DevOps Meets Formal Modelling in High-Criticality Complex Systems

Marta Olszewska
Åbo Akademi University
Joukahaisenkatu 3-5A
20520 Turku, Finland
+358442806858
mplaska@abo.fi

Marina Waldén
Åbo Akademi University
Joukahaisenkatu 3-5A
20520 Turku, Finland
+35822154675
mwalden@abo.fi

ABSTRACT
Quality is the cornerstone of high criticality systems, since in case of failure not only major financial losses are at stake, but also human lives. Formal methods that support model based-development are one of the methodologies used to achieve correct-by-construction systems. However, these are often heavy-weight and need a dedicated development process. In our work we combine formal and agile software engineering approaches. In particular, we use Event-B and Scrum to assure the quality and more rapid and flexible development. Since we identified that there are more prerequisites for a successful IT project, we use DevOps to embrace the development, quality assurance and IT operations. In this paper we show how formal modelling can function within DevOps and thus promote various dimensions of quality and continuous delivery.

Categories and Subject Descriptors
D.2.1 [Software Engineering]: Requirements/Specifications — Elicitation methods, Methodologies; D.2.4 [Software Engineering]: Verification methods — Formal methods; D.2.10 [Software Engineering]: Design — Methodologies.

General Terms
Management, Design, Reliability, Verification.

Keywords
Agile, Scrum, formal modelling, Event-B, DevOps.

1. INTRODUCTION
Development of complex systems of high-criticality requires not only vast domain knowledge, but also use of appropriate methods and tools that would ensure high quality of the produced systems. Therefore, formal methods are often employed to build software models in the early phases of the software development and guarantee that the system is correct by construction. However, nowadays there are other requirements added to the development, i.e. reducing friction in the development time, delivering artefacts faster, improving communication within development team and with stakeholders. Formal methods alone provide firm software engineering approaches; however, they need support on behalf of the development process and resource management.

Formal methods are mature enough and ready for being integrated in the development with other methods [1]. Agile methods, on the other hand, are the most appropriate means for engineering such a merge [2]. However, they both need some more support for the interdependencies within the development, including issues regarding a project set-up (tools, methods and decisions on collaboration), overcoming the learning curve of formal methods, identifying bottlenecks and handling standardization issues, etc. The merge of formal methods and agile approaches is meant, among others, to speed up the delivery of artefacts while ensuring their quality.

In our previous work we have deepened the understanding of agile concepts set in the context of safety-critical development by (i) providing evidence of such development through related work and (ii) relating agile principles, practices and values to formal environment in order to create a synergy between these two (FormAgi framework) [3]. We chose Event-B as a formal method to be used within the agile development process.

Here we continue our work with the FormAgi framework by setting up a Scrum-based process for development of systems of high criticality with the use of Event-B. We present the merge of formal modelling with Scrum in DevOps perspective. We show how formal modelling can function within DevOps and contribute to its quality assurance and continuous delivery. We also point out how DevOps can support certain aspects of formal development.

This paper is structured as follows. Section 2 provides description of approaches for achieving shorter release cycles leading towards continuous delivery and integration. Section 3 describes methods supporting correctness and quality of development. In section 4 we present how we adapted Scrum to a formal modelling context. Section 5 depicts formal modelling with Scrum in perspective of DevOps. We conclude our paper in section 6 and provide directions for future work.

2. STRIVING FOR RESPONSIVENESS

2.1 Iterative and Flexible Development
Agile software development philosophy [4] introduced in 2001 occurred to be successful due to the capabilities of substantially increasing the project success rate, while reducing the development time and cost. Although criticised for disregarding existing practices of traditional software engineering, it is still popular, mainly because of its responsiveness and ability to meet stakeholders' needs within the given time. Agile methods are known for facilitating the collaboration within the development

QUDOS'15, September 1, 2015, Bergamo, Italy
ACM. 978-1-4503-3817-2/15/09...
http://dx.doi.org/10.1145/2804371.2804373

team and boosting internal morale. Moreover, the communication between the team and stakeholders is strengthened, thus leading to higher customer satisfaction [5].

