The Role of Formal Methods in Software Engineering Education and Industry

Colin J Burgess
University of Bristol

Department of Computer Science
University of Bristol
Bristol, BS8 1TR
England
Tel. +44 117 9287960
Fax +44 117 9251154
Internet Colin.Burgess@bristol.ac.uk

Abstract. Today, nearly every Software Engineering or Computer Science Degree Course given in either a British University, or in any University around the world, includes the teaching of Formal Methods. In contrast, there are comparatively few industrial or commercial projects in which Formal Methods are used. This paper outlines some of the benefits of including formal methods as part of the degree curriculum, even if they are not used in subsequent careers. It then describes some of the ways in which more formal ideas of software development can be introduced early on in a degree course, and then built on in the subsequent years. Finally, it speculates on how the situation in industry may change in the years to come, and how formal methods themselves may need to evolve before there is more widespread acceptance of them as a viable development technique, both from the point of view of software quality and also the financial implications.

1. Introduction

In every Software Enginering or Computer Science Degree Course given today, whether in a British University or in any other University around the world, at least one formal method for software development is likely to be taught, either as a separate course, or integrated into part of another course. This is an important part of the students' education as it helps to train them to treat the specification of software as a very important stage of software development, and also to appreciate the advantages and problems associated with this approach for future projects. For a wide variety of reasons, only a very small percentage of commercial software projects use formal methods, although their use seems to be increasing very slowly. This paper examines some of the issues involved in both the education and industrial use of formal methods and speculates on how the situation will evolve in future years.

2. What is a Formal Method?

Although the term formal method is widely used, there does not seem to be any clear definition as to exactly what constitutes a formal method. On many occasions, the term is used simply to imply the use of a formal specification language, but not to include any prescription of how, or the extent of its usage, e.g. if the whole or just part of the program to be specified; is there any requirement for successive refinement, or reasoning or proof. Even the term formal specification language is not precise, as it is not clear whether languages which are aimed more specifically at the design of the implementation, rather than the specification of the problem, should be included, e.g. SDL (CCITT, 1988). In other words, how much abstraction should a language allow, in order for it to be called a formal specification language.
This paper will try and use the term formal methods to describe any approach which utilises a formal specification language and specifies the role of that formal specification during the software development process. In other words, the use of a formal method will not necessarily imply any formal refinement process, formal reasoning or proof. The term formal specification language will then be used to refer to any language in which it is possible to specify fully the functionality of the whole or a part of a piece of software in a way amenable to formal reasoning. The emphasis will be on the fact that it is based on a firm mathematical basis, rather than whether it is sufficiently abstract. CASE tools will not be considered directly as their relevance depends on the extent to which they are based on traditional structured design methods or on formal methods as outlined above.
The theoretical basis for the formal specification languages vary considerably. The most commonly taught formal methods, and the main methods that are used in industry, outside the communications sector, are VDM(Jones, 1990) and Z(Spivey, 1992). VDM was the first, or one of the first formal methods to be used for reasonably large scale projects, but Z seems to have gained in popularity recently. These two methods are closely related, even if their syntax is different, and are both based primarily on mathematical set theory.
There are a number of other approaches to formal specification, but these are used less frequently and are much less likely to be taught in any depth as part of a degree course.
Some of these other approaches are:

- Process Algebras

  These are methods which explicitly allow allow concurrency, and the modelling of processes which need to communicate with one another. A major application area for this approach is the handling of communication protocols. The most well known of these is CSP, or Communicating Sequential Processes (Hoare, 1985), which also has a similar mathematical basis to Z. Other methods of this type are LOTOS, which uses CSP to specify the concurrency aspects, but also allows a number of additional definitions ISO, 1989), and CCS.

- Algebraic Techniques

  These are methods that are based on algebra rather than set theory. The easiest example of this type is the use of Algebraic Specification for Abstract Data Types, where the behaviour of the operations on an abstract data type is governed by a set of rules expressed as equations. This type of specification tends to be a lot closer to the implementation but also fits very well with the ideas behind object-orientated design and object-orientated programming. The best known formal language for this type of approach is OBJ (Goguen, 1988).

- Functional Programming

  This is the use of a functional programming language for writing the specification, and also possibly a prototype implementation of the software. This is still more at the research stage as a viable specification technique and is slightly different from the others, in that there seems to be a strong emphasis in this approach that the specification should be executable, which is in many ways desirable, but it conflicts with the desire for abstraction. At present, the only main language for this approach seems to be Haskell(Hudak, 1993), or a partial implementation of Haskell, called Gopher(Jones, 1994).

There are other formal techniques, but these are rarely taught in any course and are very rarely used in industry for more than one specific type of application.

