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 March 11, 2007.
Available at cs.joensuu.fi/~mmeri/teaching/2007/philcs/

2.3 Recent Discourses: 1990s-2000s

In the beginning of the 1980s personal computers had spread to many homes in the Western countries. Interface design specialists were turning their focus towards understanding the context of computer use, the goals and activities of end users, and the group interactions of end users (Grudin, 1990). Since the 1990s the concept of *human-centered computing* has gradually developed. Also the attention devoted to studies of the social implications of computing has continued to increase. A new understanding of the importance of application areas to the development of computer science has also arisen. In this section, the development of the discipline of computing since the mid-1980s is traced.

2.3.1 The Denning Report

In the 1980s, computer science was diversifying rapidly, and the old debate about the essence of computer science continued. In 1981 a famous report *A Discipline in Crisis* (also called *The Snowbird Report*) was published (Denning et al., 1981). The crisis depicted in the Snowbird Report was the severe manpower shortage in the computing field, especially at the PhD level. Respectively, computer science was in the Snowbird report connected strongly with aspirations towards the computer revolution, technological society, and information processing industry. In the Snowbird Report, computer science was characterized as the study of representation, transformation, nature, and philosophy of information. Computer science was elevated to the status of a *core science*:

> [1981] *Computer Science studies the processes of information flow and transformation that underlie many professions, such as medicine, economics, business, social sciences, physical sciences, life sciences, and engineering. Like mathematics, it is an indispensable tool. It is a core science whose influence is spreading across all of society.*
>
> (Denning et al., 1981)

The authors of the Snowbird Report noted that computer science is both a theoretical and an experimental science, and in that way similar to the natural sciences. They also argued that computer science is an indispensable tool in other disciplines, and in that way similar to the mathematical sciences. After its publication the Snowbird Report was quoted widely and accompanied by a number of similar views (see especially Traub, 1981: *Quo Vadimus*: *Computer Science in a Decade*).

In the spring of 1985, the ACM and the IEEE Computer Society formed together a task force to describe the intellectual substance of the computing field in a “new and compelling way” (hereafter referred to as *Denning's task force*). The report, finished in 1989 and named *Computing as a Discipline* (hereafter referred to as the *Denning Report*), defined computing as a discipline as:

> [1989] [... T]*he systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, “What can be (efficiently) automated?”*
>
> (Denning et al., 1989[9])

9 The Denning Report was published in the January 1989 issue of *CACM* and February 1989 issue of *IEEE Computer*.

Denning's task force noted that in the *discipline of computing* (which includes "*all of computer science and 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 disciplinary history of computing.

In addition to the short definition above, the task force characterized *computing as a discipline* as the intersection of three major research traditions: *theory*, *abstraction* (modeling), and *design*. The Denning Report includes the following characterizations of each major research tradition (verbatim quotes; emphases added):

- "*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." (Note that design deals with *artifacts* which are created by people, as opposed to naturally occurring things.)

In the Denning Report it is argued that many debates about whether computer science belongs more to one of the above mentioned traditions 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 revealed that the three traditions 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 (Denning et al., 1989). The task force noted that although the three traditions are inseparable, they are distinct from one another because they represent separate areas of competence. Although design is sometimes considered to be a non-academical subject, it has been argued that without the "*science of design*", many of the opportunities of computing would be thrown away (Freeman & Hart, 2004).

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 the Purdue University's first computing curriculum in 1962—computer science in Purdue was divided to *theory*, *numerical analysis*, and *systems* (Rice & Rosen, 2004). Those 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* (Wegner, 1976; see p.40 of these lecture notes). Finally, it should be noted that having these three intertwined but inseparable aspects to form a discipline is not unique in sciences—it can easily be argued that, say, quantum mechanics is also a combination of mathematical, scientific, and engineering aspects. Denning Report is discussed into more detail later in these lecture notes.

