Is Computer Science a Science?

Cristian S. Calude

Does Computer Science Exist?

According to

- Britannica (1979)
- Grolier (1993)
- McGraw-Hill Encyclopedia of Science and Technology (1992)
- Webster (1996)

computer science simply does not exist!

For example, a search conducted at
`http://gs213.sp.cs.cmu.edu/prog/webster?computer+science`
has produced the following result:

```
Hypertext Webster Interface

Webster Definition for "computer science"

No definition for 'computer science'.
```

Is Computer Science a Science?

The answer is affirmative: Computer science is a **science**.

Reason: as its name is indicating, computer science is a particular science, i.e. it is a science.

What's wrong with the above argument?

- Computer science is not necessarily a science just as "red herring" is not necessarily a herring, and a "hot dog" is usually not a dog.
- The name of a field is expressing a "programmatic" view and cannot be substituted for the subject itself.
- "Computer science" is particularly a bad name (we don't call mathematics "pen science" nor surgery "lancet science"). Informatics might be a better name, but it seems that it is too late to make a change...

But What Makes a Subject Science?

Due to the obvious circularity, Britannica's explanation is not very useful:

> **[Science] is a discipline in which the elements involved in** scientific **inquiry ... are analyzed and discussed; and the grounds of their validity are evaluated from the points of view of formal logic, practical methodology, and metaphysics.**

Grolier's definition

> **Science may be broadly defined as the development and systematization of positive knowledge about the physical universe.**

is suggesting a naive approach: Say that

> the major goal of computer science is to discover the laws of **information** processing and then to use them to the best design of computer hardware and software.

What Is Information?

Data *represents* or *encodes* information.

Information means value and power.

Information is subjective.

Information exists for human beings, not machines.

Information looks like a powerful *metaphor*.

But, suppose, that information is **only** a metaphor? Might it be to computer science what ether was for physics in the last century?

Do we really need information as a measurable quantity?

Hartmanis' 1994 Opinion

In his "Turing Award Lecture" Hartmanis says:

> *I personally believe that computer science is not only a rapidly maturing science, but that is more. Computer science differs basically from the other sciences... Computer science deals with information, its creation and processing, and with the systems that perform it, much of which is not directly restrained and governed by physical laws.*

Deutsch's 1982 Opinion

Hartmanis' suggestion directly contradicts a much repeated "slogan":

The reason why we find it possible to construct, say, electronic calculators, and indeed why we can perform mental arithmetic, cannot be found in mathematics or logic. *The reason is that the laws of physics "happen" to permit the existence of physical models for the operations of arithmetic* such as addition, subtraction and multiplication. If they did not, these familiar operations would be non-computable functions. We might still know *of* them and invoke them in mathematical proofs (which would be presumably be called "non-constructive") but we could not perform them.

Is Computer Science an Engineering Activity?

Engineering

- **is the art of applying science to the optimum conversion of the resources of nature to benefit man. (Britannica)**
- **is the profession that deals with the design and building of machines, devices, and structures. (Grolier)**

Much of computer science can be viewed as an engineering activity. However, it appears that most engineering problems in computer science have different characteristics, e.g. as a result of the fact that computer science problems are not “directly” constrained by physical laws. Computer science might suggest new engineering paradigms and methodologies.

Brooks' 1975 Opinion

The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of imagination...

Yet the program construct, unlike the poet's words, is real in the sense that it moves and works, producing visible outputs separate from the construct itself.

Knuth's 1992 Opinion

Computer Science and Engineering is a field that attracts a different kind of thinker. I believe that one who is a natural computer scientist thinks algorithmically. Such people are especially good at dealing with situations where different rules apply in different cases; they are individual who can rapidly change levels of abstractions, simultaneously seeing things "in the large" and "in the small".

The “What” and The “How” Attitudes

In general, the *What* is reflecting the scientific approach while the *How* is reflecting the engineering approach.

I believe that a characteristic of computer science is the fantastic “intertwining” between the What and the How attitudes: they co-exist in most projects.

For example, in embarking in an analysis of the limits of discrete computation we try to understand the nature of computation (What) by measuring the complexity of computations (How).

