# Balancing Insight and Effort: the Industrial Uptake of Formal Methods

John Fitzgerald[1] and Peter Gorm Larsen[2]

[1] School of Computing Science, Newcastle University, UK
[2] Engineering College of Aarhus, Denmark
John.Fitzgerald@ncl.ac.uk, pgl@iha.dk

**Abstract.** Our goal is to help the developers of computer-based systems to make informed design decisions on the basis of insights gained from the rigorous analysis of abstract system models. The early work on model-oriented specification has inspired the development of numerous formalisms and tools supporting modelling and analysis. There are also many stories of successful industrial application, often driven by a few champions possessing deep a priori understanding of formalisms. There are fewer cases of successful take-up or adoption of the technology in the long term. We argue that successful industrial adoption of this technology requires that potential users strike a balance between the effort expended in producing and analysing a model and insight gained. In order to support this balancing act, tools need to offer a range of levels of effort and insight. Further, educators need to recognise that training in formal development techniques must support this trade-off process.

## 1 Introduction

"Start by being systematic. Specify crucial facets — of your application domain, your requirements and your software designs — formally. Then program (i.e., code) from there! ...
... a few customers are willing to accept today's rather high cost of formal development"

*Dines Bjørner [1]*

Formal methods are not immune from commercial reality [2,3,4,5]. They must be applied in a cost-effective manner so that the effort invested in building precise and abstract models yields insight that will "pay back" during system development [6]. We share with Dines Bjørner the position that even a little rigour, carefully applied, can bring substantial benefits. Yet, in order to give developers the option of applying "a little rigour", we must offer techniques and tools that are adaptable to lightweight or heavyweight use, as the application and business demand.

A development engineer is faced with a wide range of formal techniques and tools. Each demands a certain *effort*, by which we mean the combination of time and resources required to use the technique or tool. Each also promises some *insight* into the ways a particular system may behave and the mental energy that must be released to produce the final documented product. Generally, deeper insight demands greater effort; the skill is to balance the two, defining a systematic approach that yields sufficient insight for the task for a reasonable investment of effort. Beyond a certain level of effort, the gain in insight may not be valuable for the application, and the engineer should not be forced into unnecessary analysis or verification. The balance between effort and insight has been central to our work supporting industry adoption of formal techniques by evolutionary steps rather than revolutionary change. Although we are very positive about the benefits of formalism, we do see the overt (or covert) forcing of formal approaches into industrial practice as counterproductive [7].

In this paper, we examine a range of techniques and tools associated with model-oriented specification of the kind pioneered by Bjørner and many others. In each case, we review the effort/insight balance afforded by the technique and try to identify the future developments that will allow developers the freedom to choose the appropriate technology.

We have deliberately used the word *uptake* in the title of this paper in contrast to *application*. There are numerous successful applications of formal methods in many domains [8,9]. The approach with which we are most closely associated, VDM (the Vienna Development Method) has also seen some significant and instructive applications in recent years [10]. It is worth stressing that we are here interested in the long-term, sustainable industrial adoption of formally-based techniques than their successful application in isolated cases driven by specialist champions with deep a priori knowledge of specific methods. We freely admit to having been involved in many applications but few cases of take-up.

The formal methods community has developed a wide range of formalisms tailored to rather specific application environments and built on distinct semantic foundations. Our background is in model-oriented formalisms that emphasise precision obtained by applying (usually discrete) mathematics and logic to the semantics of languages used for expressing system models. The approach that we have developed, based on VDM, emphasises the use of abstract and rigorous models to help developers manage complexity and explore the consequences of alternative design decisions in early stages of the life cycle. Thus, abstraction and rigour in modelling have been more significant for us than code verification.

A good model guides your thinking, a bad one warps it.

*Brian Marick*

## Tools, Techniques, People and Processes

Successful systems development businesses need to recruit the right people, employ an appropriate development process and make use of the tools and techniques that fit the development challenge at hand. It is very hard to find companies that excel in all three areas at the same time. Typically, small specialist companies with a niche market can place an emphasis on special techniques, including formal ones. In such organisations,

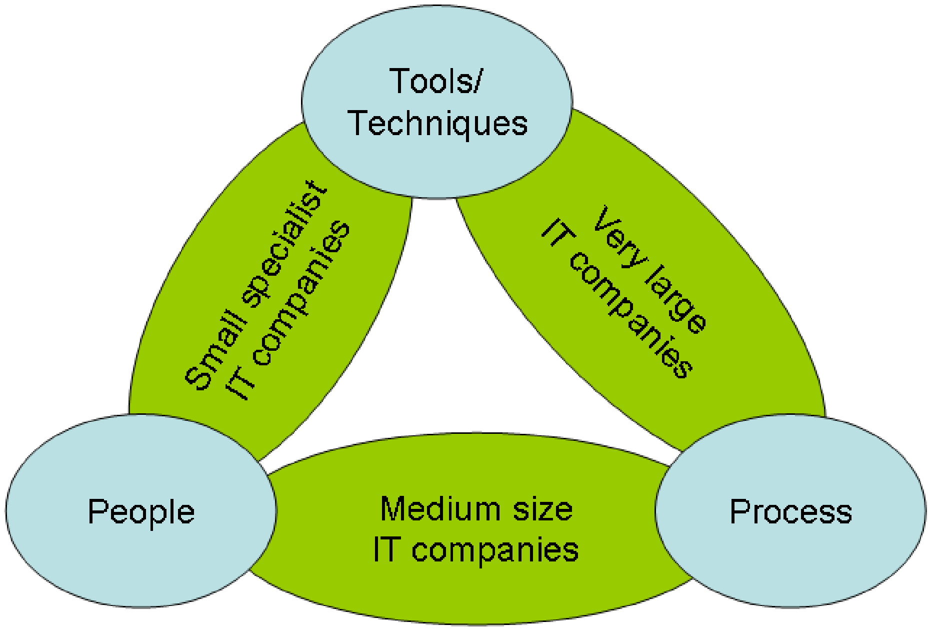

**Fig. 1.** The People, Process and Tools/Techniques Triangle

the focus is on the tools/techniques themselves and the highly skilled individuals who are needed to apply them. Medium sized software companies, where the distance from the bottom to the top of the corporate ladder is short, focus primarily on processes and the people. Very large software companies often focus on process and tools/techniques but from a long-term strategic perspective in which the dependence on small numbers of very highly skilled individuals is diminished. These are three rather independent dimensions are illustrated in Figure 1.

