The Role of Modelling in Teaching Formal Methods for Software Engineering

A. J. Cowling

Department of Computer Science
University of Sheffield
Sheffield, England

A.Cowling@dcs.shef.ac.uk

Abstract. This paper argues that the teaching of formal methods within software engineering must aim to equip students to apply the kinds of methods that may be in use throughout their future careers, rather than just those that exist at present. In particular, it argues that this use will increasingly need to be tied closely to the activity of modelling systems, and to the various kinds of non-formal models that underpin the approach of model-driven development. To support this argument, it identifies the points in the development process where formal methods should make particular contributions, and considers how this support can be covered within the constraints of the curriculum for a typical software engineering degree programme, showing how identifying formal methods paradigms can contribute to this.

Keywords: System Models · Model-Driven Engineering · Formal Methods Paradigms · Curriculum Requirements

1 Introduction

Ever since the NATO conference in 1968 [1] where the term Software Engineering (SE from now on) was coined, a key question (particularly for educators) has been how to distinguish SE from the other branches of computing that are now identified in the Computing Curricula Overview Volume [2]. The most important part of the answer is, of course, that SE is concerned not just with studying software systems, but with producing them as artefacts that will meet customer requirements. On its own, though, this simply describes a craft activity (whatever the scale of the systems that are being produced), and is not sufficient to justify the use of the term engineering. Rather, this term also implies that the design and production is being carried out in a fashion which is disciplined, particularly in the sense of making appropriate use of underlying scientific principles. Indeed, over 20 years ago this author published a paper on SE education where the title asked the question "*Where's the Appliance of Science?*" [3], and that paper then identified formal methods (FM from now on) as an essential component in any degree programme that actually aimed to teach SE, as opposed to some less systematic approach to developing software.

Sadly, the situation identified in that earlier paper has not changed much, in that SE practice still rarely makes proper use of FM [4], and many degree programmes which claim to teach SE do not make an adequate effort to include FM [5], even though they should be leading the way in so doing [6]. To some extent, therefore, this paper can be seen as a retrospective view of that earlier one, in that some of the principles which it identified are still valid, and need repeating. On the other hand, some aspects of the situation have changed significantly, as SE practice has developed, and so the details of how those principles should be applied certainly do need to be updated.

In particular, that earlier paper made little reference to the importance of modelling systems, whereas in the next 10 years it became very obvious that this is actually a key unifying concept within the SE curriculum [7]. This paper therefore examines how this concept affects the way in which FM might in future be used within SE practice, and how FM therefore need to be taught within degree programmes in SE, in order to prepare students for this future use. To do this, the next section considers how the concept of modelling is currently used in SE practice, and the likely future developments of this, and section 3 then discusses how these may affect the use of FM. In the light of this, section 4 discusses the requirements for teaching FM, and how these differ from typical current approaches. Section 5 then considers which aspects of FM therefore need to be taught, and by implication which aspects are less significant. Finally, section 6 summarises the conclusions of these arguments.

2 Modelling in SE Practice

The main argument in [7] was that the proposals for what became the SE2004 model curriculum for SE [8] were tying the concepts of modelling and analysis too closely to requirements analysis, and so were not giving adequate recognition to the importance of modelling the structures of the products being created at other stages within the SE processes, or indeed to the modelling of these processes themselves. The paper then went on to argue that the combinations of these aspects of product and process meant that the curriculum itself should be viewed as a matrix structure, with dimensions corresponding to the key activities within the process, and to the different kinds of models (structural, functional, qualitative, cost, etc) that are used.

While the 2013 revision of SE2004 had not been published formally at the time of writing this paper, one of the features of the draft for public review [9] is that the general activities of modelling systems and analysing these models are now treated as separate from the specific activity of analysing and documenting requirements for software. This recognition of modelling as a key general activity reflects developments in SE practice, of which two in particular are relevant to this paper.

One of these developments is the SEMAT project [10], which is concerned with how models and the associated theories for different aspects of SE (termed *alphas* within this project) can be integrated to provide general models and theories for SE. Since one of these alphas for any SE project is the software system being created in that project, defining ways of modelling the structure of this system is therefore a key element of this project. (As an aside, it should be noted that the linking of models and theories raises interesting questions about the actual relationships between them, but sadly these questions have to be treated as outside the scope of this paper.)

The other development within SE practice is more general, and is the creation of approaches and tools for model-driven engineering (MDE from now on), notably OMG's model-driven architecture approach (based round standards such as UML [11] and associated tools), and the Eclipse modelling framework [12] and its related tools. The common aim of these MDE approaches is to move from a situation where models of systems are built, but then used little (or perhaps even thrown away) once the concepts in them have been expressed in program code, to one where the models are instead made sufficiently detailed that code can be generated automatically from them. This is intended to raise the level of abstraction of the whole of the SE process, with the expectation of significant gains in productivity.

