Journal of Universal Computer Science, vol. 15, no. 8 (2009), 1642-1658
submitted: 14/3/09, accepted: 27/4/09, appeared: 28/4/09 © J.UCS

Computing as Engineering

Matti Tedre
(Tumaini University, Iringa, Tanzania
firstname.lastname@acm.org)

Abstract: Computing as a discipline is often characterized as a combination of three major traditions: theoretical, scientific, and engineering tradition. Although the three traditions are all considered equally necessary for modern computing, the engineering tradition is often considered to be useful but to lack intellectual depth. This article discusses the basic intellectual background of the engineering tradition of computing. The article depicts the engineering aims manifest in the academic field of computing, compares the engineering tradition with the other traditions of computing as a discipline, and presents some epistemological, ontological, and methodological views concerning the engineering tradition of computing. The article aims at giving the reader an overview of the engineering tradition in computing and of some open questions about the intellectual foundations and contributions of the engineering tradition in computing.
Key Words: information technology, philosophy of computer science, philosophy of technology, computing, engineering
Category: K.7, K.7.1, K.7.m

1 Introduction

The juxtaposing of science and technology is perhaps nowhere else as marked as in the computing disciplines. The division of computing into its mathematical/theoretical, scientific/empirical, and design/engineering traditions ([Wegner, 1976], [Denning et al., 1989]) has spurred fiery debates about the merits and shortcomings of each tradition. In those debates, the theoretical tradition leans on the recognition of mathematics and logic as the theoretical cornerstones of computing, the scientific tradition draws support from arguments from the philosophy of science, but the design/engineering tradition is usually only recognized for its utility and not for its intellectual foundations. The intellectual foundations and intellectual contributions of the engineering tradition are often ignored.

This dismissal of the intellectual basis of engineering tradition in computing might derive from the focus of epistemology in the philosophy of science in the 20th century. The scientific tradition rides on the crest of the deductive-nomological wave fueled by Hempel and Popper, and it utilizes vocabulary familiar from Kuhn, Popper, and other philosophers of science. The logico-mathematical roots of computing are rarely challenged, as mathematics is generally seen as the language of science and technology. However, the public image of engineering is vague and poor, and often secondary to even technology research

[Malpas, 2000]. There is a widespread recognition of the societal significance of technology, and scientific findings frequently grab the headlines. However, when engineers do their job well, the engineers and their intellectual accomplishments usually disappear from public association, and only the successful artifacts remain in the public [Malpas, 2000].

It is not the case that engineering would generally be considered unimportant in computing: it is usually agreed that production of useful, efficient, and reliable computational tools is a well-justified aim that is societally important. Rather, it seems to be the case that the engineering aspects of computing are considered to be based on rules of thumb and anecdotal evidence, to be less intellectually challenging than scientific and mathematical branches of computing, to be theoretically vague, and to be philosophically shallow. Those kinds of critique of engineering, however, often miss the target, as they are done from the viewpoint of science or mathematics, where evaluation criteria are largely incommensurable with evaluation criteria in engineering.

This article is aimed at depicting some of the intellectual background of engineering in the field of computing; a background often ignored by the critics of engineering in computing. This article is not a critique of the scientific/empirical tradition or the mathematical/theoretical tradition of computing. Those traditions occupy a very important place in the discipline of computing. Instead, this article describes and defends the intellectual foundations of the design/engineering tradition of computing.

2 Aims of Engineering in the Field of Computing

Most accounts of the engineering tradition of computing share the view that unlike mathematicians, engineers, who design working computer systems, have to cater to material resources, human constraints, and laws of nature [Tedre & Sutinen, 2008]. Unlike natural scientists who deal with naturally occurring phenomena, engineers deal with artifacts, which are created by people. In computing disciplines engineers design complex, cost-effective systems with minimal resource consumption ([Hamming, 1969], [Loui, 1995], [Wegner, 1976]). Indeed, what seems to be common to all the different engineering branches is that they all aim at producing useful things that are directed towards some social need or desire [Mitcham, 1994, 146–147]. Carl Mitcham, who is a philosopher of technology, wrote:

> *Engineering as a profession is identified with the systematic knowledge of how to design useful artifacts or processes, a discipline that* [...] *includes some pure science and mathematics, the "applied" or "engineering sciences" (e.g., strength of materials, thermodynamics, electronics), and is directed toward some social need or desire. But while engineering*

> *involves a relationship to these other elements, artifact design is what constitutes the essence of engineering, because it is design that establishes and orders the unique engineering framework that integrates other elements.* [Mitcham, 1994, 146–147]

The purposes of engineering and technology are practical; to manipulate and control the world [Skolimowski, 1972]. Engineers "*invent useful things or, at least, add to our knowledge of how to do it*" [Davis, 1998, 7–8]. Michael Davis distinguished engineers from applied scientists when he wrote that whereas engineers are primarily committed to human welfare, applied scientists are primarily committed to theoretical or applied knowledge [Davis, 1998, 15–16]. Unlike mathematicians and scientists, who embrace the precision and rigor of their disciplines and who can infinitely hone their products, engineers must come up with working solutions within some time limits (e.g., [Florman, 1994, 178], [Kidder, 1981]). That creates intellectual challenges of a unique kind. In addition to usefulness, usability and reliability are also crucial to computing disciplines.

