

Scrum Goes Formal: Agile Methods for Safety-Critical Systems

Sune Wolff

Terma A/S

Terma Defense and Security

Hovmarken 4, 8520 Lystrup, Denmark

Email: sw@terma.com

Abstract—Formal methods have had a relative low penetration in industry but have the potential for much wider use. The use of agile methods has been highly limited in development of safety-critical systems due to the lack of formal evaluation techniques and rigorous planning. A combination of formal methods and agile development processes can potentially widen the use of formal methods in industry as well as enabling the use of agile methods in development of safety-critical systems. This paper describes a way to add the use of formal methods to the agile development process Scrum. Experiences from using a variant of the strategy in an industrial case are summarised.

Keywords-Scrum; formal methods; combined method

I. INTRODUCTION

Formal methods are seen by many as the holy grail that ultimately will allow us to write error-free code which complies completely to its specifications, in turn ensuring that the software is the most reliable component of any system [1]. Over the years formal methods have been applied in industry [2] especially for development of mission or safety critical systems, in order to ensure correctness of the system and to prove that safety constraints are met, but still has a long way to go before becoming the standard for software development. Many of the industrial cases have been in larger European projects (RODIN¹ and DEPLOY² to name a few) where academic research groups have been in charge of the formal modelling. To move the usage of formal methods beyond these “protected” environments they need to be less intimidating to companies by, for example, presenting the use of formal methods in well known development processes widely used in industry.

While the formal methods community will claim that the application of formal methods, keeps projects within budget and time, one of the reasons why formal methods have not been more widely accepted into industry is the steep learning curve. The risk of adding cost and not delivering systems on time keeps many companies from applying formal methods unless strictly necessary. The added workload in the analysis and design phases often outweighs the potential benefits of more seamless integration and less errors in the final product.

Formal methods are perceived as being mystical and only used by a selected few computer scientists or mathematicians, and that the learning curve is too steep to be useful by more “ordinary” people for developing non-critical systems. In order to change these prejudices, and eventually ensuring a broader industrial usage of formal methods, a wider range of engineers must be “exposed” to formal methods to discover the benefits first-hand.

In a highly competitive industry companies strive to deliver faster better and cheaper software solutions. A broad spectrum of agile methods, ranging from individual methods like eXtreme Programming and test driven development to management-oriented process frameworks like Scrum [3] have emerged as a means to achieve this. Many of these methods have seen wide acceptance in the software industry as a means of ensuring that systems are created in time and within budget. The customer is heavily included in the development process, in order to ensure that the correct functionality (and only that) is developed. So agile methods can be seen as a way of adjusting the quality of the system to satisfy the user needs while staying within budget and delivering on time.

A major drawback of agile methods is the exclusive use of informal evaluation techniques which often are insufficient for establishing the quality of safety-critical systems [4]. As a result, agile methods are rarely used to develop these types of systems where more formal and rigorous development and testing techniques are often required.

Combining formal methods and agile development can potentially bring the best of both worlds as has been hinted in the past [5]. In previous research [6], we have reviewed the agile manifesto and the principles of agile development in order to evaluate to which extend formal and agile methods can be combined, and to determine the main obstacles in doing so. Here we show that light-weight formal analysis techniques can be used in agile software development and potentially can help ensuring that systems can consistently be created on time, within budget and with formally verified functionality.

In this work the use of formal specification techniques as part of the agile process Scrum is presented. The goal is to describe a concrete example of how formal specifications can be added to a widely used agile management process.

¹<http://rodin.cs.ncl.ac.uk/>

²<http://www.deploy-project.eu/>

Our hope is that this will help break down the barriers between formal and conventional software development by introducing the use of lightweight formal analysis techniques in a well known agile project management process like Scrum.

In the next section related work is described followed by a short introduction to formal methods. In Section IV an overview of the agile management method Scrum is given. The addition of formal specifications techniques to Scrum is described in Section V and the method is evaluated and discussed in Section VI. Finally some concluding remarks are given and future work is described.

II. RELATED WORK

More than 20 years ago, Richard Kemmerer investigated how to integrate formal methods into conventional development processes [7]. The work presented in this paper is somewhat inspired by the work of Kemmerer, but where he focused on adding formality to the different stages of a traditional and methodologically suspect waterfall model, the focus of the work presented here is on integrating the use of formal specification techniques to the agile project management method Scrum.