There is a plethora of agile software development methods available, all of which encourage adaptive planning, evolutionary development, early delivery, continuous improvement, and promote rapid and flexible response to change [3]. The methods themselves need to be carefully chosen and tailored to the needs of the case development in order to be successful.

Even though agile methods are meant to improve product quality by e.g. leading to less defects in the development, there are still ongoing discussions whether this is really the case [6]. Some projects, for instance the ones of high-criticality, need to adhere to certain standards and follow well-defined practices [7] [8].

Boehm et al. [9] stated that neither the agile nor the traditional disciplined approach can alone be the optimal solution. Although they seem contradicting [10], a software project needs both agility and discipline [9] [11]. Therefore, a comprehensive approach is essential to embrace all the necessary methods and raise the understanding what is needed in the development – from the perspective of tools, methodology and people.

2.2 A Synergy Demand – DevOps

A rapid creation of software products and services, as well as improvement in operations performance, is at the core of today's IT. DevOps [12] answers these needs by putting special emphasis on collaboration, integration, communication and automation. There is no clear definition of DevOps yet [13], however, it is described as a software development method that combines quality assurance mechanisms with IT operations within software engineering practices (see Figure 1). It is based on the ideas from agile development movements and supports rapid development and deployment cycles.

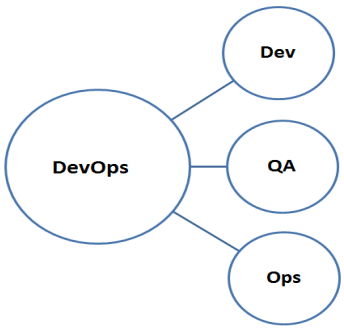

Figure 1. DevOps concept

Since systems become more complex, we need to increase our *comprehension* of what needs to be mapped from the empirical world to the model or code, as well as how to do it effectively. Moreover, we should improve our *communication and collaboration*, so that we can (i) raise the understanding of the system to be developed, (ii) foster awareness between people involved in the development and operations, (iii) progress with our development, and (iv) build our infrastructure. We require tool support for our work, to *automate* some of our activities. And finally we need to understand that our models, code, teams and organization are constantly in flux, thus we need to continuously address all the above mentioned issues.

3. CORRECTNESS AND QUALITY

There are various methods available for assuring quality in systems of high-criticality. Application of formal methods [14] brings high quality to critical systems, in particular when certain system behaviour and properties need to be guaranteed. However, some experience and mathematical background is needed in order to properly utilise the existing modelling solutions.

3.1 Iterative Formal Development

Refinement [15] [16] [17] [18] is a stepwise formal development method, which allows the system to be created iteratively following certain rules called *refinement rules* (also referred to as proof obligations) [19] [20]. *Stepwise refinement* is a top-down approach [16], which aids handling all the implementation matters and *complexity* by splitting up the problems to be specified and gradually introducing details of the system to the specification. In the refinement process, an abstract specification is created from requirements. It is then transformed into a more concrete and deterministic system that preserves the functionality of its specification in consecutive refinement steps. For each refinement step an invariant is given that states the properties of the system. Hence, also the invariant is created in an iterative manner. The refinement process is presented in Figure 2.

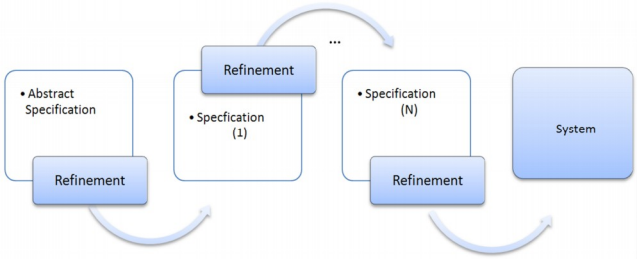

Figure 2. Refinement process

The correctness of each step, resulting in a system that is correct by construction [15], is ensured by mathematically proving that the abstract model is consistent and feasible. It involves proving that each refined model preserves its invariant. Even if proving is tool-supported, there are still some proofs that cannot be automatically discharged, but will require human interaction. The amount of involvement needed heavily depends on the chosen modelling strategy and is a subject of our current work.

