Know Your Discipline: Teaching the Philosophy of Computer Science

Matti Tedre
University of Joensuu, Dept. of Computer Science and Statistics
Joensuu, Finland

matti.tedre@cs.joensuu.fi

Executive Summary

The diversity and interdisciplinarity of computer science and the multiplicity of its uses in other sciences make it hard to define computer science and to prescribe how computer science should be carried out. The diversity of computer science also causes friction between computer scientists from different branches. Computer science curricula, as they stand, have been criticized for being unable to offer computer scientists proper methodological training or a deep understanding of different research traditions. At the Department of Computer Science and Statistics at the University of Joensuu we decided to include in our curriculum a course that offers our students an awareness of epistemological and methodological issues in computer science, and we wanted to design the course to be meaningful for practicing computer scientists. In this article the needs and aims of our course on the philosophy of computer science are discussed, and the structure and arrangements—the whys, whats, and hows—of that course are explained.

The course, which is given entirely on-line, was designed for advanced graduate or postgraduate computer science students from two Finnish universities: the University of Joensuu and the University of Kuopio. The course has four relatively broad themes, and all those themes are tied to the students' everyday work or their own research topics. I have prepared course readings about each of those four themes. The course readings describe, in a compact and simple form, the cruces of the topics that are discussed in the course. The electronic version of the course readings includes hyperlinks to a large number of articles that are available on-line. The course readings are publicly available on the course home page, and they are licensed under the creative commons license.

The first theme in the course is centered around a fundamental question—What is computer science? Students are introduced to the disciplinary history of computer science, to a number of characterizations of computer science made by the pioneers of the discipline, and to some methodological and epistemological viewpoints on computer science. The second theme is centered around the question—What is science? Students are introduced to, for instance, the concepts of pure and applied science, "hard" and "soft" sciences, the aims of science, the scientific method, scientific reasoning, the formation of scientific concepts and theories, and the Science Wars.

The third theme concerns the division of computer science into its theoretical, engineering, and empirical traditions. The lecture notes introduce the students to descriptions of computer science that emphasize the mathematical tradition

Editor: Jancie Whatley

over other traditions and to descriptions that emphasize engineering or empirical traditions. The fourth theme is the philosophy of science. Throughout the course terminology of the philosophy of science is used, and the students are introduced to a number of central issues in the philosophy of science, to some of the most notable schools in the philosophy of science, and to some critical views of science.

This course is aimed at providing a broad understanding of the different traditions of computer science, of the methodological differences between the branches of computer science, of the strengths and limitations of the different traditions in computer science, and of how the philosophy of science can be of help to computer scientists. In the course, critical reading and well-argumented writing are encouraged. The students learn that there are many problems that do not have clear-cut answers; they learn that there are many open problems where multiple incompatible, yet credible viewpoints can be defended. The students also learn to articulate their own positions, to defend those positions, to comment and criticize other positions, and to reflect and re-think their positions according to criticism. The students also get the chance to think about the intellectual foundations of their own work and their own research studies.

Keywords: philosophy of computer science, foundations of computer science, computer science education, course description

Background

Computer science is a relatively young discipline. Its birth can be traced to the 1940s, when wider academic interest in automatic computing was triggered by the construction of the first fully electronic, digital, Turing-complete computer, ENIAC, in 1945 and the concomitant birth of the stored-program paradigm (see, e.g., Aspray, 2000). It still took some 20 years for computer science to achieve a disciplinary identity distinct from fields such as mathematics, electrical engineering, physics, and logic (cf. Atchison et al., 1968; Rice & Rosen, 2004). Throughout the short history of electronic digital computing, there has been a great variety of approaches, definitions, and outlooks on computing as a discipline. Arguments about the content of the field, its methods, and its aims have sometimes been fierce, and the rapid pace of extension of the field has made it even harder to define computer science (see Tedre, 2006, pp. 255-351).

Over the last 60 years, researchers in the fields of computing have brought together a variety of scientific disciplines and research methodologies. The resulting science—computer science—offers a variety of ways for explaining phenomena; most notably it offers computational models and algorithms. The increased investments in research efforts in computer science have been paralleled by the growth of the number of computing-centered fields, such as computer engineering, scientific computation, electrical engineering, decision support systems, architectural design, and software engineering.

Although interdisciplinarity has made the development of computer science possible in the first place (cf. Bowles, 1996; Puchta, 1996; Williams, 1985, p. 209), it also poses a very real challenge to computer scientists. Firstly, it is not certain what kinds of topics should be considered to be computer science *proper*. The attempts to describe computer science are invariably either very narrow and applicable to only some subfields of computer science (e.g., Dijkstra, 1974), or so broad that they do not exclude much (e.g., Newell, Perlis, & Simon, 1967). Secondly, it is very difficult to come up with an overarching set of rules of how computer science research should ideally be done. The subjects that computer scientists study include, for instance, programs, logic, formulæ, people, complexity, machines, usability, and systems. An overarching set of rules for computer science research should cover research in fields such as software engineering, complexity theory, usability, the psychology of programming, management information systems, virtual reality, and architectural design. It is uncertain if an overarching, all-inclusive definition of computer science is possible, and if such definition is even necessary.

It is important for computer scientists to understand the challenges (and possibilities) that the vast diversity of computer science research can cause. Many disputes about how computer scientists *should work* have their roots in different conceptions about what computer science *actually is* (cf. Denning et al., 1989). Many misunderstandings and controversies between scientists from different branches of computer science might be avoided by their understanding the research traditions within which people in those branches work.

Even more importantly, computer scientists must know that the same approaches cannot be used with the whole variety of subjects that computer scientists study. 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 physical and social reality. Narratives and ethnographies are rich in dimension 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 has been argued that there are three particularly lucid traditions in computer science: the *theoretical* tradition, the *empirical* tradition, and the *engineering* tradition (cf. Denning et al., 1989). Already having those three research traditions in computer science raises some major problems. The variety of research approaches within and among those traditions might bring about ontological, epistemological, and methodological confusion. For instance, the theoretical (logico-mathematical) tradition and the engineering tradition entail different ontological views about their subjects of study, the empirical tradition and the theoretical tradition employ different methodologies, and the engineering tradition and the empirical tradition have different views about the epistemological status of their research results. It is notoriously difficult to conduct research in the intersection of research traditions without making a mess of it (e.g., Denzin & Lincoln, 1994, pp. 2-3, 99-100).

Some authors who have conducted meta-research in the field of computing (e.g., Glass, 1995; Tichy, Lukowicz, Prechelt, & Heinz, 1995) 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. Note that those authors do not actually argue that the analytical research tradition would be seriously flawed and alarming in *theoretical computer science*, where the analytical research tradition arguably suits best. What those authors seem to consider alarming is taking the analytical research tradition outside its conventional logico-mathematical and philosophical domains. That is, those authors argue that it is dubious to support scientific hypotheses only through forceful argumentation and anecdotal evidence, without any empirical testing.

