Newcastle University

COMPUTING SCIENCE

Are Formal Methods Ready for Agility? A Reality Check

Peter Gorm Larsen, John S. Fitzgerald, Sune Wolff

TECHNICAL REPORT SERIES

No. CS-TR-1218 **August 2010**

TECHNICAL REPORT SERIES

No. CS-TR-1218 August, 2010

Are Formal Methods Ready for Agility? A Reality Check

P. G. Larsen, J. S. Fitzgerald, S. Wolff

Abstract

The integration of agile software development techniques with formal methods has attracted attention as a research topic. But what exactly is to be gained from attempting to combine two approaches which are seen as orthogonal or even opposing, and to what extent do formal methods already support the principles of agility? Based on the authors' experience in applying lightweight tool-supported formal methods in industrial projects, this paper assesses the readiness of formal methods technologies for supporting agile techniques and identified areas in which new research could improve the prospects of synergy between the two approaches in future.

Printed and published by the University of Newcastle upon Tyne,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

LARSEN, P.G., FITZGERALD, J.S., WOLFF, S.

Are Formal Methods Ready for Agility? A Reality Check
[By] P. G. Larsen, J. S. Fitzgerald, S. Wolff
Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2010.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-1218)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-1218

Abstract

The integration of agile software development techniques with formal methods has attracted attention as a research topic. But what exactly is to be gained from attempting to combine two approaches which are seen as orthogonal or even opposing, and to what extent do formal methods already support the principles of agility? Based on the authors' experience in applying lightweight tool-supported formal methods in industrial projects, this paper assesses the readiness of formal methods technologies for supporting agile techniques and identified areas in which new research could improve the prospects of synergy between the two approaches in future.

About the authors

Peter Gorm Larsen is Professor at the Engineering College of Aarhus and Adjunct Professor at the University of Aarhus in Denmark.

John Fitzgerald is Reader in Computing Science at Newcastle University and Deputy Director of the Centre for Software Reliability.

Sune Wolff is an industrial PhD student at with Terma A/S, Denmark.

Suggested keywords

FORMAL METHODS
AGILE METHODS
SOFTWARE ENGINEERING, VDM

Are Formal Methods Ready for Agility?
A Reality Check

Peter Gorm Larsen, Aarhus School of Engineering, Denmark, pgl@iha.dk
John Fitzgerald, Newcastle University, UK, John.Fitzgerald@ncl.ac.uk
Sune Wolff, Terma A/S, Denmark, sw@terma.com

Abstract: The integration of agile software development techniques with formal methods has attracted attention as a research topic. But what exactly is to be gained from attempting to combine two approaches which are seen as orthogonal or even opposing, and to what extent do formal methods already support the principles of agility? Based on the authors' experience in applying lightweight tool-supported formal methods in industrial projects, this paper assesses the readiness of formal methods technologies for supporting agile techniques and identified areas in which new research could improve the prospects of synergy between the two approaches in future.

1 Introduction

Formal methods are a response to the challenge of complexity in computer-based systems, and the defects that arise as a result. They are techniques used to model and analyse complex computer-based systems as mathematical entities. Producing a mathematically rigorous model of a complex system enables developers to verify or refute claims about the putative system at various stages in its development. Formal methods can be applied to models produced at any stage of a system's development, from high-level models of abstract requirements to models of the characteristics of running code, such as memory usage [WLBF09]. The motivations for including formal methods in software development are to minimise defects in the delivered system by identifying them as soon as they arise, and also to provide evidence of the verification of critical system elements. Formal methods are highly diverse, in part because of the variety of domains in which they have been applied. Notable applications have been in the areas of communications, operating system and driver verification, processor design, the power and transportation sectors.

In spite of their successful application in a variety of industry sectors, formal methods have been perceived as expensive, niche technology requiring highly capable engineers [Sai96]. The development of stronger and more automated formal analysis techniques in the last decade has led to renewed interest in the extent to which formal techniques can contribute to evolving software development practices.

