Lecture Notes in
The Philosophy of Computer Science

Matti Tedre

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

This .pdf file was created March 11, 2007.
Available at cs.joensuu.fi/~mmeri/teaching/2007/philcs/

3.3 Is Computer Science a Science?

There are debates of the relationships between computer science and mathematics, computer science and engineering, and computer science and sciences. Of those debates, the *scientific* nature of computer science has been debated the most. The question is usually about whether computer science is a science in the same sense *natural sciences* are. It is sometimes argued that computer scientists do theoretical work similar to mathematics, and hence they are not doing science. Other people have asked computer scientists to be honest about the engineering nature of their work. And even other people have argued that computer scientists should indeed aspire to work the way physicists and other natural scientists do. Sometimes it is even argued that computer science *is* indeed a natural science in the sense that it studies naturally (but also artificially) occurring information processes (Denning, 2005b). Moreover, Donald Knuth called computer science an *unnatural* science (Knuth, 2001:167) and Herbert A. Simon called it an *artificial* science (Simon, 1981).

Often the pro-science argument is that although computer science is not a natural science, it is still an *empirical science*, because computer scientists follow the scientific method (they explore and observe phenomena, form hypotheses, and empirically test those hypotheses). Paul Rosenbloom argued that computer science is a new, *fourth domain of science*, distinct of 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 (Rosenbloom, 2004). Note that many interdisciplinary branches, such as behavioral sciences and cognitive science, span across those domains, but Rosenbloom considered computer science to constitute a new domain altogether.

It was noted earlier in these lecture notes that formal verification, or proving programs correct, cannot prove that a program's behavior corresponds to how the program was intended to behave. There are fundamental gaps between models, algorithms, computer systems, and the real world, and those gaps are the strongest arguments for *empirical* computer science (or, more strictly speaking, *experimental* computer science or *laboratory-based* computer science) and the *modeling* and *abstraction* aspects of computer science. Before dealing with the scientific tradition in computer science, another look at *science* is taken.

3.3.1 Science, Generally Speaking

It was mentioned earlier in these lecture notes (Section 1.2) that it is difficult to precisely define the concept *science*. Nonetheless, for the purposes of the rest of this section, in this short subsection a general characterization of science is given. It should be noted, however, that the image of science portrayed here is highly idealized and somewhat contested. Many modern philosophers and sociologists of science argue that scientists do not actually work according to any rigid guidelines, and that there is no fixed set of rules suitable for all branches of science (much less all branches of intellectual inquiry). This section is based on views which were originally conceived and promoted by philosophers such as Karl Popper in the early 1900s. Take this section with quite some reservations —later in these lecture notes some of these things are discussed critically and in more detail.

One of the less contested ideas connected with science is that the aim of science is to *describe, explain*, and *predict* phenomena in the world we live in (cf. Okasha, 2002:1; cf. Wright, 1971:6). One

could argue that having those aims is a necessary condition of being science. The argument that "description and explanation of phenomena without prediction is not science" has often been used in attempts to undermine the status of some branches of social sciences and humanities. But adopting such view is quite a strict statement, and by the same token many other established intellectual fields are excluded—fields such as linguistics, archeology, history, and mathematics. In addition, those three aims are certainly not a sufficient condition of being science. For instance, many religions as well as astrology aim at explaining and predicting phenomena, yet they are not considered to be sciences. Aims alone do not seem to define science. However, for the purposes of this section it is adopted that the aims of science are *description, explanation*, and *prediction* of phenomena.

Quite some people agree that striving to *objectivity* is a sine qua non of true science (for a vehement defense of the objectivity of science, see Couvalis, 1997). The term *objective* refers to different things in different contexts, and quite many quarrels about objectivity of science arise from different ways of interpreting the term. One can talk about, for example, objective statements, objective knowledge; reality, methods, standards, and justification. For instance, in the context of *scientific observations* objectivity can refer to the idea that those observations can be confirmed by anyone who carries out the same experimental procedure under the same conditions. That is, observations and results in science should not be relative or open to interpretations. In the context of *scientific theories* objectivity can refer to the idea that all the logical consequences of those theories can be tested by anyone (cf. Kemeny, 1959:96). For instance, any logical consequence of atomic theory can be tested by anyone, and those tests can either support atomic theory or conflict with atomic theory. Few natural scientists would disagree that objectivity is a characteristic of science. Quite a few of them might, however, face a problem if they were asked to define what objectivity, strictly speaking, means. (Later in these lecture notes a number of views of objectivity are presented with their counter-arguments.)

Some philosophers have argued that the most useful way of defining science is by its method (Kemeny, 1959:174). *The scientific method* is, however, not a strictly defined method, but a broad collection of principles and techniques. In its most basic sense, the term *scientific method* refers to a cycle of observations, descriptions, predictions, and experiments that test those predictions (Figure 3).

