ed.) (2004) The Information Revolution: Opposing Viewpoints . Greenhaven Press: USA.

Eglash, Ron (1997) Bamana Sand Divination: Recursion in Ethnomathematics. American Anthropologist 99(1): pp. 112-122.

Eglash, Ron (1999) African Fractals: Modern Computing and Indigenous Design . Rutgers University Press: New Jersey, USA.

Eisenberg, Michael (2003) Creating a Computer Science Canon: a Course of "Classic" Readings in Computer Science. (ACM) Proceedings of the SIGCSE '03 Conference. February 19-23, Reno, Nevada, USA: pp. 336-340.

Encyclopædia Britannica Online (2004). Available at www.eb.com

Ensmenger, Nathan L. (2001) The 'Question of Professionalism' in the Computer Fields. IEEE Annals of the History of Computing 23(4): pp. 56-74.

Farrell, Robert P. (2001) Feyerabend's Metaphysics: Process-Realism, or Voluntarist-Idealism?. Journal for General Philosophy of Science 32(2001): pp. 351-369.

Feferman, Solomon (ed.) (1986) Kurt Gödel: Collected Works, Volume I, Publications 1929-1936 . Oxford University Press: New York, USA.

Ferry, Georgina (2003) A Computer Called LEO: Lyons Teashops and the World's First Office Computer . Fourth Estate: London, UK.

Fetterman, David M. (2004) Ethnography. In Lewis-Beck et al. (eds.) (2004): "The SAGE Encyclopedia of Social Science Research Methods Vol.1": pp. 328-332.

Fetzer, James H. (1988) Program Verification: The Very Idea. Communications of the ACM 31(9): pp. 1048-1063.

Feyerabend, Paul (1970) Consolations for the Specialist. In Lakatos and Musgrave (eds.): "Criticism and the Growth of Knowledge": pp. 197-230.

Feyerabend, Paul (1975) How to Defend Society Against Science. Radical Philosophy 11 (Summer 1975): pp. 3-8.

Feyerabend, Paul (1987) Farewell to Reason . Verso: London, Great Britain.

Feyerabend, Paul (1993 [1975]) Against Method (3rd. edition). Verso: New York.

Feyerabend, Paul (1994) Art as a Product of Nature as a Work of Art. World Futures: The Journal of General Evolution 40(1994): pp. 87-100.

Feyerabend, Paul (1995) Killing Time: The Autobiography of Paul Feyerabend . The University of Chicago Press: Chicago and London.

Finerman, Aaron (1970) Comment on Amsterdam IFIP Conference on Computer Education. Communications of the ACM 13(11): pp. 702.

Firestone, Joseph M.; McElroy, Mark W. (2003) Key Issues in the New Knowledge Management . Elsevier Science: Burlington, MA, USA.

Fiske, John (1994) Audiencing: Cultural Practice and Cultural Studies. In Denzin, Norman K.; Lincoln, Yvonna S. (eds.) (1994): "Handbook of Qualitative Research": pp. 189-198.

Flamm, Kenneth (1988) Creating the Computer: Government, Industry, and High Technology . Brookings Institution: Washington, D.C., USA.

Fleck, James (1999) Learning by Trying: the Implementation of Configurational Technology. In MacKenzie, Donald; Wajcman, Judy (eds.) (1999): "The Social Shaping of Technology": pp. 244-257.

Florida, Richard (2003) The Rise of the Creative Class: And How It's Transforming Work, Leisure, Community and Everyday Life . Basic Books: New York, USA.

Floridi, Luciano (1999) Philosophy and Computing: an Introduction . Routledge: London.

Floridi, Luciano (2002) What is the Philosophy of Information?. Metaphilosophy 33(1-2): pp. 123-145.

Floridi, Luciano (2003) Two Approaches to the Philosophy of Information. Minds and Machines 13(2003): pp. 459-469.

Floridi, Luciano (ed.) (2004) The Blackwell Guide to the Philosophy of Computing and Information . Blackwell Publishing: Cornwall, UK.

Floridi, Luciano (2004b) Information. In Floridi, Luciano (ed.) (2004): "The Blackwell Guide to the Philosophy of Computing and Information": pp. 40-61.

Floyd, Robert W. (1979) The Paradigms of Programming. Communications of the ACM 22(8): pp. 455-460.

Forsythe, George E. (1967) A University's Educational Program in Computer Science. Communications of the ACM 10(1): pp. 3-11.

Forsythe, George E. (1968) What to Do Till the Computer Scientist Comes. American Mathematical Monthly 75(May 1968): pp. 454-461.

Forsythe, George (1969) Computer Science and Education. (IFIP) Proceedings of IFIP Congress 1968. August 5th-10th 1968, Edinburgh, UK: pp. 92-106 (Volume 2).

Forsythe, Diana E. (1993) Engineering Knowledge: The Construction of Knowledge in Artificial Intelligence. Social Studies of Science 23(3): pp. 445-477.

Fox, Richard G. (ed.) (1991) Recapturing Anthropology: Working in the Present . School of American Research: Santa Fe., New Mexico, USA.

Frankston, Bob (1997) Beyond Limits. In Denning, Peter J; Metcalfe, Robert M. (eds.) (1997): "Beyond Calculation: The Next Fifty Years of Computing": pp. 43-57.

Freeman, Peter; Hart, David (2004) A Science of Design for Software-Intensive Systems. Communications of the ACM 47(8): pp. 19-21.

Freksa, Christian; Jantzen, Matthias; Valk, Rüdiger (eds.) (1997) Foundations of Computer Science: Potential-Theory-Cognition (Lecture Notes in Computer Science 1337). Spinger-Verlag: Berlin, Germany.

Frenkel, Karen A. (1988) The Art And Science of Visualizing Data. Communications of the ACM 31(2): pp. 110-122.

Fuller, Steve (2003) Kuhn vs. Popper: The Struggle for the Soul of Science . Icon Books: UK.

Gabora, Liane (1995) Meme and Variations: A Computational Model of Cultural Evolution. In Nadel, Lynn; Stein, Daniel L (eds.) (1995): "1993 Lectures in Complex Systems": pp. 471-485.

Gadamer, Hans-Georg (1976) Philosophical Hermeneutics (ed. Linge, David E.) . University of California Press: Berkeley, CA, USA.

Gadamer, Hans-Georg (1982) Truth and Method . Crossroad: New York, NY, USA.

Gal-Ezer, Judith; Harel, David (1998) What (Else) Should CS Educators Know?. Communications of the ACM 41(9): pp. 77-84.

Galler, Bernard A. (1974) Distinction of Computer Science. Communications of the ACM 17(6): pp. 300.

Galliers, Robert D.; Land, Frank F. (1987) Choosing Appropriate Information Systems Research Methodologies. Communications of the ACM 30(11): pp. 901-902.

Gardner, Howard (1993) Frames of Mind (2nd edition). Fontana Press: Glasgow, Great Britain.

Garfinkel, Harold (1967) Studies in Ethnomethodology . Prentice Hall: Englewood Cliffs, NJ, USA.

Gelernter, David (1998) Machine Beauty: Elegance and the Heart of Technology . Basic Books: New York, USA.

Gelernter, David (1998b) The Aesthetics of Computing . Weidenfeld & Nicolson: London, UK.

Gergen, Kenneth J. (1985) The Social Constructivionist Movement in Modern Psychology. American Psychologist 40(3): pp. 266-275.

Gibbs, Norman E.; Tucker, Allen B. (1986) A Model Curriculum for a Liberal Arts Degree in Computer Science. Communications of the ACM 29(3): pp. 202-210.

Gingras, Yves (1997) The New Dialectics of Nature. Social Studies of Science 27(2): pp. 317-334.

Glaser, George(1974) Education 'Inadequate' for Business DP. In Computerworld VIII(45) (November 6th). By Holmes, Edith: pp 1-2.

Glasersfeld, Ernst von (1995) Radical Constructivism: a Way of Knowing and Learning . The Falmer Press: London.

Glass, Robert L.; Ramesh, V.; Vessey, Iris (2004) An Analysis of Research in Computing Disciplines. Communications of the ACM 47(6): pp. 89-94.

Glass, Robert L. (1995) A Structure-Based Critique of Contemporary Computing Research. Journal of Systems and Software 28(1995): pp. 3-7.

Glass, Robert L. (2005) "Silver bullet" Milestones in Software History. Communications of the ACM 48(8): pp. 15-18.

Gneiting, Tilmann; Raftery, Adrian E. (2005) Weather Forecasting with Ensemble Methods. Science 310(5746): pp. 248-249.

Godin, Benoît (1997) The Rhetoric of a Health Technology: The Microprocessor Patient Card. Social Studies of Science 27(6): pp. 865-902.

Godwin, Mike (1999) From Washington: net to worry. Communications of the ACM 42(12): pp. 15-17.

Goldweber, Michael; Impagliazzo, John; Bogoiavlenski, Iouri A.; Clear, A.G.; Davies, Gordon; Flack, Hans; Myers, J. Paul; Rasala, Richard (1997) Historical Perspectives on the Computing Curriculum. ACM SIGCUE Outlook 25(4): pp. 94-111.