The principles of agile software development emerged as a reaction to the perceived failure of more conventional methodologies to cope with the realities of software development in a volatile and competitive market. In contrast with some established development approaches, which had come to be seen as necessary fictions [PC86], agile methods were characterised as incremental (small software releases on a rapid cycle), cooperative (emphasising close communication between customers and developers), straightforward to learn and modify, and adaptive to changes in the requirements or environment [ASRW02]. Four value statements[1] summarise the principles of the approach. Proponents of agile techniques value *Individuals and interactions* over processes and tools, *working software* over comprehensive documentation, *customer collaboration* over contract negotiation and *responding to change* over following a plan.

Agile methods have received considerable attention, but as Turk et al. have pointed out, they do appear to make some underlying assumptions [TFR02]. For example, close customer interaction assumes the ready availability of an authoritative customer; a lower value placed on documentation assumes that documentation and software models are not themselves first-class products of the process; the emphasis on adaptation to changing conditions assumes a level of experience among developers. Since not all projects satisfy these assumptions, it has been suggested that agile approaches are unsuited for distributed development environments, for developments that make extensive use of subcontracting or that require to develop reusable artifacts, that involve large teams or involve the development of critical, large or complex software.

Given the range of both formal and agile methods, their respective pros and cons, can the two work to mutual benefit, or is the underlying principle of rigorous model-based analysis incompatible with the rapid production of code and the favouring of code over documentation? This paper briefly examines some of the existing work on this question (Section 2). A review of the authors' experience in the focused "lightweight" application of the formal method VDM in industry (Section 3) is followed by a review of the four value statements of the agile manifesto (Sections 4). In each case, we ask whether formal methods as they are now are really able to help achieve the value goal, and what research might be needed to bridge the gaps between the two approaches. Section 5 provides concluding remarks.

2 Formal Methods and Agility

The relationship between formal and agile methods has been explored for more than five years. However the issues have recently been brought into focus by Black et al. in their article for IEEE Computer in 2009 [BBB$^+$09]. Some researchers have sought to develop hybrid methods that benefit from both rigour and agility. For example, Ostroff et al. [OMP04] seek to harness Test-Driven Design, a well-known agile technique, with a more formal method of Design by Contract. Niu and Easterbrook [NE05] argue for the use of machine-assisted model checking to address the problem of partial knowledge during iterations of an agile process. López-Nores et al. [Mar06] observe that evolution steps in an agile development typically involve the acquisition of new information about the system to be constructed. This new information may represent a refinement, or may introduce an inconsistency to be resolved through a retrenchment. Solms and Loubser [SL10] describe a service-oriented analysis and design methodology in which requirements are specified as

[1] http://agilemanifesto.org/

formal service contracts, facilitating auto-generation of algorithm-independent tests and documentation. Model structure is formally defined and can be verified through a model validation suite. del Bianco et al. [dBSK10] weave agile and formal elements together into an incremental development process.

Some work seeks to develop more agile formal methods. For example, Eleftherakis and Cowling [EC03] propose XFun, a lightweight formal method based on the use of X-machines and directed at the development of component-based reactive systems. Suhaib et al. [SMSB05] propose an iterative approach to the construction of formal reference models. On each iteration, a model is extended by user stories and subjective to regressive verification. Liu [Liu09] suggests that Structured Object-Oriented Formal Language is a viable tool for use in an agile process.

But is there really a need to combine formal and agile techniques? Certain product types call for an integration of agile and formal techniques in their development, particularly where the need to respond to competitors in a lively market leads to requirements volatility, or where the characteristics of underlying components change become available. The embedded systems market is one field with both of these characteristics[2]. Indeed our current project DESTECS[3] addresses the need to improve the rate at which engineers from different disciplines iterate through early design stages.

For the developer wishing to combine agile and formal practices, it is wrong to think that the term "formal methods" refers to a single development process whether based on refinement or on post-factor verification. Similarly, it is inappropriate to think of agile software development as a set of practices that must be adopted wholesale. Both agile and formal techniques are just that – sets of techniques that should be combined to suit the needs of the product and the character of the development team. The developer is not, however, helped much by the existing literature. It is not always clear whether researchers aim to promote the use of formal methods by showing that they can be added to agile processes, or whether the aim is to produce more agile formal methods to be applied in the usual domains. Perhaps most importantly, there are few empirical results on which to base methods decisions.