The practical aims of engineering are often underlined by emphasizing the role of computing machinery in the discipline of computing. In his 1968 Turing Award lecture, Richard Hamming argued that at the heart of computing disciplines lies a technological device, the computing machine [Hamming, 1969]. Without it, Hamming argued, almost everything that professionals in computing fields do would become idle speculation, hardly different from that of the notorious Scholastics of the Middle Ages. In Hamming's opinion, much of what computing professionals do is "*not a question of can it be done as it is a question of finding a practical way*" [Hamming, 1969]. That is, the question for computing professionals is usually not whether there *can* exist a monitor system, algorithm, or compiler, but usually the professionals work on creating one with reasonable expenditure and effort. The theoretician's question "Can there be x?" is less frequent than the practitioner's question, "What is the most cost-effective way of building x?" Therefore, in Hamming's vision, the focus should not be on the abstract ideas about computation, but on the practical implementations of computing systems.

Also the kinds of questions that computing professionals ask reveal the deep rooted engineering aims in computing disciplines. In his 1993 Turing Award lecture, Juris Hartmanis argued that generally speaking, researchers in computing disciplines concentrate more on the *how* than the *what* [Hartmanis, 1993]. He wrote that natural scientists concentrate more on questions of *what*, and that computing fields, with their bias on *how*, reveal their engineering concerns and considerations. Hartmanis further argued that whereas the advancements in natural sciences are typically documented by dramatic *experiments*, in computing disciplines the advancements are typically documented by dramatic *demonstrations*. In some branches of computing the scientists' slogan "publish or perish" indeed has turned into the engineers' slogan "demo or die." Engineering aims are also visible in Hartmanis' view that whereas the physical scientists ask *what exists*, computer scientists ask *what can exist* [Hartmanis, 1993]. Notably, Hartmanis characterized automatic computing as the "engineering of mathematics" [Hartmanis, 1993].

The role of knowledge acquisition in computing also gives a hint about the nature of the discipline. In his ACM Allen Newell Award lecture Frederick P. Brooks, Jr., argued that although scientists and engineers both may spend most of their time building and refining their apparatus, the scientist *builds in order to study*, and the engineer *studies in order to build* [Brooks, 1996]. Brooks wrote that unlike the disciplines in the natural sciences, computing is a synthetic, engineering discipline. He noted that science is concerned with the *discovery* of facts and laws, whereas engineering is concerned with *making* things, be they computers, algorithms, or software systems. The human-made nature of computing has been noted by a number of prominent figures in computing. For instance, Donald Knuth called computing an *unnatural science* [Knuth, 2001, 167] and Herbert A. Simon called it an *artificial science* [Simon, 1981].

Already in the 1960s a number of prominent figures in computing started to refer to computing as a combination of art and science (e.g., [Forsythe, 1967], [Knuth, 1968]). The argument was that "*science is knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it*" [Knuth, 1974]. Donald Knuth described the scientific approach with terms such as logical, systematic, impersonal, calm, and rational, and described the artistic approach with terms such as æsthetic, creative, humanitarian, anxious, and irrational ([Knuth, 1974], see also [Snow, 1964]). For Knuth, both of those apparently contradictory approaches are valuable to computer programming. The lack of insight to artistic aspects of programming was also noted by Hamming: "*To parody our current methods of teaching programming, we give beginners a grammar and a dictionary and tell them that they are now great writers. We seldom, if ever, give them any serious training in style*" [Hamming, 1969, 10]. Concerns about excessive emphasis of theory led Knuth later to urge computing practitioners to turn some of their attention to theoretical things and computing theoreticians to turn some of their attention to practical things [Knuth, 1991].

The line between theoretical and practical aspects of computing is sometimes drawn following the "physical vs. nonphysical" lines (e.g., [Arden, 1980, 7]). Engineers, who work with practical things, are supposed to work with physical things (like hardware or machinery), whereas computer scientists work with abstract things (like algorithms and programs). But that distinction is vague and difficult. Firstly, programs have a dual nature: they are at the same time physical and abstract ([Colburn, 2000], [Fetzer, 2000, 267], [Smith, 1998, 29–

32]). And secondly, establishing a strict line between hardware and software is also difficult [Moor, 1978].

The engineering-oriented branches of computing can also be further subdivided. In his review of the three traditions of computing—mathematical, scientific, and engineering traditions—Peter Wegner underscored the goal-orientation of the engineering-oriented aspects of computing, yet he divided the engineering part of computing into two parts: practical engineering and research-based engineering [Wegner, 1976]. Wegner wrote:

> [The problem-solving paradigm of the practicing engineer] *generally involves a sequence of systematic selection of design decisions which progressively narrow down alternative options for accomplishing the task until a unique realization of the task is determined.* [The research engineer] *may use the paradigms of mathematics and physics in the development of tools for the practicing engineer, but is much more concerned with the practical implications of his research than the empirical scientist or mathematician* [Wegner, 1976].