The Computer

Turing “blessed” computer science with a universal device (universal Turing machine) which can

- be instructed to perform any conceivable discrete computation (Turing Thesis—syntactic claim)
- simulate, in principle, any physical process as described by our current laws of physics (Turing Thesis—physical claim)

Church-Turing's Thesis is almost universally accepted:

- *Philosophical argument*: Due to Turing's analysis it seems very difficult to imagine some other method which falls outside the scope of his description.
- *Mathematical evidence*: Every mathematical notion of computability which has been proposed was proven equivalent to Turing computability.
- *Sociological evidence*: No example of computing device which cannot be simulated by a Turing machine has been given, i.e. the thesis has not been disproved despite having proposed over 60 years ago.

Universality is challenged by the inherent chaotic nature of machines: the slightest change in their instructions or data can result in arbitrarily large differences in the results. In fact, the global behaviour of a universal Turing machine is not only beyond any possibility of computation (Turing), but, in fact, completely unpredictable (Chaitin). These machines have to be controlled with unprecedented precision, even when they perform the simplest computations!

There is an amazing disparity between the complexity of problems a universal computer can solve and the simplicity of this machine! Some problems require millions of code lines, yet our computer executes only a small number of different instructions.

Do We Need Algorithmic Abstraction?

Computer science has privileged relations with mathematics. (I see mathematics as an art, a science, a language, a game, and a tool.)

Two ideas look very appealing:

- Computer science = mathematics + time. (Doran, 1987)
- Computer science is the engineering of mathematics. (Hartmanis, 1995)

In computer science theory might give both **insight** and **utility**.
Here are some examples:

Turing's theoretical contribution was essential for computer science.
Today, similarly, "blue-sky" research concentrates on DNA and
Quantum computations.

The naive algorithm for the discrete Fourier transform, a key step in many digital signal-processing applications, requires $2n^2$ floating-point operations to process n waveform samples. The fast Fourier transform needs only $2n \log n$. The improvement is of the order of $1/11,000$ for processing 10 seconds of speech. (No CPU can today process speech in real time with the naive algorithm!)

Gödel's incompleteness phenomenon has a counterpart in logic programming, where completeness must be sacrificed to realize reactiveness. Concurrent logic programs abandon complete exploration of the search space in favor of incomplete committed-choice (don't care) nondeterminism. And fuzzy logic is incorporated in cameras and video-cameras.

The computer science community was taken in by local successes (Samuel's checker-playing program, early programs aiming to translate from one natural language to another one) to believe that hard problems are in fact easy, that the success or failure is a matter of using more powerful machines.

After a more careful examination it has been realized that one cannot make a significant progress without a "theoretical understanding" of the inner structure of these problems (e.g. the formal grammar of natural languages). At this point, theory entered the game.

Theory forms an enduring core of the field (I believe that automata will still form the basis of various software components at a time in the future when C++ or WIMP interfaces are forgotten).

Of course, as the theory fields grew there was a tendency to attack problems obliquely, loosing completely, at some stage, the original problems motivating the whole research. This is a real danger and the theory community should watch this phenomenon closely.

Theory should uphold its standards of rigour and move closer to practice.

Demonstration vs Examination?

Building large scale hardware and software in an incredible short time is a defining characteristic of computer science. As a consequence, products are fairly often *demonstrated* (**not in the sense given in mathematics!**) to argue possibility and feasibility.

This has generated a rivalry between

demo or die

and

publish or perish

Experimentation in computer science is the current programming of large pieces of software, using various methodologies, formalism, and tools, quite often without having any idea what these tools will offer until they are actually used. In a sense, computer science is **experimental**, not only because currently we know no better way to do it, but because—by Turing and Chaitin results—-there is no better way. Some phenomena happen to have no cause!

How dangerous is to rely only on demo?

Think only on applications in aviation or medical sciences: Is it worth taking the risk?

Will Computer Science Become Irrelevant or Disappear?

Computer science has been extraordinarily successful. Computer technology is in almost every home, car, and business. Computers are being used by many practitioners, most, if not almost, without degrees in the field. So, a couple of questions appear naturally:

Do we need computer science?

Has the field any future?

A computer scientist is needed when the computational problem is **hard**.

There are unlimited many hard computational problems.

How many of them are really of interest?

No Conclusions Are Offered

I find nothing wrong with the statement

> *Is computer science a science or engineering? Who cares. It's important, difficult and fun.*

But—in spite of all doubts generated by the above un-answered (un-answerable?) questions—I am **interested** in what computer science is.