3 Experience with VDM

The Vienna Development Method (VDM)[4] is a well established set of techniques for the construction and analysis of system models. VDM models in their most basic form consist of type and value definitions, persistent state variables, pure auxiliary functions and state-modifying operations. Data abstraction is supported by abstract base types such as arbitrary reals and finite but unconstrained collections (sets, sequences and mappings) [FLV08]. Data types can be restricted by arbitrary predicates as invariants. Functional abstraction is supported by implicit pre/post specification of both functions and op-

[2] We note that del Bianco et al. [dBSK10] use an example from this domain.

[3] `www.destecs.org`

[4] `www.vdmportal.org`

erations. The basic VDM language has been extended with facilities to support object-orientation, concurrency, real-time and distribution [FLM+05].

VDM's rich collection of modelling abstractions has emerged because industrial application of the method has emphasised modelling over analysis [FL07]. Although a well worked-out proof theory for VDM exists [BFL+94], more effort has gone into developing tools that are truly "industry strength". The VDMTools system, originated at IFAD A/S in Denmark, and now further developed and maintained by CSK Systems in Japan, supports the construction, static analysis and type checking of models, generation of proof obligations and, above all, highly efficient testing and debugging [FLS08]. These facilities have been essential to VDM's successful industry application in recent years. The same is true of Overture[5] [LBF+10], the new community tools for VDM, in which the development has been in part driven by a desire to interface VDM models of discrete event systems with quite heterogeneous models from other engineering disciplines, such as continuous time models of controlled plant [BLV+10]. Although research on proof support has been ongoing for some years [CJM91, DCN+00, VHL10], the tools have never fully incorporated proof, mainly due to a combination of lack of demand from industry users, and a lack of robustness and ease of use on the part of the available tools.

The majority of the industrial applications of VDM can be characterised as uses of "lightweight" formal methods in the sense of Jackons and Wing [JW96]. Early experience of this was gained in the 1990s when the ConForm project compared the industrial development of a small message processing system using structured methods with a parallel development of the same system using structured methods augmented by VDM, supported by VDMTools [LFB96]. A deliberate requirements change was introduced during the development in order to assess the cost of model maintenance, and records were kept of the queries raised against requirements by the two development teams. The results suggested that the use of a formal modelling language naturally made requirements more volatile in the sense that the detection of ambiguity and incompleteness leads to substantial revision of requirements. To that extent, advocates of formal methods have to embrace the volatility of requirements by the very nature of the process that they advocate.

The use of a formal method to validate and improve requirements has been a common theme in VDM's industrial applications [LDV97, SL99, PT99]. For example, two recent applications in Japanese industry, the TradeOne system and the FeliCa contactless chip firmware [KCN08, KN09], have used formal models primarily to get requirements right. In some situations, such as the FeliCa case, the formal model itself is *not* primarily a form of documentation – it is merely a tool for improving less formally expressed requirements and design documents that are passed on to other developers. In these applications, the trade-off between effort and reward of proof-based analysis has come down against proof, but in favour of a carefully scoped use of the formalism.

Several VDM industry applications have made successful use of high volume testing rather than symbolic analysis as a means of validating models. So for example the FeliCa chip was tested with 7000 black box tests and around 100 million random tests all with both the executable VDM specification as well as with the final implementation in C. The in-

[5] www.overturetool.org

dustrial VDM applications mentioned here can be characterised as processing aspects that are similar to the approach taken in an agile software development. Throughout there has been a continued focus on the needs of the customer and the process of applying VDM has been one for removing more and more uncertainty rather than focusing on obtaining a perfect system by verification. So, although we have never described our approach as "agile" it addresses some agile development principles.

4 The Agile Manifesto Meets Formal Methods

The four value statements of the agile manifesto are supported by 12 principles[6]. In this section we consider each value statement and the principles that relate to it. For each value statement, we review the extent to which formal methods support it today, discuss some deficiencies and suggest future research to remedy these.