This expectation has been discussed in [13], where it is argued that such gains could turn out to be as significant as those which arose in the early 1960s as a result of the switch from assembly languages to what were then called problem-oriented languages. Indeed, it is relevant to repeat here the prediction that was made in [13] for the likely consequence of these changes, that "*even current students of SE will find, before the ends of their careers, that the activity of writing by hand what we now regard as conventional program code will have become as rare and exotic a part of ordinary SE as writing significant portions of code in assembly language is today*".

3 Formal Methods in SE Practice

Since FM are based on the use of formal models of software systems, this likely change in the role of modelling in SE practice can also be expected to have some impact on the use of FM within SE, and so this impact needs to be analysed. Here a key issue is the relationship between the models that underpin FM and those that are widely used in SE, which was summarised by Robinson [4, 5] as "*... the use of formal methods in system development appears to have had very little penetration into practice. One reason we submit is because Formal Methods has developed as though it is a separate discipline sitting in glorious isolation from the rest of Computer Science, Software Engineering and Computer Engineering.*". In particular, this means that the models on which FM are based have often not been designed to fit well with models that are commonly used in SE, even where the latter are intended to support the levels of detail that would then be needed if they were to be used in MDE.

These problems can be illustrated by considering one common type of software system, namely a web-based application that is constructed using some object-oriented framework that enforces the model-view-controller (MVC) paradigm, on top of a relational database. Such a system will therefore incorporate at least five layers of abstraction, corresponding to the underlying database, to the models, views, and controllers, and to the use cases which the system implements. Each of these layers of abstraction will therefore need to be modelled, where the most common form of model for the database would be an entity-relationship diagram. For the model layer of the system the main feature that would need to be captured in a model would be the mapping from relations to classes, and typically this could be expressed as a UML class diagram. For the views in the system, which consist essentially of its web pages, the most common form of design model would be a set of wire-frames. The controller layer of the system is then responsible for co-ordinating the interactions of the model and view components, and these might well be represented by some form of co-ordination model, such as UML sequence diagrams. Finally, for the use case layer the normal practice would be to supplement a use case model by some structured representations of the required behaviours for each use case.

Out of these five layers of models, the only ones that are naturally and directly supported by formal methods are the database layer (where the use of relations to model tables is well established) and the model layer (where object-oriented specification languages incorporate the necessary concepts). Beyond this, wire-frame models do not have any obvious formal equivalents, and while the various process algebras can model the interactions of different system components, they do not have a particularly natural relationship with models expressed as sequence diagrams, nor do they link in any natural fashion with specification languages such as Object Z or VDM++. Finally, while it has been demonstrated by the author and colleagues [14, 15] that a variant on the X-Machine model known as the eXtreme X-Machine can be used to specify the behaviour of use cases, this method can not yet be regarded as well-established, not least because there is still much more work to be done to link it effectively to other formal specification methods.

This example therefore demonstrates a problem with the use of FM within SE that this author has found to be fundamental, namely that the existing methods (or at least those that are well-established) simply do not fit well enough with the modelling techniques that are in common use for these FM actually to be employed widely. This problem is not, however, caused by the models underpinning the FM necessarily being inappropriate, but simply by the methods having been created without sufficient understanding of how practicing software engineers might need to use them. Hence, the root cause of this problem is indeed the regrettable separation between the disciplines of FM and SE that is referred to in the quotation from Robinson with which this section began.

4 Requirements for Teaching FM in SE

Fortunately, this separation between FM and SE is not so substantial as to make it impossible to link FM with the models that are commonly used within SE. It does, however, pose a very real challenge for educators who take the view advocated at the beginning of this paper, that it is important to try to teach FM as part of SE. This challenge is that they not only have to motivate the study of FM despite the technical difficulties of the topic, but they also have to overcome the reluctance of students to engage with material that they will soon discover does not appear to be widely used within current SE practice.

This challenge was discussed in [16], where it was concluded that there are three parts to the solution. The first part is to ensure that FM are introduced right at the beginning of the curriculum, along with the introduction to SE concepts. The second part is to link the FM with the corresponding SE concepts, so that students appreciate that a formal specification of a system needs to be developed from the requirements for it in order to provide a precise description of how each of the system functions will affect the persistent data that the system must store. The third part is then to link this need for a formal specification with the uses that need to be made of it later in the SE process, such as in validating the designs for individual functions and in generating test cases for them.

These three parts can therefore be regarded as requirements for how FM should be taught within SE, and the concept of modelling systems is fundamental to each of them. Specifically, while the basic introduction to SE concepts commonly begins with the idea of a process model, the activities of analysis, design, construction and testing that are the primary components of such a model are all concerned essentially with creating or manipulating models of a system that is being developed. Hence, as these activities are studied in more detail, so that students learn how to carry them out, it is now standard practice for this introduction to focus on the relevant diagrammatic models of data and behaviour.