Gottschalk, Louis (1950) Understanding History: A Primer of Historical Method (2nd ed.). Alfred A. Knopf, Inc.: New York, USA.

Grandin, Karl; Wormbs, Nina; Widmalm, Sven (eds.) (2004) The Science-Industry Nexus: History, Policy, Implications . Science History Publications / USA & The Nobel Foundation: Sagamore Beach, MA, USA.

Greenfield, Martin N. (1984) The Impact of FORTRAN Standardization. IEEE Annals of the History of Computing 6(1): pp. 33.

Grier, David Alan (1996) The ENIAC, the Verb "to program" and the Emergence of Digital Computers. IEEE Annals of the History of Computing 18(1): pp. 51-55.

Grier, David Alan (2002) Twin Pillars of Computing. IEEE Annals of the History of Computing 24(3): pp. 87-88.

Grosch, Herb (1959) Plus & Minus. Datamation 5(6): pp. 51.

Grudin, Jonathan (1990) The Computer Reaches Out: The Historical Continuity of Interface Design. (ACM) Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Empowering People. April 1.-5., 1990, Seattle, Washington, United States: pp. 261-268.

Gruninger, Michael; Lee, Jintae (2002) Special Issue on Ontology Applications and Design: Introduction. Communications of the ACM 45(2): pp. 39-41.

Guba, Egon G.; Lincoln, Yvonne S. (1994) Competing Paradigms in Qualitative Research (Chapter six in Denzin & Lincoln (eds.), 1994: pp.105-117) . SAGE Publications: .

Guntheroth, Kurt (2006) Only More Original Research Can Save Computer Science (Letter to CACM Forum). Communications of the ACM 49(2): pp. 11.

Gödel, Kurt (1931) On Formally Undecidable Propositions of Principia Mathematica and Related Systems. In Feferman, Solomon (ed.) (1986): "Kurt Gödel: Collected Works, Vol. I": pp. 145-195.

Hacking, Ian (1999) The Social Construction of What? . Harvard University Press: Cambridge, Mass..

Hamming, Richard W. (1969) One Man's View of Computer Science (ACM Turing Lecture). Journal of the Association for Computing Machinery 16(1): pp. 3-12.

Hamming, Richard W. (1997) How to Think About Trends. In Denning, Peter J.; Metcalfe, Robert M. (eds.) (1997): "Beyond Calculation: The Next Fifty Years of Computing": pp. 65-74.

Hampden-Turner, Charles; Trompenaars, Fons (1997) Response to Geert Hofstede. International Journal of Intercultural Relations 21(1): pp. 149-159.

Hansen, Wilfred J. (1981) The Structure of "Data Structures". (ACM) ACM '81 Conference. November 9.-11., ACM Press, New York, NY, USA: pp. 89-95.

Hantrais, Linda; Mangen, Stephen (eds.) (1996) Cross-national Research Methods in the Social Sciences . Pinter: New York/London.

Haraway, Donna J. (1999 [1985]) Modest_Witness@Second_Millennium. In MacKenzie, Douglas; Wajcman, Judy (eds.) (1999): "The Social Shaping of Technology": pp. 41-49.

Harel, David (1980) On Folk Theorems. Communications of the ACM 23(7): pp. 379-389.

Harré, Rom (1972) The Philosophies of Science: An Introductory Survey . Oxford University Press: Oxford, UK.

Harris, Don; Duffy, Vincent; Smith, Michael; Stephanidis, Constantine (eds.) (2003) Human-Centred Computing: Cognitive, Social and Ergonomic Aspects, Vol. 3 . Lawrence Erlbaum Associates: New Jersey, USA.

Hartmanis, Juris (Chairman) (1992) Computing the Future. Communications of the ACM 35(11): pp. 30-40.

Hartmanis, Juris (1994) Turing Award Lecture: On Computational Complexity and the Nature of Computer Science. Communications of the ACM 37(10): pp. 37-43.

Hartswood, Mark; Procter, Rob; Slack, Roger; Voß, Alex; Büscher, Monika; Rouncefield, Mark; Rouchy, Philippe (2002) Co-Realization: Towards a Principled Synthesis of Ethnomethodology and Participatory Design. Scandinavian Journal of Information Systems 14(2): pp. 9-30.

Heilbroner, Robert L. (1967) Do Machines Make History?. Technology and Culture 8(1967): pp. 335-345.

Heilbroner, Robert L. (1994) Technological Determinism Revisited. In Smith and Marx (eds.) (1994): "Does Technology Drive History? The Dilemma of Technological Determinism": pp. 67-78.

Heim, Michael (1993) The Metaphysics of Virtual Reality . Oxford University Press: New York, USA.

Hennessy, John L.; Patterson, David A. (1996 [1990]) Computer Architecture: A Quantitative Approach (2nd ed.). Morgan Kaufmann Publishers: San Francisco, USA.

Himanen, Pekka (2001) The Hacker Ethic - and the Spirit of the Information Age . Random House: New York, NY, USA.

Hirschheim, Rudy; Klein, Heinz K. (1989) Four Paradigms of Information Systems Development. Communications of the ACM 32(10): pp. 1199-1216.

Hitchcock, Christopher (ed.) (2004) Contemporary Debates in Philosophy of Science . Blackwell Publishing: Oxford, UK.

Hodder, Ian (1994) The Interpretation of Documents and Material Culture. In Denzin, Norman K.; Lincoln, Yvonna S. (eds.) (1994): "Handbook of Qualitative Research": pp. 393-402.

Hofstadter, Richard (1968) History and Sociology: Some Methodological Considerations. In Lipset & Hofstadter (eds.) (1968): "Sociology and History: Methods": pp. 3-19.

Hofstede, Geert (1997) Cultures and Organizations: Software of the Mind . McGraw-Hill: New York.

Holloway, C. Michael (1995) Software Engineering and Epistemology. ACM SIGSOFT Software Engineering Notes 20(2): pp. 20-21.

Holstein, James A.; Gubrium, Jaber F. (1994) Phenomenology, Ethnomethodology, and Interpretive Practice. In Denzin, Norman K.; Lincoln, Yvonna S. (eds.) (1994): "Handbook of Qualitative Research": pp. 262-272.

Holstein, James A.; Gubrium, Jaber F. (eds.) (2003) Inner Lives and Social Worlds: Readings in Social Psychology . Oxford University Press: New York, USA.

Honey, Margaret (2004) Technology Has Improved Education. In Egendorf (ed.) (2004): "The Information Revolution: Opposing Viewpoints": pp. 70-78.

Hopcroft, John E. (1987) Computer Science: The Emergence of a Discipline (Turing Award Lecture). Communications of the ACM 30(3): pp. 198-202.

Hopper, Grace Murray (1978) Keynote Address ACM SIGPLAN History of Programming Languages Conference, June 1-3, 1978. In Wexelblat, 1981 (ed.): "History of Programming Languages": pp. 7-20.

Horgan, John (1996) The End of Science: Facing the Limits of Knowledge in the Twilight of the Scientific Age . Broadway Books: New York, USA.

Horowitz, Ellis; Morgan, Howard Lee; Shaw, Alan C. (1972) Computers and Society: a Proposed Course for Computer Scientists. Communications of the ACM 15(4): pp. 257-261.

Householder, Alston S. (1956) Presidential Address to the ACM . Journal of the ACM 3(1): pp. 1-2.

Householder, Alston S. (1957) Retiring Presidential Address. Journal of the ACM 4(1): pp. 1-4.

Hughes, Pat M.; Cosier, Graham (2001) What Makes a Revolution? Disruptive Technology and Social Change. BT Technology Journal 19(4): pp. 24-28.

Hughes, Thomas Parke (1983) Networks of Power: Electrification in Western Society, 1880-1930 . The Johns Hopkins University Press: London.

Hughes, Thomas Parke (1994) Technological Momentum. In Smith and Marx (eds.) (1994): "Does Technology Drive History? The Dilemma of Technological Determinism": pp. 101-114.

Hume, David (1739) A Treatise of Human Nature (ed. Penguin Books, 1969). Penguin Books: Aylesbury, UK.

Hume, David (1777[1748]) An Enquiry Concerning Human Understanding . Selby-Bigge: London, UK.

Hurley, Patrick J. (2000 [1982]) A Concise Introduction to Logic (7th edition). Wadsworth Publishing: Belmont (California).

Irving, Larry (2004) The Information Revolution Has Created a Digital Divide. In Egendorf (ed.) (2004): "The Information Revolution: Opposing Viewpoints": pp. 21-30.

Jacobs, Struan (2003) Misunderstanding John Stuart Mill on science: Paul Feyerabend's bad influence. The Social Science Journal 40(2003): pp. 201-212.

Jehn, Lawrence A.; Rine, David C.; Sondak, Norman (1978) Computer Science and Engineering Education: Current Trends, New Dimensions and Related Professional Programs. ACM SIGCSE Bulletin 10(3): pp. 162-178.

