Lecture Notes in
The Philosophy of Computer Science

Matti Tedre

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

This .pdf file was created February 5, 2007.
Available at cs.joensuu.fi/~mmeri/teaching/2007/philcs/

2.2 Emerging Interdisciplinarity: 1970s-1980s

In the beginning of the 1970s, the *users* of computers were still chiefly *computer programmers*. Human-computer interaction (HCI, or CHI) was not yet a research field of its own, but there was significant research on the psychology of computer programming and programmer-computer interaction (Baecker et al., 1995:41). Between 1965 and 1975 integrated circuit electronics plummeted the cost of computer power by a factor of a hundred (Campbell-Kelly & Aspray, 2004:198). The reduced costs led to the computer breaking out of the laboratory, which changed the user base dramatically. Whereas the old user base consisted of highly trained specialists, now people with increasingly diverse backgrounds were able to get their hands on computers. Although personal computing was originally pioneered by electronics hobbyist groups (Ceruzzi, 1999), a growing number of new users of computing technology were no longer committed to the technology per se.

The uses of computers diversified as the user base of computers diversified. By the same token the academic field of computer science diverged. The divergence of computer science has by no means been an uncontroversial process. Debates of whether computer science is a part of mathematics or engineering were gradually replaced by debates around the legitimation of branches of computer science. The valuation of some branches of computer science at the cost of some other branches is well-visible in the debates about the aims, content, and form of computer science—and even in the naming of computer science.

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 academic discipline of computing throughout the existence of the discipline; for instance, there are topics of computing such as programming languages, operating systems, human-computer interaction, and software engineering that are considered to belong to academic discipline of computing today, but that were once rejected. Computing as a discipline has 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. In this section some changes in the disciplinary identity of computing during the 1970s and 1980s are traced.

2.2.1 Shifts in User Base, Status, and Content of Computing

The psychology of programming was a hot topic especially in the 1970s and 1980s (a groundbreaking work was Gerald M. Weinberg's 1971 book *The Psychology of Computer Programming*; see Weinberg, 1971). The psychology of programming studies the psychological aspects of different kinds of programming activities, and the topics span a vast range of issues such as program comprehension and debugging, all phases of software development and maintenance, individual differences between programmers, collaboration and teamwork in programming, programming language paradigms, and teaching programming. However, as programmers became a smaller and smaller fraction of computer users, the psychology of programming became less central to the studies of interaction between humans and computers (Baecker et al., 1995:41)[5].

5 Although HCI, which is a broader topic than the psychology of programming, gained more visibility and attention during the 1980s and 1990s, the psychology of programming is still a strong branch of HCI. It seems that the focus

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 human-computer interaction, management information systems and information systems, operating systems, and networks were born and the field of computing was again stretched in a number of directions. Human-computer interaction is an innately interdisciplinary undertaking, including fields such as design and ergonomics, psychology, sociology, æsthetics, and cognitive science (e.g., Myers et al., 1996; Carroll, 2003:2-9). Interdisciplinarity of HCI has arisen debates about the scientific value of interdisciplinary and multidisciplinary science, as well as if HCI can be considered to be a distinct discipline or a part of computer science. The field of management information systems has gone through similar, serious crises of disciplinary identity (Banville & Landry, 1989).

> MULTIDISCIPLINARITY AND INTERDISCIPLINARITY
>
> *Multidisciplinary approaches* to science offer a variety of perspectives of the same phenomena by employing research approaches from a number of disciplines. In multidisciplinary approaches, the different research approaches are utilized without modification. Multidisciplinary approaches do not create a new field. *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), quantum computing (modern physics, computer science, and material science), and computer science.

The media had, in the early days of computers, portrayed computers to the general public with phrases such as "*the robot Einstein*" and the "*tireless ally of science*" (Bowles, 1996; see also Martin, 1993). But as early as in 1959 some visionaries had already understood that computers do not just replace human computers[6], but that they offer some unprecedented opportunities—for instance, Herb Grosch wrote in *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.*
>
> (Grosch, 1959)

Yet, it still took some fifteen to twenty years before researchers at large stopped regarding computers as merely giant calculators. Towards the end of the 1970s an understanding that computer science is indeed an interdisciplinary science, and not easily definable as such, emerged. In the *Second International Conference on Software Engineering*, Peter Wegner presented his three-fold characterization of computing:

---

is nowadays turning to *end-user programming*, *i.e.*, those kinds of commonplace activities that bear the characteristics of programming activity, such as recording and editing macros in Microsoft Word or setting the timer in VCRs (see the special issue on *end-user development tools* in the September 2004 issue of *Communications of the ACM*). Perhaps the fraction of computer users who are programmers (end-user ones) is increasing again?