Formal methods form a part of the tools/techniques area of the picture, employed by good developers when it makes business sense to do so. Indeed, the proponents of formal methods may have concentrated too much of their efforts on tools and techniques at the expense of people (making methods accessible to the majority of professionals) and processes (integrating the technology with existing practice). Formal techniques may not always be the right choice for essential parts of systems, so it is important to have a good understanding of the interaction between a formally developed component and parts that are developed using other means. In addition entire systems are seldom developed from scratch. In many projects large legacy components form a part of the solution and so it is important to be able to easily understand how such legacy parts fit with a formal model.

In this paper, we consider this balance between effort and insight, especially as it has been found in model-oriented specification and in VDM. We first review our own involvement by giving a brief account of VDM and developments in the formalism in recent years (Section 2). In Section 3, we consider the range of tool features now becoming available. For each, we discuss the insights to be gained and the effort to be invested in using them. In Section 4 we discuss the consequences for education and training if future generations of engineers and research scientists are to take advantage of the full range of formalisms and tools becoming available now. Finally in Section 5 we give a few concluding remarks.

## 2 Background: VDM

We have been active in the use of the Vienna Development Method (VDM), one of the longest established model-oriented formalisms. We studied under Dines Bjørner and Cliff Jones[3]. Larsen studied in "the Danish School" of VDM, which emphasised explicit specification of functionality, leading to the possibility of executable models. Fitzgerald was rooted in "the British School" which gave greater prominence to the need for proof and, where possible, implicit specification by postconditions denoting relations on inputs, results and persistent state variables. The Danish School emphasised large-scale systems and compiler technology; the British School was focussed on validation through proof and refinement-based development. Jones provides an interesting account of the scientific development of VDM [11]. Our collaboration has, so some extent, been the story of an accommodation between these two schools.

We worked together briefly on the BSI/ISO standardisation panel of VDM's specification language (VDM-SL) [12,13]. Larsen took a leading role in completing the denotational semantics of the full language [14]. He went on to pioneer the development of industry-strength tool support for VDM-SL in IFAD. Fitzgerald was meanwhile working on the interaction between modular structuring mechanisms and user-guided proof in the typed Logic of Partial Functions [15] and started work with the British Aerospace Dependable Computing Systems Centre at Newcastle University.

Our first close collaboration was on the ConForm project funded under the European Systems and Software Initiative and conducted at British Aerospace Systems and Equipment (as it then was) in Plymouth. The project involved the concurrent development of a security-related software component using, in one stream, current best practice and, in the other stream, model-oriented specification. The specification was developed in VDM-SL by BAe engineers, with the IFAD tools and several ad hoc tools to integrate the formal model with structured methods already in use in the company. A study of the two parallel developments, while far from being a controlled experiment, indicated how formally-based tools might be used in practice. As a consequence of our logging all queries raised against the requirements documents, the study also provided evidence of the kinds of insight that arise when formal models are constructed [16].

The ConForm experience led us to develop a lightweight and tool-supported approach to formal modelling that we first presented in 1998 [17]. Subsequently both of us have used VDM with many different industrial users in various application domains. Some of the work has been reported in public, e.g. [18,19,20,21].

[3] Variously referred to as "the VDM twins" and, with Peter Lucas, as "the Ancient Greeks" behind the original VDM, latterly FME and FM, Symposia! Both of them frequently emphasise the crucial contributions of Lucas, Bekič and many others to the foundations of VDM.

Fitzgerald spent most of the following ten years in academia working with the aerospace industry and, for a couple of years, in a start-up company, Transitive. Larsen, by contrast, spent most of his time in industry at IFAD and Systematic, recently joining academia at the Engineering College of Aarhus. IFAD, the company that developed the original VDM tools [22], sold the technology on to the major Japanese company CSK [23]. The tools remain under very active support and development today. The new Overture initiative [24] is developing an open-source tools platform and plug-ins to deliver at least the same functionality as the rather more monolithic VDMTools.

VDM today is a well-established formal method based on the ISO standardised specification language VDM-SL [12] and its object-oriented extension VDM++ [25]. Further extensions have provided facilities for description and analysis of distributed real-time embedded systems [26,27,28], including explicit modelling of alternative system architectures and deployment of functionality to computation and communication resources.

We have reported elsewhere on the current state of VDM's tools and given data on industrial applications [10]. However, it is worth briefly reviewing a leading current application as an indication of how VDM is used today. FeliCa Networks Inc.[4] has been developing a next generation mobile integrated circuit chip, based on a contactless card technology developed and promoted by Sony [29]. The specification development process was carried out in three phases:

1. Writing an informal definition of the requirements in Japanese (383 pages).
2. Creating UML diagrams based on this document.
3. Modelling the system in VDM++ with over 100kLOC of VDM++ (677 pages).

Validation of the VDM++ model involved over 10 million tests. During phases 1 and 2 reviews found only 93 contradictions and faults in requirements and specifications in total. In phase 3, 162 faults were found through the process of writing and reviewing the VDM++ model. In addition 116 faults were found by executing the formal model in VDMTools. Finally, an extra 69 faults were found by combining the evaluation team and the specification writing teams in reviews. The discovery of these faults are all examples of insight gained by the formulation of the abstract model and the analysis on it. No refinement or formal verification has taken place, but the use of formal modelling has been viewed by the company as a considerable success, so balance of effort and insight seems to have been appropriate. The FeliCa development team included more than 50 people and the three year project has been completed on time, which is remarkable in itself. The product is produced in high volume, with potentially high recall costs in case of defects. By the end of November 2006 more than one million chips had been shipped.

