

Is Computer Science Becoming a Cookbook Science?

Bengt Nordström

Computing Science, Chalmers and University of Göteborg

Ämneskonferens i datavetenskap och numerisk analys i
Stockholm
12-14 juni 2006

Cookbook Science

Cookbook:

- Study a number of cooks.
- Summarize what they do by writing rules.

Cookbook Science:

- Study how things work in practice (how nature is).
- Summarize by writing rules.

Carried over to Computing Science:

- Study how things work in practice:
 - study existing programmers how they program.
 - study existing programming languages (what properties they have)
 - study existing ways of structuring big programs (program libraries in C or Java).

Explanatory Science

The alternative to Cookbook Science is Explanatory Science:

- You not only summarize by rules,
- but also try to explain the causes.

A good cook knows not only that you put just some drops of oil in the egg yolk, but also why!

Poincaré:

Knowledge is the end and action is the means.

Computer Science as a science

First observation:

- Computing Science is not a science in the sense of natural science.
- We do not study Nature.
- We are very little interested in physical, chemical, biological, geological properties.
- The material reality which we are interested in has very simple properties.

The objects of our study are creations of human beings. In natural science we study Nature, it is there and we want to understand more and more about it. But in Computing Science we also create our objects of study.

All man made objects are not worthy of study.

Things taken for given (as part of Nature?):

- Programming languages (C, Java, ML, Haskell, ...)
- Specification languages (UML, VHDL, ...)
- Data description languages (XML, ...)
- Be friend with your bugs!

Programming language example: C, Java

C is a good assembly language, and Java is widely used. These languages have caused a lot of practical problems, some of them are scientific interesting, but many of them are just problems from bad design. Two fields of Computing where this causes problems:

- analysis of programming languages (semantics, static analysis)
- program verification

Programming language example: Haskell

Official propaganda:

You reason about Haskell programs exactly like you reason in mathematics, you can replace equals by equals etc.

The truth:

This is simply not the case. Examples:

- The **seq** construct.
- Definition by pattern matching. (Well typed programs can go wrong. Bad conversion rule)
- Confusion between coinductive and inductive data.

There is no formal semantics of Haskell.

Specification language example: UML

- First standardize something fuzzy.
- Then impose its use in industry.
- Then try to give semantics to it.
- General problem with specification language independent of programming language.

Data description language example: XML

Some problems:

- Unorthogonal design. Strange restrictions based on poor understanding or inefficient implementation.
- Ad hoc typing system
- To know if an XML document is well-typed it is necessary to prove some properties of an FSA which can be constructed from the description.
- Notation for an empty list of elements?

What are the reasons for this situation?

- Researchers' genuine practical interests.
- The funding system. The purpose of the research is to have quick solutions to today's problems.
- Confusion between development and research.

What are the problems with this situation?

- A lot of good researchers are wasting their time studying things not worthy of studying.
- A lot of not-so-good researchers are wasting our money studying things which are too complicated. There will be no contribution to our common knowledge.
- Innovation and development is slowed down.
- Good research has to be disguised. It is never good for ones self-esteem to pretend to do things. Some examples:
 - Denotational semantics is disguised as precise description of Java or C.
 - Static analysis disguised as security.
 - Category theory disguised as UML.
- We have to give up the strive for complete understanding. Badly designed artefacts like Java, UML, XML leads to scientific "theories" which are too complex to be useful.

What are the problems with this situation?

- We should not forget the esthetics of science. A good theory is beautiful. Looking at ugly things makes it difficult to create a beautiful theory.
- Good researchers want to work with beautiful theories.

What should be done instead?

- The minimum: When we study badly designed artefacts, we should point out the problems and suggest alternative design decisions. We should be more critical!
- Even better: Suggest new designs without trying to be compatible with old ones.

Computer Science education

- Cookbook science has a strong tradition in engineering subjects.
- A funding system where we get paid for each point a student makes is a strong impetus for cookbook teaching.
- The only way to educate engineers to a high standard is to make them understand. This requires good theory. They should use programming languages, description languages, specification languages which can be understood. (I am not saying that this exists yet).

References

References:

- George Gale: Scientific Explanation,
<http://cas.umkc.edu/philosophy/gale/CambridgeHistory.pdf>
- Peter Denning: Great Principles of Computing, CACM Nov 2003/Vol. 46, No. 11

Acknowledgements:

I want to thank John Allen for an email discussion on this topic.