It seems that the interest in producing useful things is a necessary condition of engineering. That is, an activity cannot be considered to be engineering unless the person's aim or goal is to produce useful things. But certainly interests (goals or aims) are not a sufficient condition of engineering. In most accounts of engineering, not all activities that aim at building things are considered to be engineering. Most accounts of engineering impose a number of necessary conditions, such as how the work is done (methodology) or what the outcomes of the work are (e.g., innovations, processual knowledge, or artifacts).

3 Pure Science, Applied Science, Engineering, and Technology

One view that emerges often in the philosophy of engineering and technology is the division between pure science, applied science, engineering, and technology (e.g., [Vincenti, 1990]). Unfortunately, each of those terms is vague. Especially the term *science* is exceedingly vague and can refer to many different things [Tedre, 2007], and to make matters even more complex, science is further divided into *pure* and *applied*. Research in pure science is often called *basic research*; it is research that is performed without thought of practical ends, and it yields general knowledge and knowledge about how the world works [Bush, 1945]. In other words, pure science seeks knowledge for its own sake [Malpas, 2000]. Applied science is usually connected with design, engineering [Arden, 1980], and technology. It is often argued that the focus of research projects is narrower in applied science than in pure science [Gruender, 1971], and that applied science extends scientific knowledge with a specific purpose in mind [Malpas, 2000].

Similar to the term *science*, in computing disciplines also *engineering* is a broad and ambiguous term: it can be considered to encompass a plethora of things such as requirements engineering, software development, interface design, computer engineering, robotics, operating systems, signal processing, software / hardware testing, maintenance, and project management. Epistemologically speaking, engineering commonly refers to both "*the knowledge required, and the process applied, to conceive, design, make, build, operate, sustain, recycle or retire, something of significant technical content for a specified purpose*" [Malpas, 2000]. That is, engineering concerns facts ("know-that"), experience, and skills, and it concerns the ability to use science, engineering, experience, and contextual knowledge to implement a solution to a problem ("know how") [Malpas, 2000].

Some consider *technology* to be still a bit more practical endeavor than engineering is [Feibleman, 1961]. However, similar to terms *science* and *engineering*, the term *technology* is used in a diversity of meanings. Stephen Kline distinguished between four meanings of the term: *artifacts*, which are objects made by people; *sociotechnical systems of production*, which are systems that include people, machinery, resources, processes, economic, and other aspects of production; *knowledge, technique, or know-how*, which refers to technology as a field or discipline similar to terms like psychology, sociology, and geology; and *sociotechnical systems of use*, which refers to using artifacts and knowledge about them to extend human capabilities [Kline, 1985]. In addition, the term *technology* is used to refer to things, actions, processes, methods, systems, working procedures, and progress [Kline, 1985].

The meaning of the term *technology* is often ambiguous in the language of computing professionals. In computing fields the term is used for referring to artifacts ("The new MacBook is a nice piece of technology"), sociotechnical systems of use ("Technology can eradicate poverty"), and knowledge ("With our current technological knowledge we can map the human genome"). Finally, *technological research* is sometimes distinguished from engineering research. Whereas engineering research emphasizes the development of processes, technological research emphasizes the quality of the outcomes [Malpas, 2000]. Among computing practitioners the term *technology* is most commonly used to refer to artifacts.

But even if one considered computing to be a kind of science or a kind of engineering, it is still hard to tell exactly where on the map of sciences or engineering fields does computing belong. Juris Hartmanis argued that computing differs so fundamentally from the other sciences that it has to be viewed as a new species among the sciences, especially because it deals with human-made phenomena that are explored by human-made paradigms and methods [Hartmanis, 1994]. Hartmanis noted that theory and experimentation in computing 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 of doing things that were earlier thought to be impossible or unfeasible. Hartmanis' view portrayed computing as a combination of science and engineering. Indeed, Michael C. Loui [Loui, 1995] in response to Hartmanis, noted that instead of a new species of science, it would be more appropriate to call computing *a new species of engineering.*

As long as the juxtaposition between applied science, pure science, engineering, and technology is purely descriptive, the main question is about where the dividing lines are drawn. But when the juxtaposition becomes normative—for instance, when it is implied that theoretical and scientific aspects of computing are intellectually superior to applied and engineering-oriented aspects of computing—then justification for that normative assessment should be given. That issue was addressed in the 1990s, when the Computer Science and Technology Board (CSTB) of the National Research Council of the U.S. published a report where they argued that in computing fields the traditional separation of basic research, applied research, and development is dubious [Hartmanis, 1992]. They argued that both basic and applied research call for the exercise of the same kinds of judgment, creativity, skill, and talent. The committee recommended that academic computing disciplines should abandon artificial distinctions among basic research, applied research, and engineering. In addition, the committee urged the discipline of computing to earn its governmental support by showing that computing research will have significant societal benefits [Rice, 1993].