Johansson, Frans (2004) The Medici Effect: Breakthrough Insights at the Intersection of Ideas, Concepts, and Cultures . Harvard Business School Press: Boston, Mass., USA.

Johnson, Deborah G.; Nissenbaum, Helen (eds.) (1995) Computers, Ethics & Social Values . Prentice-Hall: New Jersey, USA.

Johnson, R. Burke; Onwuegbuzie, Anthony J. (2004) Mixed Methods Research: A Research Paradigm Whose Time Has Come. Educational Researcher 33(7): pp. 14-26.

Johnson, Robert R. (1998) User-Centered Technology: A Rhetorical Theory for Computer And Other Mundane Artifacts . State University of New York Press: Albany, NY, USA.

Johnson, Ralph (2005) Share Open Source Sources (Letter to CACM Forum). Communications of the ACM 48(1): pp. 13.

Kamppuri, Minna; Tukiainen, Markku (2004) Culture in Human-Computer Interaction Studies: A survey of ideas and definitions. (CaTac) Cultural Attitudes Towards Technology and Communication '04. June 27. - July1., Karlstad, Sweden: pp. 43-57.

Kamppuri, Minna; Tedre, Matti; Tukiainen, Markku (2006) Towards the Sixth Level in Interface Design: Understanding Culture. (ACM) Proceedings of the CHI-SA 2006, 5th Conference on Human Computer Interaction in Southern Africa (ed. van Greunen, Darelle). January 25th-27th 2006, Cape Town, South Africa: pp. 69-74.

Kamppuri, Minna; Tedre, Matti; Tukiainen, Markku (2006b) A Cultural Approach to Interface Design. (Koli Calling 2005) Proceedings of the 5th Annual Finnish/Baltic Sea Conference on Computer Science Education. Nov. 17th-20th 2005, Koli, Lieksa, Finland: pp. 149-152.

Kandel, Abraham (1972) Computer Science - A Vicious Circle. Communications of the ACM 15(6): pp. 470-471.

Kant, Immanuel (1966 [1781]) Critique of Pure Reason . Hackett: Indianapolis, USA.

Kavipurapu, Krishna M.; Frailey, Dennis J. (1979) Quantification of Architectures Using Software Science. ACM SIGARCH Computer Architecture News 7(10): pp. 2-6.

Kay, Alan C. (2000) The computer revolution hasn't happened yet (keynote session). (ACM) Proceedings of the eighth ACM international conference on Multimedia. March 10-14. 2001, Marina del Rey, California, USA: pp. 1.

Kelly, Kevin T.; Glymour, Clark (2004) Why Probability does not Capture the Logic of Scientific Justification. In Hitchcock, Christopher (ed.) (2004): "Contemporary Debates in Philosophy of Science": pp. 95-114.

Kelsey, John; Schneier, Bruce; Wagner, David; Hall, Chris (2000) Side Channel Cryptanalysis of Product Ciphers. Journal of Computer Security 8(2/3): pp. 141-158.

Kernighan, Brian W.; Ritchie, Dennis M. (1988) The C Programming Language (2nd ed.). Prentice Hall: Englewood Cliffs, NJ, USA.

Kevles, Daniel J. (1987) The Physicists: The History of a Scientific Community in Modern America . Harvard University Press: Cambridge, Mass., USA.

Khalil, Hatem; Levy, Leon S. (1978) The Academic Image of Computer Science. ACM SIGCSE Bulletin 10(2): pp. 31-33.

Kidder, John Tracy (1981) The Soul of a New Machine . Little, Brown, and co.: New York, NY, USA.

Kincaid, Harold (2004) There are Laws in the Social Sciences. In Hitchcock, Christopher (ed.) (2004): "Contemporary Debates in Philosophy of Science": pp. 168-185.

Kitchenham, Barbara Ann (1996) Evaluating Software Engineering Methods and Tool (Part 1: The Evaluation Context and Evaluation Methods). Software Engineering Notes 21(1): pp. 11-15.

Klawe, Maria; Shneiderman, Ben (2005) Crisis and Opportunity in Computer Science. Communications of the ACM 48(11): pp. 27-28.

Kline, Ronald; Pinch, Trevor (1999) The Social Construction of Technology. In MacKenzie, Douglas; Wajcman, Judy (eds.) (1999): "The Social Shaping of Technology (2nd edition)": pp. 113-115.

Kline, Stephen J. (1985) What is Technology?. Bulletin of Science, Technology & Society 5(3): pp. 215-218.

Kling, Rob (1980) Social Analyses of Computing: Theoretical Perspectives in Recent Empirical Research. ACM Computing Surveys 12(1): pp. 61-110.

Kling, Rob (ed.) (1996 [1991]) Computerization and Controversy: Value Conflicts and Social Choices (2nd ed.). Academic Press: San Diego, CA, USA.

Knuth, Donald E. (1964) Backus Normal Form vs. Backus Naur Form. Communications of the ACM 7(12): pp. 735-736.

Knuth, Donald E. (1967) The Remaining Trouble Spots in ALGOL 60. Communications of the ACM 10(10): pp. 611-618.

Knuth, Donald E. (1968) The Art of Computer Programming Vol. 1 : Fundamental Algorithms (1st edition). Addison-Wesley: Reading, Mass..

Knuth, Donald E. (1972) Ancient Babylonian Algorithms. Communications of the ACM 15(7): pp. 671-677.

Knuth, Donald E. (1972b) George Forsythe and the Development of Computer Science. Communications of the ACM 15(8): pp. 721-727.

Knuth, Donald E. (1974) Computer Science and its Relation to Mathematics. American Mathematical Monthly 81(Apr.1974): pp. 323-343.

Knuth, Donald E. (1974b) Structured Programming with go to Statements. ACM Computing Surveys 6(4): pp. 261-301.

Knuth, Donald E. (1974c) Computer Programming as an Art. Communications of the ACM 17(12): pp. 667-673.

Knuth, Donald E. (1985) Algorithmic Thinking and Mathematical Thinking. American Mathematical Montly 92(March): pp. 170-181.

Knuth, Donald E. (1991) Theory and Practice. Theoretical Computer Science 90(1991): pp. 1-15.

Knuth, Donald E. (1992) Computer Programming and Computer Science. In Morris (ed.) (1992): "Academic Press Dictionary of Science and Technology": pp. 490.

Knuth, Donald E. (1997 [1968]) The Art of Computer Programming Vol. 1 : Fundamental Algorithms (3rd edition). Addison-Wesley: Reading, Mass., USA.

Knuth, Donald E. (1998) The Art of Computer Programming Vol. 2 : Seminumerical Algorithms (3rd edition). Addison-Wesley: Reading, Mass..

Knuth, Donald E. (1998b) The Art of Computer Programming Vol. 3 : Sorting and Searching (2nd ed.). Addison-Wesley: Reading, Mass., USA.

Knuth, Donald E. (2001) Things a Computer Scientist Rarely Talks About . CSLI Publications: Stanford, California.

Koen, Billy Vaughn (1987) Definition of the Engineering Method . American Society for Engineering Education: Washington, D.C., USA.

Koen, Billy Vaughn (2003) Discussion of the Method: Conducting the Engineer's Approach to Problem Solving . Oxford University Press: Oxford, UK.

Koepsell, David R. (2000) The Ontology of Cyberspace . Open Court: Chicago, USA.

Krampen, Martin; Seitz, Peter (eds.) (1967) Design and Planning II - Computers in Design and Communication . Hastings House: New York, NY, USA.

Kugel, Peter (1988) Computer Science Departments in Trouble (Letters: ACM Forum). Communication of the ACM 31(3): pp. 243.

Kugel, Peter (2005) It's Time to Think Outside the Computational Box. Communications of the ACM (11): pp. 32-37.

Kuhn, Thomas (1970) Reflections on My Critics. In Lakatos and Musgrave (eds.): "Criticism and the Growth of Knowledge": pp. 231-278.

Kuhn, Thomas (1996 [1962]) The Structure of Scientific Revolutions (3rd edition). The University of Chicago Press: Chicago, USA.

Lai, Vincent S.; Mahapatra, Radha K. (1997) Exploring the Research in Information Technology Implementation. Information & Management 32(1997): pp. 187-201.

Lakatos, Imre; Musgrave, Alan (eds.) (1970) Criticism and the Growth of Knowledge . Cambridge University Press: London, UK.

Lakatos, Imre (1970) Falsification and the Methodology of Scientific Research Programmes. In Lakatos, Imre; Musgrave; Alan (eds.) (1970): "Criticism and the Growth of Knowledge": pp. 91-196.

Lakatos, Imre (1976) Proofs and Refutations: The Logic of Mathematical Discovery (eds. Worrall, John; Zahar, Elie). Cambridge University Press: Cambridge, UK.

Land, Frank (2000) The First Business Computer: A Case Study in User-Driven Innovation. IEEE Annals of the History of Computing 22(3): pp. 16-26.

Lash, Scott; Urry, John (1994) Economies of Signs and Space . Sage: London, UK.

