Marta Olszewska | Marina Waldén

FormAgi – A Concept for More Flexible Formal Developments

Turku Centre *for* Computer Science

TUCS Technical Report

No 1124, November 2014

FormAgi – A Concept for More Flexible Formal Developments

Marta Olszewska
Åbo Akademi University, Department of Information Technology
Marina Waldén
Åbo Akademi University, Department of Information Technology

TUCS Technical Report
No 1124, November 2014

Abstract

It is increasingly important that the software system that is developed is not only of good quality, but also delivered within the anticipated cost and deadlines. Formal methods ensure that the system is correct and created according to specification, and thus fulfils the requirements and preserves certain quality attributes. Since the methods are based on establishing a heavy design upfront, the development schedule can be quite well estimated. However in reality, when the customer is changing the requirements during the development, neither the schedule nor the cost can remain constant. Therefore, a more flexible development process is needed in order to facilitate a well-established development method and, thus, enable efficient tackling of the volatility of requirements.

In this report we propose an adaptable development framework called FormAgi, which merges the strengths of formal methods and agile development methods. We base our study on concepts that are vital in formal development and combine them with ideas from well-known agile methods. Furthermore, we indicate how the framework can be utilised in the Event-B environment. Finally, we identify the challenges and opportunities for this kind of fusion.

Keywords: Formal methods, agile development process, flexibility, adaptable development framework, Event-B

TUCS Laboratory
RITES – Resilient IT Infrastructures
Distributed Systems Laboratory
Integrated Design of Quality Systems group

1. Introduction

When developing a product there are plenty of factors to be considered and most of them are related to one (or more) of the categories cost, quality and time. Although projects always strive for achieving all three [1], as they are interrelated, it is nearly impossible to optimise all of them – one will always suffer. Cost is thought to be more likely to be fixed (or estimated with some margin) at the beginning of the project, thus "only" quality of the end product (and used development process) and time to produce and deploy the product is what is left to be tackled separately.

Quality is an important matter in software development. Nevertheless, it can be understood differently by people with diverse backgrounds e.g. software engineers, business managers, and researchers. Software quality, according to the definition by IEEE Standard 1061, "is the degree in which software possesses a desired combination of quality attributes".

There are several decisive factors that impact the quality of a software product and the related development process, which in consequence determine the success of the whole development. Firstly, it is the broad knowledge of the domain and developers' commitment to a development. Secondly, it is the choice of an appropriate development methodology to be applied. Finally, it is the support for suitable technologies, processes and tools that bring the reinforcements for the system development. All of the aforementioned factors are, however, impacted by the real-life dynamics, such as for example requirement change, sociological circumstances, economic situation on the market and the continuous race with competitors. Therefore, the contemporary software development has to deal with a variety of non-technical issues. The technical issues mainly regard requirements, which are increasingly volatile.

Efficiency and effectiveness of the development method, which is necessary in the perspective of the listed factors, are two of the most important aspects in existing software developments, particularly with respect to the concept of the "need for speed" in the ICT field. Naturally, end-users, customers and stakeholders should be satisfied with the software they obtain, just as the developers should be satisfied with the software they created. Providing a supportive development environment and a well-defined development method brings benefits in obtaining feasible development

processes, which in turn results in quality products that meet the needs of the business customer and the stakeholder, and satisfies the developers.

Formal methods and Agility – why together?

Developing systems in safety-critical domains differs from developing, e.g. games or office tools, although in both cases the size and complexity of systems may be significant. Quality, as a broad aspect, is especially important with regard to the former, since failure or malfunctioning of these systems can cause some hazardous effects. For instance, it can endanger human lives by causing death or considerable injury, cause loss or serious damage to equipment, or lead to environmental harm. Furthermore, severe financial losses can be involved. Therefore, creating high quality software systems, which can be depended upon, is of essence.

Development of safety-critical systems requires special treatment, e.g. methods and tools that would ensure that the system functions correctly, according to requirements. Traditional development methods cannot assure achieving high enough quality of a critical system. However, this can be, and often is, provided by rigorous methods, i.e. formal and semi-formal methods by eliminating ambiguity and thus making the specification or a model of a system more precise. Application of formal methods brings high quality to critical systems, but at the same time it may cause complexity issues. These issues can be managed by suitable choice and application of modelling strategies, i.e. patterns, decomposition and abstraction mechanisms. However, some experience and mathematical background is needed in order to properly utilise the existing modelling solutions.

Using formal methods brings high level of rigour to the development, but at the same time limits its flexibility. Due to the needs of standardisation or certification, criticality of software adds conditions to the product life-cycle and development, i.e. tools and methods, design, documentation etc. Therefore, there is a need to complement the development and activities involved and enable shorter iterations and faster delivery, better complexity control and customer-driven approach.

While academia puts emphasis on establishing formal methods that would serve their purpose of achieving high quality software, and at the same time be accepted by industry, the development process that would facilitate projects is pushed far behind the main scope. Industry, on the other hand, would be eager to utilise what is offered by academia as is. However, some tailoring often needs to be done in order to make the use

of formal methods efficient, i.e. tailoring it to a specific project or organisation. Therefore, there is still a noticeable gap between industry and academia, which could be diminished by making formal methods more flexible in their applicability.

In order to make the development of large systems pragmatically feasible, timely and supportive to the developers, as well as resulting in satisfaction to all involved parties, e.g. developers, stakeholders and users, the development methods that currently are used might not suffice. They need to be supported by employing an efficient development process, which would facilitate chosen development methods and techniques. This can be achieved by injecting elements of agile development methods in the safety-critical development, minding that correctness and assuring high quality is still the priority.

Since agile methods can be seen as a conceptual framework, many of the development ideas can be brought into this framework and adjusted to the needs and characteristics of the environment, domain and developers. Given that there is a plethora of methodologies, practices and techniques that are to be used simultaneously, combining them may cause serious organisational problems. It is often the case that the initial process set up in the project needs to be tailored and iteratively improved in order to find a perfect mix. We intend to use a number of existing methods and capture the practices and values that best suit the needs of the formal setting.

Well-founded and established formal methods, which bring necessary rigour to the safety-critical developments, and the widely-used agile principles and methods, which introduce iterative and transparent development, need to be combined into a hybrid approach that emphasises the strengths from both of the concepts. Naturally, the hybrid approach should comprise a risk assessment considering characteristics of a particular project, e.g. balance the high risk of system failure versus an overwhelming need for speed to market [2] [3].

Our aim is not to limit ourselves with a particular formal method. Rather, we aim at proposing a generic approach, which can be utilised by various types of formal development. However, we are strongly linked with the Event-B formal method and modelling language [4], both by experience and history, and therefore when giving examples in our study, we will show how agile principles can be merged into Event-B. We strongly believe that successfully injecting agile principles to formal methods (also other than Event-B) can prove to be efficient and feasible due to elasticity of agile approaches. Agile methods will thus act as facilitators not only in the combining and fine-tuning of methods, techniques and practices, but also in the adjustment of the

proposed framework to the formal methods of choice. Agility brings no reference model for the decision making, as principles and practices of software development are still evolving. Therefore, formal methods with their design patterns etc. to guide the development will complement agility in this respect.

In this report we propose to merge a formal method with elements of agile software development principles and values. The former provides a well-defined development methodology, whereas the latter brings tailorable elasticity to the development process, in particular to modelling. Our goal is to increase usability and flexibility of formal development methods, and in particular to extend the flexibility of formal modelling process. By flexibility we mean a degree to which a process or a method can be used effectively, free from risk and with satisfaction in contexts beyond those initially specified in the development setting[1]. Additionally, we see a potential in using agility in formal approaches, since it may widen the take-up of formal methods in industry.

2. Related work

Formal methods can be traced back to 1970s, whereas agile methods date back to year 2001 (Agile Manifesto [5]). Although these two methodologies seem to be on the completely opposite sides of the line, their various combinations have been reported for almost a decade now.

Agile and Formal – separate worlds

There is a variety of formal methods, application of which differs with respect to the domain, and properties of the modelled systems. Application of formal methods involves knowledge of theoretical computer science fundamentals, in particular logic calculi, formal languages, automata theory, and program semantics, but also type systems and algebraic data types in order to solve problems in software and hardware specification and verification [6]. To our knowledge, there is no single source providing a comprehensive overview of existing formal methods, which is most likely caused by the substantial diversity of formal methods. However, there is a plethora of material

[1] Definition based on standard ISO/IEC SQuaRE [78]; rephrased from product-oriented to process and methodology-oriented

regarding certain type of formal methods, just to mention formal methods for verification [7], architecture [8], modelling [4], etc.

An overview of research on agile methods, which aims at explaining its role in software development throughout a decade (involving transformation from e.g. plan-driven, waterfall, etc.) is presented in [9]. Authors present different viewpoints and definitions of agility, as well as research themes and theoretical perspectives used in agile research. However, several common characteristics are identified, such as prioritising people over development processes (emphasis on communication), documenting only necessary artefacts (no wasteful documentation) and acceptance of uncertainty (volatile requirements). The authors mention that many practices, including merging agile with other development methods, have been reported as successful, however without empirical validation or deeper research of such claims. Therefore, further investigation is necessary to support the statements posed in experience reports and industrial white papers. In another paper regarding agile practices for large-scale developments [10] authors clearly state that in case of agile development the practice is inherently ahead of research; therefore, it needs the support and reinforcements from academia.

Agile and Formal – a common direction

Until recently, a question of maturity of formal and agile methodologies was broadly discussed in different contexts, mostly separately. For instance, formal methods were examined with respect to their adoption in industry [11], while agile methods were investigated in connection with their role in software development [9]. The readiness of formal methods for adaptation of agile principles and vice versa has already been considered for several years. However, only recently the discussion has become more systematised and coordinated, e.g. via events like International Workshop on Formal Methods and Agile Methods (since 2009) or International Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA, since 2012).

