Ubiquity, an ACM publication
June 2014

Ubiquity Symposium

The Science of Computer Science

Closing Statement

by Richard Snodgrass and Peter Denning

Editors' Introduction

Where does computer science as an intellectual discipline fit in human discourse? The traditional curricular distinction of theory and systems points to computer science being both a mathematical and an engineering discipline. What is perhaps more controversial is whether computer science can also claim to be a true science. Over a dozen contributors have looked at this question of identity from as many viewpoints. In this closing statement, we emphasize six themes running through these 16 commentaries and draw some conclusions that seem to be supported by the symposium.

Ubiquity Symposium

The Science of Computer Science

Closing Statement

by Richard Snodgrass and Peter Denning

Where does computer science as an intellectual discipline fit in human discourse? Is it a robust tributary to mathematics, Mother of all Sciences, borrowing formalisms and fundamental notions and contributing its own as well? Is it a science, sharing the mindset and philosophical underpinnings of physics, chemistry, biology, and psychology? Or is it engineering, bringing constructive approaches and wondrous creations?

Most computer scientists feel that the proper reply is not *either-or* among these choices, but rather *and*, all of them. The traditional curricular distinction of theory and systems points to computer science being both a mathematical and an engineering discipline. What is perhaps more controversial is whether computer science can also claim to be a true science.

Over a dozen contributors have looked at this question of identity from as many viewpoints. In this closing statement, we emphasize six themes running through these 16 commentaries and draw some conclusions that seem to be supported by the symposium.

We refer to those authors appearing just once in the symposium by their last names: Cuny (contribution No. 4), Carlson (No. 5), Yaffe (No. 6), Anderson (No. 7), McGeoch and Moret (No. 8), Sjøberg (No. 9), Guzdial (No. 10), Alderson (No. 11), Gelenbe (No. 12), Tichy (No. 13), Rosenbloom (No .14), and Bell (No. 15). We refer to Vint Cerf's *CACM* essay "Where is the Science in Computer Science" as Cerf/A and in his follow-on essay "Computer Science Revisited" (No. 2) as Cerf/B, and Peter Denning in his Opening Statement as Denning/A and in his essay on "Performance Analysis: Experimental computer science at its best" (No. 3) as Denning/B.

What is Science?

Denning/A got us started by emphasizing “experimental methods.” Tichy quoted Richard Feynman: “The fundamental principle of science, the definition almost, is this: The sole test of the validity of any idea is experiment.”

Cerf/A stated, “in the physical world, science is largely about models, measurement, predictions, and validation. Our ability to predict likely outcomes based on models in fundamental to the most central notions of the scientific method.” Denning/B offered the following definition: “The key affects of experimental science are an apparatus for collecting data, a hypothesis, and systematic analysis to see whether the data supports the hypothesis.” (One of us has argued elsewhere that debugging, a task familiar to all computer scientists, is in fact a highly specific example of the scientific method [1].) Carlson's definition is broader: “Science is everything we know about the physical Universe, and how we know it.” He further explained, scientists “always fit theories to facts, never facts to theories.” And added, “Doing science is all about making sure you're not fooling yourself.” This gets back to the notion of science as uncovering “empirical truth.” Yaffe went into more detail by listing 16 character traits of science, emphasizing Einstein's assertion that “The whole of science is nothing more than a refinement of everyday thinking.”

The symposium writers seem to share a commonly-held definition of science.

Is There a Science of Computation?

Denning/A gave a brief history of the field of computing from the 1940s to the present, a stretch of 70 years. He suggested “computing began as science, morphed into engineering for 30 years while it developed technology, and then entered a science renaissance about 20 years ago. Although computing had subfields that demonstrated the ideas of science, computing as a whole has only recently begun to embrace those ideals.”

Moret's history displays a similar arc. “In the pioneering era of computing (the 1960s and early 1970s), theoretical and experiment algorithms were much the same, and thorough experimental assessments were common across all of computer science … Along the way, the theory and experiment became decoupled, perhaps because of enormous complexities in setting up meaningful experiments We are move now toward reintegration of these two aspects of our field.”

Denning/A mentioned the famous letter in the journal *Science*: “In 1967, Allen Newell, Alan Perlis, and Herbert Simon famously defended the new field as a true science concerned with all aspects of ‘phenomena surrounding computers.’“ (Guzdial brought up Perlis in another context: “[Perlis] argued that computer science is the study of *process*, [which] is important to everyone.”) Cerf/A argued “the term ‘computer science’ raises expectations, at least to my mind, of an ability to define models and to make predictions about the behavior of computes and computing systems,” and concluded, “as a group of professionals devoted to the evolution, understanding, and application of software and hardware to the myriad problems, opportunities, and activities of modern society, we have a responsibility of pursue the science of computer science.”