Lee, Ed (1989) Some Suggestions on a Computer Science Undergraduate Curriculum. (IEEE) Proceedings of the COMPCON Spring '89: 34th IEEE Computer Society International Conference: Intellectual Leverage. Feb.27-Mar.3, San Francisco, CA, USA: pp. 366.

Lee, John A. N. (1996) "Those Who Forget the Lessons of History Are Doomed To Repeat It", or, Why I Study the History of Computing. IEEE Annals of the History of Computing 18(2): pp. 54-62.

Lee, John A.N. (1996b) History in the Computer Science Curriculum. ACM SIGCSE Bulletin 28(2): pp. 15-20.

Leeser, Miriam (2004) Digital Logic. In Tucker, Allen B. (ed.) (2004): "Computer Science Handbook": pp. 16-1ff..

Lemon, M.C. (2003) Philosophy of History . Routledge: London / New York.

Leone, Bruno; Stalcup, Brenda; Barbour, Scott; Winters, Paul A.; Williams, Mary E. (eds.) (1998) The Information Revolution: Opposing Viewpoints . Greenhaven Press: San Diego, California, USA.

Lerman, Steven R. (1993) Problem Solving and Computation for Scientists and Engineers . Prentice-Hall: New Jersey, USA.

Lethbridge, Timothy C. (2000) What Knowledge is Important to a Software Professional?. Computer 33(5): pp. 44-50.

Lévi-Strauss, Claude (1966 [1962]) The Savage Mind (2nd edition). University of Chicago Press: Chicago, USA.

Levinson, Paul (1998) Society Is Not Suffering from Information Overload. In Leone et al. (1998): "The Information Revolution: Opposing Viewpoints": pp. 32-39.

Lewis, Harry R.; Papadimitriou, Christos H. (1998) Elements of the Theory of Computation . Prentice-Hall: New Jersey, USA.

Lewis-Beck, Michael S.; Bryman, Alan; Liao, Tim Futing (eds.) (2004) The SAGE Encyclopedia of Social Science Research Methods . Sage Publications: Thousand Oaks, CA, USA.

Lévy, Pierre (1997) Collective Intelligence: Mankind's Emerging World in Cyberspace . Perseus Books: Cambridge, Mass., USA.

Lindsey, Charles H. (1996) A history of ALGOL 68. In Bergin, Thomas J.; Gibson, Richard G., Jr. (eds.) (1996): "History of programming languages II": pp. 27-96.

Lipset, Seymour Martin; Hofstadter, Richard (1968) Sociology and History: Methods . Basic Books: New York / London.

Liu, Jiming (2004) Forum (Response to Denning). Communications of the ACM 47(2): pp. 12.

Loui, Michael C. (1995) Computer Science is a New Engineering Discipline. ACM Computing Surveys 27(1): pp. 31-32.

Loui, R.P. (1998) B.C. Smith's, On The Origin of Objects (Book Review). Artificial Intelligence 106(1998): pp. 353-358.

Lynch, Michael (1996) Ethnomethodology. In Kuper, Adam & Kuper, Jessica (eds.) (1996): "The Social Science Encyclopedia": pp. 265-266.

Lynch, Michael (2004) Ethnomethodology. In Lewis-Beck et al. (eds.) (2004): "The SAGE Encyclopedia of Social Science Research Methods, Vol.1": pp. 332-333.

MacKenzie, Donald; Wajcman, Judy (eds.) (1999) The Social Shaping of Technology (2nd edition). Open University Press: England.

MacKenzie, Donald (1993) Negotiating Arithmetic, Constructing Proof: The Sociology of Mathematics and Information Technology. Social Studies of Science 23(1): pp. 37-65.

MacKenzie, Donald (1999) Theories of Technology and the Abolition of Nuclear Weapons. In MacKenzie, Douglas; Wajcman, Judy (eds.) (1999): "The Social Shaping of Technology (2nd edition)": pp. 419-442.

MacKinnon, Bryan (1988) The Computer Science Decline: What's Wrong (ACM Forum: Letters). Communications of the ACM 31(6): pp. 634-635.

MacLennan, Bruce (1999) Principles of Programming Languages: Design, Evaluation, and Implementation . Oxford University Press: New York, NY, USA.

MacLennan, B.J. (2003) Transcending Turing Computability. Minds and Machines 13(2003): pp. 3-22.

Madani, Kurosh; de Tremiolles, Ghislain; Tannhof, Pascal (2001) ZISC-036 Neuro-processor Based Image Processing. In Mira, José; Prieto, Alberto (eds.): "IWANN '01: Proceedings of the 6th International Work-Conference on Artificial and Natural Neural Networks": pp. 200-207.

Mahmood, Mo Adam (2002) Advanced Topics in End User Computing . Idea Group: Hershey, PA, USA.

Marcus, Mitchell; Akera, Atsushi (1996) Exploring the Architecture of an Early Machine: The Historical Relevance of the ENIAC Machine Architecture. IEEE Annals of the History of Computing 18(1): pp. 17-24.

Marcus, Aaron; Gould, Emilie West (2000) Crosscurrents: Cultural Dimensions and Global Web User-Interface Design. ACM Interactions 7(4): pp. 32-46.

Marcuse, Herbert (1964) One-Dimensional Man . Beacon Press: Boston, USA.

Martin, C. Dianne (1993) The Myth of the Awesome Thinking Machine. Communications of the ACM 36(4): pp. 120-133.

Masterman, Margaret (1970) The Nature of a Paradigm. In : "Criticism and the Growth of Knowledge (Lakatos and Musgrave, 1970)": pp. 59-.

Matloff, Norman (2004) Globalization and the American IT Worker. Communications of the ACM 47(11): pp. 27-29.

Mauchly, John W. (1942) The Use of High Speed Vacuum Tube Devices for Calculating. In Randell, Brian (1975): "The Origins of Digital Computers: Texts and Monographs in Computer Science": pp. 329-332.

Mauchly, John W. (1979) Amending the ENIAC Story. Datamation 1979(October): pp. 217-220.

McConnell, Steven C. (1993) Code Complete . Microsoft Press: Redmond, Washington, USA.

McGuffee, James W. (2000) Defining Computer Science. ACM SIGCSE Bulletin 32(2): pp. 74-76.

McKee, George (1995) Computer Science or Simply 'Computics'?. Computer (IEEE JNL) 28(12): pp. 136.

McLuhan, Marshall (1975 [1964]) Understanding Media: the Extensions of Man (fifth impression). Routledge: London, UK.

McSweeney, Brendan (2002) Hofstede's Model of National Cultural Differences and Their Consequences: A Triumph of Faith - a Failure of Analysis. Human Relations 55(1): pp. 89-118.

Merrill, M. David (2000) Write Your Dissertation First and Other Essays on a Graduate Education. Available at http://cito.byuh.edu/merrill/text/papers/GraduateEducation.pdf (accessed February 24th, 2006).

Michalewicz, Zbigniew; Fogel, David B. (2002 [2000]) How to Solve It: Modern Heuristics (corrected 3rd printing). Springer-Verlag: Berlin, Germany.

Miller, George A. (1956) The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. The Psychological Review 63(1956): pp. 81-97.

Mills, C. Wright (2000 [1959]) The Sociological Imagination (Fortieth Anniversary Edition). Oxford University Press: New York, USA.

Mingers, John (2001) Combining IS Research Methods: Towards a Pluralist Methodology. Information Systems Research 12(3): pp. 240-259.

Mingers, John (2003) The Paucity of Multimethod Research: A Review of the Information Systems Literature. Information Systems Journal 13(3): pp. 233-249.

Minsky, Marvin (1967) Why Programming is a Good Medium for Expressing Poorly Understood and Sloppily-Formulated Ideas. In Krampen, Martin; Seitz, Peter (eds.) (1967): "Design and Planning II - Computers in Design and Communication": pp. 120-125.

Minsky, Marvin (1970) ACM Turing Lecture: Form and Content in Computer Science. Journal of the Association for Computing Machinery 17(2): pp. 197-215.

Minsky, Marvin L. (1979) Computer Science and the Representation of Knowledge. In Dertouzos, Michael L.; Moses, Joel: "The Computer Age: A Twenty-Year View": pp. 392-421.

Mitcham, Carl (1994) Thinking Through Technology: The Path Between Engineering and Philosophy . The University of Chicago Press: Chicago, USA.

Moor, James H. (1978) Three Myths of Computer Science. The British Journal for the Philosophy of Science 29(1978): pp. 213-222.

Morris, Christopher (ed.) (1992) Academic Press Dictionary of Science and Technology . Academic Press: San Diego, CA, USA.

Morrison, Joline; George, Joey F. (1995) Exploring the Software Engineering Component in MIS Research. Communications of the ACM 38(7): pp. 80-91.

Muller, Michael J.; Wildman, Daniel M.; White, Ellen A. (1993) Taxonomy of PD Practices: A Brief Practitioner's Guide. Communications of the ACM 36(4): pp. 26-28.

Mumford, Lewis (1962 [1934]) Technics and Civilization . Harcourt, Brace & World: New York.