It is hard to find a convincing argument for the argued intellectual superiority of pure science over applied science and engineering. Especially recently, many modern philosophers and sociologists have argued that the old idea that "technology is applied science" is no longer true (if it ever was). Many argue that the inverse direction might be even stronger: most of the progress in modern science can be attributed to technological development. For example, astronomy took giant leaps after the invention of the telescope. The theoretical progress in particle physics is inextricably linked with the development of instruments such as different kinds of particle detectors and particle accelerators [Pickering, 1995]. Similar, thermodynamics followed the invention of the steam engine [Malpas, 2000]. In computing it is often difficult to separate theoretical from technological progress.

Some authors have even begun to use the term *technoscience* instead of the phrase *science and technology* because they believe that the two have become inseparable ([Haraway, 1999], [MacKenzie & Wajcman, 1999]). In the idealist attitude towards technology, technology is considered to be applied science, whereas in the materialist attitude towards technology, science is considered to be theoretical technology [Mitcham, 1994, 76]). It must be remembered, though, that technological and engineering knowledge are not subsets of scientific knowledge. Technology and engineering both have distinct cognitive content separate from scientific knowledge [Frey, 1991].

Rather than seeing pure science, applied science, engineering, and technology as separate categories, one could see technology as a new cognitive method for science, and see science as a source of new principles for technology [Mitcham, 1994, 86]. But although science and engineering share some similarities—for instance, they both have to conform to the laws of nature, they both are cumulative, and they both share the scaling problem—they still utilize and produce different kinds of knowledge and employ different methodologies [Tedre & Sutinen, 2008]. Whereas scientific knowledge consists of a set of observations, laws, and theories; engineering and technological knowledge consist of actions, rules, and theories (cf. [Mitcham, 1994, 193–194,197]).

4 Knowledge of Engineers

In the engineering branches of computing there is abundance of implicit metaphysical, epistemological, ethical, and methodological views. For an example about metaphysics, artificial intelligence research has reawoken the debate between the Ancient skeptic view (artifacts are less real than natural objects) and Enlightenment optimism (nature and artifacts operate by the same mechanical principles) [Mitcham, 1994, 298]. In epistemology, the debates about the nature of computing as a discipline bring up the 2500 years old debates between those who argue that technical information is not true knowledge and those who argue that technical engagement with the world produces true knowledge [Mitcham, 1994, 298]. Yet, the characteristics of typical epistemological questions in the engineering tradition of computing are markedly different from those in the mathematical and scientific traditions of computing. Questions about technological and engineering knowledge discussed in this section are familiar to philosophers of technology (e.g., [de Vries, 2003], [Herschbach, 1995], [Layton, 1974]).

4.1 Engineering Methodology

Denning et al. wrote that engineers share the methodological notion that progress is achieved primarily by posing problems and systematically following the design process to construct systems that solve them [Denning et al., 1989]. The engineering method, as seen by Denning et al., is a cycle that consists of defining requirements, defining specifications, designing and implementing, and testing. In their work, engineers often follow the method of parameter variation—that is, they repeatedly measure the performance of a device or process, while they systematically adjust the parameters of the device or its conditions of operation

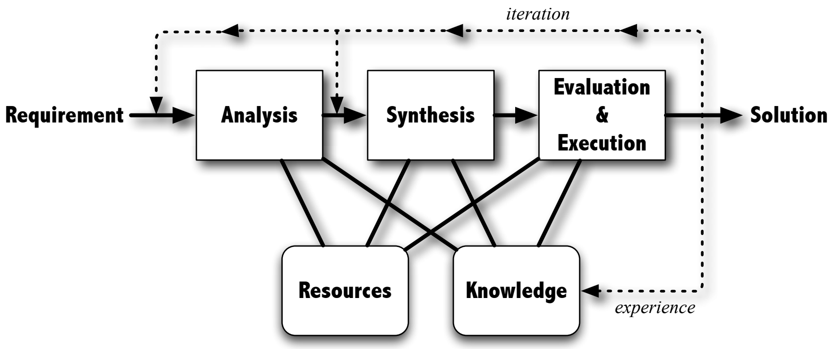

Figure 1: Generic Engineering Process ([Malpas, 2000], Reprinted with the permission of the Royal Academy of Engineering, UK)

[Vincenti, 1990, 139]. An engineer should have good skills in design, construction, testing, planning, quality assurance, problem solving, decision-making, and communication [Malpas, 2000]. The engineering method can be described as iterative cycles of analysis, synthesis, and evaluation and execution (Figure 1; see [Malpas, 2000]).

The generic engineering process portrayed in Figure 1 starts from requirements and proceeds through cycles of analysis, synthesis, and evaluation and execution. The engineering process draws from available resources and all kinds of available knowledge. Engineers often integrate many competing demands, theories, data, ideas, and knowledge from several fields and domains [Malpas, 2000]. The process typically works to meet the requirements within some time and budget constraints. The engineering process is aimed at good decisions and optimized solutions instead of conclusions [Malpas, 2000]. The knowledge gained in the process (experience) adds to the common knowledge about the phenomenon, process, theory, or domain.

