Peter J. Denning

Is Computer Science Science?

Computer science meets every criterion for being a science, but it has a self-inflicted credibility problem.

What is your profession?

Computer science.

Oh? Is that a science?

Sure, it is the science of information processes and their interactions with the world.

I'll accept that what you do is technology; but not science. Science deals with fundamental laws of nature. Computers are man-made. Their principles come from other fields such as physics and electronics engineering.

Hold on. There are many natural information processes. Computers are tools to implement, study, and predict them. In the U.S. alone, nearly 200 academic departments recognize this; some have been granting CS degrees for 40 years.

They all partake of a mass delusion. The pioneers of your field genuinely believed in the 1950s that their new field was science. They were mistaken. There is no computer science. Computer art, yes. Computer technology, yes. But no science. The modern term, Information Technology, is closer to the truth.

I don't accept your statements about my field and my degree. Do you mind if we take a closer look? Let's examine the accepted criteria for science and see how computing stacks up.

I'm listening.

JASON SCHNEIDER

Common Understandings of Science

Our field was called computer science from its beginnings in the 1950s. Over the next four decades, we accumulated a set of principles that extended beyond its original mathematical foundations to include computational science, systems, engineering, and design. The 1989 report, *Computing as a Discipline*, defined the field as:

> "The discipline of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, 'What can be (efficiently) automated?'" [3, p. 12]

Science, engineering, and mathematics combine into a unique and potent blend in our field. Some of our activities are primarily science—for example, experimental algorithms, experimental computer science, and computational science. Some are primarily engineering—for example, design, development, software engineering, and computer engineering. Some are primarily mathematics—for example, computational complexity, mathematical software, and numerical analysis. But most are combinations. All three sets of activities

The objection that computing is not a science because it studies man-made objects (technologies) is a red herring. Computer science studies information processes both artificial and natural.

draw on the same fundamental principles. In 1989, we used the term "computing" instead of "computer science, mathematics, and engineering." Today, computing science, engineering, mathematics, art, and all their combinations are grouped under the heading "computer science."

The scientific paradigm, which dates back to Francis Bacon, is the process of forming hypotheses and testing them through experiments; successful hypotheses become models that explain and predict phenomena in the world. Computing science follows this paradigm in studying information processes. The European synonym for computer science—informatics—more clearly suggests the field is about information processes, not computers.

Science	Art
principles	practice
fundamental recurrences	skilled performance
explanation	action
discovery	invention
analysis	synthesis
dissection	construction

Table 1. Science vs. art.

The lexicographers offer two additional distinctions. One is between pure and applied science; pure science focuses on knowledge for its own sake and applied focuses on knowledge of demonstrable utility. The other is between inexact (qualitative) and exact (quantitative) science; exact science deals with prediction and verification by observation, measurement, and experiment.

Computing research is rife with examples of the scientific paradigm. Cognition researchers, for example, hypothesize that much intelligent behavior is the result of information processes in brains and nervous systems; they build systems that implement hypothesized information processes and compare them with the real thing. The computers in these studies are tools to test the hypothesis; successful systems can be deployed immediately. Software engineering researchers hypothesize models for how programming is done and how defects arise; through testing they seek to understand which models work well and how to use them to create better programs with fewer defects. Experimental algorithmicists study the performance of real algorithms on real data sets and formulate models to predict their time and storage requirements; they may one day produce a more accurate theory than Big-O-Calculus and include a theory of locality. The nascent Human-Computer Interaction (HCI) field is examining the ways in which human information processes interact with automated processes.

By these definitions, computing qualifies as an exact science. It studies information processes, which occur naturally in the physical world; computer scientists work with an accepted, systematized body of knowledge; much computer science is applied; and computer science is used for prediction and verification.

The objection that computing is not a science because it studies man-made objects (technologies) is a red herring. Computer science studies information processes both artificial and natural. It helps other fields study theirs too. Physicists explain particle behavior with quantum information processes—some of which, like entanglement, are quite strange—and verify their theories with computer simulation experiments. Bioinformaticians explain DNA as encoded biological information and study how transcription enzymes read and act on it; computer models

of these processes help customize therapies to individual patients. Pharmaceutical and materials labs create man-made molecules through computer simulations of the information processes underlying chemical compositions.

To help define the boundaries of science, lexicographers also contrast science with art. Art refers to the useful practices of a field, not to drawings or sculptures. Table 1 lists some terms that are often associated with science and with art. Programming, design, software and hardware engineering, building and validating models, and building user interfaces are all "computing arts." If aesthetics is added, the computing arts extend to graphics, layout, drawings, photography, animation, music, games, and entertainment. All this computing art complements and enriches the science.