Given this, a key part of the argument of this paper is that the formal equivalents of these models can and should be introduced as well. This is because previous work by this author [17] has shown that the approach of methods integration, which links the formal and diagrammatic versions of such models, does help students to gain a better understanding of both, since the diagrammatic and formal versions of at least the basic models for common concepts are fundamentally very similar. For instance, databases are commonly relational, associations between classes are either relational or functional, and the behaviour of abstract data types can be represented in terms of state machine models. As long as the educational approach focuses on these similarities, rather than on the details of how the underlying formal models are then utilised within particular FM, the desired goal can be achieved of helping students to understand why FM are important within SE.

5 Teaching Topics for FM in SE

The experiences described above have shown that the basic concepts of FM can be introduced effectively in the early stages of an SE curriculum, and need to be introduced early if students are to begin to appreciate their importance. The next question is therefore what topics related to FM need to be taught in order to provide this necessary coverage of basic concepts, and in particular whether it is either appropriate or necessary to teach any one of the established FM as a complete method, or whether a more selective approach should be taken. As such, this question arose early in the author's experience of teaching FM, and attempting to answer it led to a programme of action research which resulted in the work that is described in [17].

The starting point for this experience was the approach of teaching Z as a formal specification method, as presented in the standard textbooks. The problem that was soon found with this approach was that these texts did not suggest any method for constructing specifications, but instead focused on the various mathematical constructions that could be employed in the specifications. This focus left the students feeling a bit like the audience at a magic show, asking the question "*where did that bit of the specification come from?*", meaning that they were gaining little understanding of how they could actually use such methods themselves. Since a key objective of teaching FM was that the students should be able to use the methods themselves, this approach therefore had to be classed as inadequate.

As the search for better approaches progressed, so it became apparent that using FM did not actually mean that the students needed to be able to use all of the features that the methods offered. Rather, they needed to understand the key concepts, so that they could then use them to express models that were sufficient for the purposes within SE that provided the motivation for teaching the methods. The approach that was therefore developed, as described in [17], was to illustrate these formal concepts by linking them closely with the corresponding diagrammatic models, so that the process of developing a specification for a system became one of translating features in the diagrams into the corresponding formal constructions. This approach allowed a complete specification to be built in a systematic fashion, but it became apparent that actually this complete specification was of less importance for the subsequent uses within SE processes than the various fragments that were derived from the diagrams.

Another important result from this research programme was the recognition that the key SE skills that students needed to develop form a sequence of stages, where the most important boundary in the sequence is between a restricted subset of SE, which is termed software development (SD from now on), and SE itself. Here, the main restriction that defines SD is that it is concerned just with qualitative characteristics such as functional correctness, and does not consider quantitative issues such as the quality of designs or processes, beyond the need to achieve at least the level of basic usability that should be provided by a sensibly structured graphical user interface.

This model of a sequence of stages underpins the results for how the teaching of FM should be fitted into an SE curriculum that are presented in [16]. In particular, the arguments that have been presented above for how FM need to be introduced within an SE curriculum apply specifically to the teaching of FM within this SD stage in developing students' SE knowledge and skills. Hence, for this stage the conclusion that comes from the experience described in [16, 17] is that it is certainly not necessary to teach any complete FM in its currently form. Rather, what is necessary is to develop students' understanding of how the concepts that are common to various FMs relate to the concepts of models of software system structures. This should then equip the students to apply these formal concepts in working with these models, as part of the core activities within the SE processes about which they are learning.

Of course, a degree programme in SE needs to develop students' SE knowledge and skills beyond this SD stage, and this further development obviously needs to apply to FM as well as to other topics in SE. At this more advanced stage the arguments for or against trying to teach a complete FM are much less clear, but what is clear is that the more advanced coverage of SE will involve introducing students to a wider range of models, both for SE products and SE processes, and the same needs to apply to FM as well.

Specifically, FM are based on a variety of different kinds of models, which can be regarded as giving rise to various different paradigms for FM. Thus, the use of data models based on sets, relations and functions gives rise to a relational paradigm that includes Z and VDM; the extension of these data models to include objects and identities gives rise to a corresponding object-oriented paradigm that includes Object Z and VDM++; the definition of abstract data types in terms of axioms for operations gives rise to an algebraic paradigm that includes OBJ and Alloy; the description of interactions between processes gives rise to a process algebra paradigm that includes CCS, CSP and Lotos; and the description of behaviour of processes in terms of state machine models gives rise to an abstract state machine paradigm that includes ASM, X-Machines, B and Event-B.