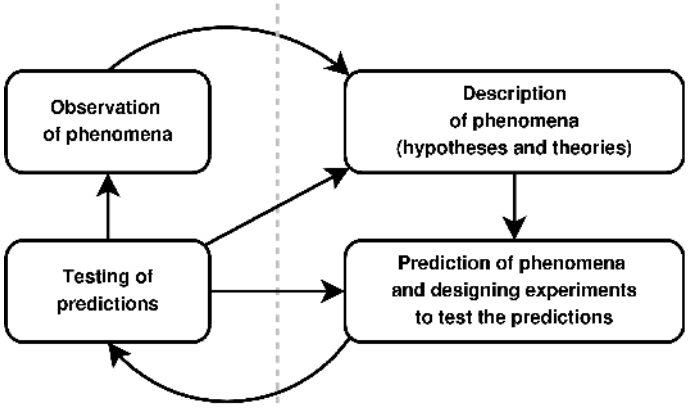

Figure 3: Cycle of Scientific Research

In the cycle of phases that belong to the scientific method, a scientist begins by observing a phenomenon. When enough observational data has accumulated, the scientist proceeds to describe the

phenomenon by forming *hypotheses* about the phenomenon, possibly supported by some other the – ories. Scientific hypotheses have logical consequences that can be tested, which allows predictions of phenomena that have not yet come to pass. The next step in the cycle is to design experiments to test some predictions, and to carry out those experiments. If the experimental results support the hypothesis, the scientist goes back to design more experiments to test the hypothesis. If the experi – mental results conflict with the hypothesis, the scientist goes back to rethink the hypothesis (or to think whether the anomalous results might have been caused by other things, such as flaws in in – struments or test setting). Note that in Figure 3 the applied, empirical parts of the scientific method are on the left side of the picture and the theoretical parts are on the right side of the picture.

The cycle in Figure 3 is, of course, an idealized version of the scientific method. Many modern philosophers of science argue that that is not how scientists really work (Kuhn, 1996). Some argue that that is not how scientists *should* always work either (Feyerabend, 1993). Nevertheless, the sci – entific method is indeed a widely supported ideal of how science should be done, yet it must be noted that the cycle is unsuited for some well-established branches of science. Especially experi – mentation that relies on the researcher's ability to freely manipulate the subject matter and to freely control the test setting does not seem to be a necessary condition of science. For instance, astro – nomers have little means of manipulating the subject matter they study; astronomers work on their tools and theories, and they try to find a good fit between their observational data, their instruments, their theories, and their theories on how their instruments work. Again, in social sciences it might be considered unethical to conduct experiments on society and in psychology one cannot test just any hypothesis on the participants.

Another rarely contested thing that is often connected with science is the notion that science is, or should be, a *self-correcting* enterprise. The argument that science is self-correcting relies on a cycle of hypotheses and theories; predictions; experiments; corroborations, corrections, or refutations; and new hypotheses and theories. Scientists, generally speaking, understand that any theory *could* be wrong. Furthermore, many scientists hold the view that scientists should actively seek to find flaws in scientific theories, try to unearth sources of bias in research settings, question all presuppositions, and so forth. Usually the idea of self-correcting science relies on the idea that in the course of time science gets closer to true understanding of the world. Both the scientists' search for truth as well as competition between scientists might indeed encourage some self-correcting processes. (Again, the outcomes of the self-correcting nature of science can be called into question.)

3.3.2 The Status of Models in Computing

In 1989 Denning et al. named three major research paradigms in computing as a discipline: *theory*, *abstraction* (modeling), and *design* (Denning et al., 1989). They wrote that modeling is the bedrock of the natural sciences: "*Scientists share the notion that scientific progress is achieved primarily by formulating hypotheses and systematically following the modeling process to verify and validate them*" (Denning et al., 1989). Denning et al. connected the modeling aspects of computer science with the scientific method and noted that modeling consists of four stages:

1. Form a hypothesis;
2. Construct a model and make a prediction;
3. Design an experiment and collect data; and
4. Analyze results.

Although modeling in this sense is taken to be characteristic of the empirical research tradition in computer science, models are an inherent part of each of the three constituents of computer science that Denning et al. listed: *theory*, *abstraction*, and *design*. In this section, *modeling* refers to model – ing work that is especially intertwined with experiments and design. For instance, many program – mers design programs that are based on models of some aspects of the world. In constructing those programs *modeling* and *designing* (engineering) are so intertwined with each other that it is difficult to say when modeling ends and designing begins.