2.3.2 A New Era of Growth

By the late 1980s, the discussion about the disciplinary identity of computer science had still not 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 (Dijkstra, 1987). 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.*
>
> (Dijkstra, 1987)

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 was that those to whom Dijkstra referred as computing scientists were already 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 the absence of bugs. Note the similarity of this debate to the debate between logical positivism and falsificationism (positivism and falsificationism are discussed into more detail later).

POSITIVISM VS. FALSIFICATIONISM

In the beginning of the 20th century philosophers of science were generally concerned about how to *prove* scientific claims. For instance, the *inductivists* would use observation and experimentation in trying to find universals: "(Black-box) 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 come to a universal conclusion: "Program p produces correct output with any input x".

In 1934 Karl Popper (1902-1994) seized a fundamental problem in the inductivist thinking: There can always be a special case where the claim turns out to be false. Hence, Popper argued that there could not be a way to prove universal truths, yet observations could be used in *falsifying* statements (Popper, 1959). If the statement is "program p produces the correct output on any input x", then one single observation of erroneous functioning can falsify the statement. A falsificationist never says that a theory is true, but a falsificationist can say that a theory is the best theory currently available—since it has not yet been successfully falsified.

Dijkstra argued that comput*ing* scientists did not face the problem of iterative design because computing scientists' programs have been rigorously proven correct (Dijkstra, 1987). Note, however, that human errors can occur in any part of the program-construction process. Proving a program correct does not mean that the resulting program corresponds to the proof, that is, that the transla -

tion from proof to program was done correctly. More importantly, proving a program correct does not guarantee that the program actually does what it is supposed to do, or that the model corresponds to the thing it is supposed to model. This topic will be discussed later in connection with the relationship of *programs*, *algorithms*, *models*, and *the world*.

So Dijkstra wanted to distance the computing scientist from the real world—he wanted computing scientists to work with the abstract aspects of computation. But Dijkstra also criticized software engineers for making a mess out of programming. In Dijkstra's ideal world, there are no bugs because programs are proven correct. It seems unfair that Dijkstra criticized other practitioners of making a mess of a job Dijkstra was not ready to undertake. By denying the link between computing and the surrounding world, Dijkstra only foisted the problems upon others. The moment one wants to utilize computing scientists' programs, someone needs to connect the program with whatever task it is supposed to achieve; convert the program to some programming language (which immediately risks bugs in the program); and take the executable program into the unpredictable, uncontrollable, uncertain, inexact, and ambiguous physical world.

Even though Dijkstra's critique is valid to some extent, the practice of computing did not develop as Dijkstra wanted it to. The field of computing was already, at the time of Dijkstra's writing, much broader than Dijkstra's vision of the discipline of computing. 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. Hopcroft raised a concern that the scientific base of computer science must not fall 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.*
>
> (Hopcroft, 1987)

Hopcroft recognized that there is a dualistic relationship between computer science and the fields that utilize computer science as a tool:

> [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.*
>
> (Hopcroft, 1987)

Hopcroft saw that *application areas* were the key to a new era of growth of computer science. His vision was that ideas and methodologies from disciplines other than computer science can contribute to computer science. That is, he believed that *extending the discipline of computing with perspectives from other disciplines is beneficial to computing*—Hopcroft expected it to lead to a new era of growth of 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, "*It is now our responsibility to formulate a new vision, to shape the goals for the next generation of researchers*." (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*.". (Compare Hopcroft's dream to Herb Grosch's early vision of computing on p.39 of this handout.)

Interestingly, even though Hopcroft wrote that computers had penetrated almost every aspect of life, including medicine, education, and economics, and even though Hopcroft predicted that computers would alter the people's lives even more dramatically in the future, Hopcroft's "commitment" did not include responsibility or accountability for any of those 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. Then again, it is not certain if computer scientists *should* be responsible for what kinds of computer science they make. This is discussed later in this course.