Clearly, it would not be realistic to even try to cover all of these methods in any one curriculum, and it would be equally unrealistic to attempt to predict which of them (if any) might become more widely used in future. On the other hand, it would not be so unrealistic to aim at covering a representative of each of these paradigms, and the differences between them suggest that it would be valuable for students to be exposed to each, in the same way that it is regarded as valuable for them to have been exposed to the different paradigms for programming languages.

6 Summary and Conclusions

In summary, therefore, the argument that this paper has made is firstly that FM must be an important part of the SE curriculum, and secondly that modelling is already an important aspect of this curriculum, and one which will become much more important as the methods of MDE become more widely adopted. This means that the curriculum needs to be structured round the relationships between FM and SE that arise from linking formal models to the diagrammatic ones that are currently used in SE practice. While making these links would obviously be easier if current FM were actually related more closely with SE practice, the paper has argued that it is essential to introduce students to FM in the early stages of teaching SE. At this introductory level, the differences between the various FM are less significant than the similarities in the underlying concepts, as these concepts are used in modelling the structures of the software systems that students are learning to develop.

Specifically, the paper has argued that students need to appreciate firstly the role of a formal specification of a system in providing a precise description of how each of the system functions will affect the persistent data that the system must store, and secondly how this description is then used in later activities in the SE process, such as validating the designs for individual functions and generating test cases for them. It has then shown that these vital objectives do not require a complete FM to be studied in detail, and that at the stage of introducing SE concepts the approach of trying to do so may actually hinder achieving them. At a more advanced level, however, there may be more justification for teaching a complete FM, although here the paper has argued that it is probably more important to cover the various paradigms for FM that have been identified, so as to ensure understanding of the models that underpin them.

References

1. Naur, P., Randell, B. (Eds.): Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968. Brussels, Scientific Affairs Division, NATO (1969)
2. ACM/IEEE, Joint Task Force for Computing Curricula 2005: Computing Curricula 2005: The Overview Report. ACM/IEEE, 2005.
3. Cowling, A.J.: Software engineering education: where's the appliance of science? In: King, G., Brebbia, C.A., Ross, M., Staples, G.: Software Engineering in Higher Education 1994, pp. 385–392. Computational Mechanics Publications, Southampton (1994).
4. Robinson, K.: Embedding Formal Development in Software Engineering. In: Dean, C.N., Boute, R.T. (Eds.): Teaching Formal Methods 2004. LNCS vol. 3294, pp. 203–213. Springer, Heidelberg (2004).
5. Robinson, K.: Reflecting on the Future: Objectives, Strategies and Experiences. In: Istenes, Z (Ed.): Formal Methods in Computer Science Education 2008, pp 15-24. ETAPS (2008).
6. Lutz, M. J., Naveda, J. F., Vallino, J. R.: Undergraduate software engineering. Commun. ACM 57(8), 52-58 (2014).
7. Cowling, A. J.: The Role of Modelling in the Software Engineering Curriculum. J. Syst. Software 75, 41-53 (2005).
8. IEEE-CS/ACM, Joint Task Force on Computing Curricula: Computing Curriculum - Software Engineering: Final Report. ACM/IEEE, 2004.
9. IEEE-CS/ACM, Joint Task Force on Computing Curricula: Draft For Public Review: Software Engineering 2013. ACM/IEEE, 2013.
10. Jacobson, I., Ng, P.-W., McMahon, P.E., Spence, I., Lidman, S.: The Essence of Software Engineering: The SEMAT Kernel. CACM 55 (12), 42-49 (2012).
11. Object Management Group: Unified Modeling Language™ (UML®), http://www.omg.org/spec/UML/
12. Eclipse foundation: EMF Documentation, http://eclipse.org/modeling/emf/docs/
13. Cowling, T.: Model-Driven Development and the Future of Software Engineering Education. In: 26th International Conference on Software Engineering Education and Training, pp 329-331. IEEE Computer Society Press (2013).
14. Thomson, C., Holcombe, M.: Applying XP Ideas Formally: The Story Card and Extreme X-Machines. 1st South-East European Workshop on Formal Methods, pp 57-71. South-East European Research Centre, Thessaloniki (2003).
15. Cowling, A.J.: A Semi-Formal Approach to Introducing Formal Methods. 3rd South-East European Workshop on Formal Methods, pp 126-140. South-East European Research Centre, Thessaloniki (2007).
16. Cowling, A.J.: Stages in Teaching Formal Methods. In: 23rd International Conference on Software Engineering Education and Training, pp 17-24. IEEE Computer Society Press (2010).
17. Cowling, A.J.: Translating Diagrams: A New Approach to Introducing Formal Methods. In: 18th International Conference on Software Engineering Education and Training, pp 121-128. IEEE Computer Society Press (2005).