The complexity of proofs depends not only on the problem and the complexity of the system to be modelled, but also on the refinement strategy utilised and e.g. on the decomposition mechanisms [21]. Therefore, assisting the modelling activity by facilitating the development process would help dealing with the complexity issues.

3.2 Modelling in Event-B

Event-B [22] is a formal method and modelling language for stepwise system-level modelling and analysis, based on the Action Systems formalism [18] [23] [24]. It is derived from the B-Method [25], with which it has several commonalities, e.g. set theory and the refinement idea. Event-B is dedicated to model complete systems, including hardware, software and environment [26] and has gained appreciation in industrial settings [27].

An Event-B specification uses a pseudo-programming notation – Abstract Machine Notation – and consists of a dynamic and a static part, *machine* and *context* respectively. The formal development starts from specifying an abstract machine from a set of requirements and then refining it in a number of steps (see Figure 2). It identifies the machine being refined, so that the refinement chain and the modelling process can be tracked and

controlled. The static part of the specification is also extended with respect to the development of the machine.

Event-B utilises refinement to represent systems at different abstraction levels, which enables us to gradually introduce details to the constructed system and to represent new levels of a system with more functionality. Mathematical proofs are used to verify consistency between the refinement levels. Event-B provides rigour to the specification and design phases of the development of critical systems. It is effectively supported via the *Rodin platform* [28], an Eclipse based tool, which is an open source "rich client platform" that is extendable with plug-ins, e.g. ones providing simulation and animation, as well as visualisations of the model. Finally, code generation from models to various programming languages is supported.

3.3 Standardisation

Systems of high criticality typically need to be qualified or certified in order to be deployed. Formal methods are recommended practices when licensing critical software, just to mention the IEC 61508 standard ("Functional Safety of Electrical, Electronic and Programmable Electronic Safety-related Systems") [29] or ISO 26262 standard for automotive domain ("Road vehicles — Functional safety") [30]. The application of formal methods is additionally followed by measures and techniques that are obligatory for the product to be certified.

Although agile methods are not explicitly considered in standards (e.g. IEC 61508 standard), their use in safety-critical development is already present and needs to be transferred to standards (see e.g. the goals of the European Project RECOMP [31]).

3.4 Need for Speed

Developing a system has never been an easy task, but now, when the complexity of the surrounding world is increasing and the requirements given by the stakeholders are expanding, it has become even more intricate [32]. There is a constant struggle between development cost, quality and time, where the choice is made on the expense of one of these characteristics (note that we do not consider cost in this paper).

Achieving high quality seems like a prerequisite for stakeholder satisfaction. However, the time pressure and need for fast delivery of a product can lead to "good enough" solutions and acceptable quality. This attitude cannot suffice for the systems of high criticality, where human lives or major financial losses can be at stake. Yet, development of this kind of systems is also required to be responsive to change, actionable, providing faster delivery, as well as enabling communication and collaboration. This as a consequence inevitably leads to the amendments in the development process, as well as organisational changes.

The formal modelling process itself is thought to be heavy-weight and thus may seem far from the *need for speed* that we experience nowadays. Therefore, a suitable development process, along with some principles and practices tailored for the specifics of the formal modelling environment and development domain is necessary to facilitate the change and aid in adapting to the challenges set by the contemporary IT world.

4. TOWARDS AGILE

A combination of (semi) formal and agile approaches has been discussed in a number of publications [9] [11] [7] [8] [33], where capabilities of traditional software development process models or development methods were merged with agile methods, principles and practices.

We observed that having a well-described and flexible development process is not sufficient for formal methods to be successful. Additional mechanisms are needed, which are specific to the development setting. In this section we first shortly describe the *FormAgi* [3] framework, which is the basis of our work. Next, we depict Scrum as the agile software development method of our choice and then adapt it to the formal modelling setting. Finally, we describe the merge of formal modelling of critical systems with the Scrum development process in the context of DevOps.

