



## On constructive research in computer science

19.9.2007  
Jyrki Nummenmaa

<http://www.cs.uta.fi/>



## Starting points

- Interesting or important problem in the field of software development
  - Methods, tools, languages, processes, ...
- Typically a constructive solution
- Implementation
- How to make science out of this?
- Science vs. standard software development?
- Notice that the observations/instructions in this talk apply generally to constructive computer science.

<http://www.cs.uta.fi/>



## “Scientification”

- Description, abstraction, and formalisation of the problem and the solution
- Analysis of the solution.
- Comparison of the solution with other solutions.
- Both analysis and comparison need to be performed in a scientifically acceptable way – more on this in the following slides.

<http://www.cs.uta.fi/>



## Description of the problem and the solution

- Extract the essential and leave the standard/non-interesting details.
- Form a suitable abstraction (pick general concepts to describe the essential).
  - Suppose we are running a number of concurrent programs. Model as processes? Threads? ...
- Describe the abstraction with a suitable formalisation.
  - Let  $P = \{P_1, \dots, P_n\}$  be a set of processes. ...
- You may shortly describe technical details of the implementation, user interface, etc.

<http://www.cs.uta.fi/>



## Extracting the essential

- In addition to your central contribution, you may need to solve several kinds of things, which are of minor importance.
  - E.g. fairly standard choice of data structures for your implementation.
- You have to understand what your main contribution (the “beaf”) is and present it as such.
  - Many readers will start by only reading the abstract and the introduction. If they can't find the “beaf” there, their motivation will drop and they may even stop reading.

<http://www.cs.uta.fi/>



## Extracting the essential

- You may discuss other things as well, but keep things into perspective.
  - Minor things deserve only minor space in your paper.
- Do not over-emphasize the trivial things and don't be too shy about your real achievements.
  - If you have invented something, say so.

<http://www.cs.uta.fi/>





## Forming an abstraction

University of Tampere, CS Department

- Find a similarity with some well-known abstract structure (graph theory, set theory, formal languages, automata, etc.)
  - The abstract structures come with lots of useful concepts, results, etc.
- Map the elements of your problem to elements of the abstract structure.
- Do not choose anything too general, though.
  - Etc. if you can always represent your data as a tree, there is no need and no point to represent it as a general graph.

<http://www.cs.uta.fi/>



## Formalization

University of Tampere, CS Department

- Take a formal notation for the abstraction.
  - If you can find a standard notation, which fits you, use it (rather than invent a completely own).
- Describe the problem and the solution with this notation.

<http://www.cs.uta.fi/>



## Evaluation of the solution

University of Tampere, CS Department

- Correctness
- Efficiency
- Expressive power
- Ease of expression
- Stylistic issues
- Meaning for relevant applications
  - You need to identify the relevant applications / application areas.

<http://www.cs.uta.fi/>



## Efficiency

University of Tampere, CS Department

- Asymptotic efficiency efficiency when inputs grow larger
  - Examples: worst case, average case, amortized efficiency (efficiency changes because we make good operations)
  - Typically calculated theoretically.
  - Simulations or test runs can be used to support theory.
  - Sometimes some theoretical hard cases do not appear in practice.
- Throughput
  - Ability to process inputs per time unit.
  - Typically used with complicated systems and evaluated using simulation.
- Response time
  - Time to produce outputs from inputs.
  - Complicated systems, simulation (like throughput)

<http://www.cs.uta.fi/>



## Expressive power

University of Tampere, CS Department

- Theoretical analysis: what can be done and what can not.
- Relationship to practical needs.
- Compare with *best alternative solutions*
  - It is not sufficient to just find something, which is even worse than what you have done.

<http://www.cs.uta.fi/>



## Ease of expression

University of Tampere, CS Department

- Theoretical analysis
  - Lengths of expressions
  - Number of necessary constructs
  - Complexity measures (such as measuring programming code complexity)
- Practical tests
  - Realistic environment.
  - Justify the choice of environment.

<http://www.cs.uta.fi/>





## Case studies

University of Tampere, CS Department

- Quite often only 1 or 2 case studies are performed.
  - It should be kept in mind that those are just examples, no statistical inferences can be made.
- In case studies one should pay particular attention to the cases.
  - " I implemented two systems using my framework. It was easy and they worked ok, when I tried them."
  - How do you know if anyone else would be able to implement anything using your framework? And if anyone else would get the solutions to work?

<http://www.cs.uta.fi/>



## Statistical inference

University of Tampere, CS Department

- Case studies represent examples.
- Statistical inference represents generality.
- Keep in mind the statistical basics
  - Rule of thumb: 30 cases minimum for an analysis. Sometimes this is genuinely too much in practice and a case study is justified.
  - Know the limitations of your data.
  - Plan your analysis before collecting the data (so that the analysis can be done from the data you collected).

<http://www.cs.uta.fi/>



## MSc thesis works

University of Tampere, CS Department

<http://www.cs.uta.fi/>



## General expectations

University of Tampere, CS Department

- Not just a work report of an engineering work.
- Has some description of background concepts or theory.
- Has a clear research problem.
- Has some sort of analysis of the solutions presented (regardless whether the solutions are yours or come from other sources).
- Use good language in an MSc style.

<http://www.cs.uta.fi/>



## Some typical issues

University of Tampere, CS Department

- Study a new technology .
  - You may test the technology.g. developing a new application.
  - You may compare the technology with others in practice or in principle (by analysis).
- A challenging design task
  - You need to explain where the intellectual challenge is.
    - There is a risk that this is just an engineering work
- Develop the methodologies, tools, etc
  - You may find a possibility for this, but if good ideas do not come along, don't get stuck

<http://www.cs.uta.fi/>



## Starting up

University of Tampere, CS Department

- Do not expect to start the work full-time
- Start up by reading and thinking about the issue.
  - At this time you just need to have a more general research idea or a couple of them. Talk about this with your supervisor.
  - Of course, you may have a more specific title in mind.
  - Also, the supervisor may propose a specific subject.
- The ideas will mature in your head little by little.
- After a couple of months, typically people come up with a more specific idea.

<http://www.cs.uta.fi/>





## Continued...

- After this, there typically is a more intensive work period.
  - Dig deeper in the subject.
  - You may analyse, design, write programs, test ideas, etc.
  - At the same time, you get a more precise idea of the actual content of your work
- Writing period
  - Start creating the structure of the document.
    - Could be just lists eg.
  - Write the first 10-15 pages so that your supervisor can check the language.

<http://www.cs.uta.fi/>