In a similar manner, when Frederick P. Brooks Jr. (1996) warned that computer scientists should not confuse their products with *laws*, his concern was that computer scientists mix up engineering and natural science. His message was that if researchers follow the engineering tradition and aim at building things, then they should not confuse *novelty* as a valuable feature of a product (unlike the natural sciences where the smallest *new* findings about the world can be considered to contribute to knowledge about the world). In the engineering tradition, the value of a product can be measured in many ways (such as operationality, usability, cost-effectiveness, effectiveness, or efficiency) but how value is measured in the engineering tradition is different from how value is measured in the analytical tradition and in the empirical tradition.

Methods from the analytical/theoretical tradition can certainly be utilized in empirical research, methods of the empirical tradition can be utilized in engineering, and all other combinations of the traditions may turn out useful, too. Yet, some caution is necessary when one moves between traditions. For instance, one cannot formally prove either that an engineered product has the intended qualities or that an engineered product will not fail (these notions were the coup de grâce of the formal verification debate—see Fetzer, 1988; Smith, 1996[1985]). One can rarely empirically demonstrate the correctness of a theorem. Showing that a product can be built does not demonstrate its utility or any other qualities—that is, showing that it is possible to build something does not mean that it is feasible or necessary to build it. The computer scientist who mingles traditions or disciplines must know each tradition or discipline well, or otherwise the result may look flawed from the point of view of each tradition and discipline (Figure 1; cf. Denning et al., 1989).

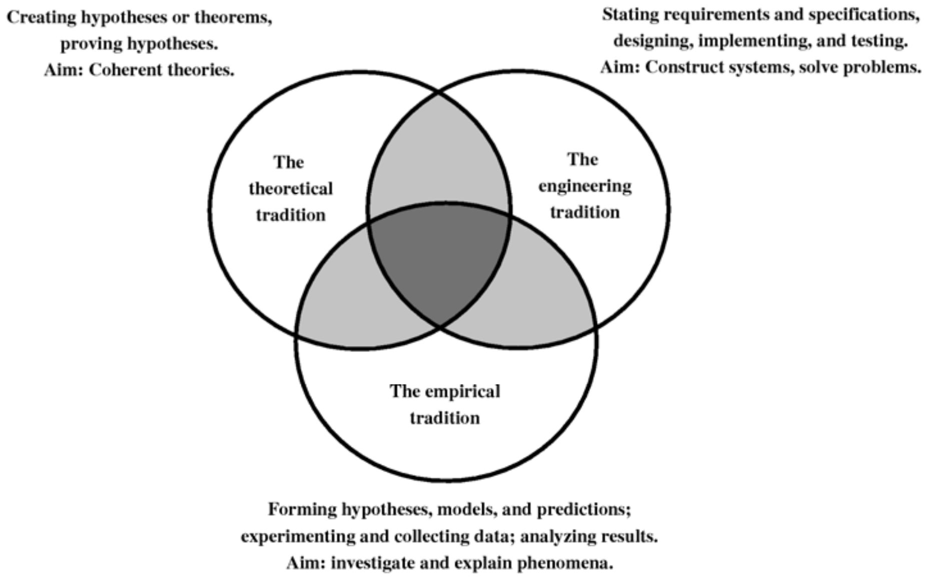
Creating hypotheses or theorems,
proving hypotheses.
Aim: Coherent theories.
Stating requirements and specifications,
designing, implementing, and testing.
Aim: Construct systems, solve problems.
The theoretical tradition
The engineering tradition
The empirical tradition
Forming hypotheses, models, and predictions;
experimenting and collecting data; analyzing results.
Aim: investigate and explain phenomena.

Figure 1: Overlapping Intellectual Traditions in Computer Science

Although mixing disciplines can potentially lead to problems, to cope with the gamut of topics in computer science, computer scientists often need to employ approaches and methods from a variety of fields. In this sense, computer scientists are expected to be *bricoleurs*, sort of academic jacks-of-all-trades. But usually one cannot adopt only one aspect of a paradigm. If one adopts, for instance, the methods from one paradigm, it is usually necessary to also adopt the standards, boundaries of applicability, validation procedures, and even the epistemological and ontological assumptions from that paradigm. If computer scientists choose to borrow from other fields only the parts they like, the resulting computer science may not be valid in any of the fields that are utilized. That can be a serious problem.

In addition to its intra-disciplinary diversification, computer science is increasingly applied in other fields such as biology, physics, chemistry, and even psychology, sociology, and anthropology. Algorithmic and computational models are utilized in an increasing variety of fields (Easton, 2006). It has been argued that this sort of versatility is an advantage for the development of computer science (Brooks, 1996). It is quite plausible that a better understanding of the assumptions, constraints, limitations, and premises of computer science can help computer scientists to accommodate computer science for the uses of other disciplines. It is also plausible that a better understanding of computer science can make it easier to utilize knowledge from other disciplines for the benefit of computer science. Understanding the strengths and weaknesses of one's own intellectual tradition can, at the very least, create a realistic image of one's discipline, and perhaps increase one's valuation of other intellectual traditions.

However, it is uncertain if computer science education can provide computer scientists the methodological understanding and the disciplinary understanding that interdisciplinary work requires. It has been 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" (Glass, 1995). From a curricula point of view, research methodology courses in a typical computer science curriculum are rare; for instance, the official ACM/IEEE curriculum recommendations (Denning et al., 2001) do not include a course on methodology, research design, or research paradigms.

In addition, it has been argued that computer scientists publish relatively few papers with experimentally validated results (Tichy et al., 1995). Other researchers have argued that some computing branches are seriously inbred (Glass, Ramesh, & Vessey, 2004; Ramesh, Glass, & Vessey, 2004). Even other researchers have argued that research reports in computing fields rarely include an explanation of the research approach in the abstract, key word, or research report itself (Vessey, Ramesh, Glass, 2002), which makes it difficult to analyze how computer scientists generally arrive at their results. Finally, despite the noted deficiencies in computer scientists' research training, practicing computer scientists utilize a vast array of methods in their work. The methodological diversity in computer science has been described by a number of authors (e.g., Alavi & Carlson, 1992; Choudrie & Dwivedi, 2005; Galliers & Land, 1987; Kitchenham, 1996; Lai & Mahapatra, 1997; Mingers, 2003; Muller, Wildman, & White, 1993; Ramesh et al., 2004; Randolph, 2007; Tichy et al., 1995; Vessey et al., 2002; Walsham, 1995; Zelkowitz & Wallace, 1997).

The charges above are serious. In other words, the charges are that (1) the official computer science curriculum does not include courses in research paradigms or research methodology; (2) many computer scientists learn their research skills from examining earlier research (mostly in computer science); (3) research papers in computer science often lack a description of methodology; yet (4) computer scientists utilize a large variety of methods and approaches (sometimes without having sufficient knowledge for using them).