4.1 FormAgi Framework

In our previous work we explored the values, principles and practices of agile development methods and placed them in the context of formal, refinement-based developments. We analysed several of the agile methods with respect to their feasibility in development of critical systems. [3]

We provided a mapping between the characteristics of these two, which established FormAgi [3], a high-level framework consisting of (i) guidelines on what concerns should be tackled before committing to a certain agile method and (ii) pointers in which aspects an agile method can be a facilitator in the formal development. Additionally, we mentioned Event-B as a formal method of our choice to be utilised within the agile process. Although Event-B is thought as being far from lightweight approach, we argue that by conducting the development in small refinement steps [22], and by decomposing the models [21] e.g. using abstractions [40], it is a good candidate to be applied in a rapid manner. Our current work on an example is meant to investigate this claim.

4.2 Scrum

In this work we chose to use Scrum [34], an iterative agile development framework, which relies on frequent releases and short development cycles, as well as supports process improvement. Since some of the characteristics conceptually overlap with the ones of Event-B, e.g. iterations and refinement steps, we consider the integration of the two to be seamless.

One of the reasons for which we chose Scrum was the clear definition of time frames for iterations (organisation of sprints) and the set of meetings to be held during the development process [35]. An overview of a sprint within Scrum is shown in Figure 3.

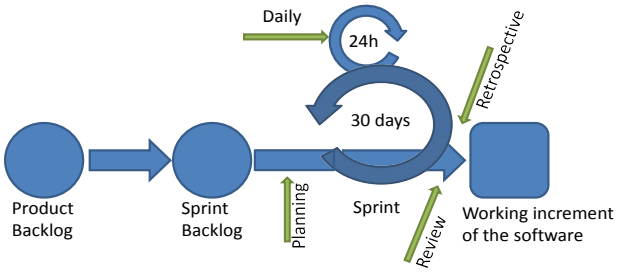

Figure 3. The overview of a Scrum sprint

During each sprint the development team takes a set of features from the product backlog, which is a set of high-level requirements. Then the stakeholder informs the developers of the features that should be completed in this iteration. From this subset the team selects the features that are realistic to be implemented. Next, when the goals for the sprint are determined, they are shifted to the sprint backlog, which remains fixed for the time of the sprint. The sprint lasts for a specific amount of time (2-4 weeks) and at the end it produces a potentially shippable product increment, which is presented by the team to the

stakeholder. Any of the unimplemented features are returned to the product backlog.

Communication, one of the cornerstones of agile approaches, is also important within Scrum. Various meetings are held throughout the development: before the sprint starts (Planning), during the sprint (Daily Scrum), after the sprint (Review and Retrospective). Retrospective is a process improvement-oriented meeting, while the other meetings concern the development itself.

In Scrum there is a strong involvement of the representative of the end user (or the stakeholder), so that at the end of each sprint the directions for further development can be indicated. The issue of how much functionality will be implemented during each iteration is controlled and decided by the development team. The contents of a sprint do not change during its duration. Thus, the moments at which functionality changes can occur are limited and well-defined.

The goal of this development methodology is to increase the relative effectiveness of development practices for improvement purposes and at the same time delivering a framework for development of complex products [36].

4.3 Adapting Scrum

One of the purposes of agile methods is to be tailored to fit the characteristics of the environment they are utilized in. These adjustments aid in getting the most out of the used methodologies and tools. In case of developing systems in Event-B we aim at smoothening the development by supporting shorter iterations, as well as facilitating the intra-project communication. Moreover, we consider supporting the communication between the team and the stakeholders as crucial to obtain high quality software.

Although formal modelling differs from coding in programming languages, creation of *artefacts* (be it a subset of requirements, specification, model on specific abstraction level or implementation of a feature) remains as the main goal.

We adapt Scrum to fit the specifics of Event-B development, which is depicted in Figure 4. In practice, formal modelling involves not only transforming requirements into models, which are then proven to be correct, but also elicitation and modelling of requirements themselves. For this reason we use the term *item*.