4.1 Individuals and Interactions over Processes and Tools

This value statement emphasises the technical competence and collaboration skills of the people working on the project and how they work together. If this is not considered carefully enough, the best tools and processes will be of little use[7]. Two of the 12 principles supporting the agile manifesto are relevant:

- Build projects around motivated individuals. Give them the environment and support their need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

The most important resources available to a project are the people working on it and not the tools or methods they use. But once the right people have been chosen, neither the tools nor the processes should be disregarded.

A criticism of our work on ConForm, as of other demonstrations of the effectiveness of formal modelling, is that we employed clever "pioneering" people to do the formal methods work and they were bound to do a better job than those applying traditional techniques. Our industry colleagues have frequently refuted this claim, arguing that, while highly skilled engineers will perform tasks well given the most elementary of tools and processes, the world is not full of excellent engineers. Most of us benefit from having tools and processes that embody the wisdom gained by previous projects and, dare we say it, more capable colleagues.

To live up to this value statement, it is required that the team members are technically competent in using efficient tools to develop working software for the customer in short

[6] `http://agilemanifesto.org/principles.html`

[7] The aphorism *"A fool with a tool is still just a fool"* is sometimes attributed to Grady Booch.

periods of time. We wonder if all our fellow formalists appreciate the levels of competence among software engineers and hence the work required to deliver methods that can be used in the majority of products. Numerous research projects have demonstrated the potential of new formal methods and tools, such as Rodin[8]. However, the process of bringing them to a state where they are deployable even by R&D engineers requires a major effort, as seen in Deploy[9].

The second principle above refers to “soft” skills and particularly to collaboration and communication. Given engineers who have a willingness and ability to communicate easily among themselves, the tools supporting formal modelling have to make it easy to share models and verification information where appropriate. But how many formal methods tools integrate well with existing development environments, allow models to be updated and easily transmitted, or even exchanged between tools? We feel that collaborative modelling and analysis is not given enough attention in formal methods research.

Most formal methods tools originated in academic research and few have been matured for industrial use. As a result, the focus has been on the functionality afforded by the tools at the expense of the accessibility or user friendliness. If formal methods are to move a step closer to agility, the tool support needs to become easier to pick-up and start using, so attention can be put back on the people actually doing the formal models instead of the tools’ limitations. This argues for increased automation and research effort being put into the interaction between modelling and verification tools and the human.

4.2 Working Software over Comprehensive Documentation

The second value statement of the agile manifesto asserts that, whilst good documentation is a valuable guide to the purpose and structure of a software system, it will never be as important as running software that meets its requirements. The value statement is supported by the following principles:

- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Working software is the primary measure of progress.
- Simplicity – the art of maximizing the amount of work not done – is essential.
- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

We suggest that adherence to these principles is probably easier in a project that is kept in-house, rather than a major distributed software development with extensive subcontracting. For large projects with many person-years of work involved, documentation is indispensable and is often a crucial part of the contract between the developer and customer.

[8] `rodin.cs.ncl.ac.uk`

[9] `www.deploy-project.eu`

For the formal methods practitioner, these must be some of the more difficult agile principles to accept. Much of the work on model-oriented formal methods emphasises the quality and clarity of models and the production of proven refinement steps on the way to well engineered code, where proofs document the correctness of the design steps. An agile process adhering to the principle above is more likely to be based on rapid development cycles in which the quality of the formal models produced takes second place to the rapid generation of code. In turn, this may mean that models will tend to be more concrete and implementation-specific than necessary. There is a risk that the development of system models by accretion on each development iteration will end up with models, and of course proofs, that are much too hard to understand or verify to serve a useful purpose.

Although we have reservations about the wisdom of combining formal modelling with rapid iterative code development, our experience would not suggest it should be written off as impossible. However, current formal modelling technology is not geared to such rapid processes. Indeed our own iterative methodology for developing VDM models of real-time systems defers code production to a late stage (although the models themselves can be executable) [LFW09]. Methods like ours should certainly be updated if support for a more iterative approach is desired.