One of the main activities of engineering is to compare solutions and select alternatives. In engineering, those comparisons are often made in terms of costs and efficiency. Interestingly, a lot of theoretical computer science, which one might describe as one of the least engineering-oriented branches of computing, is focused on the cost and efficiency of algorithms (the costs are expressed in resources such as time and storage) (cf. [Arden, 1980, 7]). But although branches of theoretical computer science study optimization of resources and cost/efficiency concerns, the focus of theoretical computer science is not on producing useful, cost-effective things, but on understanding properties of algorithms—properties that are expressed in terms such as cost and efficiency.

Timothy R. Colburn, who is a philosopher of computing, portrayed the engineering approach in computing in form of *solution engineering* [Colburn, 2000, 167]. In some branches of computing the usual scenario includes rigorous requirements, and the task of the computing professional is to engineer an algorithmic solution. Colburn portrayed an analogy between the scientific method and the problem-solving approach in computing (see Table 1; adapted from [Colburn, 2000, 168]).

Table 1: Analogy Between the Scientific Method and Problem-Solving in Computing

The Scientific Method	***Problem-solving in Computing***
1. Formulate a *hyphothesis* for explaining a phenomenon	1. Formulate an *algorithm* for solving a problem
2. *Test* the hypothesis by conducting *an experiment*	2. *Test* the algorithm by writing and running *a program*
3. *Confirm* or *disconfirm* the hypothesis by evaluating the results of the experiment	3. *Accept* or *reject* the algorithm by evaluating the results of running the program

In Colburn's analogy portrayed in Table 1, what is being tested with the scientific method is not the experiment, but the hypothesis [Colburn, 2000, 168]. The experiment is a tool for testing the hypothesis. Similar, what is being tested in problem-solving in computing is not the program, but the algorithm. The program is written in order to test the algorithm. In this analogy, writing a program is analogous to constructing a test situation. Similar, Khalil and Levy argued, "*Programming is to computer science what the laboratory is to the physical sciences*" [Khalil & Levy, 1978]. Although Colburn noted that his analogy does not hold very far, that analogy displays another view of engineering in computing disciplines.

The aim of engineering in computing—producing useful things within the boundaries of available resources—sets some significant constraints on engineers' work. Engineers often have to work relying on information that scientists would not consider adequate for scientific purposes (cf. [Vincenti, 1990]). Scientists, on the other hand, are loath to make conclusions without adequate information about the phenomenon under study. In addition, for the scientist, natural phenomena are not desirable or undesirable—they "just are", but for engineers natural phenomena can be desirable or undesirable [Frey, 1991]—for instance, in the field of electronic communication thermal noise is an unwanted natural phenomenon. Unlike theoreticians, engineers cannot abstract away aspects of the physical world.

4.2 Ontology and Epistemology

The meaning and status of knowledge differ between science and engineering. Scientific knowledge is about description, explanation, prediction, and understanding of natural (or artificial) phenomena, and scientists are concerned whether their knowledge is true. Engineering knowledge is about heuristic prescriptions (best practices) of how things should be done, and engineers are concerned whether their knowledge works (cf. [Mitcham, 1994, 197]). Mitcham divided technological knowledge into four kinds of knowledge: (1) sensorimotoric skills of making ("know-how"), (2) technical maxims ("rules of thumb" or "recipes", which offer heuristic strategies for successfully completing tasks), (3) descriptive laws (that is, "If A then B"-kind of rules, based on experience—yet those rules do not go into explaining why A and B seem to be connected in some way), and (4) technological theories (applications of scientific theories to practice; e.g., the theory of flight is an application of fluid dynamics) [Mitcham, 1994].

The mechanisms of growth of knowledge in engineering and technology are, in some senses, different from the mechanisms of growth of knowledge in science. In science there are occasional revolutions when a previous scientific theory is replaced with a new one [Kuhn, 1996]. The theories that have been replaced have often been not merely inaccurate, but plain wrong. For instance, the phlogiston theory of combustion and the optical æther theory, which are nowadays considered to be false, were once considered to be true ([Kuhn, 1996], [Laudan, 1981]). However, when new technology replaces old technology it is not because the old technology would not have worked; it is because the new technology does the task faster, more efficiently, has some other benefits over the old technology, or offers something that no previous technology could offer. This has led some philosophers of engineering to argue that pragmatic technological "what works" and "know-how" kinds of procedural and normative knowledge are rigorous, "secure knowledge" [McCarthy, 2006]. In addition to descriptive and normative knowledge, there is a strong component of tacit knowledge in technology [Vincenti, 1984].

The domain of knowledge in computing as a discipline is different from the domain of knowledge in mathematics. Already in the early days of modern computing, many pioneers adopted the position that computing as a discipline is not situated in the ideal, infinite world of mathematics, but is situated within the finite boundaries of available resources [Forsythe, 1967]. *Design* in computing disciplines is rooted in engineering and deals with constructing systems or devices to solve a given problem [Denning et al., 1989]. Design, as an engineering activity, has to cater not only to material resources but also to human constraints.