2.3.3 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*" (McKee, 1995). (Note that McKee did not further discuss the concept of *science*.) McKee argued that the term *computer science* should be replaced with something 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.*
>
> (McKee, 1995)

McKee argued that mathematicians have acknowledged the nonscientific nature of mathematics by choosing the name of the field to end with "-ics". Whether computer science resembles science, mathematics, or something else is one issue, but considering the naming of other established fields of scientific inquiry, McKee's critique seems confused. Very few natural sciences include the term science in their name (cf. chemistry, geology, astronomy), but some 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 laws of nature and things that exist regardless of human beings, but linguistics and economics deal with constructed realities and phenomena that owe their existence to people and societies.

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*." (Brooks, 1996). By Brooks' criterion, physics, chemistry, history, and anthropology *may* be sciences or not; 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 asked, in Shakespeare's words:

> [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.*
>
> (Brooks, 1996, emphasis in original)

The first (1) 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.

The "pecking order" is currently implied in academical terminology at large, not only in the naming of computer science. The juxtaposition of *hard science* and *soft science* is particularly tense; hard science with its quantitative results is often valued over soft science with its qualitative results. Hard science is usually seen as objective, rigorous, and precise, contrasted with soft science that is often seen as descriptive, interpretive, and faithful to the subtlety and complexity of reality. No matter how arbitrarily the terms *hard science* and *soft science* are used, how philosophically unsound the distinction is, or how little either term tells about quality of any particular study, the term *soft science* has often been used in a derogatory sense and the term *hard science* in a laudatory sense, at least by hard-line natural scientists.

Another juxtaposition is that between *pure science* and *applied science*. Pure science has the sound of being free of impurities that applied science contains. However, although the common connotations of the term *pure science* may be a mixture of descriptive aspects (when the term refers to a set of activities or ways of conducting research) and normative aspects (when the term implies that pure science is intellectually superior to applied science), the names are not the crux of the matter. It is difficult to see how the pecking order that Brooks mentioned would cease to exist, no matter how *computer science* 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 and between hard and soft sciences.

The second (2) unhappy trend that Brooks mentioned comes from the fact that in the natural sciences, the discovery of facts and laws (however facts and laws may be conceived) is taken as an end itself. Brooks claimed that a *new* fact, 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. (Note that Brooks coupled *computers*, *algorithms*, and *languages* with *laws*, although it is not really certain if computer scientists in reality associate things such as programs, computers, and languages with things such as theories, laws, and facts.)

At first it seems that Brook's second unhappy trend is well alive. Publications in computer science very often include the word *novel* in their title. "*Novel approach to*" is perhaps the most oft-used opening of conference paper title in computer science (barely winning "*Towards a...*"). This is troubling because *novelty* does not have any value as such. Insofar as the title should tell what is worthwhile or interesting in an article, the abundance of articles that advertise *novelty* is a sign of Brook's second unhappy trend. (Perhaps if Turing had lived today, instead of naming his seminal article "*On Computable Numbers, With an Application to the Entscheidungsproblem*", he might have named his article "*A Novel Perspective to Automatic Computation and Decision Problem*").

There again, Brooks' view of "*the computer scientist as a toolsmith*" seems to be equally appropri-ate description of the field today. The trends towards *human-centered* or *value-centered* 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 end per se.

The third (3) 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 abstrac-tions may have left the essence of the real problem behind. Brooks wrote that the surfacing of eso-teric vocabularies is a sign of this trend.

Brooks' concern about the distance between scientists' work and everyday problems is understand-able. Increasingly esoteric vocabularies indeed pose a problem to computing, or at least those vocabularies are symbols of the distance between the domain of computing and the domain of everyday problems. But there again, if computer science deals with concepts that are incommensur-able with older concepts (concepts that are radically different from any older concepts), there just might not be a way to avoid esoteric vocabularies. In addition to Brooks' concern, the theft (or at least an unauthorized use) of established intellectual terminology, such as *ontology* and *agency*, is another embodiment of this unhappy trend.