The analysis of readiness of formal methods for agility has taken place in [12], where authors notice that the integration is already taking place and, thus, is an interesting emerging research topic. They confirm the claim of agile enthusiasts Torgeir Dingsøyr and Nils Brede Moe given in [10] that the practice precedes research. In [12] benefits of future synergy between two approaches are portrayed. The potential for the combination of formal and agile techniques in software development is described in [13]. The authors observe that the merge, if applied cautiously, can minimise change-related problems and aid in evolution of the system being built. We believe that the synergy

already has taken place and that it needs to be documented, along with its benefits and drawbacks, and then possibly increase the usefulness of this merge. In this paper we focus on the former and provide our vision of possible improvement of this mix.

A powerful combination

One can perceive the merge of formal methods and agile methods in a twofold manner:

1. Agile teams applying formal methods in their development;
2. Formal development being enriched by agile method principles and practices.

In the following subsections we will first concentrate on the adoption of formal methods to the agile setting and then follow with utilising formal methods in agile environment. Finally, we give some remarks about the role of standardisation and the industrial aspect in the discussed merge of formal and agile approaches.

Agile teams applying formal methods

It has been demonstrated that agile teams are able to employ highly disciplined formal methods [14]. A proposal of a way to employ formal development methods in the agile development process, in particular Scrum, is described in [15]. The motivation of the author is to make advantages of formal specification visible and at the same time enable the use of agile development techniques for critical systems. The main idea is to fulfil the time and budget obligations with agile methods and assure quality with formal methods. We agree with these goals and statements, however, we are interested in a two-fold scenario: the developers utilising formal methods and making use of agile elements, as well as users of agile methods who include formal methods in their development, if there is a need for employing rigour in their system. Furthermore, in [15] the discussion evolves mainly around Scrum (however not limited to it) and VDM methodologies (Event-B is shortly mentioned as a related work). In our work we discuss several agile methods and integrate the best of concepts to a formal development methodology, considering the Event-B method as the primary formal method.

A proposal of formal methods being “injected” into the agile development was presented in [16], where the author was arguing for supporting agile development with formal system modelling and transformation. He puts emphasis on refactoring, e.g. by proving transformation for parts of a system and having patterns or rules to check the correctness of a system, which can be done in an agile way. Also the importance of possibility of standardisation of such an agile and formal combination is required in

order for the mixed method to be fully utilised. In our work we also support the concept of having guidelines on how to proceed with the development so that it is effective. However, we also focus on supporting the developers, who are using formal methods, with advice on development process that originate from agile methods.

Formal development with an agile twist

An example of using an agile method, in particular XP, in order to improve a software process for the development of avionics software is presented in [17]. Authors apply agile principles to the certification driven process (DO-178B) of avionics software development and identify three agile principles that need re-interpretation in order to be able to apply them in this safety-critical setting. These three principles regard producing valuable software (here software needs to be more than tested), face to face communication (here software needs to be well documented) and the issue of working software (not only should the software be working, but it should also be certifiable).

Using agile methodologies to build safety-critical software is described in form of a case study in [18]. Agile methods, particularly Scrum, were used to develop an open source image-guided surgical toolkit. The software project has been utilised by teaching hospitals and research labs, and used for clinical trials. In the paper the authors claim that "agile methods have matured since the academic community suggested almost a decade ago that they were not suitable for safety-critical systems". Therefore, we believe that it is the right time to experiment with them in the context of e.g. Event-B.

Blending agile methodologies into plan-driven software development was discussed in several publications within a special issue of Computer magazine [19]. Topics such as mixing agile and plan-driven development, as well as identification how and when to mix these two approaches were discussed among others in perspective of Scrum and XP methodologies.

Employing agile methodologies can be set not only in the context of a development process or a product, but also when engineering new (development) methods [20], and in particular for integrating formal methods. In the referred paper authors presented a framework for establishing a development method utilising agile principles and practices. Specifically, they used technical practices of XP method to integrate parts of Eiffel formal development method with CSP. They demonstrated via a case study the idea of using an agile method for formal method engineering. In our work we aim at creating a framework for flexible and adaptable development that uses formal methods. We do not limit ourselves with a particular agile method. Rather, we combine best

suited (for our purpose) practices from a subset of agile methodologies with the intention to make them as generic as possible. Having Event-B formal method in the background serves as an example for our proposals.

Standardisation aspect

Regardless if the discussion is on agile methods utilising formal development methods or the other way around, the issue of certification often arises. This is due to the type of systems where formal methods are applied, namely safety-critical systems. These typically need to be qualified or certified in order to be deployed into use. Although agile methods are not explicitly considered in standards (e.g. IEC 51608 standard), their use in safety-critical development is already present.

Project RECOMP [21], a European funded FP7 project (2010-2013), reported three companies which were working with different agile methods in the safety critical domain. The research, among other topics, concerned certification issues, including IEC 61508, DO-178 and CMMI standards. The particular project deliverable discussing these topics is confidential, thus we can only mention information that was approved to be made public or is made public by companies Skov, Kone and Symtavision themselves.

Skov is a Danish company developing systems handling climate control and production monitoring of animal production. They work on systems and components for ventilation systems, livestock house air cleaning and production control, e.g. ventilation systems for animal stables. The industry is safety-critical, however, the development is not certified, since no standards for this particular industry exist. Skov does all of their development with Scrum.

A Finish company developing various types of elevators and escalators, Kone, uses Scrum only for development of new functionality. The well-known functionality and development parts that have to be certified are still developed with traditional processes.

Finally, a German company Symtavision provides timing analysis solutions for planning, optimizing and verifying embedded real-time systems. They make software tools for safety critical development (aerospace, automotive), however, nothing of their development is included in safety critical products. Thus, they do not have certifications. Their development is built around Scrum.

The aforementioned companies are using agile methods (in one way or another) on a daily basis, meaning that the concept of having agile principles and practices in safety-critical development did not remain in the conceptual area only. It is rather the practicality of the methodology merge, i.e. increase of development effectiveness, enhancing communication between developers and providing better development transparency, among other factors, that caused the adoption of this mix in real life. However, most of the evidence and details regarding successes and challenges of this kind of hybrid approaches is not public due to the industrial secrecy. Therefore, we investigate the merge of agile and formal methodologies by taking into account the perspective of potential standardisation (see e.g. the discussion on documentation in later sections of this paper).

3. Formal methods

Formal methods are mathematically based techniques for the specification, development and verification of software and hardware systems [22]. The application of formal methods for system (software and hardware) design is motivated by the expectation that doing proper mathematical analysis can lead to reliable and robust design [23].

Formal development is a development that requires the application of formal methods, while semi-formal methods may omit the proofs in favour of e.g. simulation, as main analysis technique [24]. We are interested in both formal and semi-formal techniques, since both of them contribute to the correctness of designed systems. Moreover, we consider them equally suitable to be made more flexible.

Quality

Safety-critical systems are required to be of high quality. Primarily, they need to be dependable, which means that they should have the ability to avoid service failures that are more than acceptably frequent and severe [25]. Dependability, as a quality attribute, can be decomposed into availability, reliability, safety, confidentiality, integrity and maintainability [25].

According to the ISO / IEC 9126 standard [26] (now relabelled to ISO 25000 series), the quality attributes are reliability, maintainability, functionality, usability, efficiency and portability. The two first ones are dependability attributes. All of these signify aspects of the end product quality for the software to be developed [27].

In our work we are interested in providing a development framework, which would facilitate development of high quality safety-critical systems. In our context the quality encapsulates dependability with special focus on maintainability, as well as other attributes like functionality, usability and efficiency. Our main goal is to establish a framework that tackles not only the product related artefacts (be it a specification, a model or a final system), but also the way how to achieve high-quality (software) systems. The latter basically means tailoring a development process, which in principle should be adaptable, flexible and reactive, while still retaining the necessary degree of rigour. Therefore, ensuring high-quality is a requirement not only for the final product, but also for the development process.

Refinement

Refinement [28] [29] [30] 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) [31] [32] [33]. *Stepwise refinement* is a top-down approach [29], which aids handling all the implementation matters and complexity by decomposing the problems to be specified and gradually introducing details of the system to the specification. In the refinement process, an abstract specification A is created from requirements. Specification A is then transformed into a more concrete and deterministic system C that preserves the functionality of A in consequtive refinement steps. The refinement process is shown in Figure 1.

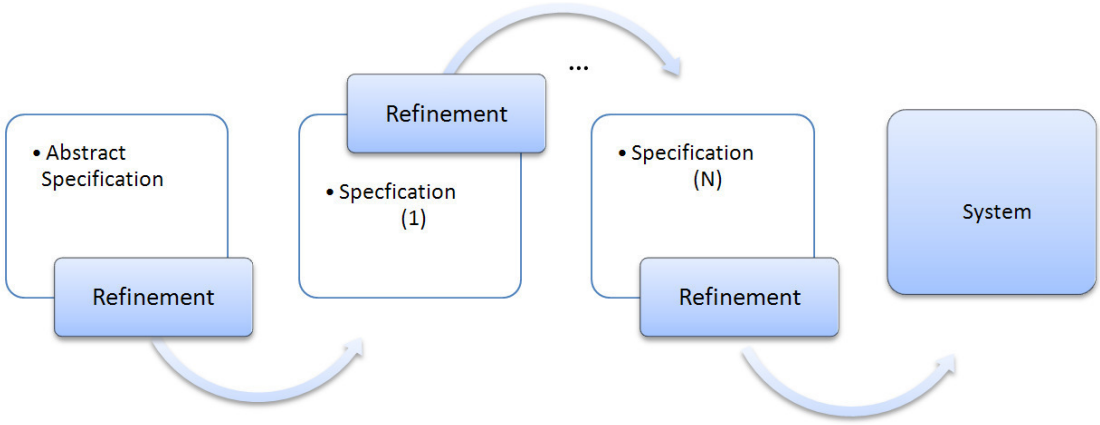

Figure 1 Refinement process

In our work we mainly use superposition refinement [34]. This is a certain kind of refinement that allows new variables and related events that operate on them to be

added within each refinement step. The existing behaviour of the system is not changed during the refinement.