In addition, whereas the Turing Machine does not have space constraints, computing professionals must take into account the limits of computing resources of actual computers [Forsythe, 1967]. Efficiency, effectiveness, and reliability are all emphasized in the field of computing (e.g., [Arden, 1980], [Hartmanis, 1993], [Knuth, 1997, 6], [Parnas, 1998], [Wegner, 1976]). However, there are two ways of looking at the domain of computing fields: firstly, there is the view that computing as a discipline studies abstract information processes and algorithms, and secondly, there is the view that computing as a discipline studies actual implementations of computing systems. Different views to the subject of computing as a discipline also led to different names of the discipline—such as computer science, informatics, and computing (e.g., [Gal-Ezer & Harel, 1998]).

One of the interesting notions about the ontology of computing is the notion that programs can be seen either as abstract constructions or as physical things that do physical work. That dual nature of programs makes disciplinary distinctions between engineering and other branches of computing difficult. On one hand, one can claim that computer programs are physical things—computer programs are swarms of electrons in the circuits of a computer ([Smith, 1998, 29–32], [Tedre, 2006, 122]). They are physical phenomena and they can make physical phenomena happen. Computer programs are a part of the causal world, and computer programs can affect the causal world. For instance, computer programs make monitors blink and printers rattle, and computer programs land airplanes and guide missiles to their targets. On the other hand, one can claim that programs are abstract things in the same way that mathematical objects are abstract. Programmers can construct procedures and programs in the same way mathematicians construct functions, theorems, and proofs—in their minds or with a pen and paper. Computers are not necessary for creating computer programs, and computer programs need not have any executable physical counterparts (such as programs in computer memory or on hard drive).

The vagueness of the hardware-software distinction blurs the intra-disciplinary borders even further. Those parts of the computer system that one can touch are often considered to be hardware, and respectively, software is often considered to be the "non-physical" parts of a computer system. This line is vague, too. Firstly, programs (when stored as electrical charges in memory, or as blips on a magnetic disc) *are* physical phenomena. More importantly, most hardware can be implemented as software and most software can be implemented as hardware. For instance, codecs are sometimes implemented as hardware, sometimes as software. In the end, a hardware-software distinction is a pragmatic distinction, and it is a subjective distinction [Moor, 1978]. For the user of a microwave oven, the whole thing is hardware. But for the engineer of a microwave oven there is often software and hardware. For the systems programmer, circuitry is hardware, but a circuit designer can see microprograms as software. A graphics programmer may not even know which parts of his or her program are going to be hardware-accelerated and which run as software.

The ontological issues in computing disciplines can be seen in a number of ways, one of which is well described by philosopher John R. Searle [Searle, 1996]. In their work, engineers work with what Searle called *brute facts* and *institutional facts* [Searle, 1996]. For instance, it is a fact that the silicon atom has 14 electrons. That is, silicon atoms have 14 electrons, no matter how people choose to call those atoms and no matter what people think about them. But there are also facts that are facts only because people (individually or collectively) agree that they are facts. For instance, it is a fact that the OSI model has seven layers, but that fact is a fact only because people collectively maintain that the OSI model has seven layers. The former type of facts is called brute facts and the latter type of facts is called institutional facts. On one hand, Searle's division allows the researcher of computing to take into account the characteristics of the physical world, because the laws of nature determine some characteristics and limits of automatic computing. On the other hand, Searle's division allows explanations of socially constructed phenomena that would not exist without people (e.g., conventions, standards, and programs).

Engineers in computing disciplines work with many things that are based on brute facts. For instance, how semiconductors work is a fact that is independent of any attitudes people may have towards semiconductors. The principles behind the functioning of the basic elements of the computer are brute facts about how the world works. But many other aspects of computing are based on institutional facts. For instance, all the standards in computing are agreements among some stakeholders in computing. It would certainly be odd to argue that the IEEE standard for floating-point arithmetic would reveal anything about how the world works (cf., e.g., [MacKenzie, 1993]). Standards, as well as many other things in computing, are constructions, which are useful but which can be abandoned or re-negotiated at any time. The reader should note that even though the mechanisms and fabric of the physical world are observer-independent, *concepts*; such as atoms, quarks, and gravity; are human constructs, and thus necessarily socioculturally influenced. Although the physical and chemical properties of silicon are brute facts, the concepts that people use to describe those properties (concepts such as hardness and atomic weight) are human-made institutional facts, i.e., social constructions.

5 Conclusions

The importance of the engineering tradition in computing is rarely contested, but there is still debate about the intellectual content and output of the engineering tradition as well as about its academic status. That is largely because the engineering tradition is significantly different from the theoretical and scientific traditions. Those traditions have different goals and aims, concerns and foci, methods and modi operandi, and, to some extent, even different epistemological and ontological views [Eden, 2007]. Some of the major differences that are commonly associated between the scientific and engineering traditions of computing are portrayed in Table 2.