Automation is once again a key factor: where code can be automatically generated, the model may become the product of interest. Further, the benefits of the model must be seen to justify its production costs, for example by allowing automatic generation of test cases, test oracles or run-time assertions. An agile process that wants to gain the benefits of formal modelling techniques has to be disciplined if the formal model is to remain synchronised to the software produced. It is worth noting that nothing in this approach precludes the use of formal methods of code analysis, for example to assist in identifying potential memory management faults. Here again, the high degree of automation can make it an attractive technology.

In general one can say that this value statement is most applicable with executable models where one then needs to be careful about implementation bias [HJ89, Fuc92, AELL92]. From a purist's perspective this is not a recommended approach. As a consequence, formal refinements from non-executable models cannot be considered agile since potentially many layers of models may be necessary before one would be able to present anything to the customers that they can understand [Mor90, BW98, Abr09].

4.3 Customer Collaboration over Contract Negotiation

The third value statement of the agile manifesto, while recognising the value of a contract as a statement of rights and responsibilities, argues that successful developers work closely with customers in a process of mutual education. Two of the agile principles would appear to relate to this value statement:

- Business people and developers must work together daily throughout the project.
- Agile processes promote sustainable development. The sponsors, developers, and

users should be able to maintain a constant pace indefinitely.

This value statement has been criticised on the grounds that large projects will not be initiated before contracts are drawn up and prices agreed. Changes involve renegotiation, and this is not done lightly.

Close customer collaboration has been a feature of several successful industrial formal methods projects [LFB96, vdBVW99, SL99, KN09]. However, in these projects formal methods have only been used as a high-level executable model of the system and no advanced formal methods techniques such as verification have been applied. However, in order to successfully exploit collaboration between business people and formal methods specialists interpersonal skills for this kind of multi-disciplinary teamwork is essential.

A major weakness of many formal methods tools is the inability to attach a quick prototype GUI to a model giving domain experts the opportunity to interact directly with the model and undertake early validation without requiring familiarity with the modelling notation. Actually, this general idea was implemented over 20 years ago in the EPROS prototyping environment [HI88]. Recent extensions to Overture [LLR$^+$10] allow the designer to create a Java applet or JFrame that can act as a graphical interface of the VDM model. Many formal methods modelling tools could benefit from similar extensions if they aim to support some of these agility principles.

4.4 Responding to Change over Following a Plan

The fourth value statement of the agile manifesto acknowledges that change is a reality of software development. Project plans should be flexible enough to assimilate changes in requirements as well as in the technical and business environment. The following two agile principles are relevant here:

- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Formal methods have no inherent difficulty in coping with requirements change. Indeed formal models may form a better basis for predicting the consequences of radical changes than attempting an analysis of complex code. However, when formal models are adjusted, the associated validation and verification results will need to be redone if full value is to be gained from the model. Thus, the speed of repeat analysis, and the extent of automated tool support are paramount.

As can be seen, applying the agile principles directly is not something that fits every type of project, but this does not mean that some agile practices cannot be applied to large projects. On the contrary, our experience is that agile development is most often used as an element in a hybrid configuration. For example, initial requirements might be analysed in a model-based, documented way, while day-to-day development might employ an agile method like SCRUM [Sch04].

We are not convinced that formal methods and formalists “embrace change”. Black et al. state [BBB$^+$09] that “formal methods cannot guarantee against requirements-creep”. While the concept of requirements creep has negative connotations associated with confused and drifting functionality, being ready to address changing requirements is a necessary part of the agile mindset. Coping adequately with requirements change in a formal setting requires models that are well structured and mechanisms for checking validity that are rapid. Further, to be able to respond to changes in a quick and seamless manner, formal methods practitioners need to accept that models can be less than perfect, especially in the early stages of an iterative development.

4.5 Two remaining principles of technical excellence

Two of the 12 principles do not fit the value statements quite so easily as the others listed above. Both of them deal with aspects of the technical quality of the product:

- Continuous attention to technical excellence and good design enhances agility.
- The best architectures, requirements, and designs emerge from self-organizing teams.