Proving

The correctness of the system development, resulting in correct by construction [28] systems, is ensured by mathematically proving that the abstract model is consistent and feasible. It involves proving that an invariant is established after the initialisation of the machine and that each event preserves the invariant. Although proving is nowadays tool-supported, there are still some proofs that require human involvement (which can sometimes be cumbersome).

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 decomposition mechanisms [35]. Therefore, providing an aid in modelling activity by facilitating the development process would bring reinforcements with the complexity matters.

The Event-B method

Event-B [4] [36] is a formal method and modelling language for stepwise system-level modelling and analysis, based on the Action Systems formalism [37] [38] [39]. It is derived from the B-Method [40], with which it has several commonalities, e.g. set theory and the refinement idea. Nevertheless, the methods differ with respect to the refinement rules. Event-B is meant for distributed systems, while the B-Method can only prove sequential systems. Moreover, Event-B is dedicated to model complete systems, including hardware, software and environment [41].

Event-B utilises refinement to represent systems at different abstraction levels, which enables us to gradually introduce more 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 process of (critical) systems. It is effectively supported via the Rodin platform [42], an Eclipse based tool, which is an open source "rich client platform" that is extendable with plug-ins.

An Event-B specification uses a pseudo-programming notation – Abstract Machine Notation (AMN) – and consists of a dynamic and a static part, called machine and

context respectively. An abstract Event-B machine consists of its unique name and has the following constructs: context, which links the machine with its static context via the SEES relationship, a list of distinct variables that give the attributes of the system; invariants – stating properties that the machine variables should preserve; a collection of events – depicting operations on the variables, where INITIALISATION is an event that initialises the system. The context encapsulates the sets and constants of the model with their properties given by axioms and theorems.

The formal development starts from specifying an abstract machine from a set of requirements and then refining the machine in a number of steps (as presented in Figure 1). Each consecutive machine is called REFINEMENT. 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 can also be refined, which is indicated by the EXTENDS clause.

The best out of the formal world

There are several key elements of formal methods that we want to emphasize, especially in the light of further discussion:

- Quality as a main concept
    - Correctness given with formal modelling and proofs
    - A strive for technical excellence and good design
    - Traceability as a part of the formal process
- Iterative modelling
    - Provided by refinement mechanisms
- Complexity control
    - Stepwise introduction of functionality and properties
    - Simplicity as a key element when it comes to modelling and proving
- Used for system development
    - Including hardware and software
- Tool support
    - Enhancing the efficiency of a development (design) method
    - Enabling application to larger problems
    - Aid for finding inconsistencies and conflicts in an efficient manner
    - Essential for adoption of a method to the industry setting

It is common to associate formal methods with "heavy" design methods. However, many of the newer formal methods are already presenting some elements of agility [43], for instance: Design-by-Contract (Eiffel, JML, Spec#), extended static checking based on contracts (ESC/Java, Boogie) and automatic test generation. Formal methods can be linked very well with many of the agile method principles.

In this paper we consider formal methods in general. However, our examples are related to the Event-B method and the Rodin tool.

4. Agile methods

Agile software development methods (in various forms) have been present on the arena of IT some time before year 2001. However, it was only in 2001 that Agile Manifesto [5] was written down and its principles and practices were established.

Agile Manifesto – values, principles and practices

The Manifesto for Agile Software Development recognises several values for improvement of the way the software is being developed. The primary values are given in the following statements on the left hand side:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

There are twelve *principles* behind the Agile Manifesto [5]. We will refer to these when discussing the enhancements in the development based on formal methods.

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Agile development is facilitated by *practices*, which are concrete techniques proposed for certain areas of development, like requirements, design, modelling, coding, testing, project management process, quality. The most known practices are pair programming, refactoring, use cases and test-driven development [44].

The strength of agile methods lies in their flexibility and adaptability, which is also referred to as *method tailoring*. Adjusting the development to the organisational needs of the project and developers, as well as to the specifics of development context, distinguishes agile methods from traditional development methods (including formal methods).

Agile development methods are described as iterative and incremental, promoting teamwork and collaboration, as well as endorsing the idea of adapting the process according to the needs of the project. The development is considered evolutionary and as it builds gradually on the close collaboration with customer. Figure 2 shows the most common characteristics of agile methods. Here we have chosen to look closer at four agile development methods: Dynamic Systems Development Method (DSDM), Kanban, Scrum and eXtreme Programming (XP). These were selected with respect to their features and their capabilities to fit the formal development specificity. In the following sections we will shortly describe each of them and emphasise the characteristics that have potential for our FormAgi adaptive framework.

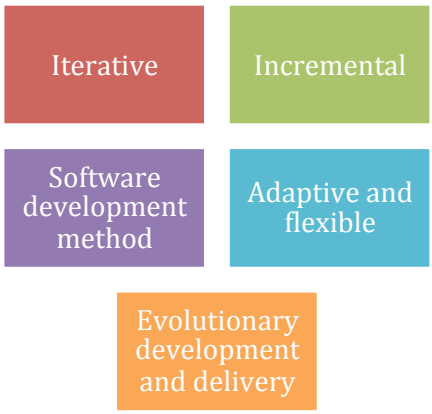

Figure 2 Agile methods – commonalities

Agile principles emphasise the importance of human factor in the development, i.e. the role of a customer and a team, as well as their collaboration. They propose to deal with change through requirements and consider simplicity as the core of development [20]. All these characteristics combined can facilitate complexity management, which is much needed in the development of contemporary (software) systems.

DSDM

Dynamic systems development method (DSDM) [45] [46] is an agile project delivery framework. It was first released in 1994 and was primarily used as a software development method (bringing some control to the rapid application development – RAD, where minimal planning and rapid prototyping are the priorities). The latest version, DSDM Atern, was established in 2007 and aims at helping people to collaborate more effectively to achieve business goals, and thus is most feasible for information systems projects.

DSDM involves eight principles supporting the idea of continuous delivery and team work. If not completely complying with the rules, some other control mechanisms have to be utilised. However, some concepts may be borrowed and adjusted to the formal development setting.

Below we present the principles that we believe are most suitable in the context of adaptive development framework:

- The *quality* of (software) system is the priority, therefore there is a need for a proper design, providing documentation and performing tests (which could seem

as being opposite to principles of agile methodologies). Also reviews are advised as another mechanism for ensuring the quality.

- *Delivering product* (or feature) *on time* is strongly advocated, which is supported by iterative and incremental development, as well as collaboration with customer (along the lines of agile principles). Moreover, there are some suggested practices, like prioritisation or time-boxing to aid meeting the deadlines.
- Continuous user and/or customer involvement is considered vital when it comes to having a project that progresses effectively and in time, as it supports decision making and *collaboration*.

- *The incremental development from firm foundations* is crucial in order to benefit the user and maybe also the customer as well as the development team, regardless if it is a business profit or a personal/social benefit. Confirming that the correct solution is being developed is the foundation for having early delivery of desired solution and functionality. Re-assessing priorities and the contemporary project capability should be done in each increment.
- *Iterative development* is not only a part of agile strategy. In case of DSDM it is complemented by having strong foundations from the start by providing enough design up front. *Gradual system building* (stepwise introduction of features) is supported by the idea that more details emerge step by step. Customer feedback is needed on a regular basis (in every iteration), which aids to produce the right solution and embrace change. Moreover, DSDM encourages creativity, continuous evolvement and learning, as well as experimenting, since it builds the morale and strengthens the confidence of developer(s) and positively impacts the team.
- *Continuous and clear communication* should exist not only on the informal level (face to face communication), but also on more formal one (lean and timely documentation, modelling and prototyping) in order to benefit from an efficient and effective development. The granularity of communication should be kept at least on the iteration-basis. Some degree of *control* is necessary in order to manage proactively, measure progress (delivery of products rather than completed activities) and make plans. Progress of a project should have appropriate visibility, which can be obtained by using appropriate level of formality for *tracking and reporting*. Finally, *evaluation mechanisms* should be implemented to measure project viability with respect to business objectives.

DSDM vs formal development

DSDM has several commonalities with formal method development, as well as some principles that would well fill in the formal development needs.

DSDM is flexible enough to allow users and/or customers fill in the specific steps of the process with their own techniques and software aids of choice. This means that in case of systems, which have some components of higher criticality, these components could be designed with the use of one of the formal methods. As a consequence a proper degree of rigour would be assured for the components and the development process would remain flexible.

The main variables in the DSDM development are the requirements (and not time or resources). This is similar to the formal development, where building (software) system heavily depends on constructing it from requirements and proving certain properties of the system. Developing the system based on requirements combined with frequent and continuous communication ensures the main goals of DSDM: to stay within the deadline and the budget. The strong focus on communication between the users and/or stakeholders, as well as and the involvement of all the stakeholders in the system development could complement formal development. Among benefits of such contribution, one can list: a quicker reaction to wrongly interpreted requirements, capturing the inconsistencies in the understanding of functional and non-functional requirements in a timely manner or constant support in case some artefacts are not clear. All of the abovementioned contribute to saving time and resources during development.

Finally, commitment to the project is regarded essential in order to ensure a successful outcome. This psychological aspect is also important for other agile methods. However, it seems that formal methods do not emphasise this aspect of human involvement and concentrate mostly on the correctness of the developed system. We believe that introducing this aspect to formal development would add value to the development process, since (software) system development is fundamentally a human activity, regardless if it is tool supported.

Kanban

Kanban [47] [48] is an approach for managing knowledge work with a focus on just-in-time delivery, while not straining the involved participants. In this method the participants have clear information on the tasks to be executed and the team members

take work from a queue consisting of these tasks. This process involves all the steps from definition of a task until its delivery to the customer.