6 The term *computers* meant originally people who do computations.

> [1976] *Computer science is in part* [1] *a scientific discipline concerned with the empirical study of a class of phenomena*[...]*, in part* [2] *a mathematical discipline concerned with the formal properties of certain classes of abstract structures, and in part* [3] *a technological discipline concerned with the cost-effective design and construction of commercially and socially valuable products.*
>
> (Wegner, 1976, underlining in original)

Peter Wegner's description in the passage above does not emphasize only one viewpoint of what computer science is. Wegner drew content from connections with empirical, mathematical, and engineering traditions. Later in these lecture notes a similar tripartite by Denning et al. is discussed in more detail.

Wegner extended the "phenomena" of computer science to include computers, languages, programs, and "*other man-made* [sic] *entities and concepts which owe their existence to the development of computers*", which could be interpreted as an open invitation for interdisciplinary studies. Under a somewhat loose interpretation of Wegner's description, studies of things such as flame wars, hacker ethics, netiquette, the information revolution, and cybercrime would all belong to the field of computer science. Such description of computer science would be similar to the classic Newell et al.'s 1967 description of computer science as the "*phenomena surrounding computers*" (Newell et al., 1967).

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 all-inclusive normative accounts of computer science. Wegner, for one, interpreted Newell et al.'s *phenomena* as algorithms, programs, programming languages, and such. Sociocultural phenomena are not explicitly or (clearly) 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 explicitly stresses an interplay of empirical research, formal theories, and design and implementation, all of which are nowadays generally considered to be proper parts of computing as a discipline.

2.2.2 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.*
>
> (Hamming, 1969)

The changes in attitude were soon reflected in computer science education. In 1972, when computers were not yet available to the general public, Horowitz, Lee, and Shaw proposed a *Computers and Society* course for computer scientists (Horowitz et al., 1972). They concluded that if research is done without concern for its influences on all aspects of society, the research can become a destructive rather than constructive mechanism. Horowitz, Lee, and Shaw gave examples of technolo-

gies that have had both positive and negative effects in their application, such as the automobile, nuclear power, and pesticides. The content of their proposed course is broad and it concerns issues that are troubling today, too (including political, economic, cultural, social, and moral issues).

Computing publications such as CACM had had extensive debates over social responsibilities of computing professionals starting as early as the 1970s. For instance, in 1973 the pros and cons of ethical code of conduct for computing professionals was discussed in CACM, in 1974 social and political questions kept the CACM *Forum* busy, and in 1979 human rights and scientific freedom were at issue. In the official computing curricula the status of social, philosophical, and ethical con – siderations changed during the years between 1968 and 1978. Whereas the '68 curriculum stated that "[social] *issues are not the exclusive or even the major responsibility of computer science*", the '78 report regarded social, philosophical, and ethical issues "*of such importance to computer scient – ist that they must permeate the instruction at this level*." (Atchison et al., 1968; Austing et al., 1979). The emerging trend of examining the relationship between computers and society was evid – ent in Khalil and Levy's 1978 example curriculum, too. They wrote, "*In the spirit of our times, an awareness of the social issues and controversies is also important for the well-educated person*" (Khalil & Levy, 1978).

Although social and ethical concerns have probably been a feature of technological change ever since the Ancient Greeks (Mitcham, 1994:277), in the early days of modern computing social and ethical issues were not considered to be a concern of computer scientists. One of the pioneers of so – cial analyses of computing was Rob Kling (1944-2003), who argued that the question of *what com – puters do to people or societies* is misleading. He argued that computers do not do anything to any – body by themselves. Instead, "*computing is selectively adopted in a given social world and organ – ized to fit the interests of dominant parties*" (Kling, 1980). Hence, Kling implied that the phrase "social impacts of computing" might be invalid. The consequences of computing follow from what people use them for; and when situations, purposes, goals, aspirations, or other things in the socio-technological setting change, the consequences also change. To speak of "impacts" is, according to Kling, distracting people from the social processes of developing, adopting, and using new techno – logies.