3. Why teach Formal Methods?

When designing any complete degree course there are always a large number of constraints, and normally it is not possible to satisfy all of them. Two of the constraints on software engineering type courses are the need to equip the students with the skills they will need in their later careers and to provide them with a broad enough background to be able to evaluate new approaches, and in due course, where necessary, to bring about changes in the way problems are solved.
There are a number of advantages in including formal methods as part of the degree course. These is a great tendency for students, and to some extent practitioners as well, to 'feel' that they are not really programming until actually producing code and testing it, rather than adopting a broader 'software engineering' perspective of the whole software development process. Thus, one of the important aspects of formal methods is that, even for quite simple problems, they force the students to think very carefully about the specification, and not to get involved in the coding too quickly. Even for students who have done a lot of programming before even starting on their degree course, the ideas behind formal methods are likely to be completely new, and can provide some motivation for them during early programming courses.

In addition, formal methods help to reinforce the importance of a proper engineering approach to software development, as they relate both to the requirements specification and to the subsequent coding stage. They also provide a good application of a number of the concepts of discrete mathematics, which otherwise runs the danger of becoming taught early on in a degree course, as being of good educational value, but then only used in perhaps a small number of specialist options.
Another very important reason for teaching formal methods is that they are gradually being used in more industrial projects, and thus students should be familiar with at least the ideas associated with the approach, even if they have not learnt the specific formal specification language that their particular industry may require.

4. When do you teach Formal Methods?

4.1. The background of the students

There are a number of different influences which affect how and when, formal methods are taught during a degree course.

- Whilst many students entering a Software Engineering or Computer Science Degree Course will have had considerable prior computing experience, this is not normally a prerequisite for the course. Even when students have had formal training in computing, this is rarely taught as part of a proper software engineering discipline, mainly because of the constraints imposed by the syllabus, time and expertise available in the schools. Thus the majority of degree courses have to start by assuming no prior knowledge of programming, even if they set a fast pace. This is not something that is peculiar to computing, but is a common way for many subjects of coping with a wide variety of 'A' Level and equivalent syllabuses.

- Most of the formal methods, with the possible exception of algebraic specification, require some understanding of discrete mathematics, and in particular, either first-order predicate calculus or set theory or both. Some 'A' Level syllabuses may have included some of this background but it is not possible to assume much, if any, common background amongst the students. Thus at least some aspects of discrete mathematics must be taught either as a separate course or introduced as part of the formal methods course.

4.2. The structure of the course

There are a variety of choices as to when to teach formal methods. The following is designed to give a flavour of the different approaches that are possible rather than to be an exhaustive list. Of all the specification languages, Z is probably the one most widely taught, particularly if only one formal method is included in the course.

- One approach, which allows the earliest introduction of VDM or Z, is to teach students to read existing formal specifications of programs as part of their first programming course, which is then based on one of the traditional imperative type of programming languages. This requires some understanding of discrete mathematics, which can be easily taught, but it is a lot easier on the students than having to write the specifications themselves. This would then be followed by a second course, which would involve developing the formal specification from the requirements specification, as well as then producing the code.

- A second approach is to teach a functional programming language, either as a first programming language or as a second programming language introduced in parallel with an imperative language. The most likely languages are Miranda, Gopher, Haskell or ML. This is not designed specifically with a view to using the language as a formal specification language, as this approach is not really proven yet, but rather to expose the students to a more declarative style of programming, which is closer to the style of most specification languages. Then, as part of a later course, a specification language such as VDM or Z could be taught.

- An alternative solution is to introduce algebraic specification of abstract data types as an integral part of the teaching of elementary data structures. This can fit easily into a first programming course which attempts to combine both the teaching of basic software engineering with the introduction to an imperative programming language. This then exposes the students to the idea of a more formal approach to software design, but makes very little demand on the mathematical ability of the students. A formal method, such as Z, can then be taught in a later course, when the students have had more instruction on discrete mathematics. One of the advantages of this approach is that the students get exposed to two entirely different techniques.

- If the course is designed to teach systems analysis and traditional structured design methodologies in sufficient depth to be able to use them on realistic size projects, then a choice has to be made as to whether to teach these before, after, or in parallel with the formal method approach. This is very likely to depend on the students ability and the overall emphasis of the degree course.

There are many possible combinations and variations on the approaches outlined. A balance has to be maintained between introducing too much new material in parallel, which can confuse the students, and postponing the introduction of important topics to a later stage. This is always a difficult compromise when designing a complete syllabus for a degree course.

5. Will the students use their Formal Methods training?

The use of formal methods in real industrial projects is increasing, but is still a small percentage of the total number of project undertaken. There are several circumstances when they are very likely to be used today for new projects.

- When the procurer of the software makes it a condition of the contract.
- Where the use of formal methods is perceived as one of the means of fulfilling a specified standard.
- When the software is for a safety-critical application or any other application where the cost of failures is high.
- When the organisation has already an established group with the necessary expertise to apply formal methods, so that they can be used without a steep initial learning curve.