Cerf/B went on to state, “To the degree that some aspects of computing are subject to analysis and modeling, it is fair to say there is a rigorous element of science in our field.” He discussed both the challenges of such a science and opportunities to address these challenges: “models used in hardware analysis are more tractable than those used in software analysis,” “modeling is a form of abstraction and is a powerful tool,” and modeling “aid[s] our ability to analyze and make predictions about the behavior of complex processes.” He concluded, "there really is science to be found in computer science.”

McGeoch made the same point concerning algorithm research: “There are three main activities in algorithm research: analysis, which is about predicting performance [echoing this identification of science by others previously mentioned]; design, which is about finding new and better ways to solve problems; and models of computation, which is about understanding how design and analysis changes when the basic operations (defined by the platform) are modified.” In all three areas, theoretical methods necessarily make simplifying assumptions—worst-case inputs, dominant operation costs, and the RAM. Experimental methods fill in the gaps between those simplifying assumptions and real experience by incorporating interesting subclasses of inputs, constant factors and secondary costs, and more realistic machine models.”

Sjøberg gave many examples of empirical investigation in software engineering and Tichy observed, “Journals and conference have been started that are devoted to empirical work and methods. In software engineering as well as in other areas, such as in experimental algorithms or usability research, the scientific method is now being practiced.” He continued, “In summary, there is no question now that software research is following the scientific paradigm.”

The consensus is clear: there *is* a science of computing.

Is It a Natural Science?

Denning/A stated, “In 1969, Simon went further [than his *Science* letter], arguing in a famous book, *Sciences of the Artificial*, that several fields, including economics and computer science, met all the traditional criteria for science, and deserved to be called sciences even if their focal phenomena are 'man-made as opposed to natural.'" Simon emphasized the "artifactual" nature of computers and of computation, in that their existence was predicated on a task that needed to be done. So his reply would be: "No, computer science is not a natural science. It is artifactual, that is, artificial." Cerf/A continued that thought: "Alan Turing drew dramatic attention to the artificiality of these systems with what we now call the Universal Turing Machine. This conceptual artifact emphasizes the artificial nature of computation. ... That software is an artifact seems obvious. Moreover, it is a strikingly complex artifact filled with layer upon layer of components that exhibit dependencies and complex and often unpredicted (not to say unpredictable) behaviors."

Denning/A noted, "In 2008, Kari and Rozenberg wrote an excellent overview of natural information processes that intersected with computer science." Denning/A's conclusion was: "The old lament that computing dealt solely with artificial information processes went by the wayside." Gelenbe went further, arguing: "Evolution itself can be viewed as a computational process whose outcome is to seek to continually optimize a living species to the conditions of their environment and to other species that share the same environment." He went on to give examples of "the computation that is inherent in chemistry,natural selection, gene regulatory networks, and neuronal systems." Rosenbloom agreed: "The earlier emphasis on computers may have blinded us to the existence of natural forms of computing. Now we can see that various forms of computing are already embodied, for example, within living organisms."

Rosenbloom followed this argument against a distinction between natural and artificial science with two additional counterarguments. "First, the distinction between natural and artificial is itself artificial and increasingly meaningless." Second, "our continually improving abilities to modify natural processes at finer levels of detail makes it increasingly difficult across the board to distinguish what is due to human agency and what is not."

Let us here make another argument for computer science as a natural science, following up on Yaffe's observation that science works because it assumes "the laws that govern the real world are knowable and comprehensible. We can discover these laws and learn to use them to understand what has already happened, what is currently happening—and most importantly—what is going to happen. This is called 'prediction.'" This focus on prediction mirrored that of many of the other authors in this symposium and raises the question, how do we know that computer science theories will endure? Yaffe also said science is reproducibility. "Science is all about trying to discover things that are universally true, that do not depend on place and time."

The question arises, why would we think a computer science theory would *endure*? Why would experimental results be reproducible? The source of endurance and reproducibility is the fact that computer science artifacts are created by humans. Sjøberg emphasized "software development is a human activity carried out in organizations" (which are also human activities). Humans are not arbitrary, but rather behave and create artifacts with structures deriving from those in their brains, structures that arose out of inherent psychological abilities and limitations, which themselves arose out of biological processes such as evolution, which themselves arose out of organic chemical processes, which themselves arose out of fundamental physical processes. Humans are of nature and thus their tools and artifacts are of nature.

Rosenbloom concluded, "It is time to put the final nails in the coffin of the *argument from artificiality,* that computing isn't a true science because it studies artificial rather than natural phenomena."