Formal techniques certainly support this focus on quality. Black et al. [BBB$^+$09] also mention the potential synergy between agile and formal methods, opening up the possibility of agile methods being applied to more safety critical domains – something which is currently, to a large extent, off limits to pure agile methods. We conjecture that the second value statement of the original agile manifesto (working software over documentation) has provided a pretext for hack-fixing and ad-hoc programming to call itself an agile process. This has hurt the reputation of agile development, and we would suggest that the addition of a fifth principle favouring quality of the end product over ad-hoc solutions could prevent some of the abuses of agility.

5 Concluding Remarks

The improved analytic power of formal methods tools and greater understanding of the role of rigorous modelling in development processes are gradually improving software practice. However, the claim that formal methods can be part of agile processes should not be made lightly. In this paper, we have examined the value statements and supporting principles of the agile manifesto and have identified areas in which formal methods and tools are hard-pressed to live up to the claim that they can work with agile techniques. In doing so, we have drawn on our own experience of developing and deploying a tool-supported formal method in industrial settings.

Formal methods should not be thought of as development *processes*, but are better seen as collections of techniques that can be deployed as appropriate. For the agile developer, it is not possible to get benefits from formalism unless the formal notation or technique is focused and easy to learn and apply. Luckily, formal modelling and analysis does not *have* to be burdensome. For example, forms of static analysis and automatic verification can be used to ensure that key properties are preserved from one iteration to the next. For this to be efficient, and to fit into the agile mindset, analysis must have automated tool support. Formalists should stop worrying about development processes if they want to support agility. Instead, they should start adjusting their "only perfect is good enough" mindset, and try a more lightweight approach to formal modelling if their goal is to become more agile.

Formalists may need to remember that most engineers, even good ones, have no prior experience of formal methods technology and demand tools that give answers to analyses in seconds. Developers of formal methods must give serious attention to their ease of use if they are to claim any link with agile software development.

Tools must integrate almost seamlessly with existing development environments if they are to support agile processes and there is considerable research required to make this a reality. Progress can certainly be made by improving tools, in particular in combining with GUI building tools and for automation of different forms of analysis. In fact we would like the agile thinking to go beyond the software to encompass collaboration between different engineering disciplines involved in a complex product development, as in the embedded systems domain [BLV$^+$10].

The agile manifesto is not necessarily consistent with a view of formal methods as correct-by-construction development processes. However, there are good reasons for combining agile principles with the formal techniques. Formal methods researchers and tool builders must, however, address some deficiencies if the benefits of such a collaborative approach are to be realised.

Acknowledgements We are grateful to the organisers of the 2010 edition of the workshop on formal methods and agile methods FM+AM'10 for the invitation to give an invited talk, and to our colleagues in the EU FP7 project DESTECS. Fitzgerald's work is also supported by the EU FP7 Integrated Project DEPLOY and by the UK EPSRC platform grant on Trustworthy Ambient Systems. Wolff's work is supported by the Danish Agency for Science Technology and Innovation. Finally we would like to thank Nick Battle for proof reading a draft of this paper.

References

[Abr09] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2009. To appear. See also http://www.event-b.org.

[AELL92] Michael Andersen, René Elmstrøm, Poul Bøgh Lassen, and Peter Gorm Larsen. Making Specifications Executable – Using IPTES Meta-IV. *Microprocessing and Microprogramming*, 35(1-5):521–528, September 1992.

[ASRW02] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. Agile software development methods Review and analysis. Technical Report 478, VTT Technical Research Centre of Finland, 2002.

[BBB$^+$09] Sue Black, Paul P. Boca, Jonathan P. Bowen, Jason Gorman, and Mike Hinchey. Formal Versus Agile: Survival of the Fittest? *IEEE Computer*, 42(9):37–45, September 2009.

[BFL$^+$94] Juan Bicarregui, John Fitzgerald, Peter Lindsay, Richard Moore, and Brian Ritchie. *Proof in VDM: A Practitioner's Guide*. FACIT. Springer-Verlag, 1994. ISBN 3-540-19813-X.

[BLV$^+$10] J. F. Broenink, P. G. Larsen, M. Verhoef, C. Kleijn, D. Jovanovic, K. Pierce, and Wouters F. Design Support and Tooling for Dependable Embedded Control Software. In *Proceedings of Serene 2010 International Workshop on Software Engineering for Resilient Systems*. ACM, April 2010.