Broy et al. proposed the addition of formal refinement steps to the traditional V-model [8]. The V-model is mainly used on a systems engineering level to describe major milestones etc, whereas the work presented here focuses on describing a development methodology which can be used on a daily basis by the engineers developing the system.

In the work of Eleftherakis et al. [9], an agile formal development methodology called *XFun* is proposed, combining the unified process with the formal notation X-machine. For *XFun* to be most effective the system under development should be a component-based reactive system, where no such restrictions are set on the type of systems in the methodology presented here. Their methodology is tied to the use of X-machine as a formal specification language whereas the work presented here looks more broadly at the use of formal specifications in an agile setting without limiting the use to a single formal language.

Ostroff et al. describes an agile approach to specification-driven development which combines the agile approach of test-driven development with the formal approach design-by-contract [10]. The proposed development process requires the use of a contract-aware programming language, whereas any programming language can be used in Scrum in general and hence also in the method presented here.

The agile formal methodology eXtreme Formal Modeling described by Suhaib et al. [11] uses an extreme programming approach for capturing the specifications of a system under development from a natural language specification into a formal model. The main contribution of Suhaib et al. is developing formal method using agile methods, whereas the

methodology described in this paper adds the use of formal specifications to an agile process.

Liu et al. have described the integration of formal methods into industry development standards with the SOFL language and methodology [12]. Instead of a pure mathematical notation, they use *condition data flow diagrams* as a graphical notation of the high level architecture of the system. Through several semi-formal refinement steps the final implementation is developed. More recently Liu has applied the SOFL method using a more agile approach [13] where an incremental implementation is used. In the work by Liu the specific formal method SOFL is used in a more general agile setting, whereas the work presented here describes the use of formal specifications in general in the specific agile method Scrum.

III. FORMAL METHODS

Formal methods are a collection of mathematically-based techniques used in development of computer systems. Using a formal specification language, a system can be described precisely with regards to functionality, concurrency, completeness, correctness, etc. This means that the properties of a system can be analysed without having to actually run the system [14]. Many formal specification languages have an executable subset that can be used to specify executable models of the system. The model developer can then exercise the system model in order to investigate runtime properties of the system [15].

In the work presented here it is assumed that executable specifications are used to enable the use of light-weight analysis techniques like scenario-based tests in order to validate system functionality and properties. Refinement of specifications and full formal verification of system properties can be a very tedious process without the right tool support and automatic proof support. In order to fit the use of formal specification techniques into an agile process like Scrum it is important that the formal specification tasks of an iteration are not too long. This is the reason for the focus on executable formal specifications which can, for example, be validated using a test oracle.

A lot of different types of formal methods exist — way too many to mention them all here, but two examples are given below:

VDM is short for Vienna Development Method, and was developed in IBMs Vienna laboratories in the 1970s [16], and hence is one of the oldest formal methods around. The VDM Specification Language (VDM-SL) enables the modeller to describe system properties using abstract data types like sets, sequences and mappings. The object-oriented extension VDM++ adds the possibility of modelling class hierarchies and concurrency, while the real-time extension VDM-RT enables the specifier to model distributed systems, where specific parts of the model are deployed on different processing units.

B-method is a formal method based on abstract machine notation, developed by Jean-Raymond Abrial [17]. The B notation uses first-order logic to describe an initial abstract goal of the system and, through multiple refinement steps, adds further detail to the model, moving closer to the final implementation. In each refinement step details are added to the model describing data structures or algorithms used to obtain the goal of the model. Event-B [18] is a related formal method which uses set theory to model reactive systems, and uses mathematical proof to verify consistency between the different refinement steps.

It is important to note that any formal notation can be used in the methodology described in Section V. It is even possible to use different types formal specifications for the different sprints, if this brings any benefits to the project. The choice of formal specification language can be based on either prior experience of the team or that some notation may be more suitable to model the content of the individual sprints. Finally, the available tool support can be a decisive factor as to which formal specification language is chosen — stable tools that support the engineers in validating the models are needed to enable the use of formal specifications in short iterative development cycles.