Many modern sociologists agree 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 society (economical and material conditions) define the super – structures of society (social and cultural characteristics). Modern sociologists, however, disagree, and indeed argue that technology is not the cause of social change at all. For instance, John Fiske wrote, "*New technologies do not in themselves produce social change, however, though they can and do facilitate it*" (Fiske, 1994).

Technologies have been claimed to be one factor in social change, but changes take place only when combined with organizational, social, and economic adjustments (Florida, 2003:26). 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 (Castells, 2001:39). Machines as such do not actually do anything; unlike humans, they are not intentional. People construct the social and political characteristics of a soci –

ety, often assisted by machinery[7]. One could rephrase Robert Heilbroner (Heilbroner, 1994) and argue that new computing technologies can be used to change the material conditions of people's lives, which may affect or induce social and cultural changes.

2.2.3 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 (Minsky, 1979). 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.*
>
> (Minsky, 1979)

However, Minsky did not give a central role to 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 (Minsky, 1979). Similar, philosopher of science Paul Feyerabend argued that there is not a single universal logic that underlies all the domains of nature, perception, the human mind, and society. Feyerabend wrote: "*There is Hegel, there is Brouwer, there are the many logical systems considered by modern constructivists*" (Feyerabend, 1993:197). Many attempts to find a universal logic have been made, but Minsky set the bar for fixed logic high: All one's beliefs and knowledge must be made self-consistent, and that is "*essentially impossible to achieve*". 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:

> [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.*
>
> (Minsky, 1979)

Edsger Dijkstra's writing in the June 1974 issue of *American Mathematical Monthly* (Dijkstra, 1974) may have contributed to how Minsky regarded computer science; Dijkstra had written 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 elaborated his position further in a polemical article published in the summer 1987 issue of *Abacus* (Dijkstra, 1987), in which he made a case about the uniqueness of comput*ing* science. Dijkstra wrote that complexity can be seen also in the ratios between the time grains of phenomena in computing:

7 This topic is discussed in Eskelinen, Teppo; Tedre, Matti (forthcoming) *Three Dogmas of Technology-Driven Development*. (Working title for an article manuscript)

> [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.*[8]
>
> (Dijkstra, 1987)

Herbert A. Simon, who is a pioneer in artificial intelligence, argued that when dealing with hierarchical abstraction levels, subparts belonging to larger aggregates also interact in an aggregate fashion. He wrote in his seminal 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.*
>
> (Simon, 1981:64)

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 (in a manner that we understand thoroughly). The entities on each level of abstraction in computer science at large work in a trivial manner. For instance, the working of logic gates in a computer is straightforward. The assembly language instructions are simple and straightforward. The instructions in a high-level programming language, such as the C language, are simple and unambiguous. Computer science is a discipline (though not the *only* discipline) that makes good use of layers of abstractions.

The reason for dealing with abstractions is that if certain things are constructed into abstracted entities, one does not need to consider fine-detailed workings of small subparts. Good abstraction hides details that are not necessary for the task at hand. When abstracted, larger entities can be used without needing to know anything about the small-scale interactions. Take, for instance, the following code, which swaps two elements in an array using a temporary variable:

```
procedure swap(A : Array of Element; i,j : index);
      var temp : Element;
      temp:=A[i];
      A[i]:=A[j];
      A[j]:=temp;
```

Consider also the following code snippet, which swaps two elements in an array without using any temporary variables:

```
procedure swap(A : Array of Element; i,j : index);
      A[i]:=A[i] XOR A[j];
      A[j]:=A[i] XOR A[j];
      A[i]:=A[i] XOR A[j];
```

Swapping elements in an array can be implemented in different ways, but the implementation itself does not matter when swapping is abstracted into a function and used in its abstracted form:

8 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). Dijkstra might have meant that unlike physicists, computer scientists deal with all those levels within the same project.

```
procedure sort(A : Array of Element);
      int i,j;
      for i:=sizeof(A) downto 1
            for j:=1 to i-1
                  if A[j]>A[j+1] swap(A,j,j+1);
```

Now when a programmer needs the swap routine, he or she does not need to consider how exactly is swap implemented.