Table 2: Engineering and Science in Computing Compared

Engineering tradition	***Scientific tradition***
Concerned with whether products work	Concerned with whether claims are true
Work is value-laden	Often claimed to be value-free
Aims at working implementations; Changing the world	Aims at new findings about the world; Understanding the world
Actions	Observations
Partly generalizable	Highly generalizable
Concerned with processes	Concerned with causes
Processes, rules, and heuristics; Products and inventions	Models, theories, and laws; Discoveries
Concretizations of abstract ideas	Generalizations from particular findings
Holistic, can integrate competing ideas	Reductionist
Propositional and procedural knowledge: “Know-that”, “Know-how”	Propositional knowledge: “Know-that”
Descriptive, normative, and tacit	Descriptive
“Demo or die”	“Publish or perish”
Must be able to act under very little information	Refuse making claims if there is not enough information

The characteristics portrayed in Table 2 are somewhat stereotypical, for in computing the scientific, engineering-oriented, and theoretical traditions are deeply intertwined [Denning et al., 1989]. One should also note that many of the item pairs in Table 2 are not fully comparable. For instance, processes, rules, and heuristics are not fully comparable with models, theories, and laws. Nevertheless, the table illustrates some emphases and background assumptions commonly connected with the engineering tradition of computing, and contrasts them with those commonly connected with the scientific tradition of computing.

The engineering tradition has an established place in computing, but its special characteristics are not always appreciated. Whenever comparisons between the traditions of computing are done in terms of the mathematical tradition or the scientific tradition, the engineering tradition certainly differs much from those two traditions. That is not a problem as long as the comparison is purely descriptive, but if normative ideas are derived from the comparison—that is, if engineering is considered less valuable or intellectually inferior to the other two traditions—then the comparison is problematic. Engineering has important and intellectually valuable content that is incommensurable with the content of the scientific and mathematical traditions.

References

[Arden, 1980] Arden, B. W., editor *What Can Be Automated? Computer Science and Engineering Research Study* MIT Press, Cambridge, Mass., USA, 1980.

[Brooks, 1996] Brooks, F. P., Jr.: The computer scientist as toolsmith II; *Communications of the ACM*, 39(3):61–68, 1996.

[Bush, 1945] Bush, V.: Science: The endless frontier. A report to the president by Vannevar Bush, director of the office of scientific research and development; Technical report, United States Government Printing Office, Washington, D.C., USA, July 1945 1945.

[Colburn, 2000] Colburn, T. R.: *Philosophy and Computer Science* M.E. Sharpe, Armonk, NY, USA, 2000.

[Davis, 1998] Davis, M.: *Thinking Like an Engineer—Studies in the Ethics of a Profession* Oxford University Press, Oxford, UK, 1998.

[Denning et al., 1989] Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., and Young, P. R.: Computing as a discipline; *Communications of the ACM*, 32(1):9–23, 1989.

[de Vries, 2003] de Vries, M. J.: The nature of technological knowledge: Extending empirically informed studies into what engineers know; *Techné*, 6(3):1–21, 2003.

[Eden, 2007] Eden, A. H.: Three paradigms of computer science; *Minds and Machines*, 17(2):135–167, 2007.

[Feibleman, 1961] Feibleman, J. K.: Pure science, applied science, technology, engineering: An attempt at definitions; *Technology and Culture*, 2(4):305–317, 1961.

[Fetzer, 2000] Fetzer, J. H.: Philosophy and computer science: Reflections on the program verification debate; In Bynum, T. W. and Moor, J. H., editors, *The Digital Phoenix: How Computers are Changing Philosophy*, pages 253–273. Blackwell Publishers, Oxford, UK, 2000.

[Florman, 1994] Florman, S. C.: *The Existential Pleasures of Engineering* St. Martin's Press, New York, NY, USA, 2nd edition, 1994.

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

[Frey, 1991] Frey, R. E.: Another look at technology and science; *Journal of Technology Education*, 3(1), 1991.

[Gal-Ezer & Harel, 1998] Gal-Ezer, J. and Harel, D.: What (else) should CS educators know?; *Communications of the ACM*, 41(9):77–84, 1998.

[Gruender, 1971] Gruender, C. D.: On distinguishing science and technology; *Technology and Culture*, 12(3):456–463, 1971.

[Hamming, 1969] Hamming, R. W.: One man's view of computer science; *Journal of the ACM*, 16(1):3–12, 1969.

[Hartmanis, 1992] Hartmanis, J.: Computing the future: Committee to assess the scope and direction of computer science and technology for the National Research Council; *Communications of the ACM*, 35(11):30–40, 1992.

[Hartmanis, 1993] Hartmanis, J.: Some observations about the nature of computer science; In Shyamasundar, R. K., editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 761/1993 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, Berlin / Heidelberg, Germany, 1993.

[Hartmanis, 1994] Hartmanis, J.: Turing award lecture on computational complexity and the nature of computer science; *Communications of the ACM*, 37(10):37–43, 1994.

[Haraway, 1999] Haraway, D. J.: Modest_witness_@_second_millennium; In MacKenzie, D. and Wajcman, J., editors, *The Social Shaping of Technology*, pages 41–49. Open University Press, England, 2nd edition, 1999.

[Herschbach, 1995] Herschbach, D. R.: Technology as knowledge: Implications for instruction; *Journal of Technology Education*, 7(1):31–42, 1995.

