Linus Åkesson

Essay for Part 1 of the PhD course in research methodology, ethics and innovation.

Research methods I currently use

In this section, I will describe my work in terms of research methods. I will start with a very general overview of the work, and then focus on three different parts of it.

Overview

The core of my work can be described as *design science* on several different levels. At the most concrete level, I develop software (a compiler toolchain, and a middleware runtime system) to solve problems of a particular kind, and then I evaluate qualitatively and quantitatively how well the software solves those problems.

At a higher level, I design the programming language that serves as input for the compiler I'm writing, and evaluate this programming language, trying to determine how well it can be used to solve problems in the intended domain.

Finally, at the highest level, I try to map out some guidelines for designing such programming languages in general. That is to say, I reflect on what worked well and what didn't work, and try to crystallise some general advice that could be helpful for others who wish to explore the same design space.

Software

As part of my research, I have designed and implemented software, and *measured its performance* (relating it to the performance of similar systems). According to the classification scheme used in [1], the design and implementation would be an example of a *formulative* research approach, and measuring the performance would be an example of an *evaluative* approach. My work includes a new algorithm for solving a particular optimisation problem, and the software could then be regarded as a *proof of concept* implementation of the algorithm. That the algorithm is effective is shown using *simulation*.

In order to explain and motivate why these artefacts are useful, I perform a *literature review* (*descriptive* according to [1]).

Language

A programming language is a designed artefact; therefore the creation of one should be regarded as design science. Again, I use literature review to show how this artefact differs from existing ones. I perform a small *case study* to identify some of the design requirements for the language: From the study I obtain an

example scenario, and the language is then designed to be able to deal with the scenario. This is an example of *use-case driven design*.

Design advice

At the most abstract level, I sketch out some advice for designing programming languages similar to mine. One could regard the software and my programming language as the concrete artefact for which these guidelines are just academic commentary. Or one could regard the advice as the central artefact (it would be classified as *formulative-guidelines/standards* in [1]), and the programming language and software as a kind of substantiation, providing a concrete evaluation of the artefact. In my opinion, both of these views are equally valid, and I deliberately refrain from committing myself to just one of the interpretations.

Research methods I might consider using in the future

The design rationale for my artefacts is mainly established by means of a case study; they grew out of a need in a particular research project. Perhaps the contribution would be regarded as stronger if a more extensive set of requirements were gathered systematically by means of a *survey*. The design of the programming language itself could perhaps benefit from studies of *visual notations* [2].

As for the evaluation aspect of my research, I might want to complement my performance measurements and simulations with *field studies* or *field experiments*, in orded to judge how well the artefacts actually work in practice. A programming language is arguably a user interface, and as such may be amenable to *heuristic evaluation* [3].

One possible way forward would be to formalise the semantics of my programming language. This would make the compiler and the language (as well as programs written in it) amenable to certain kinds of *formal analysis* and *mathematical proof*. For instance, it would be possible to demonstrate the correctness of the optimisation algorithm.

References

[1] R. L. Glass, V. Ramesh, and I. Vessey. An analysis of research in computing disciplines. *Commun. ACM*, 47(6):89–94, June 2004.

[2] D. Moody. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, Nov 2009.

[3] J. Nielsen and R. Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, pages 249–256, New York, NY, USA, 1990. ACM.