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 (Knuth, 1991). Knuth wrote that theory and practice in computer science are more intimately connected than they are in any other discipline—"*They live together and support each other*". In his article, Knuth criti-cized *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 mathem-atics 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 be-comes sterile.*
>
> (Knuth, 1991)

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*". Sim-ilar, Knuth advised those computer professionals who spend most their time with practical things to start turning some of their attention to theoretical things—"*It will improve your practice*".

The fourth (4) unhappy trend that Brooks mentioned is that if computer scientists honor the more mathematical and abstract parts of computer science more, and the practical parts less, young and brilliant minds will be "misdirected" away from challenging and important practical problems (Brooks, 1996).

In the light of the history of computing machinery—and insofar as practical matters and abstract matters actually are equally important—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 (Bowles, 1996). In the U.S., where the engineer was hailed as the hero of the new century, 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 (Bowles, 1996; Kevles, 1987:293). Brooks' fear was that if an unreserved valuation of theory over practice were to come true on a large scale, field of computing could stag - nate. (Note the opposite views too—for instance, Dijkstra wrote that computing professionals should seek to fight the shoddy quality of software by concentrating on formal, theoretical meth - ods.)

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 inven - tion and publication of endless varieties of computers, algorithms, and languages as an end; (3) for - getting the users and their real problems; and (4) directing young and brilliant minds towards theor - etical subjects.

To resist those 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 (Brooks, 1996). First, (1) "it aims [the computer scientist] at *relevant problems*, not just at exercises or at toy-scale problems". Second, (2) "it keeps [the computer scientist] *honest about success and failure*", so that he or she does not fool himself or herself easily. Third, (3) "it makes [the computer scientist] *face the whole problem*, not just the easy or mathematical parts". Fourth, (4) "facing the whole problem in turn forces [the computer 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. Although Brooks did not consider ethical issues, it might be that work - ing on the problems of another discipline can make the computer scientists more sensitive to the ethical problems that their work may entail. Brooks' fifth reason is that (5) working in a field other than computer science is *just plain fun*.

2.3.4 A New Species Among the Sciences

The 1993 Turing Award winner, Juris Hartmanis, had taken an opposite view to the view of Brooks. Whereas Brooks wrote that computer science is not a proper science, Hartmanis argued that com - puter science differs so fundamentally from the other sciences that it has to be viewed as *a new spe - cies among the sciences*, especially because it deals with human-made phenomena that are explored by human-made paradigms and methods (Hartmanis, 1994). Hartmanis wrote that computer sci - ence 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*. It is uncertain if Hartmanis advocated a dualist view of the world or not; later in this course monist, dualist, and pluralist ontologies are discussed.

The difference between Brooks and Hartmanis' views is mainly due to their different views 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*" (Brooks, 1996), 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*" (Hartmanis, 1994). 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 to do things that were earlier thought to be impossible or unfeasible.

From the viewpoint of philosophy of science, Brooks' view of science as *verification* was flawed (although perhaps one should not read Brooks' argument for verification strictly as a positivist standpoint. The demise of positivism is dealt with later in these lecture notes.) But Hartmanis' view of science in his lecture has also faced criticism. While Hartmanis regarded computing as a new kind of a science, N.F. Stewart responded Hartmanis by writing that computer scientists should strive to make computer science similar to the natural sciences (Stewart, 1995). 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* (Loui, 1995). In addition to Hartmanis, Loui, and Stewart, also Marvin Minsky had 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.*
>
> (Minsky, 1979)

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 conflicting 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 axiomatic fields like mathematics, natural sciences like physics and chemistry, technical fields like electrical engineering, and life sciences such as brain research and human genome research, are also very significant (Minsky's view) (cf. Gal-Ezer & Harel, 1998).