## 3 A Tools Viewpoint

Developing and maintaining good industry-strength tool support is extremely time consuming. The formal methods community has spread its effort over many different formalisms. For the developers of the large number of specialised tools, integration with

[4] `www.felicanetworks.co.jp`

industrial users, tools and processes has not been of great importance. Thus, it is rare to find a level of tool support that is comparable with the standards for the industrial leading software development tools [30].

Tools have a strong influence on modelling style [31]. Interestingly model-oriented formalisms with similar semantic foundations such as VDM, Z [32] and B [33] have very different tool support. For VDM emphasis has been placed on the provision of an executable subset of the modelling language and, consequently, on validation of abstract models using testing techniques [34,35,36]. For Z the focus has been to a greater extent on proof support [37,38,39]. For B effort has been directed at providing automated proof support for refinement and code generation [40,41]. These approaches strike different balances between insight and effort, and between insight and concrete results such as code. In this section we review the different kinds of feature that can be included in a tool to support formal modelling and analysis, commenting on the balance for each of these.

### 3.1 Static analysis features

Very good tools now exist for efficiently developing parsers for formal languages. Once the conformance of a formal model to the language grammar has been confirmed, a variety of static checks can be performed. Probably the best known static analyser from the programming world is Lint [42] which performs simple static tests that identify potential code defects. The popularity of Lint and tools like it is partly down to the excellent balance between insight and effort. The effort is limited to running the analyser and subsequently examining the suspicious constructs 'flagged' by the tool. Provided the number of false positives is tolerable, the insight gained in spotting these defects is valuable. Similar 'push-button' technology has been advocated for formal methods tools for some time and is now becoming a reality [43,44,45,46].

The availability of a formal semantics for the modelling language enables a wide range of automatic or semi-automatic static analysis tools:

**Type checkers:** This kind of feature is a pure push-button technology where all the errors reported must be fixed by the user [47]. This kind of feature is always worthwhile because the cost of the analysis is low and the results identify genuine defects. The level of insight gained is rather shallow: a type correct model is a long way short of being validated! In languages like VDM++, in which type membership may be restricted by arbitrarily complex invariants, the full type-checking task involves the generation of proof obligations.

**Proof obligation generators:** In order to ensure internal consistency of a formal model it is typically possible to formulate a collection of "proof obligations" that indicate potential defects in a formal model [48]. Many of these surround the potential mis-application of partial operators (a kind of 'run-time error'). More subtle proof obligations also arise such as the necessity to prove that defined operations denote non-empty relations. Assuming that all of these obligations can be discharged the formal model is guaranteed to be internally consistent, i.e. it has a meaning. However, this is no guarantee that it is describing the "right" thing. The current VDM-Tools technology stops at this level, but push-button proof of obligations has been demonstrated and the technology to support this, using HOL, is once again under active development. Proof obligation generation is automatic and hence low-cost. Discharging of obligations can not be completely automated; tools that use this approach must provide a form of interactive prover unless unproved obligations are to be left for inspection. The level of insight gained is correspondingly higher in that failure to discharge an obligation may suggest a more subtle defect than can be identified by type checking alone.

**Assertions in program code:** In-line specification of contracts (including VDM-like invariants, preconditions and postconditions) provides an opportunity for enhanced static checks on program code. This kind of approach has a long history [49] and current initiatives around Java look particularly promising [50,51]. The strength of these features are that, by spending the effort in developing the assertions, the static analysis can typically provide deeper insight into subtle errors in the code that can then easily be fixed.

**Model checkers:** The model-checking concept is general and applies to many logics and models. Its particular benefit is the production of a counterexample in the case where a checked property is not satisfied. A simple model-checking problem is testing whether a given formula in the propositional logic is satisfied by a given model [52,53]. This very powerful technique is fully automated and so has potential for giving a very good balance between effort and insight. However, formulating the model in order to support efficient checking may require high effort. There are many stand-alone model checkers and increasing interest in combining them with other analysis tools. More recently small but powerful combinations of model-checking techniques have been applied in Alloy [54], so far on relatively small examples.

## 3.2 Dynamic Analysis of Formal Models

Models are not necessarily executable [55] but formal semantics for modelling languages make their symbolic execution [56] possible, albeit at high cost. It is possible to define executable subsets within which dynamic behaviour can be explored. The danger of restriction to an executable subset is that the model's abstraction level gets too low, hampering the insight gained from analysis. Our experience is that the use of an executable subset can still provide many benefits to a user with a training in abstraction [57]. Indeed, the borderline of executablity is not as clear as one might expect [58].

Dynamic analysis of formal models comes in several forms:

**Interpreters:** Interpreters are available for executable subsets of several modelling languages, including VDM [34]. Some of these tools also provide debugging capabilities similar to those provided by programming environments already familiar to software developers. The non-exhaustive testing supported by an interpreter helps a user to step into the evaluation of an unexpected result. Typically an interpreter feature is easy to use and gives deep insight into the subtleties of a formal model so although one must manually produce the test arguments to exercise there is normally a good balance between effort and insight here.

**Test case generation:** Where the use of an executable subset enables testing of models, it can also be valuable to automate the testing process in different ways. Automatic generation of test cases [59,60] can produce entire test suites [61]. Considering the balance between effort and insight, this kind of enhancement to the automation of testing is almost always favourable, particularly when the generated test cases can be used for testing the final implementation.

**Test analysis support:** In order to provide further insight into the quality of the test set used on a model, it is possible to display the coverage of the tests carried out, for example by using colour in ways similar to those used for programming languages [62]. Alternatively graphical overviews of executions can be used to give the user a deeper understanding for what is going on [27]. Depending upon the time that must be spent creating this kind of feature, it is typically worth the low effort required to monitor the coverage of tests on a model. The insight into the functional characteristics of the model is very limited, but it may lead to improved test sets that themselves prove worthwhile.