IV. AGILE DEVELOPMENT - SCRUM

Agile software development processes describe iterative and incremental software development. Everything from the initial requirements to the finished product evolves from close customer collaboration and there is a high focus on creating several sub-deliveries of working software that the customer can try out. The term agile development was first introduced by the Agile alliance who described the agile manifesto³ and the 12 principles of agile development⁴. This section gives an overview of Scrum introducing the different roles of the project participants as well as the different activities and the artifacts produced throughout the process.

Scrum has been used to develop complex software systems since the early 1990s and is one of the most widely used agile project management methods in industry [19]. Like all agile methods Scrum focuses on short feedback loops in system development, user requirements and the development process itself. By involving the user continually in the iterative process it is ensured that the final product actually fulfill the user needs. This requires great adaptive capabilities of both process and development team in order to support change in user needs.

A. Roles

In a Scrum setup, project participants take on one of three roles:

Product owner: The person who represents the customer's interest, and determines the goal of each iteration

³Can be found at: <http://agilemanifesto.org/>

⁴Can be found at: <http://agilemanifesto.org/principles.html>

called a sprint. One of the most important tasks of the product owner is to prioritise the different tasks ensuring that the most important functionality is implemented first.

ScrumMaster: It is the task of the ScrumMaster to facilitate the Scrum process and protect the agile principles. He also needs to protect the team by removing any impediments encountered, ensuring that they are not distracted from the task at hand.

Team: Five to nine people working together to create a functional system which satisfies the user needs. Usually, the team consists of people with a broad set of competences in order to ensure that the team is self-organised and contained.

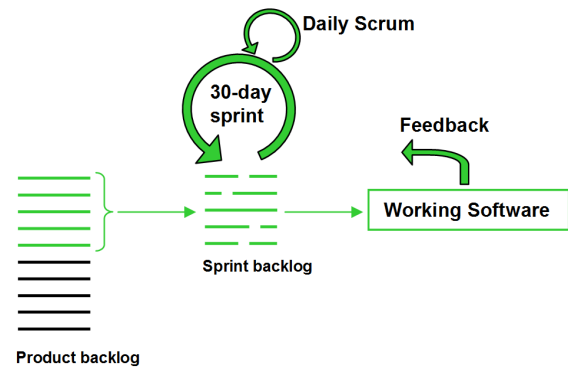


Figure 1. Overview of the Scrum process with a 30-day sprint duration

B. Activities and Artifacts

Scrum consists of several activities which describe the workflow and artifacts produced during the Scrum development:

Product vision: This is the initial idea phase, where the vision of the product is defined.

Product backlog: A prioritised list of functionality or artifacts that are needed in the final product. It is good practice to describe the items in the product backlog as user stories which add value to the user – it is then the job of the team to develop a product which fulfills these needs and supports the stories. In general, two types of tasks are defined: *implementation* and *investigation* tasks. Implementation tasks are completely defined with very little or no uncertainty. Investigation tasks often involves the use of new technology, and are used in order to investigate the feasibility of said technology and to better estimate the implementation tasks which will be derived from the investigation.

Sprint: This is a development iteration which is time-boxed to last between 2 and 6 weeks – a sprint length of one month is commonly used. An overview of a sprint is shown in Figure 1, where a duration of one month is shown.

Sprint planning: The first day of every sprint is dedicated to determine what must be done in the sprint.

Sprint backlog: The stories from the product backlog which the team has committed to implement in the sprint planning meeting.

Daily Scrum meeting: A daily meeting called a “daily scrum” with a short duration of approximately 15 minutes is held every morning. Each team member describes what was accomplished since last meeting, what will be done until the next meeting and describes any impediments discovered or encountered.

Sprint review: At the end of each sprint, a review meeting is held, where the team demonstrates what has been accomplished to the product owner.

Sprint retrospective: Following the sprint review an additional meeting is held, where focus is on people, relationships, processes and tools and not on the items produced in the last sprint.

V. FORMAL SPECIFICATIONS IN A SCRUM SETTING

The second formal development process described in Kemmerer’s work has two teams working in parallel: one using conventional development methods and the other using formal methods. At predefined milestones, the two teams synchronise their work to ensure that they work in the same direction. This approach does not go hand-in-hand with the ideals of Scrum where the team should be self-organised and self-contained having all the necessary expertise needed to solve the tasks at hand within the team itself.