Myers, Glenford J. (1978) Advances in Computer Architecture . John Wiley & Sons: New York, NY, USA.

National Research Council (chaired by Hughes, Thomas) (1999) Funding a Revolution: Government Support for Computing Research . National Academies Press: Washington, D.C., USA.

Naur, Peter & Randell, Brian (eds.) (1969) Software Engineering: Report on a Conference Sponsored by the Nato Science Committee; Garmisch, Germany, 7th to 11th October 1968 . NATO Scientific Affairs Division: Brussels, Belgium.

Naur, Peter (ed.); Backus, J. W.; Bauer, F. L.; Green, J.; Katz, C.; McCarthy, J.; Perlis, A. J.; Rutishauser, H.; Samelson, K.; Vauquois, B.; Wegstein, J. H.; Wijngaarden, A. van; Woodger, M. (1960) Report on the algorithmic language ALGOL 60. Communications of the ACM 3(5): pp. 299 - 314.

Naur, Peter (1966) The Science of Datalogy (in Letters to the Editor). Communications of the ACM 9(7): pp. 485.

Naur, Peter (1969) Programming by Action Clusters. BIT 9(3): pp. 250-258.

Naur, Peter (1972) An Experiment on Program Development. BIT 12(3): pp. 347-365.

Naur, Peter (1981) The European Side of the Last Phase of the Development of ALGOL 60. In : "Wexelblat: History of Programming Languages": pp. 92-139.

Naur, Peter (1992) Computing: A Human Activity . ACM Press: New York, NY, USA.

Needham, Joseph (1959) Science and Civilisation in China Vol.3 "Mathematics and the Sciences of the Heavens and the Earth" . Cambridge University Press: Cambridge, Great Britain.

Negroponte, Nicholas (1995) Being Digital . Hodder & Stoughton: London, UK.

Neumann, John von (1945) First Draft of a report on the EDVAC. In Randell, Brian (1975): "The Origins of Digital Computers: Texts and Monographs in Computer Science": pp. 355-364.

Newell, Allen; Perlis, Alan J.; Simon, Herbert A. (1967) Computer Science. Science 157(3795): pp. 1373-1374.

Nickerson, Eileen (ed.) (1993) The Dissertation Handbook: A Guide to Successful Dissertations . Kendall Hunt Publishing: Dubuque, USA.

Noble, David F. (1999) Social Choice in Machine Design: The Case of Automatically Controlled Machine Tools. In MacKenzie, Douglas; Wajcman, Judy (eds.) (1999): "The Social Shaping of Technology (2nd ed.)": pp. 161-176.

Norman, Donald A. (1997) Why It's Good That Computer Don't Work Like the Brain. In Denning, Peter J.; Metcalfe, Robert M. (eds.) (1997): "Beyond Calculation: The Next Fifty Years of Computing": pp. 105-116.

Norman, Donald A. (2005) Human-Centered Design Considered Harmful. Interactions 12(4): pp. 14-19.

NSR Computer Science and Telecommunications Board (1999) Funding a Revolution: Government Support for Computing Research . National Academy Press: Washington D.C., USA.

Olazaran, Mikel (1996) A Sociological Study of the Official History of the Perceptrons Controversy. Social Studies of Science 26(3): pp. 611-659.

Palvia, Prashant; Mao, En; Salam, A.F.; Soliman, Khalid (2003) Management Information Systems Research: What's There in a Methodology?. Communications of the AIS 11(16): pp. 1-33 (Article 16).

Pan, Jin; Cranefield, Stephen; Carter, Daniel (2003) A Lightweight Ontology Repository. (ACM) Proceedings of the second international joint conference on Autonomous agents and multiagent systems. July 14-18, 2003, Melbourne, Australia: pp. 632-638.

Perlis, Alan J. (1981) The American Side of the Development of ALGOL. In : "Wexelblat, 1981: History of Programming Languages": pp. 75-91.

Petersson, Tom (2005) Facit and the BESK Boys: Sweden's Computer Industry (1956-1962). IEEE Annals of the History of Computing 27(4): pp. 23-30.

Piccinini, Gualtiero (2003) Alan Turing and the Mathematical Objection. Minds and Machines 13(2003): pp. 23-48.

Pickering, Andrew (1993) The Mangle of Practice: Agency and Emergence in the Sociology of Science. American Journal of Sociology 99(3): pp. 559-589.

Pickering, Andrew (1995) The Mangle of Practice: Time, Agency, and Science . The University of Chicago Press: Chicago, USA.

Pinch, Trevor J.; Bijker, Wiebe E. (1987) The Social Construction of Facts and Artifacts: Or How the Sociology of Science and the Sociology of Technology Might Benefit Each Other. In Bijker, Wiebe E.; Hughes, Thomas P.; Pinch, Trevor J. (eds.) (1987): "The Social Construction of Technological Systems": pp. 17-50,349-372.

Pinker, Steven (2002) The Blank Slate: The Modern Denial of Human Nature . Penguin Books: New York.

Pitts, Gerald N.; Bateman, Barry (1974) A Software Oriented Computer Science Program. ACM SIGCSE Bulletin 6(1): pp. 33-36.

Planck, Max (1949) Scientific Autobiography and Other Papers . Philosophical Library: New York, USA.

Polachek, Harry (1997) Before the ENIAC. IEEE Annals of the History of Computing 19(2): pp. 25-30.

Polanyi, Michael (1964 [1958]) Personal Knowledge: Towards a Post-Critical Philosophy (Torchbook edition). Harper Torchbooks / The Academic Library: New York, USA.

Pólya, George (1957 [1945]) How to Solve It (2nd. edition). Penguin Books Ltd.: London, England.

Poon, Andy K.Y. (2006) Only More Original Research Can Save Computer Science (Letter to CACM Forum). Communications of the ACM 49(2): pp. 11.

Popper, Karl (1959 [1935]) The Logic of Scientific Discovery . Routledge: London, Great Britain.

Popper, Karl (1970) Normal Science and its Dangers. In Lakatos, Imre; Musgrave, Alan (eds.): "Criticism and the Growth of Knowledge": pp. 51-58.

Postley, John A. (1960) Letters to the Editor. Communications of the ACM 3(1): pp. A6.

Pour, Gilda; Griss, Martin L.; Lutz, Michael (2000) The Push to Make Software Engineering Respectable. Computer 33(5): pp. 35-43.

Pratt, Vernon (1987) Thinking Machines: The Evolution of Artificial Intelligence . Basil Blackwell Ltd.: Frome, Great Britain..

Preston, John (1997) Feyerabend's Retreat from Realism. Philosophy of Science 64(proceedings): pp. S421-S431.

Puchta, Susann (1996) On the Role of Mathematics and Mathematical Knowledge in the Invention of Vannevar Bush's Early Analog Computers. IEEE Annals in the History of Computing 18(4): pp. 49-59.

Pugh, Emerson W.; Aspray, William (1996) Creating the Computer Industry. IEEE Annals of the History of Computing 18(2): pp. 7-17.

Quine, Willard Van Orman (1960) Word and Object . The MIT Press: Mass., USA.

Quine, Willard Van Orman (1980 [1953]) From a Logical Point of View (2nd, revised edition). Harvard University Press: Cambridge, Massachusetts.

Raatikainen, Kimmo (1992) Meidän on kysyttävä, mitä saa automatisoida. Tietojenkäsittelytiede 3(November 1992): pp. 51-57.

Rajlich, Václav; Wilde, Norman; Buckellew, Michelle; Page, Henry (2001) Software Cultures and Evolution. IEEE Computer 34(9): pp. 24-28.

Ralston, Anthony; Shaw, Mary (1980) Curriculum '78 - Is Computer Science Really that Unmathematical?. Communications of the ACM 23(2): pp. 67-70.

Ralston, Anthony (1981) Computer Science, Mathematics, and the Undergraduate Curricula in Both. American Mathematical Monthly 81(Aug.-Sept.): pp. 472-485.

Ramesh, V.; Glass, Robert L.; Vessey, Iris (2004) Research in Computer Science: An Empirical Study. The Journal of Systems and Software 70(2004): pp. 165-176.

Randell, Brian () Software Engineering in 1968. (IEEE) Proceedings of the 4th international conference on Software engineering. September 17th-19th, Munich, Germany: pp. 1-10.

Rapaport, William J. (2005) Philosophy of Computer Science: An Introductory Course. Teaching Philosophy 28(4): pp. 319-341.

Rawls, John (1971) A Theory of Justice . Harvard University Press: Cambridge, Mass., USA.

Reisch, George A. (1991) Did Kuhn Kill Logical Empiricism?. Philosophy of Science 58(1991): pp. 264-277.

Rescher, Nicholas (1998) Complexity: A Philosophical Overview . Transaction Publishers: New Brunswick, New Jersey, USA.

Rheingold, Howard (2003) Smart Mobs: The Next Social Revolution . Perseus Books: New York, NY, USA.