[BW98] Ralph-Johan Back and Joakim Wright. *Refinement Calculus: A Systematic Introduction*. Springer, 1998.

[CJM91] Peter Lindsay Cliff Jones, Kevin Jones and Richard Moore, editors. *mural: A Formal Development Support System*. Springer-Verlag, 1991. ISBN 3-540-19651-X.

[dBSK10] Vieri del Bianco, Dragan Stosic, and Joe Kiniry. Agile Formality: A "Mole" of Software Engineering Practices. In Stefan Gruner and Bernhard Rumpe, editors, *Proc. AM+FM'2010*, Lecture Notes in Informatics. Gesellschaft für Informatik, 2010. To appear.

[DCN$^+$00] Louise A. Dennis, Graham Collins, Michael Norrish, Richard Boulton, Konrad Slind, Graham Robinson, Mike Gordon, and Tom Melham. The PROSPER Toolkit. In *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Berlin, Germany, March/April 2000. Springer-Verlag, Lecture Notes in Computer Science volume 1785.

[EC03] George Eleftherakis and Anthony J. Cowling. An Agile Formal Development Methodology. In *Proc. 1st. South-East European Workshop on Formal Methods, SEEFM'03*, pages 36–47. Springer-Verlag, 2003.

[FL07] J. S. Fitzgerald and P. G. Larsen. Triumphs and Challenges for the Industrial Application of Model-Oriented Formal Methods. In T. Margaria, A. Philippou, and B. Steffen, editors, *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.

[FLM$^+$05] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. *Validated Designs for Object–oriented Systems*. Springer, New York, 2005.

[FLS08] John Fitzgerald, Peter Gorm Larsen, and Shin Sahara. VDMTools: Advances in Support for Formal Modeling in VDM. *ACM Sigplan Notices*, 43(2):3–11, February 2008.

[FLV08] J. S. Fitzgerald, P. G. Larsen, and M. Verhoef. Vienna Development Method. *Wiley Encyclopedia of Computer Science and Engineering*, 2008. edited by Benjamin Wah, John Wiley & Sons, Inc.

[Fuc92] Norbert E. Fuchs. Specifications are (Preferably) Executable. *Software Engineering Journal*, pages 323–334, September 1992.

[HI88] Sharam Hekmatpour and Darrel Ince. *Software Prototyping, Formal Methods and VDM*. Addison-Wesley, 1988.

[HJ89] I.J. Hayes and C.B. Jones. Specifications are not (Necessarily) Executable. *Software Engineering Journal*, pages 330–338, November 1989.

[JW96] Daniel Jackson and Jeanette Wing. Lightweight Formal Methods. *IEEE Computer*, 29(4):22–23, April 1996.

[KCN08] Taro Kurita, Miki Chiba, and Yasumasa Nakatsugawa. Application of a Formal Specification Language in the Development of the "Mobile FeliCa" IC Chip Firmware for Embedding in Mobile Phone. In J. Cuellar, T. Maibaum, and K. Sere, editors, *FM 2008: Formal Methods*, Lecture Notes in Computer Science, pages 425–429. Springer-Verlag, May 2008.

[KN09] T. Kurita and Y. Nakatsugawa. The Application of VDM++ to the Development of Firmware for a Smart Card IC Chip. *Intl. Journal of Software and Informatics*, 3(2-3), October 2009.

[LBF$^+$10] Peter Gorm Larsen, Nick Battle, Miguel Ferreira, John Fitzgerald, Kenneth Lausdahl, and Marcel Verhoef. The Overture Initiative – Integrating Tools for VDM. *ACM Software Engineering Notes*, 35(1), January 2010.

[LDV97] Peter Gorm Larsen Lionel Devauchelle and Henrik Voss. PICGAL: Lessons Learnt from a Practical Use of Formal Specification to Develop a High Reliability Software. In *DASIA'97*. ESA, May 1997.

[LFB96] Peter Gorm Larsen, John Fitzgerald, and Tom Brookes. Applying Formal Specification in Industry. *IEEE Software*, 13(3):48–56, May 1996.