James Fetzer listed four kinds of models that pervade computer science: (1) *Specifications* are mod – els of problems, (2) *programs* are models of possible solutions, (3) *high-level programming lan – guages* are models of virtual machines, (4) *low-level programming languages* are models of causal (physical) machines; and he also noted that (5) *conceptualizations*, as ways of conceiving the world, are also certain kinds of models (Fetzer, 1999). Philosopher of computing Brian Cantwell Smith has emphasized the role of models in computer science as *representations* (Smith, 1996). Smith wrote that "*one way to understand the model is as the glasses through which the program or computer looks at the world: it is the world, that is, as the system sees it (though not, of course, as it necessarily is)*." (Smith, 1996).

An important notion about the concept of *model* is that there should be correspondence between some specific parts of the models and some specific parts of the thing modeled (Fetzer, 1999). Spe – cifically, in order for a computer program to be a computer model, there should be a correspond – ence between the behavior of the program and the behavior of the thing the program is supposed to model. If a computer program is argued to be a computer model of particle physics, then there ought to be a correspondence between how particles such as electrons and protons actually behave and how their counterparts in the computer program behave.

Another important notion about the concept of *model* is that all models of the world are *limited* views of the world. Different models may emphasize different aspects of the thing they model. Consider, for instance, different models of the little house between Joensuu Science Park and the Linnunlahti beach. The town planner, whose task is to plan a well-functioning and beautiful neigh – borhood, needs to know the size, shape, and color of each house, the locations of their windows and doors, and other things that are visible to outside, but the town planner needs to know little or noth – ing about the insides of single houses. The decorator of a house may need to know the locations of windows, doors, and supporting structures, but the decorator needs to know little or nothing about how the plumbing of the house is attached to the municipal sewer system. The electrician needs to know the wiring plan for the house as well as how the house's electrical system is attached to the power-distribution network, but not much more. The town planner, decorator, and electrician need very different kinds of models of the same house.

Different models serve different needs, and one of the purposes of modeling is to hide extraneous (inessential) information from the users of the model. A model simplifies things by pruning out most of the richness of the real phenomenon it models. For instance, the football-playing LEGO ro – bots at our Kids' Club rely on very simple models of playing football. A robot's actions in the world are based on that model and on the input from, for example, four touch sensors and from one light sensor.

In contrast to models, *action* is not partial (Smith, 1996). Although models can be abstract and al – though models may exclude extraneous information, *action* cannot abstract away some parts of the world. Physical action always works with the full richness of the world. Take, for instance, the football-playing robots. The representation of the world for the football-playing robot is very simple (based on four touch sensors and one light sensor). But when the robot spins its motors ac – cording to its inputs and its programming, the robot partakes in a very real and complex world (yet the robot is ignorant of most aspects of that world).

Model ≠ Theory

Philosopher James H. Moor pointed out that there is a problem in computer scientists' inclination to slide, in their discourse, between *programs*, *models*, and *theories* as if there was no distinction between them (Moor, 1978). One can have a theory (in the sense of a *set of laws used to explain and predict a set of events*) without having a model. For instance, one can have quantum theory without having a model of quantum mechanics. Also, one can have a model of a subject matter without having a theory about that subject matter (Moor, 1978). Moor noted that soap film on ir – regularly shaped wires can be regarded as a model that gives solutions to minimization problems, yet it does not provide a theory about minimization.

Moor argued that there is a widespread *model=theory myth* in computer science, whereby construct – ing a computer model of a phenomenon is considered to be the same as constructing a theory about that phenomenon. But there are a large number of examples of working computational models of phenomena which do not automatically embody a theory of those phenomena. Moor mentioned ELIZA program, which is a program that simulates the responses of a psychotherapist, and Moor noted that although a computer running ELIZA can be understood as a model of a language user hold – ing a conversation, ELIZA has never been claimed to provide a theory of human conversation (or any other theory for that matter).

In addition, Moor claimed that in computer science a confusion between *programs* and *theories* is common. Just because a robot plays football—and no matter how well it plays—it cannot be said that the program that runs in the robot is a theory of playing football. Moor took an example quote from Patrick Winston: "*Occasionally, after seeing what a program can do, someone will ask for a specification of the theory behind it. Often the correct response is that the program* is *the the – ory*."[19]. Moor argued that Winston was fundamentally wrong in claiming that the program is often the theory.

Stephen Wolfram's book *A New Kind of Science* is a typical example of an equivocation between program and theory. In his book Wolfram described a number of simple programs that demonstrate

19 Moor cited Winston, Patrick (1977) *Artificial Intelligence*. Addison-Wesley: Reading, Mass., USA:p.258.

an enormous range of behaviors (Wolfram, 2002). Based on the fact that Wolfram's programs can produce and predict a vast range of behaviors, he essentially claimed his approach to be the general theory of everything. One critic of Wolfram's book asked:

> [2002] *Just because Wolfram can cook up a cellular automaton that seems to produce the spot pattern on a leopard, may we safely conclude that he understands the mechan – isms by which the spots are produced on the leopard, or why the spots are there, or what function (evolutionary or mating or camouflage or other) they perform?*
>
> (Krantz, 2002)