Rice, John R.; Rosen, Saul (2004) Computer Sciences at Purdue University - 1962 to 2000. IEEE Annals of the History of Computing 26(2): pp. 48-61.

Rice, John R. (1993) Letter to the CACM Forum. Communications of the ACM 36(2): pp. 19.

Rogers, Everett M. (2003) Diffusion of Innovations (5th edition). Free Press: New York, NY.

Rojas, Raúl (1997) Konrad Zuse's Legacy: The Architecture of the Z1 and Z3. IEEE Annals of the History of Computing 19(2): pp. 5-16.

Rosen, Saul (1969) Electronic Computers: a Historical Survey. ACM Computing Surveys 1(1): pp. 7-36.

Rosen, Saul (1972) Programming Systems and Languages 1965-1975. Communications of the ACM 15(7): pp. 591-600.

Rosenberg, Alex (2000) Philosophy of Science: A Contemporary Introduction (2nd ed.). Routledge: New York, NY, USA.

Rosenblatt, Bruce (1984) The Successors to FORTRAN: Why Does FORTRAN Survive?. Annals of the History of Computing 6(1): pp. 39-40.

Ross, Philip E. (2004) 5 Commandments (Technology Laws and Rules of Thumbs). IEEE Spectrum 40(12): pp. 30-35.

Rubin, Frank (1987) "GOTO Considered Harmful" Considered Harmful. Communications of the ACM 30(3): pp. 195-196.

Sadeh, Eligar (2002) Space Politics and Policy: An Evolutionary Perspective . Kluwer Academic Publishers: The Netherlands.

Sage, Andrew P. (1992) Systems Engineering . John Wiley & Sons, Inc.: New York, NY, USA.

Sammet, Jean E. (1969) Programming Languages: History and Fundamentals . Prentice-Hall, Inc.: Englewood Cliffs, New Jersey.

Sammet, Jean E. (1991) Some Approaches to, and Illustrations of, Programming Language History. Annals of the History of Computing 13(1): pp. 33-50.

Samoladas, Joannis; Stamelos, Joannis; Angelis, Lefteris; Oikonomou, Apostolos (2004) Open Source Software Development Should Strive for Even Greater Code Maintainability. Communications of the ACM 47(10): pp. 83-87.

Sánchez, J. Alfredo; Leggett, John J.; Schnase, John L. (1994) HyperActive: Extending an Open Hypermedia Architecture to Support Agency. ACM Transactions on Computer-Human Interaction 1(4): pp. 357-382.

Saukko, Paula (2005) Methodologies for Cultural Studies: an Integrative Approach. In Denzin, Norman K.; Lincoln, Yvonna S. (eds.) (2005): "The SAGE Handbook of Qualitative Research (3rd ed.)": pp. 343-356.

Saviani, D. (1985) Do senso comum a consciencia filosofica. Translation in Borba, 1990 . Cortez Editora: São Paulo, Brasil.

Savitch, Walter (2004) Problem Solving with C++: The Object of Programming (5th ed.). Addison-Wesley: Reading, Mass., USA.

Scharff, Robert C.; Dusek, Val (eds.) (2003) Philosophy of Technology: The Technological Condition . Blackwell Publishing: Oxford, UK.

Scheutz, Matthias (ed.) (2002) Computationalism: New Directions . The MIT Press: Massachusetts, USA.

Scheutz, Matthias (2003) Computation, Philosophical Issues about. In Nadel, Lynn (ed.): "Encyclopedia of Cognitive Science": pp. 604-610.

Schieber, Philip (1987) The Wit and Wisdom of Grace Hopper. The OCLC Newsletter 1987 March/April(167): pp. n/a.

Schreiber, Fabio A. (2005) The Cultural Roots of Computer Science (Letter to CACM Forum). Communications of the ACM 48(11): pp. 11-12.

Schwandt, Thomas A. (1994) Constructivist, Interpretivist Approaches to Human Inquiry. In Denzin, Norman K.; Lincoln, Yvonna S. (eds.): "Handbook of Qualitative Research (1994)": pp. 118-135.

Seale, Clive (ed.) (2004) Researching Society & Culture (2nd edition). Sage: London, UK.

Searle, John R. (1964) How to Derive "Ought" From "Is". Philosophical Review 73(1): pp. 43-58.

Searle, John R. (1980) Minds, Brains, And Programs. The Behavioral And Brain Sciences 1980(3): pp. 417-457.

Searle, John R. (1983) Intentionality: an Essay in the Philosophy of Mind . Cambridge University Press: Cambridge.

Searle, John R. (1996) The Construction of Social Reality . Penguin Press: England.

Searle, John R. (2001) Refutation of Relativism. Available at http://ist-socrates.berkeley.edu/~jsearle/articles.html (accessed 22nd Nov. 2005).

Shannon, Claude E. (1948) A Mathematical Theory of Communication. The Bell System Technical Journal 27(July, October): pp. 379-423, 623-656.

Shannon, Claude E. (1950) The Lattice Theory of Information. Institute of Radio Engineers, Transactions on Information Theory (Report of Proceedings, Symposium on Information Theory, London, Sept., 1950) 1(February, 1953): pp. 105-107.

Shapiro, Stewart (2000) Thinking About Mathematics: The Philosophy of Mathematics . Oxford University Press: Oxford, UK.

Shemer, Itzhak (1987) Systems Analysis: a Systemic Analysis of a Conceptual Model. Communications of the ACM 30(6): pp. 506-512.

Shirts, Michael; Pande, Vijay S. (2000) Screen Savers of the World Unite. Science 290(5498): pp. 1903-1904.

Shneiderman, Ben (2002) Leonardo's Laptop: Human Needs And the New Computing Technologies . The MIT Press: Cambridge, Mass., USA.

Shrader-Frechette, Kristin (1992 [2003]) Technology and Ethics. In Scharff & Dusek (eds.) (2003): "Philosophy of Technology: The Technological Condition": pp. 187-190.

Siefkes, Dirk (1997) Computer Science as Cultural Development. In Freksa, Christian; Jantzen, Matthias; Valk, Rüdiger (eds.) (1997): "Foundations of Computer Science: Potential-Theory-Cognition": pp. 37-48.

Siegelmann, Hava T. (2003) Neural and Super-Turing Computing. Minds and Machines 13(2003): pp. 103-114.

Silberschatz, Abraham; Galvin, Peter Baer; Gagne, Greg (2002) Operating System Concepts (6th ed.). John Wiley & Sons: New York, NY, USA.

Sills, David L. (ed.) (1968) International Encyclopedia of the Social Sciences . The MacMillan Company & The Free Press: USA.

Simon, Herbert A. (1981 [1969]) The Sciences of the Artificial (2nd ed.). The MIT Press: Cambridge, Mass., USA.

Sloane, Neil J.A.; Wyner, Aarod D. (eds.) (1993) Claude Elwood Shannon: Collected Papers . IEEE Press: New York, NY, USA.

Smelser, Neil J. & Baltes, Paul B. (eds.) (2001) International Encyclopedia of the Social & Behavioral Sciences . Elsevier: Oxford, UK.

Smelser, Neil (1988) Handbook of Sociology . Sage: Newbury Park, CA, USA.

Smith, Merritt Roe; Marx, Leo (eds.) (1994) Does Technology Drive History? The Dilemma of Technological Determinism . The MIT Press: Cambridge, Mass., USA.

Smith, Merritt Roe (1994) Technological Determinism in American Culture. In Smith, Merritt Roe and Marx, Leo (eds.): "Does Technology Drive History? The Dilemma of Technological Determinism": pp. 1-36.

Smith, Michael L. (1994b) Recourse of Empire: Landscapes of Progress in Technological America. In Smith, Merritt Roe and Marx, Leo (eds.) (1994): "Does Technology Drive History? The Dilemma of Technological Determinism": pp. 37-52.

Smith, Brian Cantwell (1996 [1985]) Limits of Correctness in Computers. In Kling, Rob (ed.) (1996): "Computerization and Controversy": pp. 810-825.

Smith, Brian Cantwell (1998 [1996]) On the Origin of Objects (MIT Paperback ed.). The MIT Press: Cambridge, Mass., USA.

Smith, Peter B. (2002) Culture's Consequences: Something Old and Something New. Human Relations 55(1): pp. 119-135.

Smith, Brian Cantwell (2002b) God, Approximately. In Richardson, W. Mark; Russell, Robert John; Clayton, Philip; Wegter-McNelly, Kirk: "Science and the Spiritual Quest": pp. 207-228.

Smith, Brian Cantwell (2002c) Reply to Dennett. In Clapin, Hugh: "Philosophy of Mental Representation": pp. 237-265.

Snelting, Gregor (1998) Paul Feyerabend and software technology. International Journal on Software Tools for Technology Transfer (STTT) 2(1): pp. 1-5.

Snow, Charles Percy (1964 [1959]) The Two Cultures and A Second Look . Cambridge University Press: Cambridge, UK.

Snow, Charles Percy (1966) Government, Science, and Public Policy. Science 151(3711): pp. 650-653.

Solla Price, Derek J. de (1959) An Ancient Greek Computer. Scientific American June(1959): pp. 60-67.