Instead, the process proposed here is inspired by how formal methods are commonly used in industry –namely in the early concept and design phases– and then adjusted to a more agile iterative process like Scrum.

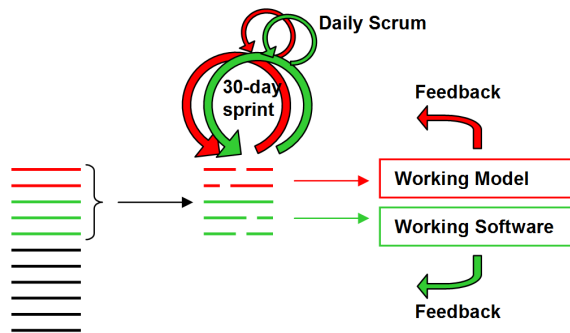


Figure 2. Integration of formal specifications into the Scrum process

The sprint tasks include both conventional software implementation tasks as well as formal specification investigation tasks of uncertain nature which cannot be directly implemented without further investigation. The focus of the formal specification tasks is the modelling and validation of high risk system properties, before they are implemented in a future sprint. The conventional software tasks are well defined tasks with little or no uncertainty and hence can be implemented without further investigation.

It is important that the formal specification helps the programmers in understanding the requirements and design of the system, and that the formalists can communicate their findings to the programmers. This communication process can be eased by using a graphical representation of the formal specification as it is done in SOFL, using a familiar syntax like VDM, or by using an executable specification using simulation results to aid in the communication.

A. Roles

Roles identical to the conventional Scrum roles are used and the responsibilities of the product owner and the Scrum-Master are unchanged.

Team: The team has several tasks: to create a high level executable formal specification of the system; to validate key concepts of the system design; to investigate the use of new technology and finally the team must implement the functionality needed to solve the tasks defined in the sprint. This means that the team needs to include engineers with expertise in formal specifications as well as conventional software developers in order to solve both tasks in a satisfactory manner.

B. Activities and Artifacts

The main changes in the method described here compared to conventional Scrum are in the individual sprints. In order to ensure agile and dynamic development no formal specification will be done up front, but will be done in the individual sprints. Hence, the product vision, product backlog and daily Scrum meeting are unchanged.

Sprint: It is important for the formal team members to have a good understanding of what the conventional software team members are working on, and vice versa, to support better knowledge transfer within the group. This ensures that more engineers get a good understanding of formal modelling techniques and opens up the option of using a formal engineer to solve some of the conventional software tasks (or the other way around) in order to balance the tasks within the team during a sprint.

Sprint planning: The sprint planning is quite similar to the conventional Scrum version, with the product owner and team deciding which stories from the product backlog go into the sprint. The most critical user requirements are chosen to be formally specified first while less critical parts of the system are implemented in parallel by the conventional software developers. This ensures that a formal specification is made of all parts of the system which has a high degree of uncertainty without hindering the agile process.

Sprint backlog: The sprint backlog will be divided into two parts: one for the formal specification investigation tasks and one for the conventional software implementation tasks.

Sprint review: After the sprint is finished, the team will present and demonstrate their work to the product owner.

Apart from customer and product owner feedback, the focus on this sprint review should be on potential input to future conventional software tasks based on the formal modelling done on the investigation tasks. It is important to decide if enough knowledge has been gained from the formal specification to fully clarify the investigation tasks so these can be turned into implementation tasks. It is also possible that new investigation tasks will be derived from the model simulation results.

Sprint retrospective: Following the sprint review, the team will have a sprint retrospective meeting discussing what went well and what could be done better process-wise in each of the parts of the sprint. It is important to assess both general interpersonal issues but also issues tied to either the formal or software tasks individually. Since knowledge transfer is a key aspect of this project setup, it is important to discuss the degree to which this is working as well.

VI. EVALUATION AND DISCUSSION

The reason for wanting to combine formal and agile methods were twofold:

- Ease the use of formal specifications in industrial projects by presenting the use of lightweight formal analysis techniques in a well known agile setting; and
- Increase the likelihood of using agile methods in the development of mission- or safety-critical systems, by introducing the validation of key concepts using formal specifications.