Science in Action

In his remarkable book about the workings of science, *Science in Action*, the philosopher Bruno Latour brings a note of caution to the distinction between science and art [7]. Everything discussed in this column (a systematized body of knowledge, ability to make predictions, validation of models), is part of what he calls ready-made-science, science that is ready to be used and applied, science that is ready to support art. Much science-in-the-making appears as art until it becomes settled science.

Latour defines science-in-the-making as the processes by which scientific facts are proposed, argued, and accepted. A new proposition is argued and studied in publications, conferences, letters, email correspondence, discussions, debates, practice, and repeated experiments. It becomes a "fact" only after it wins many allies among scientists and others using it. To win allies, a proposition must be independently verified by multiple observations and there must be no counterexamples. Latour sees science-in-the-making as a messy, political, human process, fraught with emotion and occasional polemics. The scientific literature bears him out.

Everything Latour says is consistent with the time-honored definition of the science paradigm. After sufficient time and validation, a model becomes part of the scientific body of knowledge.

Internal Disagreement

Computer scientists do not all agree whether computer science is science. Their judgment on this question seems to depend upon in which tradition they grew up. Hal Abelson and Gerry Sussman, who identify with the mathematical and engineering traditions of computing, said, "Computer science is not a science, and its ultimate significance has little to do with computers" [1]. They believe that the ultimate significance is with notations for expressing computations. Edsger Dijkstra, a mathematician who built exquisite software, frequently argued the same point, although he also believed computing is a mathematical science. Walter Tichy, an experimentalist and accomplished software builder, argues that computer science is science [12]. David Parnas, an engineer, argues that the software part of computer science is really engineering [10]. I myself have practiced in all three traditions of our field and do not see sharp boundaries.

Even the Computer Science and Technology Board of the National Research Council is not consistent. In 1994, a panel argued that experimental computer science is an essential aspect of the field [9]. In 2004, another panel discussed the accomplishments of computer science research; aside from comments about abstraction in models, they say hardly a word about the experimental tradition [8].

Paul Graham, a prominent member of the generation who grew up with computers, invented the Yahoo! store and early techniques for spam filters; he identifies with computing art. He says: "I never liked the term 'computer science'. … Computer science is a grab bag of tenuously related areas thrown together by an accident of history, like Yugoslavia. … Perhaps one day 'computer science' will, like Yugoslavia, get broken up into its component parts. That might be a good thing. Especially if it means independence for my native land, hacking" [5, p. 18]. He is not arguing against computer science, but for an appella-

Computer scientists do not all agree whether computer science is science. Their judgment on this question seems to depend upon in which tradition they grew up.

tion like computer art that is more attractive to hackers (his term for elite programmers).

Dana Gardner, of the Yankee Group, does not like this notion. He compares the current state of software development to the pre-industrial Renaissance, when wealthy benefactors commissioned groups of highly trained artisans for single great works of art [4]. He says, "Business people are working much closer to the realm of Henry Ford, where they are looking for reuse, interchangeable parts, automated processes, highly industrialized assembly lines."

OK, so computing has much art and its own science, although some of your people are not sure about the science. However, does computer science have depth? Are there fundamental principles that are non-obvious to those who do not understand the science? Who would have thought that the speed of light is the same for all observers until Einstein postulated relativity? Or that particles ride probability waves until Schroedinger postulated quantum mechanics? Is there anything like this in computer science?

Area	Problem
Computation	• Unbounded error accumulation on finite machines • Non-computability of some important problems • Intractability of thousands of common problems • Optimal algorithms for some common problems • Production quality compilers
Communication	• Lossless file compression • Lossy but high-fidelity audio and video compression • Error correction codes for high, bursty noise channels • Secure cryptographic key exchange in open networks
Interaction	• Arbitration problem • Timing-dependent (race-conditioned) bug problem • Deadlock problem • Fast algorithms for predicting throughput and response time • Internet protocols • Cryptographic authentication protocols
Recollection	• Locality • Thrashing • Search • Two-level mapping for access to shared objects
Automation	• Simulations of focused cognitive tasks • Limits on expert systems • Reverse Turing tests
Design	• Objects and information hiding • Levels • Throughput and response time prediction networks of servers

Table 2. Some non-obvious problems solved by computing principles.

Can Computer Science Surprise?

Table 2 lists six major categories of computing principles along with examples of important discoveries that are not obvious to amateurs [2]. By exploiting these principles, professionals are able to solve problems that amateurs would find truly baffling.