These accusations could, justly, offend many computer scientists. Certainly there must be many academic institutions in which computer scientists are given proper training on research design, research paradigms, working in interdisciplinary fields, epistemological and methodological conflicts, and so forth. Certainly many computer scientists meticulously report their research methodology in their publications. And certainly, many computer scientists are knowledgeable about the particular research methodologies they utilize. Also, at the Department of Computer Science and Statistics at the University of Joensuu we have come to the conclusion that it is a proper part of a computer scientist's education to be aware of the epistemological and methodological issues in computer science. Hence, we have explicitly included, in our curriculum, a course that deals with those issues—issues that properly fall in the domain of *the philosophy of computer science*.

Teaching the Philosophy of Computer Science

In this section a number of approaches to the philosophy of computer science are described, a current problem of a lack of textbooks is discussed, and the curriculum of a course in the philosophy of computer science being taught in our department is outlined. The reader should note that the view of the philosophy of computer science presented here differs to a certain degree from the philosophy of artificial intelligence, from the philosophy of information, from computing ethics, and from the philosophy of the mind. The philosophy of computer science is a "philosophy *of* a specific discipline" in the same sense as the philosophy of physics, the philosophy of biology, and the philosophy of mathematics (e.g., Colburn, 2000. pp. 129-131; Shapiro, 2000, pp. vii). Those branches of philosophy deal with the questions and problems of specific academic disciplines. The concerns typically include ontological, epistemological, and methodological issues, as well as questions about the logic, ethics, and semantics of that discipline.

The Philosophy of Computer Science Elsewhere

The term *philosophy of computer science* is not well established. The term can be found in a number of places, in different meanings. One of the original characterizations of the term is by Timothy R. Colburn. Colburn (2000) dedicated the last part of his book *Philosophy and Computer Science* to the "philosophy *of* computer science", touching on issues such as the relationships between computer science and mathematics, engineering, and experimental science; the formal verification debate; levels of abstraction in computer science and the role of abstraction; and what kinds of issues the philosophy of computer science deals with. Colburn's book offers a fruitful starting point for the topic.

The January 1999 issue of *The Monist* (vol. 82, no. 1) was a special issue on the philosophy of computer science, and the articles in that special issue constitute an eclectic array of topics loosely connected under the umbrella term *philosophy of computer science*. The European Conference on Computing and Philosophy (E-CAP) has a track on the philosophy of computer science, and the extended abstracts available at the conference website offer an overview of the multiplicity of topics that are included under the track. Amnon H. Eden and Raymond Turner from the University of Essex maintain a web collection of online and offline resources concerning the philosophy of computer science (http://pcs.essex.ac.uk/). Also William J. Rapaport's home page for his course on the philosophy of computer science offers a good collection of online and offline texts (http://www.cse.buffalo.edu/~rapaport/584S07.html).

There are a number of good book-candidates for a course on the philosophy of computer science—books by prominent people in the field, such as Luciano Floridi (1999, 2004), Timothy R. Colburn (2000), Brian Cantwell Smith (1998), Terrell Ward Bynum, and James H. Moor (Bynum & Moor, 2000; Moor & Bynum, 2003). If one widens the perspective to the philosophy of computing in general, to the philosophy of information, or to the philosophy of artificial intelligence, the number of potential textbooks grows enormous. However, I concur with William J. Rapaport's (2005) opinion in that none of those books suffices as a textbook for a course on the philosophy of computer science. The books above are outstanding monographs and anthologies, but they are not appropriate textbooks for this course.

Several universities have offered courses under the title *philosophy of computer science* or something similar. In William J. Rapaport's reading-intensive course at the State University of New York at Buffalo, USA, Rapaport preferred original articles to general overview articles, monographs, or anthologies. Rapaport's text in which he described his course (Rapaport, 2005) offers a careful description of the articles he included in the course, along with rationales for choosing those articles. In Mälardalen University, Sweden, Gordana Dodig-Crnkovic adopted a lecture-centered approach with class discussions and a mini-conference with paper presentations for students (Dodig-Crnkovic, 2006). One can easily find, in the Internet, courses on the philosophy of computing, the philosophy of artificial intelligence, the philosophy of the mind, the philosophy of information, and so forth, but those topics are significantly different from what is meant by the term *philosophy of computer science* in this course. In this course we focus on philosophical issues that should also concern those computer scientists who have no interest in philosophy in general. The focus is practical and straightforward in the sense that we deal with issues that can make a difference in our students' work, research, and writing, and in the sense that we try not to delve too deep into sophisticated speculations. In a word, we deal with philosophical issues that directly concern the everyday work of computer scientists. In the following section the approach taken in this course is described.

The Philosophy of Computer Science Course at the University of Joensuu

Our course, titled *The Philosophy of Computer Science*, is an online course with no contact teaching. Students from two Finnish universities—the University of Joensuu and the University of Kuopio—can participate in the course. An online course is the only viable alternative for a course that is held simultaneously for students in spatially distant universities in a relatively large and sparsely inhabited country. Similar to Rapaport's course, I designed this course to be reading-intensive. I decided to include weekly tasks, which include reading, writing, reflecting, and commenting on other students' writing.

The course is targeted for graduate (M.Sc) and postgraduate (PhD) students who have studied computer science long enough to have an idea of what kinds of things computer scientists do. Graduate students are required to have a M.Sc thesis topic chosen, and their coursework includes an in-depth analysis of the philosophical foundations of their work. Postgraduate students are also expected to relate their own research with the paradigms of computer science; describe the intellectual foundations of their research; and explain the applicability, limitations, and boundaries of their research results.

Since there was no obvious choice for a course textbook, and since giving non-native English speakers a large number of original texts in English seemed unreasonable, I decided to prepare course readings about a number of central themes in the course. This way I was able to pull together the essence of a good number of original texts without putting an undue burden of translation on the students. (For instance, in spring 2007 none of the 62 course participants were native English speakers; there were students from Finland, Spain, Kenya, Russia, Kazakhstan, Mexico, Zambia, Czech Republic, Nepal, Jordan, Tanzania, and Armenia.) The course readings describe the cruces of the topics that are discussed in the course, yet the readings present those crucial points in a more compact and hopefully simpler form than the original texts. To encourage critical reading and to offer pointers for further reading, the electronic version of the course readings includes hyperlinks to all the articles that are available on-line. The course readings are publicly available under the creative commons license (see Tedre, 2007).

Theme 1: What is computer science?