2.3.5 Computational Science

One of the newer topics that bring together theory and practice in computing is computational science. In the year 1993 computational science was one of the emerging topics that were searching for their place in the scientific world (cf. Stevenson, 1993), but in 2003 the topic was included in the list of core technologies of computing (Denning, 2003). According to D.E. 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.*

(Stevenson, 1993)

According to Stevenson, students in traditional computer science can easily see computer science devoid of meaning and programming devoid of empirical import (Stevenson, 1993). 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 as well as everyday things that can be attributed to computational science; take, for instance, the Human Genome Project, SETI@home, and numerical weather prediction. Those projects and fields are not considered to be computer science, but none of them would be viable without computers. 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.

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 Journal on Scientific Computing*, deal with a wide variety of topics, most of which are closely related to topics commonly considered to be computer science.

2.3.6 What's In a Name? (Part II)

Even very recently computer science has been defined through 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.*

(Brookshear, 2003:1)

But defining a discipline by listing its constituents is problematic—there are probably as many listings as there are computer scientists. Another problem with lists of topics as a definition of a discipline is the assumption that all the topics are definable themselves. In addition, 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 some 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". Because of these problems, Brookshear's definition of computer science has little normative value. Although it is close to impossible to conclusively define the field of computing by making a list of topics, listing the constituents of a field can help to understand one person's standpoint.

Another use of lists of topics, as controversial as lists of topics may be, might be to give a snapshot of the image of the field at a certain point of time. For instance, although Peter J. Denning's 2003 list of 30 *core technologies of computing* (Denning, 2003; see Figure 2) can be debatable today, it helps to elucidate Denning's viewpoint and writings on computing in general, and may perhaps serve in the future as one characterization of the field.

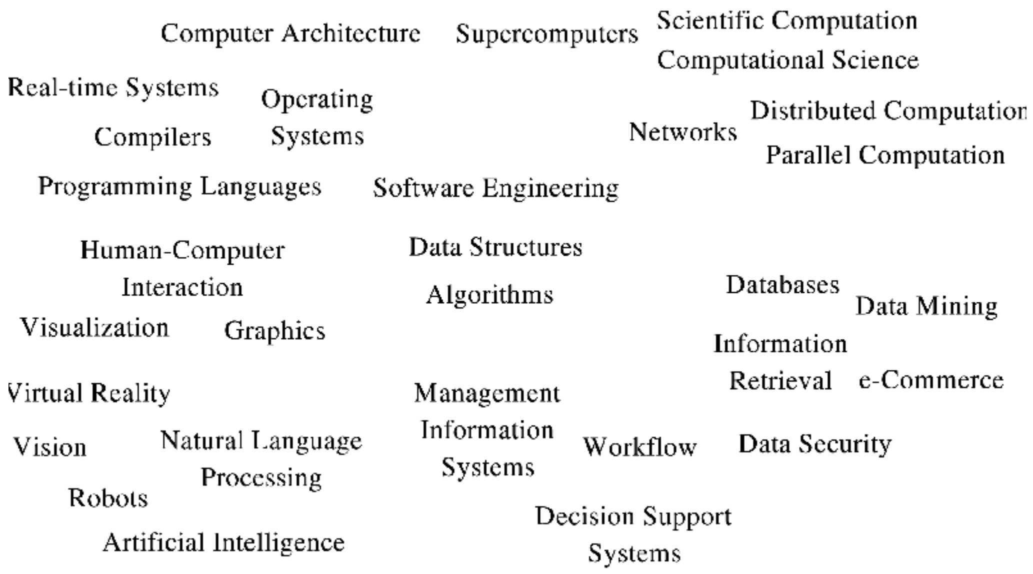

Figure 2: Core Technologies of Computing

Different core technologies are highly interconnected, so every researcher would most probably come up with a chart that looks different—and one researcher can make very different maps depending on the point of view he or she chooses to take[10]. Some computer scientists might exclude some technologies from Denning's list and include some additional technologies. For instance, some researchers might want to include *educational technology* in Figure 2 or question the status of e-commerce as a *core technology*.