Rosenbloom went further: "If computing is science, where does it fit in the traditional taxonomy of the physical, life, and social sciences? Conceivably it could sit within one or more of the three existing domains; however, its core subject matter, of information and its transformation, simply does not fit within any of these domains. Instead as laid out at some length in *On Computing* [2], it amounts to a fourth domain, and a full equal of the other three."

Computation is a *natural* science.

What Should We Call It?

As we have seen, computer science is in part a science. But as emphasized in the beginning, it is not just a science. Computer science is actually the union of three quite disparate perspectives: the mathematical perspective, the scientific perspective, and the engineering perspective.

Consider the term "theory." In the mathematical perspective a theory is a collection of mathematical objects and theorems about those objects. Examples are asymptotic complexity theory (in algorithms), program verification theory (in programming languages), and serializability theory (in databases).

In the scientific perspective, a theory as we have seen is a set of coherent predictions that have been empirically tested and shown to hold in reality.

In the engineering perspective, computer science attempts to derive systems, algorithms, languages, software engineering methodologies that are "better" in some sense: faster, cheaper, more functional. As an example, McGeoch and Moret mentioned four Dagstuhl workshops on "algorithm engineering."

It is quite awkward that the name of our discipline contains only one of its three constituent perspectives. So, what should we call the part of computer science that espouses the scientific perspective?

Denning/B very early used the term “experimental computer science.”

McGeoch made “a distinction between ‘empirical,’ which relies primarily on observation of algorithms in realistic scenarios—analogous to a field experiment in the natural sciences—and ‘experimental,’ which emphasizes systematic manipulation of test parameters, more like a laboratory experiment.” Moret shared the following: “I got a sense that most of my colleagues in both the U.S. and Europe felt that empirical had negative connotations—what came into their minds was something like ‘ad hoc’—whereas experimental had neutral to positive connotations—it brought to mind something much more systematic and thorough. This perception does not really agree with the formal definitions of the two words.” His observation points back to McGeoch's characterization. Hence the *Journal of Experimental Algorithmics*.

Paul Cohen wrote a book in 1995 entitled *Empirical Methods for Artificial Intelligence*, discussing a scientific methodology of wide application. And McGeoch's book published in 2011 is entitled *A Guide to Experimental Algorithmics*.

Sjøberg termed the area he works in as “empirical software engineering” or “empirical software research.” Others have used the term “design science.”

Cerf/B used the term “analytic” and stated “abstraction and modeling are key to making things analytic.” The term has a formal, mathematical undertone that may be confused with the mathematical perspective.

One major problem with using either of the terms empirical or experimental is that the engineering perspective utilizes experiments extensively to judge improvement of novel designs, methodologies, and algorithms; in contrast with science, which uses these same instruments not to judge improvement, but rather to further understanding, to test proposed predictive theories, and to follow Carlson's dictum to “always fit theories to facts, never facts to theories.”

One of us prefers to term the “science of computer science” or the “science of computation” with a new word, *ergalics* [3].

What are Examples of Computer Science Theories?

Denning/A mentioned “performance modeling and prediction [as] exemplifying the ideals of science.” Denning/B elaborated with two examples within the general area of performance analysis. The first concerns the virtual memory hypothesis, discussing the substantial amount of work by many teams culminating in Denning's Locality Theory [4]. McGeoch and Sjøberg also mentioned this contribution.

Denning/B's second example was queueing network models, specifically the Jackson-Gordon-Newell theory that “the steady-state probability of seeing a given apportionment of jobs among the devices of the system is of the product form—it is a series of factors, each depending on the number of jobs present at a given device and on the parameters of that device.” Cerf/B stated that Kleinrock's “most famous contribution was the recognition that the state space explosion could be tamed by [Kleinrock's] independence assumption in which the statistics of the traffic flowing through the network were regenerated at each node using statistically identical but independent variables.” Little's Law and the Bard-Schweiter equations later emerged from that work, laws that have been shown to apply to broad classes of queueing systems and that have been extensively validated on real systems.

Sjøberg enumerated many theories resulting from empirical software research. “Examples of theories, also called models, are Human Information Processing (HIP) models, the Model of Personal Computer Utilization (MPCU), the Technology Acceptance Model (TAP), and the Theory of Planned Behavior (TPB). Although their origin is not within the field of software research, they have been successfully applied or been the basis for other theories or models proposed in our field. Examples of such theories ... are theories of program understanding and comprehension, a theory of coordination in software engineering, a model of inspection processes, a model of complexity of delegation, and economic models for the business value of a development process.” He also listed some important results from that kind of research: Carefully controlled experiments have shown that skill level and task complexity are stronger predictors of pair programming performance; there were no strong indications that personality had any effect. “Inspections are consistently shown to reduce errors ... Well-applied design patterns will generally improve maintenance activities, and such patterns also improve communication among beginners. Static type-checking across module interfaces is effective in reducing the number of errors.”