The question "What is computer science?" is fundamental to this course. It is also a fundamentally multifaceted and fundamentally unresolved issue. The question of the identity of computer science has puzzled even—and perhaps especially—the most authoritative figures of computer science. Because of the question's relationship with other topics in the course, the question is explicitly visited three times during the course. During the first week of the course, the students are asked to define, in their own words, *computer science*. After this, the students vote for their favorite definition and discuss three of the most favored definitions in a discussion forum. The students are required to analyze the pros and cons of each of the three definitions in no less than 300 words per post. The aim is to lay the foundations for subsequent tasks.

The question is revisited after the students have become familiar with the disciplinary history of computer science and after they have been introduced to a good number of viewpoints of computer science by influential figures such as Newell, Simon, and Perlis (1967), George Forsythe (1967), Richard Hamming (1969), Donald Knuth (1974b), Edsger Dijkstra (1974, 1987), Peter Wegner (1976), Marvin Minsky (1979), Herbert A. Simon (1981), John E. Hopcroft (1987), Peter J. Denning et al. (1989), George McKee (1995), Frederick P. Brooks Jr. (1996), and Glenn

Brookshear (2003). The aim is to *widen* the students' perspectives on the different views of computer science.

The question is revisited a third time after the students have become familiar with the ontological, epistemological, and methodological issues in computer science. That is to say, during the course the students read about the different views on the philosophy of science, about normative and descriptive statements, about progress in science, about the limits of modeling and testing in computer science, about the different aims of science, about explanation and understanding, and so forth. At the end of the course the students are asked, a third time, to individually characterize computer science. The aim is to *deepen* the students' analysis of computer science. They are also required to submit a short text, in which they reflect on the changes in their characterizations—that is—their learning.

Theme 2: What is science and how is it done?

In order to be able to discuss the meaning of *science* in *computer science*, the students need to understand the difficulties in defining science. The term can indeed be read in a number of ways, depending on the context. For instance, science can mean (1) a class of activities such as observation, description, and theoretical explanation; or (2) colloquially, any activity that resembles or has characteristics like those: *He has gotten boxing down to a science*. Science can also refer to (3) a sociocultural and historical phenomenon: *Western science*. Science can refer to (4) knowledge (*scientific knowledge*) that is gained through experience; especially (5) knowledge that has been logically arranged in the form of general laws (Knuth, 1974a), or (6) structured knowledge derived from “the facts” (Chalmers, 1999, p. 1). Science can be understood as a (7) societal institution: *Humanity should be governed by science*, or a (8) world view: *The scientific world view*. Science can also be thought of as (9) a specific style of thinking and acting (Bunge, 1998b, p. 3), and even as (10) the profession of scientists.

To make the students familiar with the many faces of science, the students and I discuss, for instance, whether pure and applied sciences are really separate things. The lecture notes portray the problems of dichotomous positions towards science—problems such as the arguments that applied science is intellectually inferior to pure science and that pure science has become alienated from practical, everyday matters (e.g., Knuth, 1991). The lecture notes also deal with the aims of science; those aims are often considered to be exploration, description, explanation, and prediction of phenomena (Wright, 1971, 1). The students must submit a short text in which they reflect on the aim of their own research or thesis.

Because the course is specifically focused on the philosophy of computer science, the topics above are tightly linked to computer science. We discuss, for instance, if computer science is a science at all (Hartmanis, 1993), if there is a strict dividing line between pure and applied in computer science (Knuth, 1991) and where that line can be seen, whether the theoretical constructions of computer science are more like theorems or like laws (see Bunge, 1998a, p 391; Bunge, 1998b, p. 38ff), and what the aim of computer science is. The students continue to ponder whether their own PhD or M.Sc theses deal with a priori or a posteriori knowledge; if they require inductive, deductive, or abductive reasoning; and if they require causal explanations, functional explanations, or intentional explanations of phenomena, or something else. The students are also asked to analyze the aims, boundaries, and intellectual traditions of their own research—which is something that is not always explicit in computer scientists' education.

Theme 3: Mathematics, engineering, and science

Understanding the intellectual connections of computer science is of major importance in understanding the nature of computer science. Based on a well-known tripartite of computer science into its theoretical/mathematical tradition, modeling/abstraction tradition, and design/engineering tradition (Denning et al., 1989; Wegner, 1976), the roles that mathematics, engineering, and science play in computer science are discussed. The central question is whether one of those three traditions of computer science—or perhaps something else—is more central to computer science than the others. The relationships between computer science, mathematics, empirical science, and engineering are discussed separately. The aim of this theme is to clarify the contribution and the role of each of the three traditions.

In theme 3, the students are faced with the "pivotal question" of whether computer science is reducible to mathematics or logic (see Colburn, 2004). The students are first presented C.A.R. Hoare's argument that computer science and programming are indeed reducible to mathematics (Hoare, 1969). Hoare's argument is followed by the positions of the proponents of formal verification (Dijkstra, 1972; Floyd, 1967; Naur, 1966; Wirth, 1971), its critics (De Millo, Lipton, & Lipton, 1979; Fetzer, 1988), and some complementary viewpoints (Dobson & Randell, 1989; Smith, 1996). The connection between mathematics and computer science is discussed further, using a number of articles by eminent computer scientists (e.g., Dijkstra, 1974; Knuth, 1974b). The post-software crisis debates of whether there should be more or less mathematics in the computing curricula are discussed (e.g., Atchison et al., 1968; Austing, Barnes, Bonnette, Engel, & Stokes, 1977; Davis, 1977; Glaser, 1974; Hamming, 1969; Kandel, 1972; Khalil & Levy, 1978; Ralston & Shaw, 1980; Wishner, 1968). The students are asked to form a position of their own and write a 200-300 word response to Hoare; they are then required to read each others' responses and to write follow-ups to each others' texts. The aim of this theme is to portray the strong role that mathematics plays in computer science, how computer scientists should work if computer science were taken as a strictly mathematical field, and where the limits of the logico-mathematical tradition lie.

Second, the students are presented the argument that computer science is an engineering discipline; an argument which relies on the view that the goal of computer science is to design and construct useful things (cf. Loui, 1995; Wegner, 1976). The students are presented with the view that unlike mathematicians (who work with abstract things), computer scientists (who design working computer systems) are bound by material resources, human constraints, and the laws of nature. Engineers design complex, cost-effective systems with minimal resource consumption. A contrast is also made to natural scientists who deal with naturally occurring phenomena, noting that engineers deal with artifacts that are created by people. The students are asked to consider what role engineering plays in computer science and in their own research topics (e.g., an auxiliary role, a dominating role, a complementary role, etc.).