Francis Sullivan wrote that computer scientists can design a binary adder because they know what addition is, but they cannot produce consciousness because they do not know what consciousness is (Sullivan, 2006). This does not seem to be true, either. In order to build a system that implements or produces a phenomenon, it is not necessary to fully understand that phenomenon. It has been ar – gued that computer scientists build systems better than they understand those systems (Smith, 1998:21,64). For example, creating a soap film on irregularly shaped wires equips a person with an artifact that gives solutions to minimization problems, but it does not mean that the person would know much about minimization, or that the person would know he or she has a solution to minimiz – ation problem. Also Alan Turing wrote that he wished to allow the possibility that an engineer or team of engineers may construct a machine which works, but whose manner of operation cannot be satisfactorily described by its constructors because they have applied a method which is largely ex – perimental (Turing, 1950).

Computer, Model, and the World

There is a more fundamental problem with programs, models, and theories in computer science than Moor's *model=theory myth*. That is the profound difference between the model and the world, and especially difficult this problem is for program verification. In order to approach this distinction one can ask, "what does program verification actually mean?". In Figure 4 there are three concepts —a computer, a model, and the real world (do not get stuck here with the problem with what is "real"). The examples (computer, model, and real-world situation) in Figure 4 are about robots that play football.

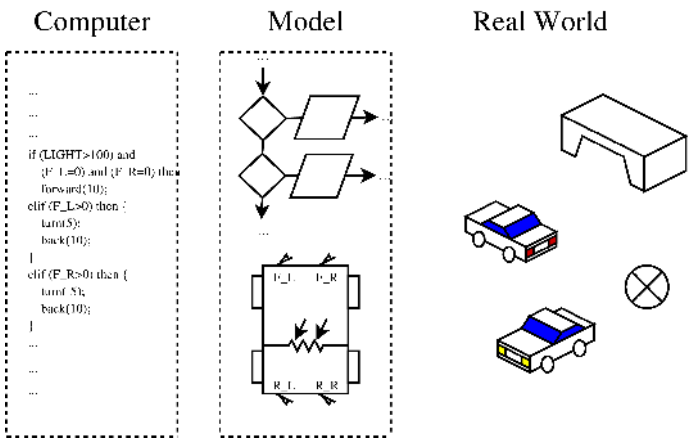

Figure 4: Computer, Model, and the World

Computer in Figure 4 refers to the hardware and software that implement a full-working football-playing robot. *Model* in Figure 4 refers to the abstraction of the world as the football-playing robot sees it, and to the model according to which the robot acts upon certain inputs (I do not mean *see* and *act* to be anthropomorphisms). *Real world* in Figure 4 refers to the actual robo-ball game situ – ation with its infinite variety. In the real world there are other robots, two goals, and the playing field; there are humans who may, at any time, grab the robot; there is the fragile toy frame and cas – ing of the robot; there is the gravity, batteries, and all kinds of inference; there is everything that the world is made of. Note that *computer* and *model* are bounded, but *real world* is unbounded.

The robot's view of the world is derived from those few input sources that the robot builder has chosen (e.g., touch and light sensors). The *model* according to which the robot plays football is an attempt of the robot designer to model robot football-playing. The robot takes *actions* according to the robot's model of football-playing and to the states of its sensors, yet the outcomes of those ac – tions are not confined to the abstract model-world. The robot's actions affect the real world. That is, although the robot acts according to its sensor inputs and according to the model it has about football playing, the actions of the robot have consequences outside the robot's model of the world. Brian Cantwell Smith wrote:

> [1985] *The point is that computers, like us, participate in the real world; they take real actions. One of the most important facts about computers* [...] *is that we plug them in. They are not, as some theoreticians seem to suppose, pure mathematical abstractions, living in a pure detached heaven.*
>
> (Smith, 1996)

Now, what is considered (formal) *program verification* is verifying that the program corresponds to the model. In the best case, formal verification might prove that *you constructed the program right*. Program verification does not say anything about how well the model corresponds to the world, and hence verification cannot tell whether *you constructed the right program*. The validity and utility of a program must be established in some other ways. (But essentially nothing can *prove* the validity and utility of a program.)

When kids in our Kids' Club come up with models of how their robots should play football, and when they implement those models as programs, it would basically be possible to mathematically prove that the *programs* (as abstract things) correspond to the models (as other abstract, formally defined things). But it is *not* possible to *prove* that the models correspond to the reality. And fi – nally, no matter how good the robots are in goal-making or football-playing, there are no "proofs of correct playing". This is a fundamental problem with correctness of computers[20].

How Can One Describe Correct Intention?