OK. I'm finding this compelling. But I still have a concern. Is it worth investing either my time or R&D dollars in computer science? In his 1996 book, The End of Science, *journalist John Horgan argues that most scientific fields have saturated. They have discovered most of their basic principles and new discoveries are less and less frequent. Why is computer science different? Once the current round of computer-science-in-the-making settles out, and assuming the hackers don't secede, will computer science die out?*

Computer Science Thrives on Relationships

Horgan argued in 1996 that new scientific discoveries require mastering ever-greater amounts of complexity. In 2004 he repeated his main conclusion: "Science will never again yield revelations as monumental as the theory of evolution, general relativity, quantum mechanics, the big bang theory, DNA-based genetics. ... Some far-fetched goals of applied science—such as immortality, superluminal spaceships, and superintelligent machines—may forever elude us" [6, p. 42].

Has computer science already made all the big discoveries it's going to? Is incremental progress all that remains? Has computer science bubbled up at the end of the historical era of science?

I think not. Horgan argues

that the number of scientific fields is limited and each one is slowly being exhausted. But computer science is going a different way. It is constantly forming relationships with other fields; each one opens up a new field. Paul Rosenbloom has put this eloquently in his recent analysis of computer science and engineering [11].

Rosenbloom charts the history of computer science by its relationships with the physical, life, and social sciences. With each one computer science has opened new fields by implementing, interacting, and embedding with those fields. Examples include autonomic systems, bioinformatics, biometrics, biosensors, cognitive prostheses, cognitive science, cyborgs, DNA computing, immersive computing, neural computing, and quantum computing. Rosenbloom believes that the constant birth and richness of new relationships guarantees a bright future for the field.

All right, I'll accept that. You have science, you have art, you can surprise, and you have a future. But you also have a credibility problem. In the 1960s your people claimed they would soon build artificially intelligent systems that would rival human experts and make new scientific discoveries. In the 1970s they claimed that they would soon be able to systematically produce reliable, dependable, safe, and secure software systems. In the 1980s it was the disappearance of paper, universities, libraries, and commuting. None of these things happened. In the 1990s you contributed to the Internet boom and then crashed with the dot-com bust. Now you're making all sorts of claims about secure systems, spam-blocking, collaboration, enterprise systems, DNA design, bionics, nanotechnology, and more. Why should I believe you?

Validating Computer Science Claims

There you have us. We have allowed the hype of advertising departments to infiltrate our laboratories. In a sample of 400 computer science papers published before 1995, Walter Tichy found that approximately 50% of those proposing models or hypotheses did not test them [12]. In other fields of science the fraction of papers with untested hypotheses was about 10%. Tichy concluded that our failure to test more allowed many unsound ideas to be tried in practice and lowered the credibility of our field as a science. The relative youth of our field—barely 60 years old—does not explain the low rate of testing. Three generations seems sufficient time for computer scientists to establish that their principles are solid.

The perception of our field seems to be a generational issue. The older members tend to identify with one of the three roots of the field—science, engineering, or mathematics. The science paradigm is largely invisible within the other two groups.

The younger generation, much less awed than the older one once was with new computing technologies, is more open to critical thinking. Computer science has always been part of their world; they do not question its validity. In their research, they are increasingly following the science paradigm. Tichy told me that the recent research literature shows a marked increase in testing.

The science paradigm has not been part of the mainstream perception of computer science. But soon it will be. C

References

1. Abelson, H.G. and Sussman, G.J. *Structure and Interpretation of Computer Programs, 2nd ed*. MIT Press, 1996.
2. Denning, P. Great principles of computing. *Commun. ACM 46*, 10 (Nov. 2003), 15–20.
3. Denning, P. et al. Computing as a discipline. *Commun. ACM 32*, 1 (Jan. 1989), 9–23.
4. Ericson, J. The psychology of service-oriented architecture. *Portals Magazine* (Aug. 2004); www.portalsmag.com/articles/default.asp?ArticleID=5872.
5. Graham, P. *Hackers and Painters: Big Ideas from the Computer Age*. O'Reilly and Associates, 2004.
6. Horgan, J. The end of science revisited. *IEEE Computer* (Jan. 2004), 37–43.
7. Latour, B. *Science in Action*. Harvard University Press, 1987.
8. National Research Council. *Computer Science: Reflections on the Field, Reflections from the Field*. National Academy Press, 2004.
9. National Research Council. *Academic Careers for Experimental Computer Scientists and Engineers*. National Academy Press, 1994.
10. Parnas, D. Software engineering: An unconsummated marriage. *Commun. ACM 40*, 9 (Sept. 1997), 128.
11. Rosenbloom, P. A new framework for computer science and engineering. *IEEE Computer* (Nov. 2004), 31–36.
12. Tichy, W. Should computer scientists experiment more. *IEEE Computer* (May 1998), 32–40.

Peter J. Denning (pjd@nps.edu) is the director of the Cebrowski Institute for information innovation and superiority at the Naval Postgraduate School in Monterey, CA, and is a past president of ACM.