Somberg, Benjamin L. (1987) A Comparison of Rule-Based and Positionally Constant Arrangements of Computer Menu Items. ACM SIGCHI Bulletin 17(SI): pp. 255-260.

Sommerville, Ian (1982) Software Engineering . Addison-Wesley: Bedford Square, London, UK.

Spier, Michael J. (1974) A Critical Look at the State of Our Science. ACM SIGOPS Operating Systems Review 8(2): pp. 9-15.

Spindler, Michael (1998) The Information Revolution Can Benefit Society. In Leone et al. (1998): "The Information Revolution: Opposing Viewpoints": pp. 17-20.

Sprankle, Maureen (1998 [1989]) Problem Solving and Programming Concepts (4th edition). Prentice-Hall: New Jersey, USA.

Stake, Robert E. (1994) Case Studies. In Denzin, Norman K.; Lincoln, Yvonna S. (eds.) (1994): "Handbook of Qualitative Research": pp. 236-247.

Staley, Kent W. (1999) Logic, Liberty, and Anarchy: Mill and Feyerabend on Scientific Method. The Social Science Journal 36(4): pp. 603-614.

Stallman, Richard (2005) Free Is Not Open Software (Letter to CACM Forum). Communications of the ACM 48(7): pp. 12-13.

Stanford Encyclopedia of Philosophy (2005). Available at http://plato.stanford.edu/

Stewart, N.F. (1995) Science and Computer Science. ACM Computing Surveys (CSUR) 27(1): pp. 39-41.

Stevenson, Johan W.; Tanenbaum, Andrew S. (1979) Efficient Encoding of Machine Instructions. ACM SIGARCH Computer Architecture News 7(8): pp. 10-17.

Stevenson, D.E. (Steve) (1993) Science, Computational Science and Computer Science: At a Crossroads. (ACM) Proceedings of the 1993 ACM conference on Computer science. February 16-18, Indianapolis, Indiana, USA: pp. 7-14. Republished in Communications of the ACM 37(12), December 1994: pp.85-96.

Stibitz, George (1946) Introduction to the Course on Electronic Digital Computers. In Campbell-Kelly; Williams, Michael R. (eds.) (1985): "The Moore School Lectures": pp. 5-16.

Strauss, Claudia; Quinn, Naomi (1997) A Cognitive Theory of Cultural Meaning . Cambridge University Press: Cambridge, UK.

Stroustrup, Bjarne (1997) The C++ Programming Language (3rd ed.) . Addison-Wesley: Reading, Mass..

Strum, Shirley; Latour, Bruno (1999) Redefining the Social Link: From Baboons to Humans. In MacKenzie, Douglas; Wajcman, Judy (eds.) (1999): "The Social Shaping of Technology (2nd edition)": pp. 116-125.

Subramanyam, Krishna (1981) Scientific and Technical Information Resources . Marcel Dekker: New York, NY, USA.

Succi, Giancarlo; Valerio, Andrea; Vernazza, Tullio; Succi, Gianpiero (1998) Compatibility, Standards, and Software Production. StandardView 6(4): pp. 140-146.

Suchman, Lucy A. (1987) Plans and Situated Actions: The Problem of Human-Machine Communication . Cambridge University Press: Cambridge, UK.

Sutinen, Erkki; Tarhio, Jorma (2001) Teaching to Identify Problems in a Creative Way. (IEEE Computer Society) Proceedings of the FIE '01, Frontiers in Education. October 10-13, : pp. T1D8-13.

Swartzlander, Earl E., Jr. (2004) High-Speed Computer Arithmetic. In Tucker, Allen B. (ed.) (2004): "Computer Science Handbook": pp. 22-1ff..

Tedlock, Barbara (2005) The Observation of Participation and the Emergence of Public Ethnography. In Denzin, Norman K.; Lincoln, Yvonna S. (eds.) (2005): "The SAGE Handbook of Qualitative Research (3rd ed.)": pp. 467-481.

Tedre, Matti; Sutinen, Erkki; Kähkönen, Esko; Kommers, Piet (2003) Is Universal Usability Universal Only to Us?. (ACM) Proceedings of the CUU 2003. November 10.-11., Vancouver BC, Canada: pp. 1-2 (web publication).Available at http://www.sigchi.org/cuu2003/program.htm (Nov.28.2004), Also available at www.ethnocomputing.org

Tedre, Matti; Sutinen, Erkki; Kähkönen, Esko; Kommers, Piet (2006) Ethnocomputing: ICT in Social and Cultural Context. Communications of the ACM 49(1): pp. 126-130.

Tedre, Matti (2002) Ethnocomputing: A Multicultural View on Computer Science . University of Joensuu Press: Joensuu, Finland.

Tedre, Matti (2004) Problem, Knowledge, and Understanding in Ethnocomputing. (University of Joensuu, Department of Computer Science) Proceedings of the Second International Conference on Educational Technology in Cultural Context (International Proceedings Series 4). September 1-2, 2003, Joensuu, Finland: pp. 85-92.

Thacker, Eugene (2004) Biomedia . University of Minnesota Press: Minneapolis, USA.

Thomas, Gary (1997) What's the Use of Theory?. Harvard Educational Review 67(1): pp. 75-104.

Thompson, Ken (1984) Reflections on Trusting Trust. Communications of the ACM 27(8): pp. 761-763.

Tichy, Walter F.; Lukowicz, Paul; Prechelt, Lutz; Heinz, Ernst A. (1995) Experimental Evaluation in Computer Science: A Quantitative Study. Journal of Systems and Software 28(1995): pp. 9-18.

Tichy, Walter E. (1998) Should Computer Scientists Experiment More?. IEEE Computer 31(5): pp. 32-40.

Trochim, William M. (2000) The Research Methods Knowledge Base (2nd ed.). Atomic Dog Publishing: Cincinnati, OH, USA.

Trogemann, Georg; Nitussov, Alexander Y.; Ernst, Wolfgang (eds.) (2001) Computing in Russia: the History of Computer Devices and Information Technology Revealed . Vieweg: Braunschweig, Germany.

Trompenaars, Fons; Hampden-Turner, Charles (1997 [1993]) Riding the Waves of Culture: Understanding Cultural Diversity in Business (2nd ed.). Nicholas Brealey Publishing: London, UK.

Tuchman, Gaye (1994) Historical Social Science: Methodologies, Methods, and Meanings. In Denzin, Norman K.; Lincoln, Yvonna S. (eds.) (1994) : "Handbook of Qualitative Research": pp. 306-323.

Tuchman, Gaye (2004) Historical Methods. In Lewis-Beck et al. (eds.) (2004): "The SAGE Encyclopedia of Social Science Research Methods": pp. 462-464.

Tucker, Allen B. (Editor and Co-chair); Barnes, Bruce H. (Co-chair); Aiken, Robert M.; Barker, Keith; Bruce, Kim B.; Cain, J. Thomas; Conry, Susan E.; Engel, Gerald L.; Epstein, Richard G.; Lidtke, Doris K.; Mulder, Michael C.; Rogers, Jean B.; Spafford, Eugene H.; Turner, A. Joe (1991) A Summary of the ACM/IEEE-CS Joint Curriculum Task Force Report: Computing Curricula 1991. Communications of the ACM 34(6): pp. 68-84.

Tucker, Allen B. (ed.) (2004) Computer Science Handbook (2nd ed.). Chapman & Hall/CRC: Florida, USA.

Turing, Alan M. (1936) On Computable Numbers, With an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, Series 2 42(1936): pp. 230-265.

Turing, Alan M. (1939) Systems of Logic Based on Ordinals. Proceedings of the London Mathematical Society, Series 2 45(1939): pp. 161-228.

Turing, Alan M. (1950) Computing Machinery and Intelligence. Mind 59(236): pp. 433-460.

Ubiquity (2002) In memoriam: Edsgar Dijkstra (1930-2002). ACM Ubiquity 3(26): pp. 2.

Urry, John (2004) The Complexities of the Global. (University of Lancaster) On-line publications of the Department of Sociology. July 2nd, 2004, Lancaster, UK: pp. http://www.comp.lancs.ac.uk/sociology/papers/urry-complexities-global.pdf (Feb 5, 2006).

Walsham, Geoff (1995) The Emergence of Interpretivism in IS Research. Information Systems Research 6(4): pp. 376-394.

Wang, Xin; Chan, Christine W.; Hamilton, Howard J. (2002) Design of Knowledge-Based Systems With the Ontology-Domain-System Approach. (ACM) Proceedings of the 14th international conference on Software engineering and knowledge engineering. July 15-19, 2002, Ischia, Italy: pp. 233-236.

Watson, Burton (trans.) (1964) Chuang Tzu: Basic Writings . Columbia University Press: New York, NY, USA.

Wegner, Peter; Goldin, Dina (2006) Interactive Computing Is Already Outside the Box (Letter to CACM Forum). Communications of the ACM 49(3): pp. 11.