A survey of the reasons for the low usage of formal methods in industry was conducted by the NPL(Austin at al., 1993). Although the total percentage of projects using formal methods is still very small, there are now a large number of projects that have been completed using this approach, and some reports published of the advantages and disadvantages of their use. One of the earliest, and best well-known is the IBM CICS project, based on VDM (Collins, Nicholls and Sorenson, 1987; Nix and Collins, 1988; Houston and King, 1991). A book describing the results from a number of other projects is about to be published(Hinchley and Bowen). Since the 1980's, more projects tend to use Z rather than VDM, and where distributed systems and concurrency are required, as for example in the communications industry, then they are more likely to use CSP, LOTOS or SDL. In a number of projects in the UK, formal methods are being used on a trial basis, or in parallel with a conventional approach, or to replace an exiting system, and these projects are often a result of collaboration between the Universities, or companies closely associated with the Universities, and industry.

6. The future for Formal Methods

It is only possible to conjecture about the role of formal methods in the future. Their use in certain applications looks certain but it is not clear whether this will spread to a large spectrum of applications or not. There are a number of possible ways of introducing formal methods into industrial projects without having to impose too large a change on the staff involved, some of which are currently being tried. Some of these are:

- Teach some of the existing staff to be able to read formal specifications which have been written by others, but also to include in the specifications

English description interleaving the different parts of the specification, where the English is designed to help the comprehension of the specification to those less familiar with the particular specification language, rather than be an informal specification in its own right.

- The development of more tools, including CASE tools that are based on formal methods, to assist with all aspects of their use. At the most basic level this involves the writing, editing and printing of the specification, but also the development of tools to assist their construction from requirements specifications, either written or diagrammatic or both; the development of more sophisticated proof tools to allow the user to perform reasoning; successive refinement; proof and satisfaction of required properties of a specification.

- To gain more experience in the industrial context, by using formal methods for perhaps a more crucial part of a project, possibly in parallel with an exising development methodology, so that the advantages and disadvantages can be seen in that application. Unfortunately, it seems very difficult to set up metrics that can assess the success of the formal methods, particularly with respect to both quality and financial implications, that are not influenced by a large number of other factors, e.g. the enthusiasm of the staff involved. However, from the limited evidence available so far, the use of formal methods does seem to result in a higher quality product being delivered to the customer, as measured by the number of errors per thousand lines of code being detected after delivery, and also, that many major errors are detected at an earlier stage of the software development process compared to the use of more conventional methods.

It is quite clear that formal methods will continue to be used and will need to be taught in software engineering degree courses. Fortunately, the number of good textbooks on the different methods is increasing, and also the production of standards for some of the languages. Rather like the situation with programming languages, there is no evidence yet of one particular specification language being used for all projects, but possibly some indication that the Z is being increasingly used for projects not requiring a large amount of concurrency, and CSP, or languages related to CSP for those projects requiring concurrency. Also it is very unclear, at present, what is likely to be the scope of the the usage of the formal specification within the project. For example, will the vast majority of projects use it purely as a means of formally documenting the specification, or will its use for reasoning, successive refinement and proof of particular properties become more common place amongst users of formal methods. There is also the possibility of the formal specification being used as a guide for the construction of test cases for the final implementation(Scullard, 1988).

7. References

Austin, S. and Parkin, G.I. 1993. Formal Methods: A Survey. *NPL Report*, Teddington, U.K.

CCITT Blue Book. 1988. Functional Specification and Description Language (SDL). Volume X-Fascicle X.1. Recommendation Z.100, Melbourne 14-25 Nov.

Collins, B.P., Nicholls, J.E. and Sorensen, I.H. 1987. Introducing Formal Methods: the CICS Experience with Z. *IBM Technical Report* TR12.260, IBM (UK) Labs. Ltd., Hursley Park, UK.

Goguen, J.A., and Winkler, T. 1988. Introducing OBJ3, SRI International.

Hinchley, M.G. and Bowen, J.P. (eds.) (in press). Applications of Formal Methods. Prentice Hall

Hoare, C.A.R. 1985. Communicating Sequential Processes, Prentice-Hall.

Houston, I. and King, S. 1991. CICS Project Report: Experiences and Results from the use of Z. In Proceedings of VDM'91, Vol. 551. Lecture Notes in Computer Science. Springer-Verlag.

Hudak, P. et al. 1993. Report on the programming language Haskell (Ver. 1.2). University of Glasgow, U.K.

ISO 8807. 1989. Information processing systems - Open systems interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour, International Organisation for Standardisation.

Jones, C.B. 1990. Systematic Software Development using VDM. 2nd. Edition, Prentice-Hall.

Jones, M.P. 1994. The implementation of the Gofer functional programming system. Tech. Rep. Yaleu/dcs/rr-1030, Yale University. U.S.A.

Nix, C.J. and Collins, B.P. 1988. The use of Software Engineering, including the Z notation, in the development of CICS, *Quality Assurance*, **13(3)**:103-110.

Scullard, G.T. 1988. Test case selection using VDM. In VDM'88 Conference Proceedings, LNCS 328, Springer Verlag.

Spivey, J.M. 1992. The Z Notation: A Reference Manual, 2nd. Edition, Prentice-Hall.