Other scientific theories within computation include Cognitive Load Theory (concerning multi-media applications), Fitts’ Law (concerning mouse movement), Manhattan World Theory (concerning image analysis), Phase-Transition Theory (applying to NP-complete algorithms), and Social Network Theory (concerning man-made graphs, such as those encountered in Facebook).

What Are Implications To Curricula And To Teaching And Doing Research?

Cuny raised the challenge of broadening participation in computing and argued “high school is the most critical point. High school should be the place where students … see computing as an empowering discipline that has the potential to impact many aspects of society.” She stated the existing AP CS course is “a year of Java programming [that] appeals to a very narrow group of students.” She noted the National Science Teachers Association and the National Math and Science Initiative are partners in the effort to increase participation. (Both groups we imagine would enthusiastically support Andersen's injunction to choose math in high school, in part because “math—or, at least, mathematical understanding—is a prerequisite for understanding computing.”) This raises the possibility that by emphasizing the science perspective as a core part of the discipline of computer science, there may be increased interest in our discipline among high school students, including underrepresented groups. Carlson emphasized “the enterprise of science [can be accomplished] by presenting it in a way that is both intellectually stimulating and emotionally compelling.”

McGeoch stated unequivocally, “I think general experimental methodology should play a role in computer science education at both levels.” She went on to assert that “experimental algorithmics is rich in examples and projects for students.” Sjøberg agreed that “empirical evaluation should become a natural part of our teaching curriculum” at both the undergraduate and graduate levels.

By changing the context of introductory undergraduate classes, Guzdial argued “we can explore scientific principles that ... are not traditionally covered in introductions to computing.” He continued, “These are pretty deep, fundamental ideas that come about naturally in a Media Computation context, and students really do engage in that part of the course. The point of the context is to explain to the students why they should care about those principles.” Rosenbloom gave several examples of how both computing specifically and in integration with the three other great domains of science could be incorporated into STEM education. And Bell went further, illustrating how some of the great ideas of computer science could be communicated effectively even to students as young as 10 years old.

Conclusions

We draw seven conclusions from the symposium.

First, the question of whether computer science is science is as old as the field. It arose because traditional scientists did not recognize computational processes as natural processes. Even though much of the early energy of the field was devoted to building systems and understanding their theoretical limits, the field developed two important scientific theories. The theory of locality studied memory reference patterns of computations and the theory of performance evaluation validated queueing network models for reliable performance predictions of computer systems.

Second, there is a growing consensus today that many of the issues we are studying are so complex that only an experimental approach will lead to understanding. The symposium documents advances in algorithmics, biology, social networking, software engineering, and cognitive science that use empirical methods to answer important questions.

Third, scientists in many fields now recognize the existence of natural information processes in their fields. This settles an early controversy that said CS deals only with the behavior of artificial information processes. CS is not off in a compartment for "sciences of the artificial" as Herb Simon thought. CS is indeed a natural science.

Fourth, we need an alternative to the awkward term "science of computer science." None of the several options mentioned by the authors are yet universally accepted.

Fifth, there now exists a core group of scientific theories about computation, with many other areas working to add to that group. They hunger for the understanding that rigorously tested theories could provide.

Sixth, because information processes are pervasive in all fields of science, computing is necessarily involved in all fields, and computational thinking has been accepted as a widely applicable problem-solving approach. Many students are now selecting computer science majors because it preserves their flexibility in choosing a career field later.

Finally, computing presented as science is very engaging to middle and high school students, belying the unfortunate and prevalent notion that computer science equals programming. A growing number of STEM teachers are embracing these new methods.

References

[1] Clayton Morrison and Richard T. Snodgrass. Computer Science Can Use More Science. *Communications of the ACM* 54, 6 (2011), 36-39.

[2] Paul S. Rosenbloom. *On Computing: The Fourth Great Scientific Domain*. MIT Press, Cambridge, 2013.

[3] Richard Snodgrass. Ergalics: A natural science of computation., University of Arizona. 2010.

[4] Peter J. Denning. The Locality Principle. *Communications of the ACM* 48, 7 (2005), 19-24.

About the Authors

Richard Snodgrass is a Professor of Computer Science at the University of Arizona, working in the areas of ergalics and temporal databases.

Peter J. Denning (pjd@nps.edu) is Distinguished Professor of Computer Science and Director of the Cebrowski Institute for information innovation at the Naval Postgraduate School in Monterey, California, is Editor of ACM *Ubiquity*, and is a past president of ACM.

DOI: 10.1145/2633608