## 3.3 Verification of Formal Models

The expressiveness of formal modelling languages places limits on the extent to which analyses can be automated. For realistic industrial applications one must normally settle for as much automation as possible and then provide support for manual analysis [63]. In the area of formal verification one can divide features into those that support formal refinement and those that support formal proof:

**Formal refinement support:** Many formal modelling languages have an associated notion of refinement enabling the description of successively more concrete models, with a formal relationship between each of them [64]. Many different tools are able to support this process [65]. Typical approaches involve the definition of a refinement relation [66,67] between concrete and abstract models. The balance between effort and insight gained here is problematic from an industrial perspective unless either there is substantial automation or the correctness of the application is sufficiently important to warrant the extra cost in such a fully formal development [33,68]. However, there is also work towards automated support for refinement [69].

**Theorem provers:** The ultimate advantage of using a formal over an informal model is the ability to verify its properties to a high level of rigour [70], even for infinite state systems. Given our concern to balance effort and insight, some degree of automation is required here [71]. A complete reliance on proof automation may lead to the use of a notation that lacks expressiveness [72]. In reality, for many formal modelling applications, we would not wish to compromise the accessibility of the modelling language in order to support a certain level of automated analysis. The balance between the formalism and the extent of automation is crucial. It depends on the projects needs: are these to ensure internal consistency by discharging as many consistency proof obligations as possible, or are they to prove system-level properties (we have used the term validation conjectures)? In the former case a high level of automation may be desirable. In the latter the real insight gained by guiding a proof may help understanding of the rest of the model, in particular detecting and eliminating defects [73,74].

Until recently, we have not seen a strong enough industrial case for bringing proof support into VDMTools because of the computational overheads and also the possibility of having to build an interface for user guidance of the specialised proof process. Our experience with manual proof support for the proof theory of VDM-SL [15] indicated that many proof obligations are generated by even a simple model and it is vital to be able to discharge as many of these as possible without user guidance. In the PROSPER project [75], it was possible to discharge the vast majority of generated proof obligations automatically (up to 90 % for a railway application [76]) and we are now working on reproducing this for VDM++ using HOL4. We leave undischarged obligations for inspection, but view the development of good human-guided but machine-assisted proof tools (based on an appreciation of the cognitive aspects of proof) as an essential research goal.

## 3.4 Connection to the Development Environment

Modelling and analysis techniques based on formal notations will rarely be used for the development of an entire system, so their products should fit with the results of applying other techniques, as well as with the processes employed in the development team. In order to balance the effort spent on producing the formal models with the insight gained this is clearly one of the areas with potential for payback in terms of minimising the time that needs to be spent in the final implementation phases. Supporting this for VDMTools meant spending major efforts on tasks that formalists might find uninteresting, but which are essential for deployment. For example, we have had to develop interfaces to proprietary WYSIWYG document editors, use ASCII syntax, build code generators, application programmer interfaces and even links to UML tools! Here we list some of the features that we consider particularly significant in connecting our tools to people and processes in the development environment.

**Code generators:** If a formal modelling language has an executable subset, there is also potential for automating a part of the coding process. This adds value to the formal models being produced and thus affects the balance between the effort spent producing the model and its value as a basis for an implementation. Generated code will rarely be as efficient as a hand-coded implementation. However, given a reliable code generator, there is some confidence that the generated code accurately reflects the properties of the model. Critical applications demand the use of certified code generators [77]. The use of code generation for production code comes at a price in terms of the degree of abstraction that can be permitted in the model.

**Combination with other notations:** It is essential for industrial uptake that a tool for formal modelling is able to support whatever standards for processes and other tools are being used. It is even better if users are able to move back and forth between the various models and notations that are used, seeing updates in model reflected consistently in others. Considerable work has gone into developing appropriate couplings between formal methods tools and UML, the de-facto standard in large parts of the industry [78]. Such a bi-directional link with developed for VDM-Tools to support interaction with the Rose UML tool. The effort/insight balance is improved by linkage between informal and formal tools. However the links do not simply have to be with classical software design tools. For example, integration with a continuous time simulator is a promising vehicle for collaboration between systems and software engineers [79], both working in different, but both formal, spheres.

**Combination with Development Environments:** Companies have development environments that must be used for for integrating the final system. It is also likely that the implementation derived from a formal model will need to be integrated with code that is developed differently (e.g. for existing GUI or legacy code). Here the effort/insight balance is affected by the ease of doing this kind of integration. Features for combining existing code with a formal model may prove valuable [80], as may facilities for combining with a GUI interface [81].

## 3.5 Past and Future for Formal Methods Tools

Tools for formal modelling and analysis have come a long way since work began on VDM-SL parsers, but competition with conventional tools is hard. In the 1980s tools for formal specification were mainly limited to basic static checks for syntax- and type-correctness. At that time it was even possible to write PhD thesis about general formal methods tool support [82]. The 1990s saw an increase in the range of tools exploiting more of the formal semantics, in particular interpreters, code generators, test case generators, model checking and proof support. In addition many of the tools had support for combining formal models with informal, usually graphical, notations. At present, we conjecture that none of the formal methods tools are as highly featured as the leading industrial software development tools.

Open source platforms with potentially closed-source plug-ins offer a promising approach for delivering tools with the capabilities that we have discussed [83,24]. If this can be achieved tool builders will not have to start from scratch whenever tool support is to be developed for a new notation. Ideally, different views or parts of a system could be described in different formalisms and verification of properties could be performed by a variety of theorem provers in a compositional way. However, before this becomes a reality there are major theoretical challenges on semantic integration that must be addressed.

Our own experience with VDMTools and the Overture initiative leads us to want to address several areas:

– Co-simulation as an extension of executable specification and the use of interpreters for control applications in embedded real-time systems.
– Modelling faults and experimenting with alternative fault detection and tolerance mechanisms inside formal models.
– User-guided proof for gaining deep insight at a higher level than conventional theorem provers.

# 4 An Educational Viewpoint

A fool with a tool is still a fool.

*Grady Booch*

We have argued that successful industry adoption of formal techniques requires the balancing of effort and insight, and that tools and development environments should support this trade-off. No matter how advanced the tools, they can only achieve a measure of industry credibility if graduate engineers possess skills in abstraction and rigorous thinking, and are open to selecting the right techniques for the job. It therefore seems appropriate to ask how this should affect the aims, content and delivery of formal methods education.