To help students arrive at a broad understanding of what engineering in computer science means, a number of central views of engineering are presented. The tenets of engineering presented include, for instance, that "progress is achieved primarily by posing problems and systematically following the design process to construct systems that solve them" (Denning et al., 1989; Wegner, 1976); that the "scientist builds in order to study, and the engineer studies in order to build" (Brooks, 1996); that instead of being concerned with the discovery of laws and facts, engineers are concerned with *making things*, be they computers, algorithms, or software systems (Brooks, 1996); and that progress in engineering is documented by demonstrations instead of experiments that support or refute hypotheses (Hartmanis, 1993). A general account of the engineering method, which relies on engineering heuristics, is presented (Koen, 2003). Also some think-pieces about the engineering argument in general and about the distinction between hardware and software are presented (Holloway, 1995; Moor, 1978; Zelkowitz & Wallace, 1997). The students are asked to make a well-grounded argument for whether computer science relies mostly on abstract ideas, theories, structures, working hypotheses, implementations, or something else. Again, the students are required to read each others' arguments and to comment on those arguments. The aim of this part is to discuss the significance, limits, and methods of the engineering tradition in computer science.

Third, the students are presented the argument that computer science is an empirical science. Some authors liken computer science to natural science, at least to some degree (cf. Tichy, 1998; Denning, 1980) or argue that computer scientists study both naturally occurring and human-made information processes (Denning, 2005). Others tend to think that computer science is an empirical science that studies artificial things (Knuth, 2001, p. 167; Simon, 1981). Many authors argue that computer scientists should strive to work like physicists and other natural scientists do (Feldman & Sutherland, 1979). Paul Rosenbloom (2004) argued that computer science is a new, fourth domain of science, distinct from the physical sciences, which focus on nonliving matter; the life sciences, which focus on living matter; and the social sciences, which focus on humans and their societies. In the light of the discussion about what science is, the students are asked to consider and report on whether computer science is indeed an *empirical science* (or *experimental science*, or *laboratory-based science*).

The students are then familiarized with the scientific method and with the model of science based on hypotheses, models, predictions, experiments, and analysis (e.g. Hempel, 1965; Kemeny, 1959; Popper, 1959[1935]). They are also presented the different ways in which one can model and represent phenomena (Fetzer, 1999; Smith, 1996). The students are asked to give examples of phenomena that are beyond the logico-mathematical tradition but that empirical research can describe, predict, and control. They are also asked to make an argument about whether *subject* or *method of inquiry* defines what is science, and to use their argument to analyze their own research.

Finally, the students are presented counterarguments about why computer science might not pass for a science. For instance, they are presented with the view that theories in computer science do not explain fundamental phenomena (cf. Hartmanis, 1993) and the view that much of computer science resembles exploration of uncharted territory rather than seeking solutions to clearly specified problems (Fletcher, 1995). A note is made that the observations about algorithm behavior, the usability of machinery and software, or information retrieval, are observations of things that computer scientists have constructed. It is noted that those observations might not bring to light anything new about the world—those observations might only indicate how well previous computer scientists had done their jobs. It is also noted that whereas research in the natural sciences is based on observations (data) that scientists can explain, predict, and replicate, there is no data in computer science beyond the computer and programs (McKee, 1995). At the end, the students are asked to form a position of their own regarding the scientific nature of computer science, and to write a 500-800 word essay on it. They are then required to read each other's essays and to write two comments on other students' essays.

Theme 4: What is the philosophy of science?

As the course title implies, the course is closely connected to the philosophy of science. Most concepts and terms used in the course indeed belong to the vocabulary of the philosophy of science. Because most of computer science students have no previous familiarity with the philosophy of science, the students are given an introduction to a number of central issues in the philosophy of science, such as the foundations of science, its central assumptions and limitations, its implications, and what constitutes scientific progress. Three basic questions that appear often in the philosophy of science are introduced: the ontological, epistemological, and methodological questions: “What is real?”, “How do we get to know about the reality?”, and “By which principles do we form knowledge?” (Denzin & Lincoln, 1994, pp. 99-100). However broad and vague those questions may appear at first, we continue from them to questions that are more specific and more easily applicable to computer science topics.

In this course specific questions are emphasized instead of very generic questions. More emphasis is given to questions about computer science than to generic questions about all sciences. Students are required to relate questions central to the philosophy of science with their own PhD or M.Sc thesis topics. The questions that are dealt with include, for instance, “What is scientific knowledge and how is it different from other kinds of knowledge?”, “With what kinds of methods is computer science research done?”, “What are the limits of scientific knowledge?”, “How does computer science develop?”, “What role do argumentation, logic, confirmations, concepts, demonstrations, and consensus play in computer science?”, “Are scientific results objective or subjective?”, and “What can be proven?”.

The course includes an introduction to some aspects of epistemology, such as the deep-rooted phrase that “knowledge is justified true belief” and its refutation (Gettier, 1963), to various forms of scientific explanation and understanding (Wright, 1971), and Hempel's model of scientific explanation as well as the deficiencies of that model (Hempel, 1965, pp. 331-496). The course also includes a short introduction to basic versions of some major schools in the philosophy of science—rationalism, inductivism, logical positivism, falsificationism, and Kuhn's theory of scientific revolutions. Some alternatives to and criticisms of those are also discussed; such as Feyerabend's anarchistic theory of science (Feyerabend, 1993) and new Bayesianism (Chalmers, 1999, pp. 174-192). The aim of this part of the course is to familiarize the students with some basic positions and some vocabulary in the philosophy of science.

Not only in this theme, but throughout the course, the students become familiar with a number of central questions in the philosophy of science, such as the problem of causation (Hume, 1739, sect. 6), the distinction between normative and descriptive arguments (Hume, 1739, Book III, pp. 507-521), the problem of growth of knowledge (e.g., Lakatos & Musgrave, 1970), the underdetermination thesis (Duhem, 1977, pp. 183-188; Quine, 1980, pp. 20-46), the problem of induction (Hume, 1777, Section IV,I:20-27,II:28-33), the problem with the theory-independence of facts (e.g., Chalmers, 1999, pp. 12-18; Smith, 1998, pp. 49-50), necessary and sufficient conditions, the problem with the progress of mathematics (Lakatos, 1976; Shapiro, 2000), and a number of other topics.

At the end of the course the students are required to write an 800-1000 word discussion forum post in which they choose and defend a philosophical position for their own research. They need to clearly state the aims of their research and to utilize the lecture material to analyze the epistemological and methodological issues in their research. Each student is then asked to critically evaluate the analyses of two other students using no less than 300 words.

Organizational issues

Grading online courses is difficult. The open questions include, for instance, if diligence should be rewarded over insightfulness, if originality should be rewarded over mastery of the course content, how many times should one be able to revise one's texts, and if critical skills should be valued over constructive skills. Although there clearly are no simple answers, I wanted to emphasize the importance of writing and responding to other people’s writings, so in this course I adopted a grading system that aims at some kind of a balance between rewarding work, or quantity, and rewarding insight, or quality. The progressive grading system that I adopted is presented in Figure 2. At the University of Joensuu, a five-point integer scale from 1 (at least 50% of maximum points) to 5 (at least 90% of maximum points) is used in grading (zero indicates a failing grade).