Kanban can be characterised in a two-fold manner:

- As a graphical *management process* telling what, when and how much to produce,
- And as an *approach to* incremental, evolutionary *process improvement* for organizations.

Kanban is based on four principles, focusing strongly on incremental and evolutionary change for the continuous improvement:

- It starts from the existing process steps and set of roles, and builds on top of that
- Since some of the existing elements in the organisation may work acceptably, they do not need to change, at least initially
- It is based on the common agreement towards improvement
- It encourages initiative and leadership on all organisational levels

There are six core practices supporting Kanban and facilitating execution of its principles:

- *Visualising the workflow and the system needs* – by increasing the understanding of the process and system by bringing the necessary level of clarity
- *Limiting work in progress* – by implementing changes in a stepwise manner
- *Managing flow* – by evaluating the changes
- *Making policies explicit* – by having a clear and understandable process
- *Implementing feedback loops* – by collaboration, use of measurements, etc. to make the process improvements and related benefits clear
- *Improving collaboratively and evolving experimentally* – by using models and scientific method.

Kanban vs formal development

Kanban suggests that a scientific approach should be used to achieve improvement by implementing continuous, incremental and evolutionary changes. However, it does not prescribe a specific scientific method to be used, thereby leaving this decision for the organisation and/or developers. Formal development can be used as such scientific

method and bring necessary rigour to the development. In Table 1we present how Kanban can be related to formal development and suggest how it can contribute to adding flexibility to a rigorous method.

Kanban	Formal development
Visualise (the workflow)	Visualise (the model)
Limit work in progress	Refinement mechanism with its incremental and evolutionary system development. Effect: limitation on defects (faults).
Manage flow	The design and its process needs to be monitored, measured and reported. Evaluation should include the modelling strategies used. Continuous workflow should be ensured by small refinement steps.
Make policies explicit	The organisational policies need to be set in order to support the discussion about the design and design strategy between the team members. Also the tool(s) used in the development should support the exchange of ideas and experiences. This visibility should be sustained for the improvement purposes.
Implement feedback loops	Feedback loops can be facilitated by including e.g. metrics, measurements and indicators for trends into the design process, as well as supporting collaboration and team work. Design reviews done by other (senior) team members would certainly strengthen the process both by providing quality to the design and aiding the learning process.
Improve collaboratively, evolve experimentally	Using formal models and formal method brings necessary rigour to the development while Kanban is there to facilitate the design process. Collaboration and experimentation would increase understandability and support creating specialised and cross-functional teams.

Table 1 Relation between Kanban practices and formal development characteristics

Scrum

Scrum was based on case studies from manufacturing companies in the automotive, photocopier and printer industries and is described in [49] as a method to increase flexibility and speed of development. The method was created to tackle complex software systems and has been used as a project management method.

Following the agile spirit, Scrum is oriented towards short feedback loops in system development, fulfilling customer requirements and providing a clear development process. Customer is heavily involved in the development on a regular basis, which gives valuable feedback on the software being developed (whether the customer needs are satisfied) and allows to timely capture divergence from customer's vision of the product. Hence, communication and adaption are cornerstones also in this agile method.

The Scrum development process is iterative and the result of each iteration should be a potentially shippable product. New requirements for the final product can be dealt with when a new iteration starts. After each iteration the development process is reviewed and necessary changes to the process may be taken at this point.

Scrum vs formal development

The relationship between Scrum and formal development are presented in Table 1. We focused on the main and comparable characteristics, suggesting how formal development can be extended or fine-tuned with Scrum in order to be more flexible and adaptable.

Scrum	Formal development
A flexible, holistic product development strategy where a development team works as a unit to reach a common goal	A rigid approach - encountering problems with organising team-work or distributed development. A holistic approach - domain knowledge is, however, a necessity when proving system properties, in order to ensure that the system is correct.
Emphasis on communication	Needs to be fine-tuned to formal methods reality. Heavy involvement of a customer might not always be possible. Formal methods are more focused on individual problem solving than communication. Via animation of the models they could, however, be discussed with others.
Divided into sprints that last a certain amount of time	Formal development does not have a time-frame structured process. However, such a process can be established to suit the needs of the project and enable establishing check-points for the refinement steps.
Teams with multifaceted know-how, role oriented	Team work on one model might face difficulties. However, a team with versatile know-how could be

	useful and easier to assemble.
Empirical approach - Quick delivery and quick response to emerging requirements	Theory based approach - Time factor is different in formal development (more emphasis on correctness than on rapid development). Changing requirements may also involve re-proving the changed models, which in turn entails delays.

Table 2 Cross-comparison of major characteristics of Scrum and formal development

XP

Extreme Programming (XP) is a software development method, and a type of the agile software development, intended to improve software quality and responsiveness to changing customer requirements [50]. The most important characteristic of XP is providing the visibility, simplicity and stability to software development by pushing the best practices to the extreme, yet in a disciplined manner [51].

Among many elements of XP there are: pair programming, extensive code review, unit testing of all code, delaying implementation of features until they are needed, simplicity and clarity in code. Moreover, anticipating changes in the customer's requirements when the problem is better understood, as well as communication with the customer and among programmers is vital for the success of XP. Therefore, XP answers such development problems as: poor system quality (making the system unusable), not meeting the requirements, and finally the situation where the resulting system is outdated by the time of deployment. As in other agile methods, XP is also based on short iterations ("inspect and adapt") and short feedback loops (interaction with users/customers).

XP provides a number of values, principles and practices, which are based on the agile principles. A combination of all of them gives a detailed picture of the method. Therefore, it is significant to evaluate the currently used approach against the complete set of values, practices and principles, in order to fully benefit from utilising XP [20].

There are four XP activities:

- *Coding* - The only truly important product of the system development process is code. No code means no working product
- *Testing* - Some testing can eliminate some flaws, but extensive testing can eliminate more flaws (unit tests, acceptance tests)

- *Listening* - Communication with customers to understand their needs and facilitate the collaboration between developers
- *Designing* - Control over the complexity and dependencies within a system

Since XP relates to the (software) system code and acknowledges testing as one of the most important practices, the method needs some fine tuning for the formal development setting. The four abovementioned activities should be related to the model, instead of to the code. Therefore, modelling and models should aid in communication between the developers. Furthermore, a model should not leave space for interpretation of system properties. In addition, keeping a model simple not only facilitates its modification (e.g. due to changed requirements), but also helps when proving properties of a model.

Testing activities, advocated by XP can be "translated into" proving activities, needed when constructing the system. As testing is treated as a mechanism for quality control in XP, in case of formal modelling proving would be a way to ensure correctness, and thus quality. Listening to the other developers, as well as communication with users and/or customers early in the development (design stage) not only aids in achieving the desirable solution or product, but also reduces the risk of producing a useless artefact and capture some potential problems in time. This in consequence decreases the possibility of money loss, waste of resources and time.

According to XP activities, having a design is a way of controlling the complexity and managing the dependencies within a system. Having formal development with clear and simple specifications and models that are being proven while constructed is contributing to complexity control. Moreover, adding features to the system in a stepwise manner helps to minimise the difficulty of proving system properties and by that reduces the possible complexity.

There are four important values in the XP method, which basically can be treated as guidelines on how to foster decisions in a system development project; communication, simplicity, feedback and courage. They guide the development as follows:

- Communication
    - Brings clarity and visibility to the development
    - Helps in sharing knowledge between development team members, as well as the users of the system
- Simplicity

    - Starts with the simplest solution
    - More functionality can then be added later control over development process and the system
    - Facilitates communication
    - A simple design with very simple code could be easily understood by most programmers in the team
    - A disadvantage is that more effort might need to be entailed later to change the system
- Feedback
    - From the system via unit tests
    - From the customer via acceptance and functional tests
    - From the team via time estimation for the feature to be implemented
    - Emphasis is given to timely feedback that is provided frequently and promptly.
- Courage
    - Design for today
    - Preparation to refactor the code
    - Persistence in finding solutions

XP methodology also emphasises respect within team members and respect of developers to their work. This helps to keep the team motivation high, as well as pushes the progress of the development from a psychological aspect, as opposed to purely implementing technical solutions.

Interestingly enough, NASA was using elements of XP methodology in their development earlier than the XP principles and practices were established [52]. Already in 60ties NASA had a practice of test-first development, which meant planning and writing tests before each micro-increment. Formal test documents for acceptance testing based on formal requirements were written before the tests for actual system were created. Furthermore, the tests were automated (to even further shorten the development time) and worked on small sections of code instead of big features.

XP vs formal development

In Table 3 we present a list of characteristics vital for eXtreme Programming and relate it to the features of formal development setting.

XP	Formal development
A software-development discipline that organizes people to produce higher-quality software more productively	Aiming at high quality (correctness) and productivity, where the former is of priority
Introduce checkpoints where new customer requirements can be adopted (feasibility of new requirements and their prioritisation consulted with customer)	No checkpoints defined. Checkpoints should be identified according to the type of organisation and development. – First the time between the checkpoints should be defined and then the actual modelling can begin, which should be supported by the collaboration with customer and aided with initial empirical insight. After that the initial assumptions can be checked and the decision on further progress of this part of the project can be made.
Improve responsiveness to changing customer requirements; requirements are considered as an opportunity	Changing requirements are the reality, but may cause a lot of work. – Modifications mean reproving parts of the system, which entails rework and costs. Models need to be well structured, so that the effort is minimised
Multiple short development cycles (cost and complexity reduction)	Small refinement steps (risk and complexity reduction)
Pair programming	Writing specifications and building models in pairs or a specification or and/or model created by one person is reviewed by another
Extensive code review	Specification and/or model review (preferably by senior staff)
Simplicity and clarity in code	Simplicity and clarity in specification and/or model
Beneficial elements of traditional SE practices are taken to extreme levels	Best practices taken to formal development

Table 3 Listing of eXtreme Programming characteristics and how formal development relates to it

As for the main mechanisms for maintaining the scope, in case of XP it is to limit the number of requirements being processed to the minimum, as well as iterate with the customer on the working software (feedback loop). Formal development, on the other hand, focuses on introducing changes in a stepwise manner, so that the proving activity can be simplified, and proposes simulation or animation as means to obtain feedback from customer.