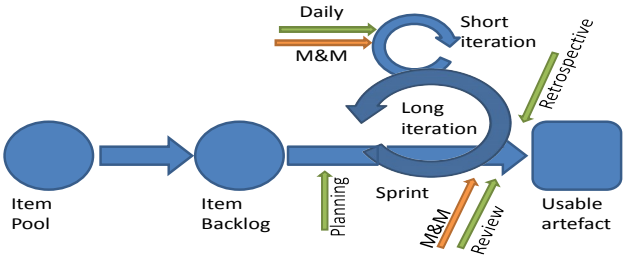

Figure 4. Scrum Sprint adapted to formal modelling (within the FormAgi framework)

The requirements in the *item pool* are considered as a set. The item pool acts as product backlog. However it contains not only high-level requirements, but also lower-level requirements, safety cases, environment descriptions. Furthermore, the *item backlog* is a subset of the item pool and consists of requirements chosen for this sprint, but not prioritised. Since prioritization may take more time than what is scheduled for regular Scrum process, we believe it should be done within sprints. The reasoning is twofold: (i) we do not want to rush decisions which would lead to a complex and hard to prove model and (ii) the work on the requirements and their structuring with respect to the modelling strategy will pay off later, when the model needs to be extended. Thus, a sprint includes modelling of the requirements, as well as developing and proving a model. For the verification purposes, model animation and simulation can also be a part of the sprint. It is well supported via plug-ins to the Rodin platform.

The duration of *long* and *short iterations* should be decided before the development starts. There is a risk that some requirement or property is too complex to be processed within a short iteration. In this case it should be discussed during the short *daily* meeting so that the team is informed. The sprint *review* resembles the discussions in a regular sprint. However, some issues like model walkthroughs, or demonstrating the results to stakeholder as model simulation or animation should also be included at this stage. The *retrospective* is meant to reflect upon the sprint and highlight the areas of the sprint for future improvement.

It can be noted that the relation between a refinement step and a development process iteration is certainly not a one-to-one mapping. There can be several refinement steps in one iteration. Moreover, a refinement step might be too large for an iteration, so that the problem to be modelled needs to be decomposed into sub-problems and only as such placed into the item backlog.

Finally, although not present in original Scrum, a feedback system is included in the sprint via the *Monitoring and Metrics* mechanisms *(M&M)*, which is to raise understanding on what is being done (short iterations), as well as to facilitate the process improvement and provide evidence on the development (long iterations). We are aware that metrics and measurements within agile developments are sometimes considered as harmful to the team morale and against agile philosophy. However, we use them for informational purposes rather than "plunger of blame".

5. EMBRACING IT ALL

Nowadays it is not only the software system development itself that matters, regardless if it is the work on requirements, specifying the system, modelling its properties and behaviour, or proving it afterwards. It needs certain methods and tools, practices and principles, as well as some external mechanisms for making the development work smoothly and progress towards creation of a quality product. In Figure 5 we present our view on formal modelling set in the DevOps context. Note that the formal development method is only a part of the whole development.

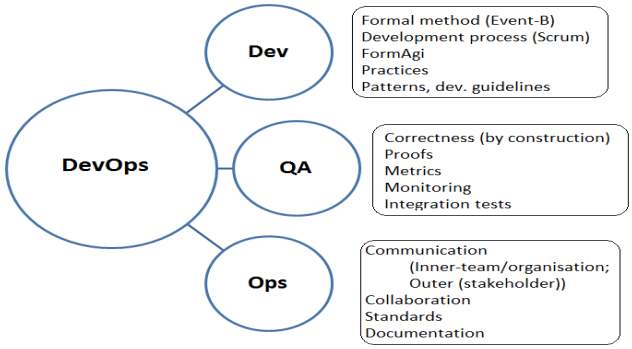

Figure 5. Formal Modelling in DevOps context

Apart from the formal method (Event-B), and the development process (Scrum) supported by the FormAgi framework, there are other necessary elements that need to be taken into consideration in order to achieve a quality product in the domain of highly-critical systems. DevOps encapsulates what we believe is vital for the development to be successful, from technical, social and quality perspectives.