I grade every task on a five-point integer scale. Students are required to complete 50% of the tasks in order to pass the course, and those students who complete more than 80% of the tasks cannot fail the course. With 50% completed tasks, the best possible grade is a 2, but the students who have completed 50% of the tasks will get grade 2 only if each completed tasks was given full marks. The best grade 5 is available for students who complete at least 80% of the tasks. Also, if the student completes more than 80% of the tasks, the minimum grade begins to increase. If the student has completed 100% of the tasks, the minimum grade is 3. Grading is straightforward: first the average of student's marks from completed tasks is calculated ($1 \leq m \leq 5$), and that average is summed with a base score that grows in a linear fashion from -3 (50% tasks completed) to 2 (100% tasks completed). During the course the students know what marks they have been given from previous tasks.

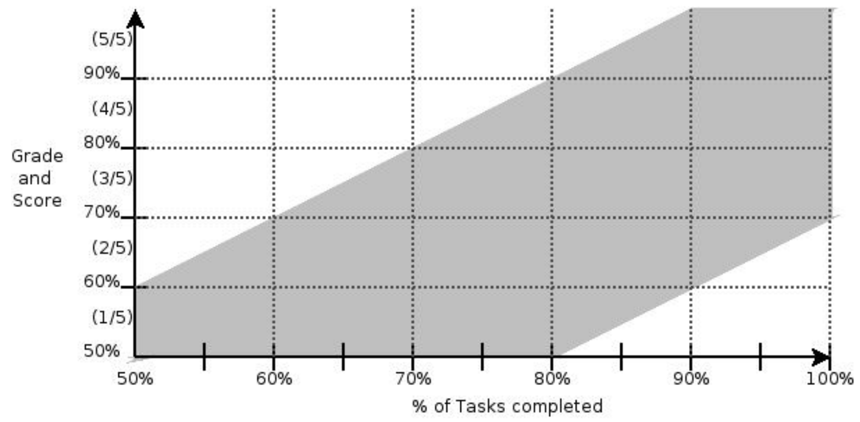

Figure 2: Completion of Tasks and Corresponding Grades

As the course is held in two spatially distant universities, there are no lectures, tutorials, or face-to-face group discussions. The course is run on an open source solution—the Moodle online learning platform. The lecture readings are licensed under the creative commons license, which is an open license that allows free copying and distribution of the material, but prohibits commercial uses and modification of that material. The lecture notes are publicly available on-line (http://cs.joensuu.fi/~mmeri/teaching/2007/philcs/).

Reading and grading students' writings is tedious work for teachers, especially in such a reading- and writing-intensive course as this. Therefore, peer evaluation is emphasized. Although I monitor and administer the discussion groups, the students spend a considerable amount of time reading and commenting on each other's texts. Students are able to revise their texts based on the comments of others, and I only evaluate the end products. This radically diminishes the amount of feedback I need to write.

Conclusions

In the first section a short disciplinary history of computer science was presented, and it was argued that the interdisciplinary nature of computer science has fueled a growth of computer science that is unparalleled by any other science, but that the same interdisciplinarity also poses problems in disciplinary understanding of computer science. Today the term *computer science* covers so many incommensurable research traditions that conflicts between paradigms are inevitable. Computer science education in general has been criticized as lacking methodological training and ignoring courses on research traditions. At the Department of Computer Science and Statistics at the University of Joensuu, Finland, we have introduced a course that addresses those issues.

In the second section a number of courses around the world on the philosophy of computer science were outlined, and some of the literature available on the topic was described. Especially the lack of suitable textbooks on the topic is a problem for many educators. My solution was to rewrite the essential readings in a simplified and compact form. The number of topics that are covered in the course is small compared to similar courses run in other academic institutions: I have excluded or given less emphasis to topics such as the philosophy of artificial intelligence, computer ethics, the philosophy of the mind, the philosophy of information, and the philosophy of computing. This course is specifically about the philosophy of computer science as the philosophy of a specific academic discipline.

This course is divided into four broader themes, which overlap to some degree. The first broad theme is the identity of computer science. There are numerous accounts of what computer science is, and in this course the merits and pitfalls of a good number of those accounts are discussed. The second broad theme is dissecting the term *science*. Different aspects of science, including its aims, methods, logic, kinds of explanations, and boundaries, are analyzed. The third broad theme is the famous tripartite of computer science into its mathematical, empirical, and engineering traditions. Each tradition is considered separately, and the pros and cons, boundaries, applicability, influences, methods, and other qualities of each are considered. The fourth broad theme is the philosophy of science. The discussions include a number of central philosophical issues that can easily be connected with computer science, such as the foundations of science, the growth of scientific knowledge, scientific revolutions, the relationship of theory and observation, underdetermination, and so forth.

This course offers students alternative viewpoints to what computer science is, how computer scientists work, and why computer scientists work as they do. More importantly, this course encourages critical thinking more than many other courses in computer science. The students learn that there are many problems that do not have clear-cut answers; they learn that there are open problems where multiple incompatible, yet credible viewpoints can be defended. Students also learn to form and defend their own positions, to comment and criticize other positions, and to reflect and revise their arguments according to criticism. Finally, the students have the chance to thoroughly think about the intellectual foundations of their own work and their own theses.

Acknowledgements

I wish to thank two experts on e-learning, Pasi Silander and Lynette Nagel, for their advice on the course arrangements. This work was funded by the East Finland Graduate School in Computer Science and Engineering (ECSE) and the National Institute for International Education (NIIED) of the Korean Government.

References

Alavi, M., Carlson, P. (1992). A review of MIS research and disciplinary development. *Journal of Management Information Systems, 8*(4), 45-62.

Aspray, W. (2000). Was early entry a competitive advantage? US universities that entered computing in the 1940s. *IEEE Annals of the History of Computing, 22*(3), 42-87.

Atchison, W. F., Conte, S. D., Hamblen, J.W., Hull, T.E., Keenan, T.A., Kehl, W.B., McCluskey, E.J., Navarro, S.O., Rheinboldt, W.C., Schweppe, E.J., Viavant W., & Young, D.M. (1968). Curriculum '68, recommendations for academic programs in computer science. *Communications of the ACM, 11*(3), 151-197.

Austing, R.H., Barnes, B.H., Bonnette, D.T., Engel, G.L., & Stokes, G. (1977). 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), 1-16.

Bowles, M.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), 5-15.

Brooks, F.P., Jr. (1996). The computer scientist as Toolsmith II. *Communications of the ACM, 39*(3), 61-68.

Brookshear, J.G. (2003). *Computer science: An overview* (7th edition). New York: Addison-Wesley.

Bunge, M. (1998a [1967]). *Philosophy of science Vol. 1: From problem to theory* (revised ed.). New Brunswick, New Jersey, USA Transaction Publishers.