The values of XP need to be adjusted to meet the needs of formal development methodologies. For instance, models and documentation can be used as communication means. Keeping design simple is also important for both kinds of environments, but is even more visible in the formal ones, as it simplifies e.g. the proving process. It also improves the team communication and overall understandability of a model. However, designing and coding according to the currently known requirements (the XP idea of "design for today") might cause that the effort of "tomorrow" will be excessive, and e.g. cause a delay in modelling crucial features and properties. Hence, formal development should not agree fully with the idea of design for today, but plan the design to make it well-structured.

The courage in XP should be understood in formal development as the courage to think ahead, but at the same time be able to balance it with prioritising what needs to be added or modified. One should aim for an iterative and steady pace of development, since refactoring a model or specification can be problematic (also as a tool issue). Therefore, it is vital to introduce changes one by one, while keeping the broader perspective on the development.

Feedback in a formal environment can be viewed in three different dimensions of the system development, just as for XP. However, the type of feedback and the way it is obtained can slightly differ between the two environments. The feedback from the system can be given by the modelling tool when modelling and proving. The feedback from the customer on the other hand can be provided by showing the customer a model simulation or animation which gives information on the implemented functionality and properties. Note that the model does not have to be executable and final at the point of validating it with the customer. The validation would serve as a checkpoint to see whether the development is progressing in the right direction. Finally, the feedback from the team can help to e.g. redesign the system (for simplification) or confirm the choice of a modelling strategy.

A cross-comparison of other characteristics of XP and formal development in perspective of potential drawbacks of XP methodology is presented in short in Table 4.

In the right column we propose solutions to these weaknesses, some of them already provided by formal development.

XP issues	Possible solutions /formal development
Unstable, iteratively defined requirements	Requirements are rather stable for safety-critical systems[2]
Requirements expressed as automated acceptance tests rather than specification documents	Requirements as specification documents[3]
No documented compromises of user conflicts	Most of the potential conflicts can be detected when proving system properties. They should be documented
Lack of an overall design specification or document	The design specification is provided, however, there is a need for better documentation
No design up-front – threat of redesigning	Stable design, created incrementally
Too many meetings	Just enough meetings
Risk of scope creep	Focused design

Table 4 Relationship between other characteristics of XP and formal development

Best out of the agile world

In this section we shortly summarise the agile principles that we consider the most beneficial in perspective of using them in combination with formal methods. First and foremost, the quality orientation and striving for excellence is the main shared aspect in both types of development, but tackled in slightly different manner.

Flexibility and adaptability of a development, which means tailoring the development process according to the existing context, be it project specifics or organisational needs, is extremely useful not only in the aspect of the development environment, teams and single developers, but also when considering the volatility of requirements.

[2] Requirements can be checked with model animation, which would correspond to acceptance testing in XP. There are some tools (and plug-ins) from the formal development domain that deal with requirements before the requirements are being modelled. For instance ProR plugin [51] within Rodin tool (idea similar to ProB) sorts requirements given in natural language in a structured way so that it is easier to model them later.

Furthermore, using the best existing practices and artefacts (model, code, experience) and building upon them covers the socio-psychological side of the development, i.e. it is easier to modify something existing than to start with something completely new. Hence, the evolution is strongly emphasised as opposed to revolution.

The steady pace of the development and exceeded control over the project is brought by the iterative and incremental development. Note that control is used in a sense of manageability and not e.g. controlling and comparing developers in teams. Iterative and incremental development entails reducing development risks and reacting to the encountered problems in time.

Communication, collaboration and user involvement is especially important, since the human aspect is inevitably present in (software) systems development. Social interaction is facilitating the knowledge exchange, as well as it helps in promptly discovering the problems and inconsistencies regarding the development (regardless if it is the development itself, team work, customer needs, etc.). Similarly, the time factor is present in having frequent releases of the system, which also helps in getting feedback from users and/or customers, as well as identifying inconsistencies and problems with development.

Maintaining a simple development has an effect of making the development activities easier and more effective. Furthermore, it helps to maximise the amount of work not done and facilitates adding more features to the system.

There is a large choice of agile methods, which basically share features within the agile concept, but at the same time are answering different types of development challenges. Thus, the organisations or developers can certainly find a method that best suits their needs.

5. FM vs AM – clash of titans or MacGyver solution?

Nowadays there is a need for tailored solutions, which would be based on well-known practices that at the same time are feasible in a specific context. This demand for hybrid approaches is based on a risk assessment of a particular project's characteristics, e.g. having high risk of failure (especially in case of safety-critical developments) versus an overwhelming need for speed to market [53] [53]. Our goal is to make formal

development more flexible and adaptable. Therefore, we aim at combining two recognized approaches: formal development methodology and agile approaches in order to establish a flexible formal development framework. So far, there have been some critical opinions about combining these two, as they were thought to behave like "oil and water" [54]. These approaches have been recently put in a new light and described as mature enough in order to be experimented with [15].

We want to use and highlight the strengths of agile and formal approaches and inject the elements of agile methodology into the formal one. On a more abstract level (meaning not focusing on a specific agile software development method), there are several agile principles and practices that can be merged directly and effortlessly with formal development, inter alia striving for high quality, iterative and incremental development. In Table 5we present a listing of characteristics that are considered crucial for agile approaches and depict how formal development can be fine-tuned to go hand-in-hand with agile processes.

Characteristic	Agile approach	Formal development and its tailoring
Approach type	Evolutionary	Refinement-based
Collaboration	Highly collaborative	Model reviews; team-work; visualisations and simulations as a support for discussion
Quality issue	Quality-focused approach	Quality is the priority
Software or system development?	Software	System (also including software)
Steady development	Potentially shippable working software is produced on a regular basis	Stepwise refinement; working SW can be understood as providing animations or generating code from the models
Executable releases	Produce working software on a systematic basis	(Stepwise) refinement combined with animation or code generation
Quality assurance	Continuous regression testing (or a better alternative, a test-driven development approach)	Quality is assured by correct-by-construction development (proving properties of the specification, verification mechanisms, etc.)

Collaboration with customer and/or user	Close collaboration, ideally on a daily basis	Collaboration is limited – development is based on expert domain knowledge[4]
Self-organising teams	Strongly supported within an appropriate governance framework	The social factors are not well supported – there is a need for mechanisms for building and sharing experience, lessons learnt etc.; tool support needs to be extended
Improvement	Regularly reflect on how the developers work together and then act to improve on their findings	Iterate on what can be improved – either with process or product – needs methodological and tool support

Table 5 Cross-comparison agile and formal development characteristics.

As depicted in Table 5, many of the ideas originating in agile principles are already present in formal development. The *iterative, incremental, evolutionary and steady* type of approach is visible through the refinement mechanism (especially stepwise refinement), where properties or features are added gradually in each of the modelling iterations. Also related proving is done iteratively, for each of the refinement step. However, producing *executable releases* for each single iteration of formal development might not be feasible. Therefore, in order to confirm or refute the properties of the produced system, one would need to provide simulations or animations or automatically generate code, so that the collaboration with the user and/or customer is facilitated.

Formal methods enable us to build *software* and *hardware*, as well as their *environment*, while agile approaches focus mostly on the software. However, one should note that many of the agile practices and specific agile methods (e.g. Lean, Kanban) originate in industrial procedures for system engineering (manufacturing lines, where software is only a fragment of the whole process).

Striving for *technical excellence* and achieving *high quality* of (software) systems are the cornerstones of formal developments. In safety-critical environments there is no room for faults or misinterpretations. Therefore, proving correctness and assuring high quality of the produced system is the priority. In this aspect, formal development is far

[4] Can be supported by verification and simulation mechanisms, which should be combined with close collaboration with user and/or customer

more recognized than the development using agile methods. However, agile methods strongly aim for improvement, e.g. through feedback mechanisms and refactoring, which is not so visible in formal development. There can be several reasons for this fact, just to mention that formal development is often related to big design upfront or so called "design for tomorrow", where almost all the details are thought through and decided before entering the next phase of development, leaving almost no space for changes. Moreover, refactoring can occur to be quite expensive in formal setting, both in sense of time and resources. Hence, some flexibility in the formal development process would enable not only easier refactoring, but also support the idea of improvement, regardless if it concerns a product or a process.

Frequent feedback and *short iterations*, as well as *artefact reviews* can be enabled in formal development by a tool support and well established modelling methods. The feedback can be twofold: from the tool itself, e.g. when proving or generating documentation or from the user and/or customer, when the tool is used to show animation or simulation of a system. In both cases it enables artefact reviews. Moreover, tools can be used as a medium for the internal communication between the development team members, e.g. for the more experienced developers to give feedback on the modelling strategy.

Self-organising teams are much encouraged by the agile methods, but not so well supported by the formal development. There is a need for organising social mechanisms for building and sharing experience, lessons learnt etc., which could also be assisted by a tool. Currently, such a knowledge exchange is usually done internally between developers and researchers and is not structured. Some elements of this kind of collaboration and self-management are visible through research papers or research projects [55] [21], but no rules have yet been proposed.

Frequent delivery and deployment need to be thought through and adapted to the type of the formal development or the organisation that runs the project. Rushing the deliveries causes a threat of lowering the quality of formal development, which in case of safety-critical systems is a serious issue and has severe consequences. When considering modelling and proving some system properties, one might need more time to either perform a proof or re-model and re-prove the system. Therefore, some checkpoints should be set, but at the same time they should not endanger the correctness of a development. As for frequent deployment of a system, tool support for code generation is necessary in order to be able to manage the fast pace of the development, as well as possible system refactoring.