We should not expect our students to share our motivations. In our experience, many students are driven by the need to develop skills that will be useful in pursuing a career and many are also driven by the satisfaction of building a working computer system and seeing it run. If we care about the industry uptake of formal techniques, we should care about all our students and not just the potential PhDs.

## 4.1 Aims

We suggest that the overriding aims of formal methods advocates in education should be: (i) to help students develop transferable skills of abstraction and rigorous modelling and analysis; and (ii) to develop the knowledge and skill needed to select tools and techniques on the basis of cost and potential effectiveness. These are not impossibly vague goals: Sobel's study [84], albeit the subject of a debate on experimental design [85,86], was a first attempt to assess whether a training in formal techniques may improve students' general analytic and problem solving skills [87]. In our teaching [88], with its origins in industrial courses, we have been led to ask whether we really know what skills we want to help our students develop and how we could establish whether our current courses are achieving this. Kramer has recently argued that abstraction skills are core to computing and that we should try to monitor the development of such skills through students' development [89]. It has been pointed out that the sorts of test we need are lacking, as most relevant tests focus on logical reasoning.

At a more practical level, we would like typical graduates, not only the most academically gifted, to at least know that next generation Integrated Development Environments will provide a higher static analysis capability than at present and will support expressive annotation-based languages in the manner of JML [50] and Spec# [90], allowing them to identify hitherto hard-to-detect errors in their code. We want them to be surprised when such technology is *not* deployed in the companies where they work and we would even like them to use it to get the edge on their fellow programmers! In that way we hope that they will be able to find an appropriate balance in the use of abstraction and rigour in their own work.

### 4.2 Content

Giving students a sense of the effort/insight trade-off means exposing them to a range of analysis techniques as well as offering them experiences that help them to see the insights that come from a range of analytic techniques, from manual to automated. In our own teaching experience [88], we apply a "lightweight" approach using VDM++. We emphasise practical applications, teaching through a range of examples derived from industry application. A strength of the approach is the relative familiarity of the structure of VDM++ to undergraduates already versed in object-oriented programming. However, analysis is so far limited to specification testing and proof obligation generation. Advanced techniques including proof and model checking, are taught separately. Practical experience, discussions centered around formal models and sharing of insights are central to the approach if we are to help students move from superficial and atomistic learning to a deeper appreciation of the costs and benefits of abstraction and rigour.

In a revision to its undergraduate curriculum in formal techniques, Newcastle is looking at beginning with JML (because Java is taught extensively in the first part of the curriculum), raising students' expectations about the forms of push-button analysis that can be applied to their programs. Similar concepts (pre-/post specification, use of invariants etc.) can then be lifted to the design level by teaching model-oriented specification in VDM++ with tool support and introducing validation through structured argument. At the final level, aimed at software engineering specialists who may well become tools developers themselves, we will introduce proof and model-checking as the technology that underpins the advanced analysis of both programs and design models.

### 4.3 Delivery

There has been much debate around the placing of formal methods in the wider computing curriculum. Should they be treated as a distinct discipline or should they be fully integrated with other material? van Lamsweerde [63] argues for the integration of formal methods into normal development activities. This surely suggests that training in formal techniques should be, to a large extent, part of the normal components of computer science and software engineering curricula. Wing's suggested approach [91] is to teach common elements (state machines, invariants, abstraction mappings, composition, induction, specification and verification) and to use tools to reinforce theory. She identifies the difficulty of winning fellow faculty members round as a real impediment to this [92].

Recent work [93] suggests that students' performance improves when they are provided with Integrated Development Environments encompassing specification support tools, static analysers and provers. It remains to be seen if this level of tooling deepens students' understanding of the models that they are creating and the formalism used. However, it may be seen as a step in the right direction by freeing students to concentrate on the meaning of a model rather than automatically checkable characteristics.

## 5 Concluding Remarks

Will formal methods remain niche technology with localised use dependent on energetic champions? We believe that some major changes are required to help the mainstream get the benefits of abstraction and rigour in system modelling and analysis. Tools are vital and, we have argued, a full range of tools have to be offered in a way that allows their gradual adoption into processes and by people already in place. This means much more collaboration between tools developers, very likely via open platforms, in order to give users the flexibility to balance effort and insight.

From a teaching perspective, we ought to be liberal in our use of a range of formalisms supported by tools in order to encourage the breadth of experience that will allow graduate engineers to select appropriate technology. We should not expect everyone to be interested in how we express formal semantics, but we should adopt course content and delivery styles that help them to feel the benefits of a little abstraction and rigour.

Formal approaches have been widely, but often quietly, adopted in modern programming languages and development environments. Industry-leading programming notations now include possibilities for increased abstraction through of abstract types such as sets, sequences and mappings and use of concepts such as invariants, pre and post-conditions formulated using predicates. None of these advances will be known as ‘formal methods’. Dines Bjørner, Zhou Chaochen and so many others have worked to create technology so fundamental that it disappears into the fabric of software and systems engineering. Surely this is an achievement to be proud of.

**Acknowledgments** We are grateful to the organisers of the Festschrift Symposium for the opportunity to prepare this paper, and most especially to Dines Bjørner for the lively inspiration, support and encouragement he has provided to us both over many years. As always we are grateful to our many colleagues involved in the current VDM Tools and Overture initiatives, especially Shin Sahara and Marcel Verhoef, for their contributions to the work reported in this paper and to Jeremy Bryans and Hugo Daniel Macedo for their helpful comments on a draft of this paper.