In *Limits of Correctness in Computers* Brian Cantwell Smith noted the difficulty of describing "cor – rect intention" (already touched in an earlier quote by Hoare, 1969; see page 69 of these lecture notes):

20 I recommend reading Smith's thought-provoking article *Limits of Correctness in Computers.* Originally Smith's art – icle was published 1985 as a technical report for Stanford Center for the Study of Language and Information (SCLI:85-35). It has also been published in Johnson & Nissenbaum, 1995:456-479 and in Kling, 1996:810-825.

> [1985] *The word "correct" is already problematic, especially as it relates to underlying intention. Is a program correct when it does what we have instructed it to do? or what we have wanted it to do? or what history would dispassionately say it should have done?*
>
> (Smith, 1996; underlining added)

For instance, when is our football-playing program correct? Is that program correct when it *does exactly what it was instructed to do*? A hasty programmer might indeed be tempted to say that a program is correct when it does exactly what it was instructed to do. But bug-free programs and erroneous programs alike work exactly as they were instructed to do. In this interpretation of correctness, all programs are "correct", because all programs work as they were instructed to work. (Only those computer systems that for some reason fail to execute programs properly are flawed.)

Alternatively, is our football-playing program correct when it *works exactly according to our model*? It seems tempting to argue so, but in that case the correctness of the program may not have anything to do with how well the robot actually plays football. Correctness of the program in that case only tells that the program corresponds to the model, and if the robot does not score, it is not because the program would be flawed (though then the model might be flawed). Although this view can be adopted as a proof of correctness of the relationship *between the program and the model*, it certainly does not say anything about how well a program does what it was *intended to do*.

Alternatively, is a football-playing program correct when the robot makes a lot of goals or when it wins? In that case, correctness is a relative concept—any program can be correct in some situations. However, it seems odd to speak about "degrees of proofs" or "relative correctness". Perhaps the word might, in that case, be not *correct* but *efficient*. Efficiency is, in fact, what computer scientists usually argue for.

Alternatively, is a football-playing program correct when the actions of the robot correspond exactly to the *specifications* of the robot? There are many problems to this view. For instance, on what level of abstraction should the specifications be? Should they be of the kind "When you get the ball, drive to the opponent's goal while evading the opponent's robots", or of the kind "If the reading from the light sensor exceeds 127, and readings from both touch sensors at the front are 0, go forward"? And when something goes wrong with the robot's playing, how can one tell if the specification or the program is flawed? Smith wrote:

> [1985] *When you show that a program meets its specifications, all you have done is to show that two formal descriptions, slightly different in character, are compatible. This is why I think it is somewhere between misleading and immoral for computer scientists to call this "correctness". What is called a proof of correctness is really a proof of the compatibility or consistency between two formal objects of an extremely similar sort: program and specification.*
>
> (Smith, 1996)

It would indeed be misleading or sometimes even immoral to formally prove that the program is consistent with the model, but to leave the relationship between the model and the world with little

emphasis. Disregarding the relationship between the model and the world could be disastrous with, for instance, programs that control nuclear reactors or programs that administer radiation therapy.

Because all models of real-world things have to exclude an infinite number of variables as superfluous, all models are deficient in some ways. Here the discussion comes back to the problem I earlier referred to as *underrepresentation problem:* "Given a number of models that all model and predict well different aspects of a phenomenon, but that all are flawed in some ways, how does one determine which model to use?". Because formal methods and traditional methods alike are hopelessly inadequate to decide the underrepresentation problem, normative stands and common sense are required in software construction. Normative stands come in because one must choose which aspects are the most important to be modeled. Common sense comes in because the number of variables to be excluded is infinite, and in practice one just needs to go on and make decisions about the specifications of the computer system.

In making specifications of, say, a football-playing robot, one can start the specifications by making a normative statement that for a football-playing robot the *most important things* to be modeled are the heading of the robot, collisions with other robots, and the location of the ball. The specifications need not include, for instance, the atmospheric pressure, noise in the arena, flying insects, or audience watching the game. Those things are not important with a football-playing robot, yet their exclusion is not a formally provable matter but a matter of commonsense thinking.

It seems that the need for empirical research in computer science arises directly from the complexity of the real world. If only those kinds of programs that are purely mathematical can be proven correct, the rest of computer programs may require different approaches. The scientific method in computer science is essentially about (1) observing phenomena, (2) formulating hypotheses, (3) constructing models and making predictions, (4) designing experiments and collecting data, (5) analyzing results, and (6) comparing the successes of competing hypotheses. It is the success of the scientific method in working with models that has led to arguments that computer science *is* an empirical science and to arguments that computer scientists *should* experiment more.

3.3.3 Is Computer Science a Science?

One of the earliest arguments for the *scientific* nature of computer science is probably Newell et al.'s argument in the September 1967 issue of *Science* (see page 14 of these notes). Newell et al. argued that phenomena breed sciences, and that computer science is the study of the phenomena called computers and the phenomena surrounding them (Newell et al., 1967). Although the term *science* has been used with a large number of characterizations of computer science, most of those characterizations do not specify what they mean by *science*.

