Who are we, and what are we doing here?

Alan Wassyng

McMaster Centre for Software Certification, Department of Computing and Software,
McMaster University, Hamilton, Ontario, Canada,
wassyng@mcmaster.ca

Abstract. Many Formal Methods researchers and practitioners seem to treat Formal Methods more as a religion than as an approach to rigorous software engineering. This fervour has a few side-effects: i) There have been spectacular advances in a few areas in Formal Methods; ii) There are a significant number of highly effective Formal Methods advocates - and practitioners; iii) The Formal Methods community at large seems to be condescendingly dismissive of any protestation of disbelief; and iv) Different methods and approaches seem to be judged on a belief basis rather than through evidence based analysis. The essential fact remains though, that after decades of research, Formal Methods are not used much in industrial software development. It is time that we, the Formal Methods community, question the basis of our existence. I argue that we exist to further the use of mathematics and rigorous analysis in the development of software applications, in the same way that electrical engineers, mechanical engineers, civil engineers, chemical engineers further the safe and effective development of a multitude of devices, buildings, manufacturing processes etc. This is clearly not a new thought. It does, however, suggest that we need to examine the link between Formal Methods and Software Engineering more carefully than is currently the case. A definition of engineering from the Academic Press Dictionary of Science and Technology is *"the application of scientific knowledge about matter and energy for practical human uses such as construction, machinery, products, or systems"*. Engineers use science as the basis for their work. This is not a one-way street. Feedback from engineering as to what are the important scientific problems to be solved is an important driver in scientific endeavours. Engineering work, in turn, forms a basis for the work done by technicians in our everyday lives. Again, feedback is an essential driver for the engineering community. In the modern digital world, Software Engineers should assume the role of the engineer. If we are truly serious about Software Engineering as an engineering profession, we need to consider the roles of Computer Scientists and Software Developers in this context. To be consistent with other domains, Software Engineers should use scientific knowledge as the basis of their work. This knowledge includes the growing domain of knowledge generated by Computer Science, and in particular, the specialized forms of mathematics that are applicable in the digital domain. In addition to Computer Scientists and Software Engineers, we also have Software Developers the technicians of our domain. This is a nice and neat correlation with other engineering fields unfortunately it is not, at this time, an accurate description of the

situation. In most countries, the difference between Computer Science and Software Engineering is decidedly blurry. Even when the difference should be obvious (for example, Canada insists that to call yourself an *"engineer"* you must be recognized as such by a professional engineering accreditation body), it is commonplace to find Computer Scientists playing the role of both engineer and technician. What does this mean for Formal Methods? Are Formal Methods people Computer Scientists, Software Engineers, Software Developers all of the above any of the above? If you look back at what I said about our *raison d'etre*, and if you agreed with what I said, perhaps you agreed too quickly!

Lately, my interests have been focused on the certification of software intensive systems: methods for building software intensive systems so that they can be certified; and methods for certifying such systems. This has made me rethink why, in spite of some amazing advances, Formal Methods are not used more often in everyday practice. I strongly believe that it is both possible and necessary to define *engineering methods* for the development of high integrity software applications, that these engineering methods must be based on mathematics, science, and well-founded heuristics, that "approved" methods should be significantly more prescriptive/objective than current software development techniques, and that these methods have to be supported by high quality tool chains. I also believe that the development of these methods is the task, primarily, of Software Engineers. What is the implication of this for the Formal Methods community? I think the answer is simple but not yet widely palatable. I think there is not enough focus on Software Engineering as opposed to Computer Science. It seems to me that the feedback from the engineering domain to the science domain is haphazard at best - non-existent a lot of the time! Over the past twenty years we have seen papers on myths of Formal Methods, Challenges of Formal Methods, the Ten Commandments of Formal Methods, experience of Formal Methods in industry, rethinking Formal Methods and the list goes on. So, why another talk on what seems to be a talked-out subject? Arrogance, of course! And, I hope, some new observations that may help us define our future path. I did not come to these conclusions all on my own. I have been extremely fortunate in my career, both in industry and in academia, to work with incredibly smart and dedicated colleagues, and I am indebted to them for teaching me so much about a very complex subject.

From a Software Engineering perspective, there are a number of fundamental principles that need to guide our design of Formal Methods: integration of the Formal Methods aspects with the rest of the software development life cycle; integration of the life cycle phases; comprehensive tool chains integrated into the methods; completeness criteria; ability to handle real-world aspects; scalability; the methods and tools must be understandable and usable by average, educated, practitioners (technicians); prescriptive/objective guidance; experimental validation of resulting methods and tools; and development with certification as a goal. In this talk I will use a running example to illustrate and discuss these principles. I hope this talk will be viewed as an exhortation to great technical

successes, and even greater success in producing powerful methods and tools that software developers will want to use.

Acknowledgements

The opinions expressed in this talk are mine, but I have been incredibly fortunate to work with many extremely knowledgeable and capable software professionals over the past twenty years, both in academia and in industry. There are too many to mention all of them, but I do need to acknowledge my gratitude to: Mark Lawford, Tom Maibaum, Paul Joannou, and Dave Parnas for the hours of discussion (not to mention arguments) and collaboration, especially over the past ten years. Also, my colleagues Rick Hohendorf, Glenn Archinoff, Dominic Chan, David Lau, Greg Moum, Mike Viola, Jeff McDougall, David Tremaine, Peter Froebel and Alanna Wong, showed me how to approach software engineering as a true engineering discipline. Thanks to all of you!