## References

1. Bjørner, D.: Software Engineering 1: Abstraction and Modelling. Springer, Berlin, Heidelberg (2006) ISBN: 978-3-540-21149-5.
2. Hall, A.: Seven Myths of Formal Methods. IEEE Software **7**(5) (September 1990) 11–19
3. Bowen, J.P., Hinchey, M.G.: Ten Commandments of Formal Methods. IEEE Computer **28**(4) (April 1995) 56–62
4. Tretmans, J., Wijbrans, K., Chaudron, M.: Software Engineering with Formal Methods: The Development of a Storm Surge Barrier Control System Revisiting Seven Myths of Formal Methods. Form. Methods Syst. Des. **19**(2) (2001) 195–215
5. Bowen, J.P., Hinchey, M.G.: Ten Commandments of Formal Methods ... Ten Years Later. IEEE Computer **39**(1) (January 2006) 40–48
6. Larsen, P.G.: On the Industrial Value of Models. In Duke, D., Evans, A., eds.: 2nd BCS-FACS Norhern Formal Methods Workshop, Ilkley, BCS-FACS, Springer (July 1997)

7. Glass, R.L.: The Mystery of Formal Methods Disuse. Communications of the ACM **47**(8) (2004) 15–17
8. Dan Craigen, S.G., Ralston, T.: An International Survey of Industrial Applications of Formal Methods. Volume Volume 1 Purpose, Approach, Analysis and Conclusions. U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, MD 20899, USA (March 1993)
9. Rushby, J.: Formal Methods and the Certification of Critical Systems. Technical Report CSL-93-7, Computer Science Laboratory, Menlo Park CA 94025 USA (December 1993)
10. Fitzgerald, J.S., Larsen, P.G.: Triumphs and Challenges for the Industrial Application of Model-Oriented Formal Methods. In Margaria, T., Philippou, A., Steffen, B., eds.: Proc. 2nd Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2007). (2007) Also Technical Report CS-TR-999, School of Computing Science, Newcastle University.
11. Jones, C.B.: Scientific Decisions which Characterize VDM. In Wing, J., Woodcock, J., Davies, J., eds.: FM'99 - Formal Methods, Springer-Verlag (1999) 28–47 Lecture Notes in Computer Science 1708.
12. P. G. Larsen and B. S. Hansen and H. Brunn N. Plat and H. Toetenel and D. J. Andrews and J. Dawes and G. Parkin and others: Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language (December 1996)
13. Plat, N., Larsen, P.G.: An Overview of the ISO/VDM-SL Standard. Sigplan Notices **27**(8) (August 1992) 76–82
14. Larsen, P.G., Pawłowski, W.: The Formal Semantics of ISO VDM-SL. "Computer Standards and Interfaces" **17**(5–6) (September 1995) 585–602
15. Bicarregui, J., Fitzgerald, J., Lindsay, P., Moore, R., Ritchie, B.: Proof in VDM: A Practitioner's Guide. FACIT. Springer-Verlag (1994) ISBN 3-540-19813-X.
16. Larsen, P.G., Fitzgerald, J., Brookes, T.: Applying Formal Specification in Industry. IEEE Software **13**(3) (May 1996) 48–56
17. Fitzgerald, J., Larsen, P.G.: Modelling Systems – Practical Tools and Techniques in Software Development. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK (1998) ISBN 0-521-62348-0.
18. Devauchelle, L., Larsen, P.G., Voss, H.: PICGAL: Practical Use of Formal Specification to Develop a Complex Critical System. In Fitzgerald, J., Jones, C.B., Lucas, P., eds.: FME'97: Industrial Applications and Strengthened Foundations of Formal Methods. Volume 1313 of Lecture Notes in Computer Science., Springer-Verlag (September 1997) 221–236 ISBN 3-540-63533-5.
19. Fitzgerald, J., Jones, C.: Proof in the validation of a formal model of a tracking system for a nuclear plant. In Bicarregui, J., ed.: Proof in VDM: Case Studies. FACIT Series. Springer-Verlag (1998)
20. Smith, P.R., Larsen, P.G.: Applications of VDM in Banknote Processing. In Fitzgerald, J.S., Larsen, P.G., eds.: VDM in Practice: Proc. First VDM Workshop 1999. (September 1999) Available at www.vdmportal.org.
21. Mukherjee, P., Bousquet, F., Delabre, J., Paynter, S., Larsen, P.G.: Exploring Timing Properties Using VDM++ on an Industrial Application. In Bicarregui, J., Fitzgerald, J., eds.: Proceedings of the Second VDM Workshop. (September 2000) Available at www.vdmportal.org.
22. Elmstrøm, R., Larsen, P.G., Lassen, P.B.: The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications. ACM Sigplan Notices **29**(9) (September 1994) 77–80
23. CSK: VDMTools homepage. *http://www.vdmtools.jp/en/* (2007)
24. Overture-Core-Team: Overture Web site. http://www.overturetool.org (2007)

