Formal Methods in Software Engineering Education
Discussion Summary

Derek Andrews

1 Introduction

Formal methods are seen as the Cinderella subject of software engineering, always hovering in the background, hoping to be invited to the ball. But exactly what are formal methods? Because they were fashionable in the eighties, the term was stolen by “informal” methods to provide themselves with a cloak of respectability—one of the claims of formal methods was that, by using them, it was possible to reduce the number of errors in a program and improve productivity; claims that any method would like to be associated with. This report defines formal methods as mathematically based methods—actually using mathematics to specify, design and implement a computer system.

2 The current situation

In many syllabuses where formal methods are taught, they are seen as a curiosity. The subject is taught in isolation and its importance is not understood by the students; nor is its importance understood by software engineering teachers who are not familiar with either the advantages and disadvantages of formal methods. Further, employers of computer scientists and software engineers do not see the need for formal methods and thus there are little or no economic pressures to teach the subject. We are now in the third era of formal methods: the subject was started in the seventies, seen as a fashionable magic bullet in the eighties and has been rediscovered in the nineties: the main application of formal methods today is perceived to be in the construction of safety-critical software.

Using formal methods to specify a system is seen by many as being too laborious. The mathematical specification of a computer system is seen in isolation and not as the starting point of a development. A formal definition is recognized as an abstract model of the real world, but it is not seen as an abstract model of the final system. A formal specification is not perceived as a bridge between the real world and a computer system that models (a part of) that real world. Formal methods are frequently taught using small examples—examples that the good student could easily code directly. The main advantage of formal methods as a way of understanding large systems is frequently missed.

A further problem is that there are pressures by the exponents of formal methods that they should be adopted in an all or nothing strategy. It is not realized by many fans of the technique that a totally formal approach is too complicated and inappropriate for some systems. It should be seen as a tool for the difficult parts of a system, not as a way of specifying and developing the whole system. Formal methods should only be used where they can be seen to provide more understanding than the usual informal approach when designing a system. An additional problem is that it is not unusual for the subject to be taught by enthusiastic amateurs who have no experience of formal methods on large software projects, and thus their advantages are not fully understood. Finally, perhaps the resistance to the adoption of formal methods is due to the fact that a current influence on teaching software engineering is teaching tools that emphasize productivity—software engineering courses should also teach methods that are for improving the quality of code rather than just the quantity.

3 A way forward

There is a need to demonstrate the worth of formal methods. This can best be done with project work and/or workshops that are based on realistic (i.e., large) problems. The main advantages of formal methods can best be appreciated with large examples. One of the original uses of Meta-IV (a precursor to VDM-SL) was the specification of large systems. The mathematics provided leverage, which can really only be appreciated with large examples—it is much easier to write a formal specification to discover potential problems than to design and write the code. Writing down
the state for a large system provides a basis for the architecture of the final executable code, and defining the operations provides the system architect with a view of the functionality of the system. This is only really meaningful for a large system.

There are ways of incorporating the ideas of formal methods into software engineering courses. Pre- and post-conditions can be used for specifying functions and procedures. It is not necessary to use mathematics, the conditions can be expressed in English, the purpose being to encourage students to specify what their programs do before they start to write the code. The (possibly misnamed) abstract syntax of Z or VDM-SL can be used to describe the information content of data—the student can be taught to express what information is to be stored or passed across interfaces, either between different components of a software system or between the user and the system (the HCI). They can then worry about exactly how that information is to be represented. These ideas can be used to teach the concept of abstraction and implementation independent design.

Formal methods can be used to specify objects as part of an object-oriented design method. The information content of an object can be specified using abstract syntax and the methods of the object specified using pre- and post-conditions. If structured analysis and structured design are being taught, processes can be specified using pre- and post-conditions and both data-flows and the contents of file stores described using abstract syntax. Formal methods can be used to teach the understanding of abstraction at a very high level. Reading papers and text books on software engineering shows that many of the higher abstractions are missed by current exponents of software engineering; the abstractions they use are still linked to implementation rather than to specification.

A major advantage of a good abstract specification is that it can be used as the basis of a development—either formal or informal. The executable code should be derived from the specification using transformations that can be informal, rigorous or even formal. That there exists a full theory of specification and development should be of interest to all software engineers. Using formal methods it can be shown how a good specification can lead to the production of elegant, high-quality and efficient code. The use of formal methods also provides a basis for reasoning about a system—it is possible to show that a system that has been specified formally has certain properties—crucial for safety critical systems (hence the resurgence of formal methods in the nineties).

4 An aside on assessment

Without attempting a reasonable sized project, (for example specify a spreadsheet, a care hire system, a hotel management system, ...) students will not appreciate the leverage or understanding of a problem domain that writing a formal specification provides. Thus a substantial project should be part of any formal methods course. The normal examination process should also be reconsidered: a two or three hour exam does not really provide a good way of testing the understanding of formal methods. An intensive exam in formal methods is not popular with students since it is felt that there is a lot of notation to learn. A trivial solution to this is to provide a formulae book in the exam so the student realizes that learning the notation is not as critical as understanding the ideas. A further approach is to give the student a substantial problem to solve in an all day, open-book, exam. Students can even be given help during the exam (and the help given noted on the exam paper). In this way the emphasis can be moved away from learning lots of notation towards learning a different way of thinking about engineering systems.

5 Conclusion

Formal methods should be seen as a software engineering tool that has applications in various areas. As for all tools, it should be used only where it is appropriate—perhaps the main reason for the lack of adoption of formal methods is that it is often sold as the silver bullet that will solve all problems, rather than just a tool which is very powerful if used appropriately. Perhaps Cinderella, rather than demanding she should go to the ball as the star turn, should sneak in the back way and show her worth as a participant.

Acknowledgements

The convenor of the discussion group would like to thank the other members for their contribution: Juris Reinfelds, Ruth Davis, Doug Grant, John Hughes, Peter Eden, Chris Ho Stuart and Liang Han.