There is one more crucial issue when considering inherently complex and large real life systems and the industrial deployment of development methods. The methods should fulfil a vital requirement – they have to be *scalable*, otherwise they are useless. As reported in [56] it is possible to improve scalability of agile methods (explicitly XP) with the help of a formal development method (explicitly VDM). As observed by the authors, lack of scalability is one of the problems of the agile development method.

Appropriate merge of agile and formal methods results in more effective and timely development, often of higher quality [56]. The major success factor of this blend is *balancing between flexibility and control*, characteristics offered separately by the two methods.

While agile methods provide software development how-to's on the project (management) and process level, formal development gives well-established systems engineering practices and rigour, which is often required in large and high-risk projects. In this work we propose a combination of agility with formal development principles on the conceptual level. We suggest injecting elements of agile methodology to the rigorous formal development process.

Additional issues formal vs agile

Formal development methods are placed on the opposite end of the scale compared to the agile ones with respect to predictive vs adaptive criterions [2]. Although they display some major differences between each other, they also have a large number of commonalities, mostly due to the fact that the creation of high-quality software is human-oriented activity. In this section we present how the differences can be reconciled and how these two supposedly distinct methodologies can support each other, when combined.

It has been reported that agile methods need to propose some mechanisms for handling the *traceability* issues [57]. Since documenting the development of a system and measuring its state was by many considered as a waste, many agile developments were encountering problems, especially when the deployed system entered the maintenance phase [1]. However traceability issues can be managed on behalf of formal modelling, where the formal models can be treated as a source of system documentation, where modelling decisions are recorded. Again, offering an appropriate tool support for the formal modelling is vital.

The *volatility of requirements* is one of the reasons that agile methods became so popular – they were proposing a solution to constantly changing requirements. Although change is welcome in agile methods, it is not so welcome in formal development, as the latter involves much more effort than in the agile setting. Daily „builds" may not occur to be reasonable in a formal setting, but having regular checkpoints e.g. twice a week can aid in structuring the development and make it more regular. Having these (less frequent) checkpoints prevents from rushing into wrong decisions, consequences of which would have to be fixed or refactored later on.

The structure and the properties of a system have an impact on when and how much of effort will be spent on system r*efactoring*. It is more feasible if the system is decomposed to many refinement steps with small abstraction gaps, which makes it a „well-proportionated" one. This type of model of a system facilitates change while minimising the effort. There are guidelines on how to deal with abstractness, as well as guidelines on system decomposition, which both facilitate "good" design [58] [59]. It is worth noting that changing the invariants is the most "expensive" change. An example of a balanced modelling and the role of a well proportionated system can be found in [60], where the authors describe the incremental development of the Mondex system in Event-B (discussed in more detail in section Challenges and opportunities specific for Event-B).

There is a need not only for well-structured systems, but also for clear and simple developments, which would decrease the problems regarding understandability of a system and facilitate development management. There is significantly more *transparency in the development process* achieved with agile methods, as there is an emphasis on communication between team members, as well as between developers and users and/or customers. In formal development the transparency is mainly provided by the tool, yet in a limited way. Furthermore, it can be obtained with measurements (metrics) and feedback mechanisms (proofs). However, the guidelines on how to communicate in a structured way (social aspect) are not defined for formal development. For instance, there exists a way of visualising the system at the design stage in the Rodin platform, e.g. via the UML-B plug-in. However, providing a visualisation of the development structure in Rodin, which is taking into account the granularity of a system, could be beneficial. Likewise, comparing the graphical representations of two development steps would add value to the development.

Event-B flavour

The biggest and most beneficial characteristic of Event-B modelling language, apart of being a formal method that has vast history and experience behind it, is its tool support. Rodin platform is an open source, Eclipse-based tool offering plenty of plugins, which not only facilitate formal modelling, but also serve as additional development aids. It is worth noticing that providing the tool support played a major role for the adoption of Event-B in industrial settings (projects RODIN [61], DEPLOY [55]).

Many plugins extend the tool concerning communication, knowledge transfer, increasing understandability, as well as tackling the concept of team work. The characteristics that are supported can be grouped as:

- Code generation to the following languages:
  - C, C++, Java and C#;
  - multi-tasking Java, Ada, and OpenMP C code
  - VHDL
  - JML-specified Java abstract classes and JML-specified Java implementations
- Animation and simulation
  - ProB
  - AnimB
  - UML-B - Statemachine Animation
- Visualisation
  - BMotion Studio
- Documentation
  - ProR
  - B2Latex

Domain knowledge from the Event-B perspective can be supported by using for example the Pro-B [62] plug-in, which is an animator and model checker for the B-Method. It enables scenario based model testing and checks the specification for a range of errors. Moreover, it can be used for model finding, deadlock checking and test-case generation. Another Event-B plug-in, BMotion Studio [63], is a visual editor for formal specifications, which allows the developer of a formal model to arrange a domain specific visualisation, e.g. for discussing it with the domain expert (user and/or customer). BMotion Studio enables to create the visualisation via the graphical user interface, instead of writing code.

6. Challenges and opportunities in the hybrid approach

The combination of formal methods and agile methods is far from straightforward, mostly due to the fact that these are perceived as two completely different approaches to software development (adaptive vs predictive). Therefore, it is not possible to merge all the elements of agile methods into the formal framework as is. Some of them need to be fine-tuned according to the specificity of the formal setting. One of these issues is for instance the matter of *communication (and implementation) vs documentation* and the issue of *frequent delivery and deployment*.

Need for speed in contemporary (software) systems development forces quick implementation in order to obtain executable code that can be shown to the customer, e.g. to gain appropriate feedback. This pressure for continuous and rapid coding discards in many cases creating documentation or design specifications, as these activities are often considered as a waste in agile setting as it is not an executable software per se. The misconception that follows the idea of producing working software and not documenting it is often a cause of many problems, as it often results in not having any documentation at all. As a consequence, difficulties accumulate when a created system is large and complex, as well as when it is deployed and enters the maintenance phase [1].

Producing documentation in safety-critical development is vital not only for the sake of recording the development process (certification and authorisation purposes), but also for documenting the steps of the development. Thus, the case when no documentation exists poses a serious threat to the development.

Furthermore, prioritising communication over documentation, although potentially making the development smoother at the time of developing a feature, has also its drawbacks. It can cause some traceability issues or the situations when e.g. decisions are not documented and cannot be reasoned about, which in turn lead to problems with maintainability of a system later on. In addition, some certification difficulties may arise due to insufficient documentation.

We want to keep documentation and support communication. Having e.g. models in formal developments is already some way to document a system. Although the specification as such is not documentation (it needs some additional notes, comments and traces), it is already a foundation for the further steps of the development. Finally,

tool (or tool-chain) support for documenting specifications and models in the formal environments is crucial not only for traceability purposes, but also to be able to use documentation as a communication medium.

Frequent delivery and deployment is regarded as one of the priorities in agile development. However, for formal development it is more important that the delivered software is correct and dependable. Therefore, it is more important to have steady, evolving and iterative development – sustainable pace – in formal development, which can be regulated by e.g. periodic checks or reviews. The control over the development process provided by these regular inspections can aid detection of design and development problems, missing requirements, wrong modelling directions etc. before too much work is invested in the development

From the practical point of view, the practice of "end-of-day working software" could be changed to an "end-of-week working software" schedule, or simply be reduced to mutually agreed regular dates, specific for instance for a project or an organisation. Such a more relaxed schedule would allow people to avoid the feeling of being rushed to generate artificial stubs just to deliver "something" working regardless of its quality. Additionally, a less-rigid schedule would allow developers working on complex features and functionality to more fully develop them over a longer period of time. Establishing guidelines on how to successfully manage formal development with frequent checks would be beneficial for providing transparent control over the development process and the development itself.

Challenges and opportunities specific for Event-B

There are also some issues with Event-B, which need further investigation. For instance, the monolithic approach to modelling, which is characteristic for Event-B, creates difficulties in developing several components separately and then integrating them later on. It can also be an obstacle to separately refine certain parts of the system (in order to investigate them more) and then integrate them in the whole project. This can be a major impediment to refactoring, as well. Moreover, there is lack of support for team-work, which means that only one person can work on a particular specification at a time. This in turn, brings difficulties with globally distributed developments, where collaborative modelling and analysis would be one of the key-features.

The strength of the Event-B method lies in the extendable tool support. Consequently, the weaknesses are linked to the lack of tool assistance for specific needs. Below we

present the opportunities for further development of the Rodin tool, as well as discuss the social aspects of the formal development process, all in order to support the needs of adaptive development framework.

Team work and globally distributed developments

Team work and self-organising teams are one of the cornerstones of agile developments. The responsibility is being spread equally in the team, but at the same time team members have strong motivation for the common goal. Therefore, the hypothetical risk that "if everybody is responsible then in the effect nobody is" is limited. In Event-B *team work* is strongly limited by the lack of proper tool support for it. Naturally, one could implement the practice of regularly reviewing the specifications and models by senior and more experienced team members. Furthermore, writing and proving specifications, in addition to constructing models in pairs, as an idea similar to the one given in XP programming, could be an option. The latter, however, would involve more resources and effort. Therefore, there is a risk that this solution does not seem attractive for the industrial environment.

Another issue which to the same degree concerns the team work and globally distributed developments is how to handle requirements for unified understanding of a specification or a model. This is especially the case, when the model is built by several developers. There are several factors that can affect perception of an artefact, starting form cultural and social characteristics, through the policy and standards set in the organisation, to the skills and domain-knowledge of the developers in each team (or unit). Naturally, these can be somewhat unified by introducing some standards for writing specifications or for modelling. However, the human dynamics combined with the tooling element, i.e. appropriate communication between and within teams, supported by the tool, is the basis for a successful development.

Additionally, there is a need for a repository for *handling models and specifications and the changes* performed on these. This type of repository should be integrated with the Rodin platform, possibly in the form of yet another plugin, and enable managing models. Specifically, it should facilitate working on the development simultaneously, which would mean managing and (enable manually) resolving conflicts when the same part of the model was changed. Moreover, the tool should support dividing the model (branching), so that the developers can work on the model at the same time and merge the changes at the end of the day (or whenever they are ready).