25. Fitzgerald, J., Larsen, P.G., Mukherjee, P., Plat, N., Verhoef, M.: Validated Designs for Object–oriented Systems. Springer, New York (2005)
26. Verhoef, M., Larsen, P.G., Hooman, J.: Modeling and Validating Distributed Embedded Real-Time Systems with VDM++. In Misra, J., Nipkow, T., Sekerinski, E., eds.: FM 2006: Formal Methods, Lecture Notes in Computer Science 4085 (2006) 147–162
27. Verhoef, M., Larsen, P.G.: Interpreting Distributed System Architectures Using VDM++ – A Case Study. In Sauser, B., Muller, G., eds.: 5th Annual Conference on Systems Engineering Research. (March 2007) Available at http://www.stevens.edu/engineering/cser/.
28. Fitzgerald, J., Larsen, P.G., Tjell, S., Verhoef, M.: Validation Support for Distributed Real-Time Embedded Systems in VDM++. Technical Report CS-TR:1017, School of Computing Science, Newcastle University (April 2007)
29. Kurita, T., Oota, T., Nakatsugawa, Y.: Formal specification of an embedded IC for cellular phones. In: Proceedings of Software Symposium 2005, Software Engineers Associates of Japan (June 2005) 73–80 (in Japanese).
30. Holloway, M., Butler, R.W.: Impediments to Industrial Use of Formal Methods. IEEE Computer **29**(4) (1996) 25–26
31. Fitzgerald, J., Larsen, P.: Formal Specification Techniques in the Commercial Development Process. In Wirsing, M., ed.: Position Papers from the Workshop on Formal Methods Application in Software Engineering Practice, International Conference on Software Engineering (ICSE-17), Seattle. (April 1995) http://home0.inet.tele.dk/pgl/icse.pdf.
32. Woodcock, J., Davies, J.: Using Z – Specification, Refinement, and Proof. Prentice Hall International Series in Computer Science (1996)
33. Abrial, J.R.: The B Book – Assigning Programs to Meanings. Cambridge University Press (August 1996)
34. Larsen, P.G., Lassen, P.B.: An Executable Subset of Meta-IV with Loose Specification. In: VDM '91: Formal Software Development Methods, VDM Europe, Springer-Verlag (March 1991)
35. Mukherjee, P.: Computer-aided Validation of Formal Specifications. Software Engineering Journal (July 1995) 133–140
36. Larsen, P.G.: Ten Years of Historical Development: “Bootstrapping” VDMTools. Journal of Universal Computer Science **7**(8) (2001) 692–709
37. Houston, I.: The IBM Z Tool. In: VDM '91: Formal Software Development Methods, Springer-Verlag (October 1991) 691–692
38. Saaltink, M.: Z and EVES. In Nicholls, J., ed.: Z User Workshop, York 1991, Springer-Verlag (1992) 223–242 Workshops in Computing.
39. I.Toyn, J.: CADiZ: an Architecture for Z tools and its Implementation. Softw.-Pract. Exp. (UK) **25**(3) (March 1995) 305–30
40. Lee, M., Sørensen, I.: B-tool. In Prehn, S., Toetenel, W., eds.: VDM '91 – Formal Software Development Methods, Springer-Verlag (October 1991) 695–696
41. Clearsy: Atelier B Web site. http://www.atelierb.societe.com/index_uk.htm (2007)
42. Johnson, S.: Lint, a C Program Checker. Computer Science 65, Bell Laboratories (December 1977)
43. Rushby, J.: Model Checking and Other Ways of Automating Formal Methods. In: Software Quality Week, San Francisco, CA (May 1995) Position paper for panel on Model Checking for Concurrent Programs.
44. Heitmeyer, C.L.: On the Need for Practical Formal Methods. In: FTRTFT '98: Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, London, UK, Springer-Verlag (1998) 18–26
45. Heitmeyer, C.: A Panacea or Academic Poppycock: Formal Methods Revisited. In: High-Assurance Systems Engineering, 2005. HASE 2005. Ninth IEEE International Symposium on, IEEE (October 2005) 3–7

46. Heitmeyer, C.: Developing Safety-Critical Systems: the Role of Formal Methods and Tools. In: SCS '05: Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2006) 95–99
47. Pierce, B.C., ed.: Advanced Topics in Types and Programming Languages. MIT Press (2005)
48. Bernhard K. Aichernig and Peter Gorm Larsen: A Proof Obligation Generator for VDM-SL. In Fitzgerald, J.S., Jones, C.B., Lucas, P., eds.: FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997). Volume 1313 of Lecture Notes in Computer Science., Springer-Verlag (September 1997) 338–357 ISBN 3-540-63533-5.
49. Luckham, D.C., von Henke, F.W.: An Overview of Anna, A Specification Language for Ada. IEEE Software (March 1985) 9–22
50. Burdy, L., Cheon, Y., Cok, D., Ernst, M.D., Kiniry, J.R., Leavens, G.T., Leino, K.R.M., Poll, E.: An overview of JML Tools and Applications. Intl. Journal of Software Tools for Technology Transfer **7** (2005) 212–232
51. Chalin, P., Hurlin, C., Kiniry, J.: Integrating Static Checking and Interactive Verification: Supporting Multiple Theories and Provers in Verification. In: Proceedings of Verified Software: Tools, Technologies, and Experiences (VSTTE). (2005)
52. Clarke, E., Emerson, E., Sistla, A.: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. ACM Transactions on Programming Languages and Systems **8**(2) (April 1986) 244–263
53. McMillan, K.L.: Symbolic Model Checking. PhD thesis, Carnegie Mellon University, School of Computer Science (1992) Kluwer Academic Publishers, ISBN: 0-7923-9380-5.
54. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press, Heyward Street, Cambridge, MA02142, USA (April 2006) ISBN-10: 0-262-10114-9.
55. Hayes, I., Jones, C.: Specifications are not (Necessarily) Executable. Software Engineering Journal (November 1989) 330–338
56. Kneuper, R.: Symbolic Execution as a Tool for Validation of Specifications. PhD thesis, Department of Computer Science, Univeristy of Manchester (March 1989) Technical Report Series UMCS-89-7-1.
57. Andersen, M., Elmstrøm, R., Lassen, P.B., Larsen, P.G.: Making Specifications Executable – Using IPTES Meta-IV. Microprocessing and Microprogramming **35**(1-5) (September 1992) 521–528
58. Fröhlich, B.: Towards Executability of Implicit Definitions. PhD thesis, TU Graz, Institute of Software Technology (September 1998)
59. Dick, J., Faivre, A.: Automating the Generation and Sequencing of Test Cases from Model-Based Specifications. In Woodcock, J., Larsen, P., eds.: FME'93: Industrial-Strength Formal Methods, Formal Methods Europe, Springer-Verlag (April 1993) 268–284 Lecture Notes in Computer Science 670.
60. Gaudel, M.C.: Testing can be formal, too. In Mosses, P., Schwartzbach, M., eds.: TAPSOFT'95: Theory and Practice of Software Development, CAAP/FASE, Springer (1995) 82–96
61. Burdonov, I., Kossatchev, A., Petrenko, A., Galter, D.: KVEST: Automated Generation of Test Suites from Formal Specifications. In: FM '99: Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume I, London, UK, Springer-Verlag (1999) 608–621
62. TestingFaqs.org: Test Coverage Tools. http://www.testingfaqs.org/t-eval.html (2007)
63. van Lamsweerde, A.: Formal Specification: a Roadmap. In: ICSE '00: Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA, ACM Press (2000) 147–159