What kinds of activities in computer science are scientific, then? There are many activities that have led to interesting findings in computer science but that are not considered to be scientific. Peter Denning wrote against the conception that *tinkering*, or *hacking*, is science (Denning, 1980b). It is not science to put together things, try them, and see what happens (no matter how much programmers might like that modus operandi; cf. Graham, 2004:18-25). In Denning's opinion, scientific work needs to be *directed* and *systematic*.

Although research of computers, the electronic devices, has always had an empirical character (see, e.g., Wilkes, 1968), the role of experimentation in computer *science* became a hot topic when Jerome Feldman and William Sutherland published their report titled *Rejuvenating Experimental Computer Science* (Feldman & Sutherland, 1979). In the report they recommended that universities and the U.S. government should recognize and support experimental computer science. It is apparent that by experimental computer science Feldman and Sutherland meant the kind of empirical research that is close to the falsificationist paradigm and the scientific method.

Peter Denning joined ranks with the Feldman committee and wrote in the October 1980 issue of CACM that no scientific discipline can be productive in the long term if its experimenters merely build components (Denning, 1980b). Also the ACM Executive Committee agreed with Feldman committee in that experimental computer science was undervalued at the time (McCracken et al., 1979). A vivid discussion in CACM ensued (e.g., Ousterhout, 1981). As to what *experimental computer science* is, participants of the debate mentioned measuring, testing (McCracken et al., 1979), making hypotheses, observing, collecting data, classifying, and sustaining or refuting hypotheses (Denning, 1981). As a prime example of experimental computer science, Denning mentioned performance analysis—the construction, validation, and evaluation of computer systems (Denning, 1981). In performance analysis researchers have a theory of how a computer system should be constructed, they have hypotheses about how that system would perform, they design and implement the system, and test the system and either find the system to perform according to their predictions or not.

It is clear that empirical research can reach to areas where formal, logico-mathematical reasoning cannot. (But there again, those two kinds of research are from different worlds. Logico-mathematical reasoning generates *a priori* knowledge whereas empirical research generates *a posteriori* knowledge (see pages 14 and 25).) For instance, branches such as software engineering and human-computer interaction are mostly beyond the reach of the logico-mathematical tradition. Because there are no axiomatic theories of human activity, HCI research must rely on empirical methods. Empirical methods are also central to branches such as numerical computation, information retrieval, genetic algorithms, and artificial intelligence.

Walter Tichy is one of the proponents of the view of computer science as an empirical and experimental science. In Tichy's point of view, *computer science* is a study of all kinds of information structures and information processes, and *computers* are devices that help people model and study those structures and processes (Tichy, 1998). Tichy argued that the only major difference between traditional sciences and computer science is just that information is "neither energy nor matter" (although Tichy did not discuss this ontological question further).

Denning and Tichy argued, in separate articles, against the objection that "computer science is not a science because it studies human-made objects and phenomena" (Denning, 2005b; Tichy, 1998). In Denning's opinion, that argument is a red herring (an irrelevant argument). Denning's and Tichy's argument is that computer science studies information processes both natural and artificial. In this sense Denning and Tichy seem to have tried to hang onto the parallel to natural sciences. Note that they could have also argued that not the *subject* but the *methodology* of inquiry is a defining feature of science. That is, they could have argued that what defines science is not *what* is being studied

but *how* the research is being done. In this sense, they could have argued that those branches of computer science that rely on the scientific method are indeed science, regardless of the nature of their subject of inquiry.

That is about what Herbert A. Simon has argued. In *The Sciences of the Artificial* Simon wrote that "*natural science is knowledge about natural objects and phenomena*", and argued that in a similar sense there can be scientific knowledge about artificial, human-made, objects and phenomena (Simon, 1981:3). Computer science, no matter how artificial or synthetic its subject may be, can be done through empirical inquiry like any other subject (Newell & Simon, 1976).

Take, for instance, research where one (1) explores different kinds of algorithms for, say, polygonal approximation and analyzes their strengths and weaknesses, (2) develops a new algorithm for the task, (3) formulates a hypothesis about how the new algorithm fares compared with the previous algorithms, (4) designs experiments for testing the hypothesis and collects data using those different algorithms, and (5) analyzes and interprets the results. Although such research bears many characteristics of the natural sciences, the subject of the study—algorithms—are not naturally occurring phenomena but human-made phenomena. Nonetheless, despite not being natural science, such research can easily be argued to be empirical science. The defining characteristic of empirical science vis-à-vis natural science might be that natural science begins with observations of naturally occurring phenomena whereas empirical science begins with analytical or formal research (Robert L. Glass suggested that the rule "*observe the world*" in natural sciences is changed to the rule "*observe the problem space*" in empirical sciences; Glass, 1995).