Wegner, Peter; Goldin, Dina (2003) Computation Beyond Turing Machines. Communications of the ACM 46(4): pp. 100.

Wegner, Peter (1976) Research Paradigms in Computer Science. (IEEE) Proceedings of the 2nd international conference on Software engineering. October 13-15, San Francisco, California, USA: pp. 322-330.

Wegner, Peter (1976b) Programming Languages - The First 25 Years. IEEE Transactions on Computers C-25(12): pp. 1207-1225.

Wegner, Peter (1993) Letter to the CACM Forum. Communications of the ACM 36(2): pp. 17-19.

Weiser, Mark; Brown, John Seely (1997) The Coming Age of Calm Technology. In Denning, Peter J.; Metcalfe, Robert M. (eds.) (1997): "Beyond Calculation: The Next Fifty Years of Computing": pp. 75-85.

Weiser, Mark (1993) Ubiquitous Computing. IEEE Computer 26(10): pp. 71-72.

Weiss, E.A. & Corley, Henry P.T. (1958) What's In a Name? (Response to a Letter to Editor). Communications of the ACM 1(4): pp. 6.

Weiss, Eric A. (1993) Letter to the CACM Forum. Communications of the ACM 36(2): pp. 20.

Wellman, Barry; Hiltz, Starr Roxanne (2004) Sociological Bob: How Rob Kling Brought Computing and Sociology Together. The Information Society 20(2004): pp. 91-95.

Wernick, Paul; Hall, Tracy (2004) Can Thomas Kuhn's Paradigms Help Us Understand Software Engineering?. European Journal of Information Systems 13(3): pp. 235-243.

Vessey, Iris; Ramesh, V; Glass, Robert L. (2002) Research in Information Systems: An Empirical Study of Diversity in the Discipline and Its Journals. Journal of Management Information Systems 19(2): pp. 129-174.

West, Dave (1997) Hermeneutic Computer Science. Communications of the ACM 40(4): pp. 115-116.

Wexelblat, Richard L. (ed.) (1981) History of Programming Languages (ACM monograph series) . Academic Press: London, UK.

Wilde, Norman; Buckellew, Michelle; Page, Henry; Rajlich, Václav (2001) A Case Study of Feature Location in Unstructured Legacy Fortran Code. (IEEE) Proceedings of the Fifth European Conference on Software Maintenance and Reengineering. 14th-16th March, Lisbon, Portugal: pp. 68-76.

Viller, Stephen; Sommerville, Ian (1999) Coherence: An Approach to Representing Ethnographic Analyses in Systems Design. Human-Computer Interaction 14(1999): pp. 9-41.

Williams, Samuel B. (1954) The Association for Computing Machinery. Journal of the ACM 1(1): pp. 1-3.

Williams, Michael R. (1985) A History of Computing Technology . Prentice-Hall: New Jersey, USA.

Williams, Rosalind (1994) The Political and Feminist Dimensions of Technological Determinism. In Smith, Merrit Roe; Marx, Leo (eds.) (1994): "Does Technology Drive History? The Dilemma of Technological Determinism": pp. 217-235.

Winegrad, Dilys (1996) Celebrating The Birth Of Modern Computing: The Fiftieth Anniversary of a Discovery At The Moore School of Engineering of the University of Pennsylvania. IEEE Annals of the History of Computing 18(1): pp. 5-9.

Winner, Langdon (1993) Social Constructionism: Opening the Black Box and Finding it Empty. Science as Culture 3,3(16): pp. 427-452.

Winner, Langdon (1999) Do Artifacts Have Politics?. In MacKenzie, Donald; Wajcman, Judy (eds.) (1999): "The Social Shaping of Technology": pp. 28-40.

Wirth; Niklaus (1971) Program Development by Stepwise Refinement. Communications of the ACM 14(4): pp. 221-227.

Wishner, Raymond P. (1968) Letters to the Editor: Comment on Curriculum '68. Communications of the ACM 11(10): pp. 658.

Wittgenstein, Ludwig (1986 [1922]) Tractatus Logico-Philosophicus (translated from German; with an introduction by Bertrand Russell). Routledge & Kegan Paul: London, Great Britain.

Wolfram, Stephen (2002) A New Kind of Science . Wolfram Media: Champaign, IL.

Wood, Helen M. (1995) Computer Society Celebrates 50 Years. IEEE Annals of the History of Computing 17(4): pp. 6.

Woolgar, Steve (ed.) (2002) Virtual Society? Technology, Cyberbole, Reality . Oxford University Press: Oxford, UK.

Worsley, B.H. (1950) The E.D.S.A.C. Demonstration. In Randell, Brian (1975): "The Origins of Digital Computers: Texts and Monographs in Computer Science": pp. 395-401.

Young, Kimberley, S. (1998) Caught in the Net . John Wiley & Sons: New York, NY, USA.

Zadeh, Lotfi A. (1968) Computer Science as a Discipline. The Journal of Engineering Education 58(8): pp. 913-916.

Zaphyr, P.A. (1959) Letters to the Editor. Communications of the ACM 2(1): pp. 4.

Zaslavsky, Claudia (1980) Count on Your Fingers African Style . Harper & Row: New York, USA.

Zelkowitz, Marvin V.; Wallace, Dolores (1997) Experimental Validation in Software Engineering. Information and Software Technology 39(1997): pp. 735-743.

Zemanek, Heinz (1979) Al-Khorezmi His Background, His Personality His Work and His Influence. (Proceedings of the Symposium on "Algorithms in Modern Mathematics and Computer Science") Lecture Notes in Computer Science vol 122/1981. September 16th–22nd, Urgench, Uzbek SSSR: pp. 1-81.

Zhang, Cuihua; Howland, John E. (2005) Brief and Yet Bountiful: The History of Computing, Why Do Students Need It?. Journal of Computing Sciences in Colleges 20(4): pp. 308-314.

Zúñiga, Gloria (2001) Ontology: Its Transformation from Philosophy to Information Systems. (ACM) Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001. October 17 - 19, 2001, Ogunquit, Maine, USA: pp. 187-197.

Dissertations in Computer Science

Rask, Raimo. Automating Estimation of Software Size during the Requirements Specification Phase—Application of Albrecth's Function Point Analysis Within Structured Methods. Joensuun yliopiston luonnontieteellisiä julkaisuja, 28 – University of Joensuu. Publications in Sciences, 28. 128 pp. Joensuu, 1992.

Ahonen, Jarmo. Modeling Physical Domains for Knowledge Based Systems. Joensuun yliopiston luonnontieteellisiä julkaisuja, 33 – University of Joensuu. Publications in Sciences, 33. 127 pp. Joensuu, 1995.

Kopponen, Marja. CAI in CS. University of Joensuu, Computer Science, Dissertations 1. 97 pp. Joensuu, 1997.

Forsell, Martti. Implementation of Instruction-Level and Thread-Level Parallelism in Computers. University of Joensuu, Computer Science, Dissertations 2. 121 pp. Joensuu, 1997.

Juvaste, Simo. Modeling Parallel Shared Memory Computations. University of Joensuu, Computer Science, Dissertations 3. 190 pp. Joensuu, 1998.

Ageenko, Eugene. Context-based Compression of Binary Images. University of Joensuu, Computer Science, Dissertations 4. 111 pp. Joensuu, 2000.

Tukiainen, Markku. Developing a New Model of Spreadsheet Calculations: A Goals and Plans Approach. University of Joensuu, Computer Science, Dissertations 5. 151 pp. Joensuu, 2001.

Eriksson-Bique, Stephen. An Algebraic Theory of Multidimensional Arrays. University of Joensuu, Computer Science, Dissertations 6. 278 pp. Joensuu, 2002.

Kolesnikov, Alexander. Efficient Algorithms for Vectorization and Polygonal Approximation. University of Joensuu, Computer Science, Dissertations 7. 204 pp. Joensuu, 2003.

Kopylov, Pavel. Processing and Compression of Raster Map Images. University of Joensuu, Computer Science, Dissertations 8. 132 pp. Joensuu, 2004.

Virmajoki, Olli. Pairwise Nearest Neighbor Method Revisited. University of Joensuu, Computer Science, Dissertations 9. 164 pp. Joensuu, 2004.

Suhonen, Jarkko. A Formative Development Method for Digital Learning Environments in Sparse Learning Communities, University of Joensuu, Computer Science, Dissertations 10. 154 pp. Joensuu, 2005.

Xu, Mantao. K-means Based Clustering and Context Quantization, University of Joensuu, Computer Science, Dissertations 11. 162pp. Joensuu, 2005.

Kinnunen, Tomi. Optimizing Spectral Feature Based Text-Independent Speaker Recognition. University of Joensuu, Computer Science, Dissertations 12. 156pp. Joensuu, 2005.

Kärkkäinen, Ismo. Methods for Fast and Reliable Clustering. University of Joensuu, Computer Science, Dissertations 13. 108pp. Joensuu, 2006.

Tedre, Matti. The Development of Computer Science: A Sociocultural Perspective. University of Joensuu, Computer Science, Dissertations 14. 502pp. Joensuu, 2006.