It is the hope to remove some of the mysticism that surrounds formal methods by having engineers trained in formal specification techniques working alongside conventional software engineers in a single team. During the daily Scrum meetings knowledge will be transferred implicitly, ensuring that the use of formal development methods is better understood by the software engineers. In time, the software engineers of the team can help out on the investigation tasks getting an even better understanding of formal validation techniques.

One of the key skills for software engineers is abstraction [20] — the ability to specify in detail the system elements needed in order to verify a certain property, while abstracting away from all unnecessary details. This is a very complex challenge often mastered by engineers with formal specification experience. Conventional software developers can have a tendency to focus on low-level implementation details of the system and not having the more abstract overview of the entire system. Working alongside formalists (who utilise more abstract thinking) will in time improve the abstraction skills of the programmers which could potentially lead to better system quality.

Balancing the number of formal and software tasks is a potential issue. For example, if too few formal modelling tasks are included in a sprint, some of the formalists will be forced to work on conventional software tasks in order

to finish the complete sprint. This will lead to lowered overall performance of the team, since a traditional software engineer would probably be more efficient at solving these tasks. This is another trade-off for the knowledge transfer which happens within a diversely composed group.

One of the main benefits of adding the use of formal specifications is to give valuable input to the implementation phase, or to enable reuse of test cases, which in turn will lessen the overall workload. Tool support for some formal methods includes code generation from the formal specification to a target programming language. It is unusual for the generated code to be seamlessly deployed without further alterations; however it can at least be used as a skeleton for further development.

As has been mentioned, the method described here is not tied to the use of a single, specific formal specification language. The author has most experience using VDM so this will be used as an example.

The C-like syntax of VDM will ensure that engineers with embedded programming experience will find it less complex because of the familiar syntax and constructs — this will help further demystify formal methods which is one of the key goals of the work presented here. VDM supports a modular modelling approach, where only part of a system is modelled and analysed while the rest of the system is abstracted away. This fits perfectly with the “investigation” formal modelling tasks described in our method, where only key elements of the entire system are specified formally.

To analyse the use of an arbitrary alternative formal specification language Event-B is chosen. In Event-B models are developed in a very structured way: initially a very simple abstract model is created and through several refinement steps the final concrete model is developed. These refinement steps are checked by a translation tool ensuring that each new model is a formal refinement of its predecessor. Once the final concrete model is created it needs to be translated into a program and compiled into executable code for the target platform — these final steps are the weak points of this approach.

Event-B uses a very monolithic approach to system modelling which makes it hard to only refine parts of the system in order to investigate this part in more detail. The use of Event-B fits best into the initial phases of system development where a formal team can make a refined model of the entire system. This model is then passed on to the software developers who implement the system based on the model. For the method described in this paper VDM is a better fit: the formalists can model only the critical parts of the system in parallel with the software developers implementing the rest of the system.

Scrum is a project management method and, as such, does not describe how the individual sprints should be structured. Agile development methods like eXtreme Programming (XP) and Test Driven Development (TDD) describe just this.

XP uses similar techniques as Scrum: iterative development, daily stand-up meetings, measuring project velocity and close customer collaboration. In addition pair programming is advocated as is the use of frequent and automated integration tests. Pair programming is a good technique for sharing the knowledge of formal methods amongst the team members and to ensure consistency between the formal specification and the final implementation.

In TDD developers are required to write automated unit tests defining the code requirements before the code is implemented. In VDM it is possible to define implicit functions by means of a signature and pre- and post-conditions. By using VDM and TDD as the daily development methods, the formal modelling tasks can generate unit tests which can be reused in the conventional implementation tasks, and hence saving time. This combination of VDM and TDD could be ideal for the method described in this paper, but more work is required before any conclusions are drawn.

VII. SUMMARY AND FUTURE WORK

A concrete approach to using formal specifications in the popular agile project management method Scrum has been presented. A single team, consisting of formalists as well as conventional software engineers, creates a formal model in addition to the software implementation in each sprint.

The work presented here is purely theoretical, but is based on industry experience using Scrum for developing large mission critical systems for self-defense systems deployed in fighter aircrafts. The author also has experience using formal models to precisely describe functionality of military self-defense systems.