It could be argued, though, that research such as the one above does not follow the normal experimental protocol. Especially, there is a clear lack of precautions against experimenter bias (Fletcher, 1995). If a researcher argues that his or her algorithm A works faster than comparison algorithms B and C with data set D and parameters P, the research set-up often does not follow the blinding principle familiar from psychology, medicine, and social sciences. The blinding principle is necessary if one wants to get rid of any experimenter's bias—in that case, the researcher should not be able to choose B, C, D, and P favorably for his or her own algorithm A. Another variation of the argument above is that in the traditional experimental protocol in natural sciences a researcher should be an outsider to the phenomenon to be explained—but in computer science one might ask, "How much of an outsider can one be to a phenomenon he or she has created?"

Although many arguments that computer science is an empirical science have been *descriptive* arguments, there is also a strong normative movement to increase the amount of experimentation in computer science. There are numerous studies in which research reports in computer science have been compared with research reports in other fields, usually natural sciences, and in which the researchers have found that computer scientists experiment significantly less than researchers in many other disciplines. This has led many authors to advise computer scientists to experiment more (Tichy et al., 1995; Denning, 2005b; Glass, 1995—in this early article Glass seems to have argued that even theoretical computer science needs experimental or observational evaluation). However, many of those authors have failed to justify why computer scientists *should* aspire to work like natural scientists or other scientists. One might justly ask, "If computer science is different from the other sciences, why should its methods be the same?"

3.3.4 Objections to the *Science* Part of Computer Science

There is plenty of criticism of the term *science* in computer science, yet it is sometimes difficult to tell when the critique is about wrong naming of computer *science*, and when the critique is about the scientific content, scientific aims, or the scientific method in computer science. Some pro – ponents of a formalist tradition of computer science, especially Edsger Dijkstra, have argued that an empirical bent is detrimental to computer science, and that a successful computer science must build on mathematics. Usually, however, the critique is not directed towards empirical research as such. The critique is usually directed towards the centrality and prospects of empirical research in computer science, or towards the parallels drawn between natural sciences and computer science.

Peter Fletcher took issue with Denning, Glass, and Tichy et al.'s preoccupation with experimenta – tion, and noted that without the theoretical principle of Turing-equivalence of all computers (that particular characteristics of computers are, in principle, irrelevant), there would be no academic dis – cipline of computing, but just eclectic knowledge about particular machines (Fletcher, 1995). Fletcher wrote that much of research in computing is not of the sort "[seek] *the best solution to a previously specified problem*" (Fletcher, 1995). He wrote that often computer scientists work with problems that are poorly understood, and where one major goal is to understand the problem and delimit it more precisely. Fletcher argued that, "*Computer science could not consist entirely of* [solving precisely specified problems], *since someone has to think up the precisely specified prob – lems in the first place and convince us that they are worth pursuing*" (Fletcher, 1995).

Juris Hartmanis characterized the difference between the natural sciences and computer science:

> [1993] *Theories in computer science do not compete with each other for which better explains the fundamental nature of information. Nor are new theories developed to re – concile theory with experimental results that reveal unexplained anomalies or new, un – expected phenomena as in physics. In computer science there is no history of critical experiments that decide between the validity of various theories, as there are in physic – al sciences.*
>
> (Hartmanis, 1993)

The difference between research of naturally occurring phenomena and artificial (human-made) phenomena has been the foremost target of criticism of those who argue that computer *science* is a wrong name. For instance, the observations about algorithm behavior, about usability of machinery and software, or about information retrieval, are observations of things that computer scientists have constructed. Those observations do not tell anything new about the world—those observations tell only about how well previous computer scientists have done their job. This, some critics argue, is not the task of science. Therefore, the thinking goes, computer science should not be called a sci – ence.

In a sense, the argument above is justified. If the phenomena that computer science studies—be it information, computers, computations, processes, usability, programs, or algorithms—is thoroughly human-made, then studies of those phenomena may not reveal anything about the natural world. If the term *science* refers strictly to studies of naturally occurring phenomena, then, similar to math – ematics, computer science might not be a science.

For instance, George McKee wrote that even if the difference between the subject matters of com – puter science and traditional scientific fields is recognized, people in computer science have differ – ent goals and methodologies than people in traditional sciences (McKee, 1995). He argued that in the natural sciences, research is based on observations (data), which scientists can explain, predict, and replicate. In the field of computing, McKee argued, there is no data beyond the computer and programs, which behave exactly as they were designed to behave. In a similar manner, also Brooks argued that computer science is not a science but a synthetic, engineering discipline (Brooks, 1996). However, neither McKee nor Brooks argued that computer science should change—they argued that the name just does not reflect what actually happens in computer science. They argued that computer scientists are not doing science although the name of the field suggests they would be do – ing science.