Bunge, M. (1998b [1967]). *Philosophy of science Vol. 2: From explanation to justification* (revised ed.). New Brunswick, New Jersey, USA Transaction Publishers.

Bynum, T.W. & Moor, J.H. (2000). *The digital phoenix: How computers are changing philosophy*. Oxford, UK: Blackwell Publishers.

Chalmers, A.F. (1999 [1976]). *What is this thing called science?* (3rd. edition). Queensland, Australia: University of Queensland Press.

Choudrie, J. & Dwivedi, Y.K. (2005). Investigating the research approaches for examining technology adoption issues. *Journal of Research Practice, 1*(1), Article D1.

Colburn, T.R. (2000). *Philosophy and computer science*. Armonk, NY, USA: M.E. Sharpe.

Colburn, T.R. (2004). Methodology of computer science. In L. Floridi (Ed.), *The Blackwell guide to the philosophy of computing and information* (pp. 318-326). Cornwall, UK: Blackwell Publishing.

Davis, R.L. (1977). Recommended mathematical topics for computer science majors. *ACM SIGCSE Bulletin, 9*(3), 51-55.

De Millo, R.A., Lipton, R.J., & Perlis, A.J. (1979). Social processes and proofs of theorems and programs. *Communications of the ACM, 22*(5), 271-280.

Denning, P.J. (1980). What is experimental computer science? *Communications of the ACM, 23*(10), 534-544.

Denning, P.J. (2003). Great principles of computing. *Communications of the ACM, 46*(11), 15-20.

Denning, P.J. (2005). Is computer science science? *Communications of the ACM, 48*(4), 27-31.

Denning, P.J. (Chairman), Comer, D.E., Gries, D., Mulder, M.C., Tucker, A., Turner, A.J., & Young, P.R. (1989). Computing as a discipline. *Communications of the ACM, 32*(1), 9-23.

Denning, P. J., Chang, C. (chairmen) and IEEE/ACM Joint Task Force for Computing Curricula (2001). *Computing curricula 2001*. Retrieved Feb 14 2007 from http://www.sigcse.org/cc2001/

Denzin, N.K. & Lincoln, Y.S. (Eds.). (1994). *Handbook of qualitative research*. London, UK: SAGE.

Dijkstra, E.W. (1972). The humble programmer. *Communications of the ACM, 15*(10), 859-866.

Dijkstra, E.W. (1974). Programming as a discipline of mathematical nature. *American Mathematical Monthly, 81*(June-July), 608-612.

Dijkstra, E.W. (1987). Mathematicians and computing scientists: The cultural gap. *Abacus, 4*(4), 26-31.

Dobson, J. & Randell, B. (1989). Program verification: Public image and private reality. *Communications of the ACM, 32*(4), 420-422.

Dodig-Crnkovic, G. (2006). What is philosophy of computer science? Experience from the Swedish national course. *European Conference on Computing and Philosophy* – E-CAP'06, June 2006, NTNU, Trondheim, Norway. Extended abstract retrieved Feb 14, 2007 from http://www.anvendtetikk.ntnu.no/ecap06/program/Dodig-Crnkovic.pdf

Duhem, P.(1977 [1914]). *The aim and structure of physical theory* (2nd edition, 3rd reprint). New York, USA: Atheneum.

Easton, T.A. (2006). Beyond the algorithmization of the sciences. *Communications of the ACM, 49*(5), 31-33.

Feldman, J.A. & Sutherland, W.R. (1979). Rejuvenating experimental computer science: A report to the National Science Foundation and others. *Communications of the ACM, 22*(9), 497-502.

Fetzer, J.H. (1988). Program verification: The very idea. *Communications of the ACM, 31*(9), 1048-1063.

Fetzer, J.H. (1999). The role of models in computer science. *Monist, 82*(1), 20-36.

Feyerabend, P. (1993[1975]). *Against method* (3rd ed.). New York: Verso.

Fletcher, P. (1995). Readers' corner: The role of experiments in computer science. *Journal of Systems and Software, 30* (1-2), 161-163.

Floridi, L. (1999). *Philosophy and computing: An introduction*. London: Routledge.

Floridi, L. (Ed.). (2004). *The Blackwell guide to the philosophy of computing and information*. Cornwall, UK: Blackwell Publishing.

Floyd, R.W. (1967). Assigning meanings to programs. In *Proceedings of Symposia in Applied Mathematics vol. 19*. Providence, Rhode Island, USA, 19-32.

Forsythe, G.E. (1967). A university's educational program in computer science. *Communications of the ACM 10*(1), 3-11.

Galliers, R.D. & Land, F.F. (1987). Choosing appropriate information systems research methodologies. *Communications of the ACM, 30*(11), 901-902.

Gettier, E.L. (1963). Is justified true belief knowledge? *Analysis, 23*, 121-123.

Glaser, G. (1974, November 6). Education 'inadequate' for business DP. *Computerworld, VIII*(45), 1-2.

Glass, R.L. (1995). A structure-based critique of contemporary computing research. *Journal of Systems and Software, 28*, 3-7.

Glass, R.L., Ramesh, V., & Vessey, I. (2004). An analysis of research in computing disciplines. *Communications of the ACM, 47*(6), 89-94.

Hamming, R.W. (1969). One man's view of computer science (ACM Turing lecture). *Journal of the Association for Computing Machinery, 16*(1), 3-12.

Hartmanis, J. (1993). Some observations about the nature of computer science. In R.K. Shyamasundar (Ed.), *Lecture notes in computer science vol. 761*, pp. 1-12.

Hempel, C.G. (1965). *Aspects of scientific explanation and other essays in the philosophy of science*. New York, NY, USA: The Free Press.

Hoare, C.A.R. (1969). An axiomatic basis for computer programming. *Communications of the ACM, 12*(10), 576-580, 583.

Holloway, C.M. (1995). Software engineering and epistemology. *ACM SIGSOFT Software Engineering Notes, 20*(2), 20-21.

Hopcroft, J.E. (1987). Computer science: The emergence of a discipline (Turing award lecture). *Communications of the ACM, 30*(3), 198-202.

Hume, D. (1739). *A treatise of human nature* (Penguin Books edition, 1969). Aylesbury, UK: Penguin Books.

Hume, D.(1777 [1748]). *An enquiry concerning human understanding*. London, UK: Selby-Bigge.

Kandel, A. (1972). Computer science - A vicious circle. *Communications of the ACM, 15*(6), 470-471.

Kemeny, J.G. (1959). *A philosopher looks at science*. Princeton, NJ, USA: Van Nost, Reinhold.

Khalil, H. & Levy, L.S. (1978). The academic image of computer science. *ACM SIGCSE Bulletin, 10*(2), 31-33.

Kitchenham, B.A. (1996). Evaluating software engineering methods and tool (Part 1: The evaluation context and evaluation methods). *Software Engineering Notes, 21*(1), 11-15.