While dependability, herein reliability, is ensured by Event-B, Scrum increases the speed of delivery of artefacts and facilitates their deployment. Iterations supported by the development process and model refinement promote continuous delivery. Once the model is created and proved throughout the development, all the modelling decisions are documented in a stepwise manner in form of refinement steps [37]. They can serve as artefacts for standardisation purposes. Moreover, runtime information in the Ops phase can be tracked all the way to requirements due to the way the system is developed. In case of a failure at runtime, it can be traced to a flaw in the modelling process, the specification or even the requirements. Modifications of a system by either adding new or altering existing components or requirements will cause the whole system to be remodeled and (partially) proved again.

We can facilitate the modelling and help continuous delivery of artefacts by application of patterns, be it the lower level ones [38], or the ones related to the modelling strategy [39]. We are currently working on a library of components to support the reuse and modularity of development.

We noticed that there is quite a natural transition from push to pull flow with respect to modelling requirements and building a model. We focus only on necessary properties related to a modelled artefact in a certain refinement step. Thus refinement helps us in (i) concentrating on what matters the most at a particular point in the development and (ii) matching the level of abstraction with the current development stage. As a consequence, we provide better control over the model complexity, and by that contribute to higher quality. Moreover, by employing requirement prioritisation and providing strategy in modelling, e.g. by decomposition and abstraction mechanisms [40], we avoid waste. So far we have identified two cases of generating it that can be avoided: (i) when insufficient time is spent on requirements modelling, it can lead to spending excessive time on modelling and then cause cumbersome proving; (ii) when detailing the model too early, it increases the complexity of the model and its related proofs.

Identifying bottlenecks and prioritising the improvement areas, which are identified as important in DevOps, are supported by the retrospectives in Scrum. We additionally propose monitoring and metrics to be used as a feedback system for improvement. So far we provide metrics for measurement of the complexity of the Event-B models [41], as well as their UML representations [42].

Visualisations of models are available via Rodin tool plugins for animation and simulation. This enables us to show the results of the modelling to team members and stakeholders after a sprint, without the need to provide executable code. The platform offers code generation to various programming languages, with different level of technical detail, once the model is at a lower level of abstraction.

Although tests are not necessary for the critical development modelled with formal methods (proofs are ensuring correctness), integration tests should be used in order to ensure quality in case the critical part is integrated with a non-critical one [31].

Post-mortems are one of the mechanisms that should be incorporated in the formal modelling and should be done with two groups (team and stakeholders). However, one would need an additional "check" mechanism that could be incorporated in the development process once a bigger milestone is achieved. Particularly, a more in-time feedback could be provided if this check is integrated with some other part of the system.

So far we have shown how formal modelling can work within DevOps, but we also see how DevOps (and agile methods) can contribute to formal modelling. What is often missing in the formal process is the emphasis on communication between the team members (and also with the stakeholders). After receiving requirements, be it safety cases or functional properties, the experts are singlehandedly working out *a good way* of modelling them, but usually without a common strategy for the development. We observed that intra-team communication largely enhances the modelling capabilities. We also observed that stand-ups are a good approach of pinpointing difficulties with the modelling or proving. Thus the collaboration means not only knowledge sharing, but also raising understanding and awareness in the project. It also leads to the concept of "reusable team", where the expertise of every group member is known and can be utilised whenever needed.

6. CONCLUSIONS AND FUTURE WORK

As observed by many DevOps followers, having no clear definition of this concept has its advantages. Firstly, it raises discussions, and secondly, it opens new possibilities of combining methods and tools within one high level development method, which in consequence enables to smoothen out the gap between development and operations. We believe that regardless of the application domain, it is important to utilise existing methodologies and tools through a well-structured merge (DevOps umbrella), so that not only the development is supported, but also the people, processes and the artefacts inevitably bound to it (operations and quality management tasks).