[LFW09] Peter Gorm Larsen, John Fitzgerald, and Sune Wolff. Methods for the Development of Distributed Real-Time Systems using VDM. *International Journal of Software and Informatics*, 3(2-3), October 2009.

[Liu09] Shaoying Liu. An approach to applying SOFL for agile process and its application in developing a test support tool. *Innovations in Systems and Software Engineering*, 6:137–143, December 2009.

[LLR$^+$10] Peter Gorm Larsen, Kenneth Lausdahl, Augusto Ribeiro, Sune Wolff, and Nick Battle. Overture VDM-10 Tool Support: User Guide. Technical Report TR-2010-02, The Overture Initiative, www.overturetool.org, May 2010.

[Mar06] Martín López-Nores and José J. Pazos-Arias and Jorge García-Duque and others. Bringing the Agile Philosophy to Formal Specification Settings. *International Journal of Software Engineering and Knowledge Engineering*, 16(6):951–986, 2006.

[Mor90] Carroll Morgan. *Programming from Specifications*. Prentice-Hall, London, UK, 1990.

[NE05] Nan Niu and Steve Easterbrook. On the Use of Model Checking in Verification of Evolving Agile Software Frameworks: An Exploratory Case Study. In *3rd International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS 2005)*, pages 115–117, Miami, Florida, USA, May 2005. INSTICC Press.

[OMP04] Jonathan S. Ostroff, David Makalsky, and Richard F. Paige. Agile Specification-Driven Development. In J. Eckstein and H. Baumeister, editors, *XP 2004*, volume 3092 of *Lecture Notes in Computer Science*, pages 104–112. Springer, 2004.

[PC86] D.L. Parnas and P.C. Clements. A Rational Design Process: How and Why to Fake It. *IEEE Transactions on Software Engineering*, 12(2), February 1986.

[PT99] Armand Puccetti and Jean Yves Tixadou. Application of VDM-SL to the Development of the SPOT4 Programming Messages Generator. In John Fitzgerald and Peter Gorm Larsen, editors, *VDM in Practice*, pages 127–137, September 1999.

[Sai96] Hossein Saiedian. An Invitation to Formal Methods. *IEEE Computer*, 29(4):16–30, April 1996. Roundtable with contributions from experts.

[Sch04] Ken Schwaber. *Agile Project Management with Scrum*. Prentice Hall, 2004. ISBN: 073561993X.

[SL99] Paul R. Smith and Peter Gorm Larsen. Applications of VDM in Banknote Processing. In John S. Fitzgerald and Peter Gorm Larsen, editors, *VDM in Practice: Proc. First VDM Workshop 1999*, September 1999. Available at www.vdmportal.org.

[SL10] Fritz Solms and Dawid Loubser. URDAD as a semi-formal approach to analysis and design. *Innovations in Systems and Software Engineering*, 6:155–162, 2010.

[SMSB05] Syed M. Suhaib, Deepak A. Mathaikutty, Sandeep K. Shukla, and David Berner. XFM: An Incremental Methodology for Developing Formal Models. *ACM Transactions on Design Automation of Electronic Systems*, 10(4):589–609, October 2005.

[TFR02] Dan Turk, Robert France, and Bernhard Rumpe. Limitations of Agile Software Processes. In *Proceedings of the 3rd international Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2002)*, pages 43–46, May 2002.

[vdBVW99] Manuel van den Berg, Marcel Verhoef, and Mark Wigmans. Formal Specification of an Auctioning System Using VDM++ and UML – an Industrial Usage Report. In John Fitzgerald and Peter Gorm Larsen, editors, *VDM in Practice*, pages 85–93, September 1999.

[VHL10] Sander Vermolen, Jozef Hooman, and Peter Gorm Larsen. Automating Consistency Proofs of VDM++ Models using HOL. In *Proceedings of the 25th Symposium On Applied Computing (SAC 2010)*, Sierre, Switzerland, March 2010. ACM.

[WLBF09] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal Methods: Practice and Experience. *ACM Computing Surveys*, 41(4):1–36, October 2009.