3.3.5 Are There Laws in Computer Science?

Frederick P. Brooks Jr. wrote that a new fact, a new law, is an accomplishment in science, and warned that computer scientists should not confuse any of their products with laws (Brooks, 1996). There is, indeed, a problem of what are laws in computer science, if there even are such things. Un – fortunately the term *law* has been used in many different senses, often inconsistently. Two of the most common meanings of the term *law* are (1) *regular patterns* of (causal) events and (2) *descrip – tions* of those regular patterns. Note the difference: in the former meaning laws are independent of whatever people may think about them—they obtain because that is how the world works; but in the latter meaning laws are constructions that people have made to describe how the world works—constructions that can be wrong. An example of the former might be the phenomenon that any two masses attract each other; an example of the latter might be Newton's law of universal gravitation:

$$F = G\frac{m_1 m_2}{r^2}$$

The first meaning of the term is clear in philosopher of science Mario Bunge's notion that *objective law* designates "*an objective pattern of a class of facts (things, events, processes), i.e. a certain con – stant relation or mesh of constant relations really obtaining in nature, whether we know it or not*" (Bunge, 1998:391-392). In that sense, *laws* refer to recurring patterns of how things in the world happen, independent of scientists' theoretical constructions (hypotheses, theories, or such). The second meaning of the term is what Bunge called a *law formula*: "*a proposition or a propositional function that is usually supposed to describe a law or a part of a law*" (Bunge, 1998:392). Bunge noted that in scientific texts laws are mostly of the second kind. But generally speaking, in natural sciences the two meanings are often conflated together. That is, the term *law* is often used to refer to generalizations of physical behavior based on empirical data, and those generalizations are be – lieved to be faithful descriptions of how the world works. But laws-as-descriptions are really just fallible attempts to portray laws-as-objective-patterns.

In the natural sciences laws-as-objective-patterns are generally understood to be, among other things, *universal* (they apply everywhere in the universe the same way), *eternal* (they appear to be timeless), *absolute* (nothing in the universe appears to affect them), and *omnipotent* (nothing es – capes them) (Davies, 1992:82-83). A problem with laws in computer science is that on one hand, a lot of computer science deals with theoretical constructions and aim to explain relations within and

between formal systems (formal systems do not exist without people creating them); but on the oth – er hand, a lot of computer science attempts to explain processes that happen regardless of people. There is a reason to ask whether the constructions in computer science are more like mathematical *theorems* or like physical *laws-as-descriptions*, or if computer science has both of those. This ques – tion will be discussed in more detail later.

Donald Knuth has argued that there is a certain difference between the subject of computer science and the subject of natural sciences. He wrote that "*like mathematics, computer science will be dif – ferent from the other sciences in that it deals with man-made* [sic] *laws which can be proved, in – stead of natural laws which are never known with certainty*" (Knuth, 1974). Knuth's position seems to be that laws in computer science are more like theorems than laws of nature. His wording im – plies that the laws in computer science exist because there are intelligent beings that make them ex – ist. By the term *man-made laws*, Knuth might mean things like theorems that are useful within a certain framework (laws-as-constructions?). But Knuth's wording "*man-made law*" could also refer to a symbol-theory system that describes universal, eternal, absolute, and omnipotent laws.

So *law* seems to be quite a confusing term. It is interesting to note that although Knuth is a math – ematician, he preferred the term *laws* to *constructions*, *theories*, *theorems*, or *hypotheses*. Perhaps it would be clearer to write, "*like mathematics, some parts of computer science deal with axiomatic human constructions (which can be proven within a certain framework) instead of naturally occur – ring phenomena (which cannot be explained with certainty)*". (But that statement has problems of its own, too.)

Finally, that (some) laws in computer science can be proven true indicates that Knuth took there to be a *universal logic*, following which, any independent thinker would proceed from the same ax – ioms and premises to the same conclusions. It seems that in Knuth's sense of the term, *laws* of com – puter science are sets of statements about computing that researchers have constructed to perfection within the bounds that researchers have set. Hence the factuality of the laws of computer science vis-à-vis the imperfect interpretations of the unbounded external world made by natural scientists.

The difficulty of the term *law* is evident in quite some texts in computer science. For instance, in his defense of experimental computer science, Peter Denning wrote:

> [1980] *The hypothesis* [in experimental computer science] *may concern a law of nature —for example, one can test whether a hashing algorithm's average search time is a small constant independent of the table size by measuring a large number of retrievals.*
>
> (Denning, 1980b)

It is hard to see in which sense the average search time of a hashing algorithm *A* (which is a human construction), implemented on a computer brand *B* (which is a human construction), both *A* and *B* relying on the theoretical-technical framework of modern computation (which is a human construc – tion), would be a law of nature in the sense that it would tell anything about naturally occurring things. Denning claimed that the argument "computer science studies man-made objects and hence is not a natural science" is a red herring (Denning, 2005b), but Denning used a confused version of laws of nature in his own argument (Denning, 1980b).