[Kidder, 1981] Kidder, J. T.: *The Soul of a New Machine* Little, Brown, and Co., New York, NY, USA, 1981.

[Khalil & Levy, 1978] Khalil, H. and Levy, L. S.: The academic image of computer science; *ACM SIGCSE Bulletin*, 10(2):31–33, 1978.

[Kline, 1985] Kline, S. J.: What is technology?; *Bulletin of Science, Technology & Society*, 5(3):215–218, 1985.

[Knuth, 1968] Knuth, D. E.: *The Art of Computer Programming*, volume 1: Fundamental Algorithms Addison-Wesley, Reading, Mass., USA, 1st edition, 1968.

[Knuth, 1974] Knuth, D. E.: Computer programming as an art; *Communications of the ACM*, 17(12):667–673, 1974.

[Knuth, 1991] Knuth, D. E.: Theory and practice; *Theoretical Computer Science*, 90(1991):1–15, 1991.

[Knuth, 1997] Knuth, D. E.: *The Art of Computer Programming*, volume 1: Fundamental Algorithms Addison-Wesley, Reading, Mass., USA, 3rd edition, 1997.

[Knuth, 2001] Knuth, D. E.: *Things a Computer Scientist Rarely Talks About* CSLI Publications, Stanford, California, USA, 2001.

[Kuhn, 1996] Kuhn, T.: *The Structure of Scientific Revolutions* The University of Chicago Press, Chicago, USA, 3rd edition, 1996.

[Laudan, 1981] Laudan, L.: A confutation of convergent realism; *Philosophy of Science*, 48(1981):19–49, 1981.

[Layton, 1974] Layton, E. T., Jr.: Technology as knowledge; *Technology as Culture*, 15(1):31–41, 1974.

[Loui, 1995] Loui, M. C.: Computer science is a new engineering discipline; *ACM Computing Surveys*, 27(1):31–32, 1995.

[MacKenzie, 1993] MacKenzie, D.: Negotiating arithmetic, constructing proof: The sociology of mathematics and information technology; *Social Studies of Science*, 23(1):37–65, 1993.

[Malpas, 2000] Malpas, R.: The universe of engineering: A UK perspective; Technical report, The Royal Academy of Engineering, London, UK, 2000.

[McCarthy, 2006] McCarthy, N.: Philosophy in the making; *Ingenia*, 26:47–51, March 2006.

[Mitcham, 1994] Mitcham, C.: *Thinking Through Technology: The Path Between Engineering and Philosophy* The University of Chicago Press, Chicago, USA, 1994.

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

[MacKenzie & Wajcman, 1999] MacKenzie, D. and Wajcman, J., editors *The Social Shaping of Technology* Open University Press, England, 2nd edition, 1999.

[Parnas, 1998] Parnas, D. L.: Software engineering programmes are not computer science programmes; *Annals of Software Engineering*, 6(1998):19–37, 1998.

[Pickering, 1995] Pickering, A.: *The Mangle of Practice: Time, Agency, and Science* The University of Chicago Press, Chicago, USA, 1995.

[Rice, 1993] Rice, J. R.: Letter to the CACM forum; *Communications of the ACM*, 36(2):19, 1993.

[Searle, 1996] Searle, J. R.: *The Construction of Social Reality* Penguin Press, England, 1996.

[Simon, 1981] Simon, H. A.: *The Sciences of the Artificial* MIT Press, Cambridge, Mass., USA, 2nd edition, 1981.

[Skolimowski, 1972] Skolimowski, H.: The structure of thinking in technology; In Mitcham, C. and Mackey, R., editors, *Philosophy and Technology: Readings in the Philosophical Problems of Technology*, pages 42–49. Free Press, New York, NY, USA, 1972.

[Smith, 1998] Smith, B. C.: *On the Origin of Objects* MIT Press, Cambridge, Mass., USA, mit paperback edition, 1998.

[Snow, 1964] Snow, C. P.: *The Two Cultures and A Second Look* Cambridge University Press, Cambridge, UK, 1964.

[Tedre, 2006] Tedre, M.: *The Development of Computer Science: A Sociocultural Perspective* PhD thesis, University of Joensuu, Department of Computer Science and Statistics, Joensuu, Finland, 2006.

[Tedre, 2007] Tedre, M.: Know your discipline: Teaching the philosophy of computer science; *Journal of Information Technology Education*, 6(1):105–122, 2007.

[Tedre & Sutinen, 2008] Tedre, M. and Sutinen, E.: Three traditions of computing: What educators should know; *Computer Science Education*, 18(3):153–170, 2008.

[Vincenti, 1984] Vincenti, W. G.: Technological knowledge without science: The innovation of flush riveting in american airplanes, ca. 1930 – ca. 1950; *Technology and Culture*, 25(3):540–576, 1984.

[Vincenti, 1990] Vincenti, W. G.: *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History* The Johns Hopkins University Press, Baltimore / London, 1990.

[Wegner, 1976] Wegner, P.: Research paradigms in computer science; In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, pages 322–330, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.