A lightweight version of the process has already been tried out in practice [21], where the author was the only engineer with formal modelling experience in a multi-team setup with more than 25 engineers implementing the system. The project involved many upgrades to an existing system, whereas the formal modelling focused on an upgrade to the interpretation of messages sent between two subsystems using an existing communication protocol. Executable models of both subsystems were specified in VDM++ and a large test suite was used in order to exercise the use of the communication protocol. The result of these tests were used to show the implications of the upgrade to the customer, and to ensure that no undesired side effects were introduced.

The outcome of the project was very positive — the customer was especially satisfied with the early and rapid feedback provided by the formal model created. The software developers gained some insight into the benefits of using formal specifications, but since the author was not fully integrated into the daily scrum meetings the knowledge transfer was limited.

The work presented in this paper was conducted with the desire of creating a more concrete approach how to integrate the use of formal specifications to an agile process.

It is the hope for the future to test a larger-scale version of the proposed strategy with a greater focus on the formal specification tasks than what has already been tested. Scrum is well suited for larger-scale projects where ScrumMasters from all teams involved synchronise the work done in so-called Scrum-of-Scrums. With the addition of formal specifications it is required that all ScrumMasters have a good understanding of the formal models being created to be able to synchronise this part of the sprint. This stresses the importance of using a formal specification technique that is easily readable by people without prior formal training — like the graphical representation used in SOFL or the familiar C-like syntax for VDM to name a few examples.

An emerging technique that crosses the borders between the formal and conventional software culture is *design by contract* languages like JML [22] and Spec# [23]. These languages enable formal interface definitions through: pre- and post conditions; invariants; and errors and exceptions. This interface definition is added as basic annotations of the individual methods and modules. This is another way of integrating the use of formal specification techniques in industry while still maintaining industrially preferred development methods and work flow.

We hope that the method presented here will provide valuable input to companies looking into adding the use of formal specification techniques to their agile software development processes. In addition to methodologies like the ones described here, several things are needed in order to ensure higher acceptance of formal methods into industry. One such thing is better tool support, automating many of the tedious tasks of especially formal verification. When formal verification needs to be done by hand, the duration of the individual iterative cycles gets too long to fit into the more agile development approaches many companies use. In addition, it is important to educate not only talented and efficient software engineers but also to ensure that they understand formal specification concepts and have the ability to work on different levels of abstraction. This will ensure better interaction between the formal and the conventional software world, and increase the likelihood that formal methods will gain acceptance in industry.

Formal methods are still struggling to gain wide acceptance and usage in conventional software development, but have seen use in verifying key properties of safety-critical systems. In a short time agile methods have gained good acceptance in industry, but are not often used to develop safety-critical systems due to lack of rigorous validation techniques. Combining the two approaches to software development will hopefully make the benefits of formal specification techniques more visible as well as enable the use of agile development techniques of critical systems. By introducing the use of formal specifications in an industrially widely used process like Scrum will make formal specifications much more accessible and hence increase the industrial pen-

etration. This will broaden the focus of agile methods which traditionally are mostly concerned with keeping projects on time and within budget, to focus more on system quality — this will take us one step closer to the holy grail of having error-free code developed on schedule and within budget.

VIII. ACKNOWLEDGMENTS

The author would like to thank Christian Schultz Ottosen, Peter Gorm Larsen, Joey W. Coleman, Claus Ballegård Nielsen, Augusto Riberio, Rasmus Lauritsen and Henning Staun Nielsen for valuable input on earlier versions of this paper. The work presented here is partly funded by “The Danish Agency for Science, Technology and Innovation”. Terma A/S is thanked for supporting the project. Finally, the author would like to thank the anonymous reviewers for their valuable feedback.