In our work quality in formal modelling within DevOps is achieved by building a correct-by-construction software system, incorporating monitoring and metrics mechanisms to the development process, providing tool support for modelling and proving, as well as facilitating communication and collaboration across organisation and with stakeholders.

We are currently planning to perform a case study, where we will employ all the methodologies and practices mentioned in this paper (see Figure 5). It will be a development of a high-criticality system, but executed in an academic environment. Moreover, we would like to investigate further what can be interpreted as a waste in formal modelling and how to minimise it. Since the information about how to model in Event-B is spread over a number of publications, we are currently working on providing more comprehensive guidelines, suitable also for inexperienced Event-B users. Finally, we believe that deeper research on how refinement steps relate to Scrum sprints has a potential of shortening the deployment time for artefacts.

7. ACKNOWLEDGMENTS

This work was carried out within the project ADVICeS, funded by Academy of Finland, grant No. 266373. We would like to thank PhD Mikołaj Olszewski, Vaadin Expert and a Scrum enthusiast, for constructive discussions on agile topics.

8. REFERENCES

[1] Larsen, Peter Gorm, Fitzgerald, John S., and Wolff, Sune. Are Formal Methods Ready for Agility? A Reality Check. In *Second International Workshop on Formal Methods and Agile Methods* (Pisa 2010), Springer.

[2] Paige, Richard F. and Brooke, Phillip J. Agile Formal Method Engineering. In *Integrated Formal Methods* (Eindhoven 2005), Springer.

[3] Olszewska, Marta and Waldén, Marina. *FormAgi – A Concept for More Flexible Formal Developments*. Åbo Akademi University, Turku, 2014.

[4] *Manifesto for Agile Software Development*. 2001.

[5] Olszewski, Mikołaj. *Scaling Up Stepwise Feature Introduction to Construction of Large Software Systems*. TUCS Dissertation Series, Turku, 2013.

[6] Olszewska, Marta, Jeanette, Heidenberg, Weijola, Max, Mikkonen, Kirsi, and Porres, Ivan. *Did It Actually Go This Well? A Large-Scale Case Study on an Agile Tranformation.* TUCS, Turku, 2014.

[7] Glazer, Hillel, Dalton, Jeff, Anderson, David, Konrad, Mike, and Shrum, Sandy. *CMMI or Agile: Why Not Embrace Both!* Carnegie Mellon University (CMU) and Software Engineering Institute (SEI), 2008.

[8] Fritzsche, M. and P., Keil. Agile methods and CMMI: compatibility or conflict? *e-Informatica, Software Engineering Journal*, 1, 1 (2007), 9-26.

[9] Boehm, Barry and Turner, Richard. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, 2003.

[10] Turner, R. Agile development: good process ot bad attitude? In Oivo, Markku and Komi-Sirviö, Seija, eds., *Product Focused Software Process Improvement*. Springer, Rovaniemi, 2002.

[11] Boehm, Barry and Turner, Richard. Observations on balancing discipline and agility. (Salt Lake City 2003), IEEE Computer Society.

[12] Loukides, Mike. *What is DevOps?* O'Reilly Media, Sebastopol, 2012.

[13] Loukides, Mike. What is DevOps (yet again)? *Radar*, radar.oreilly.com (Feb. 2015).

[14] Butler, Ricky W. What is Formal Methods? In *NASA LaRC Formal Methods Program*. 2001.

[15] Dijkstra, Edsger W. A Constructive Approach to the Problem of Program Correctness. *BIT Nmerical Mathematics*, 8(3) (1968), 174-186.

[16] Wirth, Niklaus. Program Development by Stepwise Refinement. *Communications of the ACM*, 14(4) (1971), 221-227.

[17] Back, Ralph-Johan. *On the Correctness of Refinement Steps in Program Development; PhD thesis*. University of Helsinki, 1978.

[18] Back, Ralph-Johan. Refinement Calculus, Part II: Parallel and reactive programs. Stepwise Refinement of Distributed Systems. In de Bakker, J. W. et al., eds., *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*. Springer-Verlag, 1990.