Knuth, D.E. (1974a). Computer programming as an art. *Communications of the ACM, 17*(12), 667-673.

Knuth, D.E. (1974b). Computer science and its relation to mathematics. *American Mathematical Monthly, 81*(April), 323-343.

Knuth, D.E. (1991). Theory and practice. *Theoretical Computer Science, 90*, 1-15.

Knuth, D.E. (2001). *Things a computer scientist rarely talks about*. Stanford, California, USA: CSLI Publications.

Koen, B.V. (2003). *Discussion of the method: Conducting the engineer's approach to problem solving.* Oxford, UK: Oxford University Press.

Lai, V.S. & Mahapatra, R.K. (1997). Exploring the research in information technology implementation. *Information & Management, 32*, 187-201.

Lakatos, I. (1976). *Proofs and refutations: The logic of mathematical discovery* (J. Worrall & E. Zahar Eds.). Cambridge, UK: Cambridge University Press.

Lakatos, I. & Musgrave, A. (Eds.). (1970). *Criticism and the growth of knowledge*. London, UK: Cambridge University Press.

Loui, M.C. (1995). Computer science is a new engineering discipline. *ACM Computing Surveys, 27*(1), 31-32.

McKee, G. (1995). Computer science or simply 'computics'?. *IEEE Computer, 28*(12), 136.

Mingers, J. (2003). The paucity of multimethod research: A review of the information systems literature. *Information Systems Journal, 13*(3), 233-249.

Minsky, M. L. (1979). Computer science and the representation of knowledge. In M.L. Dertouzos & J. Moses (Eds.), *The computer age: A twenty-year view*, 392-421.

Moor, J.H. (1978). Three myths of computer science. *The British Journal for the Philosophy of Science, 29*, 213-222.

Moor, J.H. & Bynum, T.W. (2003). *Cyberphilosophy: The intersection of philosophy and computing*. Oxford, UK: Blackwell Publishers.

Muller, M.J., Wildman, D.M. & White, E.A. (1993). Taxonomy of PD practices: A brief practitioner's guide. *Communications of the ACM, 36*(4), 26-28.

Naur, P. (1966). Proof of algorithms by general snapshots. *BIT, 6*(4), 310-316.

Newell, A., Perlis, A.J., & Simon, H.A. (1967). Computer science. *Science, 157*(3795), 1373-1374.

Popper, K. (1959 [1935]). *The logic of scientific discovery*. London, Great Britain: Routledge.

Puchta, S. (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), 49-59.

Quine, W.V.O. (1980 [1953]). *From a logical point of view* (2nd, revised edition). Cambridge, Massachusetts: Harvard University Press.

Ralston, A. & Shaw, M. (1980). Curriculum '78 - Is computer science really that unmathematical? *Communications of the ACM, 23*(2), 67-70.

Ramesh, V., Glass, R.L., & Vessey, I. (2004). Research in computer science: An empirical study. *The Journal of Systems and Software, 70*, 165-176.

Randolph, J.J. (2007). *Computer science education at the crossroads. A methodological review of the computer science education research: 2000-2005*. Utah State University, Logan, Utah, USA. Retrieved April 12, 2007 from http://www.archive.org/details/randolph_dissertation/

Rapaport, W.J. (2005). Philosophy of computer science: An introductory course. *Teaching Philosophy, 28*(4), 319-341. Retrieved Feb 14, 2007 from http://www.cse.buffalo.edu/~rapaport/Papers/rapaport_phics.pdf

Rice, J.R. & Rosen, S. (2004). Computer sciences at Purdue University - 1962 to 2000. *IEEE Annals of the History of Computing, 26*(2), 48-61.

Rosenbloom, P.S. (2004). A new framework for computer science and engineering. *IEEE Computer, 37*(11), 23-28.

Shapiro, S. (2000). *Thinking about mathematics: The philosophy of mathematics*. Oxford, UK: Oxford University Press.

Simon, H.A. (1981). *The sciences of the artificial* (2nd ed.) Cambridge, Mass., USA: The MIT Press.

Smith, B.C. (1996 [1985]). Limits of correctness in computers. In R. Kling (Ed.), C*omputerization and controversy: Value conflicts and social choices* (2nd ed.) (pp. 810-825). San Diego, CA, USA: Academic Press.

Smith, B.C. (1998 [1996]). *On the origin of objects* (MIT Paperback ed.). The MIT Press: Cambridge, Mass., USA.

Tedre, M. (2006). *The development of computer science: A sociocultural perspective*. Joensuu, Finland: Yliopistopaino. Retrieved Feb 14, 2007 from ftp://cs.joensuu.fi/pub/Dissertations/tedre.pdf

Tedre, M. (2007). *Lecture notes in the philosophy of computer science*. Lecture notes for the Department of Computer Science and Statistics, University of Joensuu, Finland. Retrieved Feb 14, 2007 from http://cs.joensuu.fi/~mmeri/teaching/2007/philcs/

Tichy, W.E. (1998). Should computer scientists experiment more? *IEEE Computer, 31*(5), 32-40.

Tichy, W.F., Lukowicz, P., Prechelt, L. & Heinz, E.A. (1995). Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software, 28*, 9-18.

Vessey, I., Ramesh, V., & Glass, R.L. (2002). Research in information systems: An empirical study of diversity in the discipline and its journals. *Journal of Management Information Systems, 19*(2), 129-174.

Walsham, G. (1995). The emergence of interpretivism in IS research. *Information Systems Research, 6*(4), 376-394.

Wegner, P. (1976). Research paradigms in computer science. (IEEE) *Proceedings of the 2nd international conference on Software engineering*. October 13-15, San Francisco, California, USA, 322-330.

Williams, M.R. (1985). *A history of computing technology*. New Jersey, USA: Prentice-Hall.

Wirth, N.(1971). Program development by stepwise refinement. *Communications of the ACM, 14*(4), 221-227.

Wishner, R.P. (1968). Letters to the editor: Comment on Curriculum '68. *Communications of the ACM, 11*(10), 658.

Wright, G.H. von (1971). *Explanation and understanding*. London, UK: Routledge & Kegan Paul.

Zelkowitz, M.V. & Wallace, D. (1997). Experimental validation in software engineering. *Information and Software Technology, 39*(1997), 735-743.

Biography

Matti Tedre received a PhD degree in computer science in 2006 from the University of Joensuu, Finland. Since 2002 he has been working in the Department of Computer Science and Statistics at the University of Joensuu as an assistant, researcher, and lecturer (and spent two years in South Korea visiting the universities of Yonsei and Ajou). He has also been a visiting instructor at the University of Pretoria, South Africa. Earlier, he worked as a programmer and as a software analyst. His research interests include social studies of computer science, the history of computer science, and the philosophy of computer science.