REFERENCES

- [1] T. Hoare, “The Ideal of Program Correctness: *Third Computer Journal Lecture*,” *Computer Journal*, vol. 50, no. 3, pp. 254–260, 2007.
- [2] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, “Formal Methods: Practice and Experience,” *ACM Computing Surveys*, vol. 41, no. 4, pp. 1–36, October 2009.
- [3] K. Schwaber, *Agile Project Management with Scrum*. Prentice Hall, 2004, ISBN: 073561993X.
- [4] D. Turk, R. France, and B. Rumpe, “Limitations of Agile Software Processes,” in *Proceedings of the 3rd international Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2002)*, May 2002, pp. 43–46.
- [5] S. Black, P. P. Boca, J. P. Bowen, J. Gorman, and M. Hinchey, “Formal versus agile: Survival of the fittest?” *IEEE Computer*, vol. 42, no. 9, pp. 37–45, September 2009.
- [6] P. G. Larsen, J. Fitzgerald, and S. Wolff, “Are formal methods ready for agility? a reality check,” in *2nd International Workshop on Formal Methods and Agile Methods*, S. Gruner and B. Rumpe, Eds. Lecture Notes in Informatics, September 2010, pp. 13–25, iSSN 1617-5468.
- [7] R. A. Kemmerer, “Integrating Formal Methods into the Development Process,” *IEEE Software*, pp. 37–50, September 1990.
- [8] M. Broy and O. Slotosch, “Enriching the software development process by formal methods,” in *Proceedings of the International Workshop on Current Trends in Applied Formal Method: Applied Formal Methods*. London, UK: Springer-Verlag, 1998, pp. 44–61.
- [9] G. Eleftherakis and A. J. Cowling, “An agile formal development methodology,” in *Proc. 1st. South-East European Workshop on Formal Methods, SEEFM’03*. Springer-Verlag, 2003, pp. 36–47.
- [10] J. S. Ostroff, D. Makalsky, and R. F. Paige, “Agile specification-driven development,” in *XP 2004*, ser. Lecture Notes in Computer Science, J. Eckstein and H. Baumeister, Eds., vol. 3092. Springer, 2004, pp. 104–112.
- [11] S. M. Suhaib, D. A. Mathaikutty, S. K. Shukla, and D. Berner, “XFM: An Incremental Methodology for Developing Formal Models,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 4, pp. 589–609, October 2005.
- [12] S. Liu and Y. Sun, “Structured Methodology + Object-Oriented Methodology + Formal Methods: Methodology of SOFL,” in *Proceedings of First IEEE International Conference on Engineering of Complex Computer Systems*. Ft. Lauderdale, Florida, U.S.A.: IEEE Press, November 1995, pp. 137–144.
- [13] S. Liu, “An approach to applying soft for agile process and its application in developing a test support tool,” *Innovations in Systems and Software Engineering*, vol. 6, pp. 137–143, December 2009.
- [14] I. Hayes and C. Jones, “Specifications are not (Necessarily) Executable,” *Software Engineering Journal*, pp. 330–338, November 1989. [Online]. Available: <http://www.cs.man.ac.uk/csonly/cstechrep/Abstracts/UMCS-89-12-1.html>
- [15] N. E. Fuchs, “Specifications Are (Preferably) Executable,” Institut für Informatik, Universität Zürich, Tech. Rep. 91.10, Juli 1991.
- [16] D. Bjørner and C. Jones, Eds., *The Vienna Development Method: The Meta-Language*, ser. Lecture Notes in Computer Science. Springer-Verlag, 1978, vol. 61.
- [17] J.-R. Abrial, *The B Book – Assigning Programs to Meanings*. Cambridge University Press, August 1996.
- [18] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, “Rodin: an open toolset for modelling and reasoning in Event-B,” *STTT*, vol. 12, no. 6, pp. 447–466, 2010.
- [19] E. S. F. Cardozo, J. B. F. A. Neto, A. Barza, A. C. C. Franca, and F. Q. B. da Silva, “SCRUM and Productivity in Software Projects: A Systematic Literature Review,” in *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. Keele University, UK, 2010.
- [20] J. Kramer, “Is Abstraction the Key to Computing?” *Communications of the ACM*, vol. 50, no. 4, pp. 37–42, 2007.
- [21] S. Wolff, “Using Executable VDM++ Models in an Industrial Application - Self-defence System for Fighter Aircraft,” Aarhus University School of Engineering, Tech. Rep. ECE-TR-1, March 2012.
- [22] G. T. Leavens, A. L. Baker, and C. Ruby, “Preliminary design of jml: a behavioral interface specification language for java,” *SIGSOFT Softw. Eng. Notes*, vol. 31, pp. 1–38, May 2006.
- [23] M. Barnett, M. Fähndrich, K. R. M. Leino, P. Müller, W. Schulte, and H. Venter, “Specification and Verification: The Spec# Experience,” *Communications of the ACM*, vol. 54, no. 6, pp. 81–91, June 2011.