64. Back, R.J.: On the Correctness of Refinement Steps in Program Development. PhD thesis, Åbo Akademi, Department of Computer Science, Helsinki, Finland (1978) Report A–1978–4.
65. Carrington, D., Hayes, I., Nickson, R., Watson, G., Welsh, J.: A Review of Existing Refinement Tools (1994) SVRC TR-94-8, University of Queensland.
66. Jones, C.B.: Systematic Software Development Using VDM. Second edn. Prentice-Hall International, Englewood Cliffs, New Jersey (1990) ISBN 0-13-880733-7.
67. Ah-Kee, J.: Operation Decomposition Proof Obligations. PhD thesis, University of Manchester (1989)
68. Badeau, F., Amelot, A.: Using B as a High Level Programming Language in an Industrial Project: Roissy VAL. In: Z to B Conference / Nantes. (2005) 334–354
69. L. Burdy, J.M.: Automatic Refinement. In: BUG Meeting, FM'99, Toulouse, France (1999)
70. Leavens, G.T., Abrial, J.R., Batory, D., Butler, M., Coglio, A., Fisler, K., Hehner, E., Jones, C., Miller, D., Peyton-Jones, S., Sitaraman, M., Smith, D.R., Stump, A.: Roadmap for Enhanced Languages and Methods to aid Verification. In: GPCE '06: Proceedings of the 5th International Conference on Generative Programming and Component Engineering, New York, NY, USA, ACM Press (2006) 221–236
71. Owre, S., Rushby, J.M., , Shankar, N.: PVS: A Prototype Verification System. In Kapur, D., ed.: 11th International Conference on Automated Deduction (CADE). Volume 607 of Lecture Notes in Artificial Intelligence., Saratoga, NY, Springer-Verlag (June 1992) 748–752
72. Paulson, L.C.: Generic automatic proof tools. In Veroff, R., ed.: Automated Reasoning and its Applications: Essays in Honor of Larry Wos. MIT Press, Cambridge, MA, USA (1997) 23–47
73. Harper, R.: Proof-directed debugging. Journal of Functional Programming **9**(4) (July 1999) 463–469
74. Dennis, L.A., Monroy, R., Nogueira, P.: Proof-directed Debugging and Repair. In Nilsson, H., van Eekelen, M., eds.: Seventh Symposium on Trends in Functional Programming 2006. (2006) 131–140
75. Dennis, L.A., Collins, G., Norrish, M., Boulton, R., Slind, K., Robinson, G., Gordon, M., Melham, T.: The PROSPER Toolkit. In: Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Berlin, Germany, Springer-Verlag, Lecture Notes in Computer Science volume 1785 (March/April 2000)
76. Terada, N., Fukuda, M.: Application of Formal Methods to the Railway Signaling Systems. Quarterly Report of RTRI **43**(4) (2002) 169–174
77. Dion, B., Gartner, J.: Efficient Development of Embedded Automotive Software with IEC 61508 Objectives using SCADE Drive. In: VDI 12th International Conference: Electronic Systems for Vehicles, VDI (October 2005)
78. Snook, C., Butler, M.: UML-B: Formal modeling and design aided by UML. ACM Trans. Softw. Eng. Methodol. **15**(1) (2006) 92–122
79. Marcel Verhoef, Peter Visser, J.H., Broenink, J.: Co-simulation of Real-time Embedded Control Systems. In: IFM 2007: Integrated Formal Methods, LNCS (July 2007)
80. Fröhlich, B., Larsen, P.G.: Combining VDM-SL Specifications with C++ Code. In Gaudel, M.C., Woodcock, J., eds.: FME'96: Industrial Benefit and Advances in Formal Methods, Springer-Verlag (March 1996) 179–194
81. Hekmatpour, S., Ince, D.C.: A Formal Specification-Based Prototyping System. In Barnes, D., Brown, P., eds.: Software Engineering 1986. Peter Peregrinus Ltd., London (1986) 317–335
82. McParland, P.J.: Software Tools to Support Formal Methods. PhD thesis, Queen's University Belfast (October 1989)

83. RODIN-Project-Members: RODIN Web site. http://rodin.cs.ncl.ac.uk/ (2007)
84. Sobel, A.E.K., Clarkson, M.R.: Formal Methods Application: An Empirical Tale of Software Development. IEEE Trans. Software Engineering **28**(3) (2002) 308–320
85. Berry, D.M., Tichy, W.F.: Comments on "Formal Methods Application: An Empirical Tale of Software Development". IEEE Transactions on Software Engineering **29**(6) (2003) 567–571
86. Sobel, A.E.K., Clarkson, M.R.: Response to "Comments on 'Formal Methods Application: An Empirical Tale of Software Development' ". IEEE Trans. Software Engineering **29**(6) (2003) 572–575
87. Sobel, A.E.K.: Empirical Results of a Software Engineering Curriculum Incorporating Formal Methods. In: Proceedings of SIGCSE 2000, ACM (2000) 157–161
88. Larsen, P.G., Fitzgerald, J.S., Riddle, S.: Learning by Doing: Practical Courses in Lightweight Formal Methods using VDM++. Technical Report CS-TR:992, School of Computing Science, Newcastle University (December 2006)
89. Kramer, J.: Is Abstraction the Key to Computing? Communications of the ACM **50**(4) (2007) 37–42
90. Barnett, M., Leino, R.M., Schulte, W.: The Spec# Programming System: An Overview. In: CASSIS Proceedings. (October 2004)
91. Wing, J.: Weaving Formal Methods into the Undergraduate Computer Science Curriculum. In: AMAST: 8th International Conference on Algebraic Methodology and Software Technology. (2000)
92. Palmer, T.V., Pleasant, J.C.: Attitudes Toward the Teaching of Formal Methods of Software Development in the Undergraduate Computer Science Curriculum: a Survey. SIGCSE Bulletin **27**(3) (1995) 53–59
93. Skevoulis, S., Makarov, V.: Integrating Formal Methods Tools into Undergraduate Computer Science Curriculum. In: Frontiers in Education Conference, 36th Annual, ASEE, IEEE (October 2006) 1–6