In his letter to editor in the March 1988 issue of CACM, Peter Kugel hinted back to the 1967 definition of computer science by Newell et al. when Kugel asked, "*Lots of people drive cars but that does not justify an 'automotive science'. Do computers justify a 'computer science'?*" (Kugel, 1988). Among other questions, Kugel asked whether computer science should be redefined, and wrote:

10 Denning's article listed core technologies; the arrangement is mine. However controversial the arrangement may be, a chart might here be preferred to a list of words (atomic elements), or to an ordered list (unidimensional map) because the additional dimension might give more room for imagination and intellectual associations.

[1988] *Perhaps our* [computer scientists'] *role is like that of logic in the medieval curriculum that was supposed to sharpen the mind. Perhaps we deserve 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".*

(Kugel, 1988)

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" (Hamming, 1969). It also resonates well with Brooks' above-mentioned *driving-problem approach*, which "keeps the computer scientist honest about success and failure" (Brooks, 1996).

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 his magnum opus *On the Origin of Objects*, Smith wrote that computation is not a topic of a single discipline, "*Computation is not a subject matter*" (Smith, 1998:73). Smith's argument was 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 (Smith, 1998:360). 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'?*

(Kugel, 1988)

[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.*

(Smith, 1998:74)

Earlier it was noted that Newell et al. claimed that "unlike tools like the electron microscope and the spectrometer, the computer cannot be said to be subsumed under any other science as an instrument" (Newell et al., 1967). But perhaps computer science will, after all, face the same fate as the science of microscopy—that is to say, perhaps computer science will in the future lose its status as a distinct academic and intellectual field and partly become subsumed under other disciplines, partly continue serving as a tool for other disciplines.

Towards the turn of the millennium, also Dijkstra gave some concessions to diversity in computer science. In the book *Beyond Calculation: The Next Fifty Years of Computing* (Denning & Metcalfe, 1997) Dijkstra wrote:

[1997] *Another thing we can learn from the past is the failure of characterizations like "computing science is really nothing but X," where for "X" you may substitute your favorite discipline, such as numerical analysis, electrical engineering, automata theory, queuing theory, lambda calculus, discrete mathematics, or proof theory.*

(Dijkstra, 1997)

Nonetheless, in 2001 Dijkstra still argued that there is a real problem with computer scientists' inclination to evaluate the progress of computer science by the things that computer science and technology has accomplished, yet to forget those goals that computer science and technology has failed to reach (Dijkstra, 2001). Apposite examples are easy to find: enormous amounts of resources have been spent on artificial intelligence, yet artificial intelligence research has frequently failed to meet the "within 30 years" predictions. Dijkstra argued that among the failed goals there are goals that are just too important to be ignored, such as the central challenge of computing "how not to make a mess of it" (Dijkstra, 2001).

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, other disciplines like Mathematics and Chemistry are no longer related very strongly to the etymology of their names.*

(Knuth, 1985)

In 1996, Frederick P. Brooks, Jr., asked rhetorically "*What's in a name? Much.*" (Brooks, 1996). 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 (Forsythe, 1968[11]). Also the public image of computing as a discipline, as well as its name, may have had an effect on the early development of computing, but there may be 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 (Forsythe, 1968:455).

In his famous book *Philosophical Investigations*, Ludwig Wittgenstein noted that although it is really difficult to come up with an all-inclusive definition for the term *game*, the term is still very useful (Wittgenstein, 1958:§64-67). A proper definition of the term should perhaps include, for instance, sufficient conditions and necessary conditions (see p. 12 of these lecture notes) for something to be considered a game. Consider, for instance, a definition of the term *game* that would include games such as chess, solitaire, football, Olympic games, Quake III, and catch. Perhaps the terms *science* and *computer science* are similar things—useful concepts that are just not easily definable (cf. Okasha, 2002:17).

11 A previous version of Forsythe's paper was published as Stanford's technical report CS-TR-67-77 (available online).