However, it is hard to define the boundaries of abstraction layers in computer science. It is in prin - ciple possible to construct a whole *computer program* from *microinstructions*, or at least from *ma - chine instructions*. In fact, the very distinction between hardware and software is artificial and vague—most of software can be implemented as hardware and much of hardware can be implemen - ted as software. However, if the computer program is not a very simple one, constructing a pro - gram from machine-level instructions is an enormously difficult and time-consuming task. So usu - ally there are a number of abstraction layers between microinstructions and executable programs: for instance, *machine instructions*, high-level *statements* and *expressions*, *statement blocks*, *sub - routines*, and *modules*. The complexity of computer systems is due to the complex ways in which different abstractions interweave on each abstraction level and between each abstraction level.

There are two basic ways of interpreting the source of complexity and clarity in the domain of com - puting. First, one can argue that computer science studies things that have a naturally hierarchical order, and that all complexities, irregularities, or difficulties in computer science arise from a lack of understanding of that inherent order. Secondly, one can argue that there is no inherent structure of the world, and that whatever structures are used in computer science, those structures are an at - tempt of computer scientists to make sense of the computational world. That is, in order to under - stand computation or just to get automatic computation work, computer scientists make hierarchical taxonomies of the world. Those two interpretations are discussed later in this course, in context of *ontology* and the debate between *inherent-structurism* and *nominalism*.

The experience people generally have with computer systems speaks against Dijkstra's early claim that computer scientists switch “agilely” between global and local considerations and between mac - roscopic and microscopic concerns (Dijkstra, 1974). Dijkstra's later text shows that by the year 2001 he did not believe it either—Dijkstra wrote that computer scientists should be deeply ashamed about the bug-ridden software of today (Dijkstra, 2001).

Computer scientists seem particularly deft at mastering complexity in strictly bounded realms (mi - crocosmoses), such as single algorithms, language definitions, and specialized hardware parts. However, as bugs in nearly all hardware, software, and especially computer systems show, com - puter scientists invariably fail at coalescing a number of microcosmoses into coherent systems (macrocosmoses), such as operating systems or computer networks.

2.2.4 Artificial Intelligence

The 1960s and 1970s saw also the large-scale emergence of artificial intelligence as a branch of computer science. Vernon Pratt wrote that 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 lo - gic* from symbolization to Boolean logic to predicate calculus, and (3) the development of *computa -*

tional instruments through Babbage's mechanical tools to the modern computer (Pratt, 1987:2). AI has also deep roots (at least to the 17th century) in a number of disciplines, and earlier there were some debates about whether AI is a part of computer science, of philosophy, of psychology, or of something else. AI is indeed a vast and interdisciplinary field, and it includes topics such as pattern recognition, heuristics, and adaptive systems as well as strong philosophical basis, especially in the philosophy of mind and, certainly, in the philosophy of artificial intelligence. As an interdisciplin – ary endeavor, AI brings new disciplinary viewpoints into computer science. For example, psycho – logy, linguistics, and philosophy are intertwined with computing in AI.

Pratt argued that although AI has a long history there is no continuum, plan, or story in this history, but a number of projects, each different. Some of those projects have been successful, some abort – ive, all for different reasons (Pratt, 1987:1-7). In retrospect, Frederick P. Brooks, Jr., 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 (Brooks, 1996). And in Brooks' opinion, more serious than the wasted dollars, was "*the diversion of the very best computer science minds and* [...] *academic laboratories*." (Brooks, 1996).

Although theories, concepts, and methodologies from psychology, linguistics, and philosophy are used in AI, AI is generally not considered to be subsumed under psychology, linguistics, or philo – sophy. Those disciplines are perhaps *auxiliary disciplines* that are used in research of AI. In a very similar manner, 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.

However, the fact that studies of AI may also contribute to the disciplines of psychology, linguist – ics, or philosophy is not substantive in the identity of the field of AI—AI can have a unique discip – linary identity regardless of its intellectual origins and where its results are applicable. The relation – ship of disciplines and auxiliary disciplines seems to be crucial in the debate of the disciplinary identity and character of computer science. This topic is discussed later in this course.