Collaboration between self-organising and cross-functional teams

The collaboration between the teams was shortly mentioned in previous section, however here we would like to emphasise the need for orchestrating the communication in formal developments. In Event-B the developers are working separately and thus the communication between them is quite limited. This isolation impacts the exchange of knowledge, process of learning and negatively influences the team spirit. It is not possible to create a self-organising team, which would continuously aim for improvement and where roles of the team members are diverse, when there is not enough of encouragement and support not only from the development process, but also from the tool aspect. This could be solved by implementing yet another plugin or enhancing the existing plugins with "communication features", like notes, comments. Finally, some mechanisms for assistance in specification and model reviews should be provided.

Evolvability issue

The *evolutionary approach*, one of the characteristics strongly emphasised by agile methods, is not so obvious in the Event-B development. However, refinement as being part of the Event-B methodology can be considered as a mechanism that enables correct progress of the development of a system, thus allowing it to evolve. Therefore, formal development can be regarded as evolutionary.

An example of evolution in Event-B setting is presented by Butler et al. [60]. The authors propose an incremental refinement approach to the development of a flash-based file storing system, which is modelled in Event-B. The modelling involves two types of decomposition: horizontal refinement was used in feature augmentation and vertical refinement for structural refinement. Especially the first type of refinement resembles evolutionary development; instead of specifying everything in one level (possibility of high proof difficulty), the system features were split into sub-features, which were introduced in subsequent refinement steps. Hence, the evolutionary approach is present in the Event-B method and results in a model that is easier to construct and prove. The evolutionary approach can facilitate the creation of a well-proportionated system, which can even further contribute to establishing adaptable development frameworks.

Another topic related to evolution is the support of Event-B for *product lines*. Product lines and agile methods seem to be complementing each other, but mostly from the

point in the development, where the basic functionality of a system is created. At this point agile methods may be utilised and can facilitate the development of a variety of systems within a family of products. In the Event-B setting it is possible to see the concept of product lines. In [64] authors observed the need to develop a generic requirement set in order to create the core of a family of products, which can later be developed for subsequent system instantiations. This process is even more complicated because of the demand for a high level of verification by this safety-critical domain, and standards of the avionics industry. In the paper a case study is presented and given as an engineering method, validation and verification of generic requirements is proposed using domain engineering and formal methods techniques and tools. The development method used was related the B-Method (a predecessor of Event-B) and Event-B.

Requirements, refactoring and emergent design

Handling requirements and their volatility can occur to be problematic in all developments, regardless of the development process or methods used. This concerns also agile development, which, by definition, welcomes requirement change. Managing change in requirements, although often tackled swiftly by agile methods, can become an issue when it comes to more rigid developments. In formal developments requirements change entails not only changing specification and model, but also re-proving (part of) the system. As a consequence, the required effort is much bigger and negatively impacts the schedule and human resources of the project. Moreover, the maintainability of a system after deployment is at risk and the matter of reuse of (parts of) a model may be impossible.

Event-B development is of monolithic nature; however, some proposals of modular development in Event-B were already given in order to tackle real world problems (changes and reuse in large and complex systems, scalability issues) [65]. The authors extended the Event-B method with a method to decompose system models into components that can be independently developed. They emphasised benefits of managing complexity of a model, as well as reasoned on how their approach can be useful in e.g. formal product line development. The primary goal was to enable parallel development of several independent parts of the system as well as reuse formally developed modules in other developments. This approach is tool supported by Rodin platform via a modularisation plugin [66].

Yet another issue regarding requirements is their *prioritisation*. In agile development this is usually agreed upon when having meetings with stakeholders and team members. Therefore, customers and/or users have a large impact on what features will be

implemented in the following iteration. In the Event-B development the problem of prioritisation of requirements is dependent also on the refinement strategy. The well-proportionated system and development are meant to be facilitating the choice of which requirement to take-up first, however no specific guidelines are given on how to do it. Therefore, this topic should be investigated on the conceptual level and supported with a tool.

As for *refactoring*, an example of a balanced modelling and the role of a well-proportionated system can be found in [60], where the authors describe the incremental development of the Mondex system in Event-B. The authors show that strengthening and skilfully adding new invariants to the system can help in reducing the proving effort, meaning that the mathematical proof can be much easier or even done automatically. Moreover, in the presented modelling strategy the non-proved proof obligations and the interactive prover guided the developers in refactoring (creating new invariants and constructing a gluing invariant). Furthermore, the case of finding an invalid modelling assumption when a complete system is modelled and proved is described. This is one of the most common reasons for refactoring next to the change of requirements given by the user and/or customer. Based on a case study, the authors describe the modifications and reproving of the system, which were necessary to fix the wrong modelling assumption. Also in this case a "well-proportionated" system contributed to minimising the re-proving effort, as the change was reasonably well localised. Furthermore, the high degree of automatic proof that the approach entailed after the first refactoring (invariant related) meant that the majority of the re-proof required was automatic.

Another opportunity for the Event-B development lies in *emergent design*, a concept which focuses on delivering small pieces of working code with business value. When applying this concept the organisation begins delivering functionality and lets the design emerge. At the same time when the functionality is implemented, the refactoring is done. In consequence, at the end of a release cycle, the development fulfils at least a minimum amount of design requirements. In a situation when emergent design is not utilised, meaning that the refactoring of the development is left for "later", the "size" of the design would be significantly bigger. In Event-B the concept of emergent design would be transferred to the modelling phase, where small iterations of model development are delivered and the idea of refinement is utilised. However, one cannot let the design freely emerge, as the model and related proofs would be heavily dependent on refactoring. Thus, all the challenges that originate from refactoring in formal development settings would be directly transferred to the issues regarding the

emergent design. Finally, since models and their proofs are the foundations for the correct software, the model itself cannot be reduced – it has to be taken as is.

7. Contribution and next steps

In this report we presented a study of how well the agile software development can be applied in combination with a formal development. While the former provides guidelines and flexibility to the development process, the latter gives strong and well-established methods for building correct software. There is a variety of agile and formal methods available; therefore, they need to be chosen according to the type of development. In this report we first presented some general characteristics and benefits of each of the approaches, and then discussed a number of agile methods in more detail (DSDM, Kanban, Scrum and XP). Since we have a lot of experience with Event-B and it is well suited for industrial purposes, we focused on that formal method. We cross-checked the methodologies in order to investigate how feasible the characteristics of identified agile methods are for the Event-B setting.

We determined that no particular agile method can be taken as is, but needs to be tailored for the specific context. Essentially, building the FormAgi hybrid approach is based on choosing the characteristics (principles and practices) that are appropriate for the formal development. This may also differ between the selected formal methods. Therefore, agile characteristics chosen as feasible for Event-B may differ from the ones chosen for e.g. VDM.

Formal methods are mature enough and ready for being integrated in the development with other methods [12]. Moreover, agile methods are the most appropriate means for engineering such a merge [20]. In this paper 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 (b) relating agile principles, practices and values to the Event-B environment.

We also evaluated the issues that appear when agile methods are to be adopted in environments that are not inherently encouraging or friendly to make agile. Apart from investigating bottlenecks in transferring agile concepts to the formal setting, we provided suggestions on how to tackle principles and practices that cannot be directly shifted to the formal setting. Our proposals were based on the notion of tailoring the agile method to the formal method to be used, here Event-B. The idea is to make the

formal method as feasible as possible by establishing and endorsing the adaptive development framework.

Tooling is a vital issue at every stage of (software) system development. The industrial take-up for formal methods, herein Event-B, was heavily dependent on tool support. Establishing adaptive development framework with respect to the tooling aspect is on one hand straightforward, as many of the principles and practices can be directly implemented due to existing tool support. On the other hand, in order to be able to fully utilise the benefits that come with agile approaches, some additional tool support would be required. We investigated this aspect in our report, too.

Future work directions

The hybrid FormAgi approach presented in this paper is still at the conceptual level. We would like to identify benefits and limitations to our approach when merging agile development principles with formal development in order to validate our ideas. At the same time, we would like to highlight the issues which may rise when applying agility to projects traditionally considered as being non-agile. Our purpose is to show the advantages and risks that can be involved when applying the framework, in order to enable and facilitate the decision making in organisations, which plan to make their formal development more flexible.

We aim at providing evidence on feasibility of the proposed adaptive development framework. Therefore we plan to perform empirical research on the proposed approach by creating an experimental setup, if the collaboration with industry allows, by performing a case study. As a secondary option we plan to establish formal experiment within a group of students, where the goal is to construct a safety-critical system (or a part of a safety-critical system, e.g. a component) with the use of a formal method, preferably Event-B. The development process is to be of an agile type, i.e. one of the discussed agile methods Scrum, Kanban, DSDM, XP or the mix of their principles and practices. Thus, a development setting would be created, which would include formal modelling as a part of the development. We would monitor the development itself with quality metrics.

The practicality of adaptive development framework would be assessed in quantitative and qualitative manner and the measurements would be used to isolate the factors that impact flexibility, efficiency and effectiveness in the proposed hybrid FormAgi

approach. This would contribute to investigating in practice how to balance flexibility provided by agile methods and control given by formal methods.

Acknowledgements

This work has been done within the Academy of Finland funded project ADVICeS. Authors would like to thank Prof Michael Butler and Dr Colin Snook for the valuable discussions on the Event-B method and formal modelling, as well as on the use of agile methods in industry.

References

[1] P. J. Ågerfalk i B. Fitzgerald, „Flexible and distributed software processes: old petunias in new bowls," *Communications of the ACM,* tom 49, pp. 27-34, 2006.

[2] B. Boehm i R. Turner, Balancing Agility and Discipline: A Guide for the Perplexed, Addison-Wesley, 2003, p. 304.