[19] Metayer, Christophe, Abrial, Jean-Raymond, and Voisin, Laure. *Event-B Language, RODIN Deliverable 3.2 (D7)*. 2005.

[20] Waldén, Marina and Sere, Kaisa. Reasoning about Action Systems using the B-Method. *Formal Methods in System Design*, 13 (1998), 5-35.

[21] Yeganefard, Sanaz and Butler, Michael. Problem Decomposition and Sub-Model Reconciliation of Control Systems in Event-B. In *IEEE International Workshop on Formal Methods Integration* (Turku 2013).

[22] Abrial, Jean-Raymond. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[23] Back, Ralph-Johan and Kurki-Suonio, R. Decentralization of process nets with centralized control. *2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (1983), 131-142.

[24] Back, Ralph-Johan and Sere, Kaisa. From modular systems to action systems. *Software - Concepts and Tools*, 17 (1996), 26-39.

[25] Abrial, Jean-Raymond. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[26] Event-B, http://www.event-b.org/index.html. *Home of Event-B and the Rodin Platform*. 2008.

[27] Romanovsky, Alexander and Thomas, Martyn. *Industrial Deployment of System Engineering Methods*. Springer Heidelberg, 2013.

[28] RODIN and http://www.event-b.org/platform.html. *RODIN - Rigorous Open Development Environment for Complex Systems*. 2006.

[29] Commission(IEC), International Electrotechnical. *IEC 61508 1-7 - Functional Safety of Electrical, Electronic, Programmable Electronic Safety-related Systems*. IEC, 2010.

[30] ISO/FDIS. *26262 Road Vehicles - Functional Safety*. ISO/FDIS, 2011.

[31] RECOMP. RECOMP Project - Reduced Certification Costs Using Trusted Multi-core Platforms. In *https://artemis-ia.eu/project/21-recomp.html*.

[32] Olszewska, Marta. *On the Impact of Rigorous Approaches on the Quality of Development*. Turku Centre for Computer Science, 2011.

[33] Mandal, Ardhendu and Pal, S. C. Achieving agility through BRIDGE process model: an approach to integrate the agile and disciplined software development. *Innovations on System Software Engineering*, 11, 1 (March 2015), 1-7.

[34] Schwaber, Ken. *Agile Project Management with Scrum*. Microsoft Press, 2004.

[35] Shore, James and Warden, Shane. *The Art of Agile Development*. O'Reilly Media, Sebastopol, 2008.

[36] Schwaber, Ken and Sutherland, Jeff. *Scrum. The Official Guide.* Scrum.org, 2010.

[37] Pląska, Marta, Waldén, Marina, and Snook, Colin. Documenting the Progress of the System Development. (Heidelberg 2009), Springer-Verlag.

[38] Snook, Colin and Waldén, Marina. Refinement of Statemachines using Event-B semantics. In *Formal Specification and Development in B* (Besançon 2007), Springer.

[39] Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilic, D., and Latvala, T. Supporting Reuse in Event B Development: Modularisation Approach. In *Abstract State Machines, Alloy, B and Z: Second International Conference (ABZ)* (2010), Springer.

[40] Parsa, Masoumeh, Snook, Colin, Olszewska, Marta, and Waldén, Marina. Parallel Development of Event-B Systems with Agile Methods. In Mousavi, Mohammad Reza and Taha, Walid, eds., *Proceedings of 26th Nordic Workshop on Programming Theory, NWPT '14*. Halmstad University, Halmstad, 2014.

[41] Olszewska (Pląska), Marta and Sere, Kaisa. Specification Metrics for Event-B Developments. In *13th International Conference on Quality Engineering in Software Technology (CONQUEST 2010)* (Dresden 2010).

[42] Olszewska, Marta and Waldén, Marina. Measuring the Progress of a System Development. In Petre, Luigia et al., eds., *Dependability and Computer Engineering: Concepts for Software-Intensive Systems*. IGI Publishing House, Hershey, 2011.