

Towards a mature engineering discipline

I usually start my lectures on Software Specification by the following comparison of Software Engineering with other “mature” engineering disciplines such as civil engineering or mechanical engineering:

Assume that a bridge has to be built over a river by a civil engineering firm, you are in charge of overseeing this project, and you observe the following scenario:

The engineers hold sessions with the stakeholders in this project, have a couple of internal discussions, in which they make rough sketches of the bridge on a white board, make some drawings on paper and after a few of these sessions they move to the construction site.

You rush to them and ask them where their design documents, and plots and analysis results are. To your surprise, you find out that there are no such things; the engineers say that they had a briefing with construction workers, in which a painting of the (to-be-built) bridge is shown and its different parts are discussed. Hence, engineers believe, the workers should be able to work out the details by themselves. When you ask them about calculations regarding the strength of the structure to be built and its resistance against different types of stress, you hear:

“We have thought of an ingenious test plan: we shall build the bridge and cautiously let a car drive over it; if it does not collapse, we let a truck go over the bridge, then a loaded truck and then a number of trucks. If all goes well, then we are confident that it will not collapse under any foreseeable circumstance. Of course, we cannot guarantee anything, should anything unforeseen emerge and the bridge collapse, we would

certainly be able to reconstruct it within a week time.”

I would imagine that, by any reasonable standard, this is considered unacceptable and you would definitely report this strange state of affairs to the authorities that can stop this project and bring it back to the hands of “real” engineers.

Let’s compare the situation with the practice of engineering software: what is described above is considered a very reasonable way of working in many, if not most, software development teams. The team of engineers starts with a rough sketch, puts it down on paper, discusses it with programmers and afterwards, the programmers start programming. There is not a quantitative analysis of the properties of the to-be-developed product and, at best, there is a test plan to test the product under foreseeable scenarios.

Computer systems can be as complex as the most advanced products in other engineering disciplines; a bridge or a sky-scraper is not more complex than the software you carry in your laptops (e.g., the cost of building the tallest skyscraper hitherto, known as Burj Dubai, was about 1.5 billion dollars, while the development of Windows Vista cost about 6 billion dollars; both figures are taken from Wikipedia).

What is the solution? One possible solution is obtained by observing the way of working in the other engineering disciplines and acquiring their best practices.

A common aspect of all mature engineering disciplines is the use of mathematical models for making their initial sketches more precise and the use of mathematical techniques to reason about them.

The first step towards this goal is to map the specification of software systems to an underlying mathematical theory. This mapping is called a formal semantics. After defining a formal semantics, one can proceed with devising analysis techniques to measure and judge different properties of the to-be-developed computer system. The process of measuring and judging the specifications for given properties is called formal verification. The title “formal semantics and verification” nicely summarizes my research interests in computer science.

Regarding formal semantics, I am interested in the methods for defining the actual behavior of programs running on an abstract machine; this is called “operational semantics” and the most popular method in this field is called Structural Operational Semantics, SOS for short, which defines the meaning of each composite piece of specification in terms of its components. The original course-notes introducing this field of formal semantics, authored by my second promoter prof. Gordon Plotkin (University of Edinburgh), attracted about 1000 citations before its actual publication in a journal in 2004. To my knowledge this is the most cited (unpublished) report in Computer Science ever!



As far as formal verification is concerned, I am most interested in efficient techniques for exhaustively searching all possible behaviors of a specification or program. This is done to firmly establish that the software system behaves well in all possible situations it can be confronted with.

This is called “model checking”. In 2007, the Turing Award, the Nobel prize of Computer Science, went to Clarke (CMU), Emerson (University of Texas at Austin) and Sifakis (Verimag) for their roles in “developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries”. This shows the significance of this technique for the whole field of Computer Science, in general, and for the mathematical specification and analysis techniques (called Formal Methods), in particular. As Daniel Jackson (MIT) once nicely put it:

“Model checking saved the reputation of formal methods.”

I personally believe that we are just witnessing the beginning of an exciting era in Computer Science and Engineering; we are gradually becoming capable of modeling and analyzing huge real-life computer systems, thereby turning our field into a mature engineering discipline.

Mohammad Mousavi was born in 1978 in Tehran, Iran. He did his bachelor and masters in Computer Engineering at Sharif University of Technology in Tehran and moved in 2001 to the Netherlands to start his Ph.D. studies. He got his Ph.D. degree in 2005 from TU/e. Since then, he has been an assistant professor (UD) at the department of Electrical Engineering at TU/e, post-doctoral researcher at Reykjavik University, Iceland and finally, an assistant professor (UD) at the department of Computer Science at TU/e, where he currently holds a permanent position.



onderwijs
dr. Mohammad Mousavi

