Practical Formal Methods

Jeroen J.A. Keiren
Department of Computer Science
Open University of the Netherlands
Department of Digital Security, ICIS
Radboud University
Nijmegen, The Netherlands
Delft University of Technology
Delft, The Netherlands
Jeroen.Keiren@ou.nl

1 Introduction

Today, we heavily depend on software. We do not only use the computers on our desktops and the mobile phones in our pockets. Financial infrastructures and automatic stock trading are controlled by computers, and computer systems are embedded in home appliances such as televisions, safety critical systems such as cars and airplanes, as well as systems controlling (access to) infrastructure such as bridges and tunnels.

During the development of software, inevitably, mistakes are made. In fact, on average, every 1000 lines of code contain up to 10-16 bugs [? ?]. Some of these bugs will only show up once the system is running. At that point, the consequences can range from being harmless – e.g. needing to restart your phone because it freezes –, to very severe – such as a car crashing [?], hundreds of millions of dollars being lost in in the stock markets [?]. Furthermore, a growing reliance on battery-powered devices and the effects of climate change have resulted in an increased interest in *green computing*. Bugs that lead to quick draining of batteries have gained a lot of publicity [?].

To effectively detect or avoid such bugs early (preferably before software is used) calls for a range of different tools and techniques. This ranges from exact and exhaustive verification methods such as (automated) theorem proving and model checking, to formal testing techniques such as model based testing, automated testing at the level of graphical user interfaces, as well as more traditional testing techniques such as unit- and integration testing. The formal methods and software engineering communities in the Netherlands have a long track record in developing such techniques.

When considering, especially, the more formal approaches such as model checking, theorem proving and model-based testing, there is a lot of anecdotal evidence that they are effective in finding and avoiding bugs, for instance [? ? ?]. However, the techniques typically require experts that are well-versed in both the application domain as well as in the formal methods applied. Furthermore, a clear business case for the application of formal methods is missing.

Besides the mainly theoretical research into the foundations of formal methods based verification and testing techniques, there are three research directions that deserve our undivided attention.

1. Develop formal methods that can be used by software engineers that are not formal methods experts.
2. Establish a business case for the industrial application of formal methods through empirical research.
3. Determine how formal methods can best be integrated in software engineering and computer science curricula.

I will detail each of these three points in the rest of this abstract.

2 Bringing formal methods to the masses

The application of formal methods such as model checking or model based testing to industrial cases is still very much an expert activity. Tools often rely on models of the software, instead of the software itself, and the models are specified in domain-specific languages whose syntax is far from the the programming languages software engineers are used to. Throughout the years many applications of such techniques have been reported as a success, e.g., [? ?], but large-scale industrial application has yet to gain traction.

In software model-checking, some successes have been obtained, e.g., at Microsoft using SLAM[1] [?] and TERMINATOR, continued as T2 [?], for the verification of C code, where verifying a limited set of properties on an actual (C-code) implementation seems to be one of the success criteria.

To bring formal methods to the masses, as a community we need to invest not just in new techniques, but also in the engineering of tooling and languages that can be used in the industrial software engineering process, and recognise that this step is not "just trivial engineering". We should therefore not stop at the development of prototype tools, but also look at industry needs and invest in making the tooling mature enough for use in production systems. At CERN, for example, we went from high-level academic tools, down to an integration of the verification in the IDE used by

[1] https://www.microsoft.com/en-us/research/project/slam/, accessed 9 August 2018

the developers, effectively providing developers with push-button access to model checking [**?**].

3 Empirical evidence for the merits of formal methods

Industrial application of formal methods are typically reported as succes stories. However, from an industrial perspective, it is important to know the effects on applying such techniques. How does the application of formal methods affect, for example, time-to-market, number of bugs found after deployment, etc.

Unfortunately, such information is currently lacking from the formal methods literature. Some vendors, such as Verum with its Dezyne toolkit – which is based on mCRL2 [**?**] –, do report benefits such as "50% reduction in development costs, 25% reduction in the cost of field defect and 20% decrease in time-to-market"[2], but the sources and reliability of such data are unclear. In order to build a successful business case for the application of formal methods in software engineering practice, a collaborative effort should be made to collect data about the application of formal methods. Based on the collected data, a business case for the application of formal methods should be developed. One possibility could be the collection of data in a large number of student projects, such as [**?**].

4 Teaching formal methods

A final advance in the acceptance of formal methods lies in the way we teach formal methods. It appears commonplace in software engineering and computer science curricula to present formal methods more as a research topic than as a software engineering topic. A typical question one gets from students is "so, where and how are these techniques actually applied in industry". In recent years, I have seen more critical attitudes of students towards formal methods courses. Should we do a better job at integrating formal methods into our curricula, to really show the students what the can *do with formal methods* instead of *how the formal methods work*?

Note that there is more to the teaching of formal methods. How can we effectively teach such methods using modern tools and techniques such as massive open online courses (MOOCs)? In particular, with (sometimes dramatically) increasing numbers of students, and in the context of distance learning, we need to find ways of providing feedback in formal methods education that does not rely on expert feedback. Is it possible to build interactive online tools that present feedback to students about their solutions? Are there more effective ways of teaching formal methods than we currently use in our courses? Can we, in this respect, learn from programming education research such as [**? ?**]?

[2] https://www.verum.com, accessed 9 August 2018.