[3] B. Boehm, „Software Engineering," *IEEE Transactions on Computers,* 1976.

[4] J.-R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.

[5] „Manifesto for Agile Software Development," 11-13 02 2001. [Online]. Available: http://agilemanifesto.org/. [Data uzyskania dostępu: 03 12 2013].

[6] J.-F. Monin i M. G. Hinchey, Understanding Formal Methods, Springer, 2003.

[7] J. B. Almeida, M. J. Frade, J. S. Pinto i S. Melo de Sousa, Rigorous Software Development: An Introduction to Program Verification, Springer, 2011, p. 307.

[8] M. Bernardo i P. Inverardi, „Formal Methods for Software Architectures," w *Third International School on Formal Methods for the Design of Computer, Communication and Software Systems: Software Architectures, SFM 2003*, Bertinoro, 2003.

[9] T. Dingsøyr, S. Nerur, V. Balijepally i N. B. Moe, „A decade of agile methodologies: Towards explaining agile software development," *Journal of Systems and Software,* tom

85(6), pp. 1213-1221, 2012.

[10] T. Dingsøyr i N. Brede Moe, „Research challenges in large-scale agile software development," *SIGSOFT Software Engineering Notes,* tom 38 (5), pp. 38-39, 2013.

[11] J. Woodcock, P. G. Larsen, J. Bicarregui i J. Fitzgerald, „Formal Methods: Practice and Experience," *ACM Computing Surveys,* tom 41 (4), pp. 1-40, 2009.

[12] P. G. Larsen, J. S. Fitzgerald i S. Wolff, „Are Formal Methods Ready for Agility? A Reality Check.," w *Second International Workshop on Formal Methods and Agile Methods*, Pisa, 2010.

[13] J. P. Bowen, M. Hinchey, H. Janicke, M. Ward i H. Zedan, „Formality, Agility, Security, and Evolution in Software Development," *Software Technologies,* pp. 86-89, October 2014.

[14] S. E. Black, P. P. Boca, J. P. Bowen, J. Gorman i M. G. Hinchey, „Formal versus agile: Survival of the fittest," *IEEE Computer,* tom 49 (9), p. 39–45, 2009.

[15] S. Wolff, „Scrum Goes Formal: Agile Methods for Safety-Critical Systems," w *Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, Zurich, 2012.

[16] M. Löwe, „Formal Methods in Agile Development," *Special issue of Electronic Communications of the EASST: Graph and Model Transformation 2010,* tom 30, pp. 1-6, 2010.

[17] A. Wils, S. Van Baelen, T. Holvoet i K. De Vlaminck, „Agility in avionics software world," w *Extreme Programming and Agile Processes in Software Engineering (XP)*, 2006.

[18] K. Gary, A. Enquobahrie, L. Ibanez, P. Cheng, Z. Yaniv, K. Cleary, S. Kokoori, B. Muffih i J. Heidenreich, „Agile methods for open source safety-critical software," *Software - Practice and Experience,* tom 41, p. 945–962, 2011.

[19] L. Williams i A. Cockburn, „Agile software development: it's about feedback and change," *computer,* tom 36, pp. 39-43, 2003.

[20] R. F. Paige i P. J. Brooke, „Agile Formal Method Engineering," w *Integrated Formal Methods*, Eindhoven, 2005.

[21] „RECOMP Project - Reduced Certification Costs Using Trusted Multi-core Platforms,"

[Online]. Available: http://recomp-project.eu/.

[22] R. W. Butler, „NASA LaRC Formal Methods Program," NASA, 2001. [Online]. Available: http://shemesh.larc.nasa.gov/fm/fm-what.html. [Data uzyskania dostępu: 04 11 2013].

[23] M. C. Holloway, „Why Engineers Should Consider Formal Methods," w *AIAA/IEEE16th Digital Avionics Systems Conference*, 1997.

[24] M. Olszewska, On the Impact of Rigorous Approaches on the Quality of Development, Turku Centre for Computer Science, 2011, pp. 1-218.

[25] A. Avizienis, J.-C. Laprie, B. Randell i C. Landwehr, „Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing,* tom 1 (Issue 1), pp. 11-33, 2004.

[26] I. O. f. Standardization, ISO 9126-1 - Software engineering - Product quality — Part 1: Quality model, ISO, 2001.

[27] M. Bundschuh i C. Dekkers, The IT Measurement Compendium. Estimating and Benchmarking Success with Functional Size Measurement, Springer, 2008, p. 680.

[28] E. W. Dijkstra, „A Constructive Approach to the Problem of Program Correctness," *BIT Nmerical Mathematics,* tom 8(3), pp. 174-186, 1968.

[29] N. Wirth, „Program Development by Stepwise Refinement," *Communications of the ACM,* tom 14(4), pp. 221-227, 1971.

[30] R.-J. Back, On the Correctness of Refinement Steps in Program Development, Åbo Akademi, Department of Computer Science, 1978.

[31] C. Metayer, J.-R. Abrial i L. Voisin, „Event-B Language, RODIN Deliverable 3.2 (D7)," 2005.

[32] C. Snook i M. Waldén, „Refinement of Statemachines using Event-B semantics," w *Formal Specification and Development in B*, Besançon, 2006.

[33] M. Waldén i K. Sere, „Reasoning about Action Systems using the B-Method," *Formal Methods in System Design,* tom 13, pp. 5-35, 1998.

[34] R.-J. Back i J. von Wright, Refinement Calculus: A Systematic Introduction. Graduate Texts in Computer Science, Springer Heidelberg, 1998.

[35] S. Yeganefard i M. Butler, „Problem Decomposition and Sub-Model Reconciliation of Control Systems in Event-B,” w *IEEE International Workshop on Formal Methods Integration*, Turku, 2013.

[36] J.-R. Abrial, „“Extending B without Changing it (for Developing Distributed Systems),” w *Proceesings of 1st Conference on the B Method*, Nantes, 1996.

[37] R.-J. Back, Refinement Calculus, Part II: Parallel and reactive programs. Stepwise Refinement of Distributed Systems, Åbo Akademi, 1990.

[38] R.-J. Back i R. Kurki-Suonio, „Decentralization of process nets with centralized control,” *2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing,* pp. 131-142, 1983.

[39] R.-J. Back i K. Sere, „From modular systems to action systems,” *Software - Concepts and Tools,* tom 17, pp. 26-39, 1996.

[40] J.-R. Abrial, The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.

[41] Event-B, „Home of Event-B and the Rodin Platform,” 2008. [Online]. Available: http://www.event-b.org/index.html. [Data uzyskania dostępu: 04 11 2013].

[42] „Rodin Platform,” 2006. [Online]. Available: http://www.event-b.org/platform.html or directly at http://sourceforge.net/projects/rodin-b-sharp/. [Data uzyskania dostępu: 2010].

[43] R. Hähnle, „Agile Formal Methods,” w *6th International KeY Symposium*, Nomborn, 2007.

[44] A. Alliance, „Guide to Agile Practices,” 2011. [Online]. Available: http://guide.agilealliance.org/. [Data uzyskania dostępu: 04 12 2013].

[45] D. Consortium, The DSDM Atern Handbook, DSDM Consortium, 2013.

[46] D. J. Tudor i I. J. Tudor, The DSDM Atern Student Workbook: A Guide to the Definitive Agile Framework, Galatea Training Services Ltd, 2010, p. 224.

[47] D. J. Anderson i D. G. Reinertsen, Kanban: Successful Evolutionary Change for Your Technology Business, Blue Hole Press, 2010, p. 278.

[48] P. Klipp, Getting Started with Kanban, Amazon Digital Services, 2014, p. 529KB.

[49] H. Takeuchi i I. Nonaka , „New New Product Development Game,” *Harvard Business*

Review, tom 86116, pp. 137-146, 1986.

[50] K. Beck, Extreme Programming Explained, Addison-Wesley, 2000, p. 224.

[51] K. Beck, Extreme Programming Explained: Embrace Change, 2nd edition, Addison-Wesley Professional, 2004, p. 224.

[52] C. Larman i V. Basili, „Iterative and Incremental Development: A Brief History," *Computer,* tom 36 (6), p. 47–56, 2003.

[53] B. Boehm, „Get Ready for Agile Methods, with Care," *IEEE Computer,* tom January, 2002.

[54] S. Gruner, „Innovations in Systems and Software Engineering," w *2nd Workshop on Formal and Agile Methods*, 2009.

[55] „DEPLOY Project - Industrial deployment of system engineering methods providing high dependability and productivity," 1 February 2008. [Online]. Available: http://www.deploy-project.eu/. [Data uzyskania dostępu: 18 April 2011].

[56] H. Mochio i K. Araki, „VDM++ as a Basis of Scalable Agile Formal Software Development," w *9th Overture Workshop on VDM (colocated with FM2011)*, Limerick, 2011.

[57] A. De Lucia i Q. Abdallah, „Requirements Engineering in Agile Software Development," *Journal of Emerging Technologies in Web Intelligence,* tom 2 (3), pp. 212-220, 2010.

[58] J.-R. Abrial, „Mathematical Models for Refinement and Decomposition," *Modeling in Event-B: System and Software Engineering,* 2009.

[59] M. Butler, „Decomposition Structures for Event-B," w *Integrated Formal Methods (iFM2009)*, 2009.

[60] M. Butler i D. Yadav, „An incremental development of the Mondex system in Event-B," *Formal Aspects of Computing,* tom 20 (1), pp. 61-77, 2008.

[61] „RODIN - Rigorous Open Development Environment for Complex Systems," [Online]. Available: http://rodin.cs.ncl.ac.uk/.

[62] ProB, „The ProB Animator and Model Checker," 18 11 2013. [Online]. Available: http://www.stups.uni-duesseldorf.de/ProB/index.php5/Main_Page. [Data uzyskania dostępu: 20 01 2014].

[63] B. Studio, „BMotion Studio,” 23 11 2012. [Online]. Available: http://www.stups.uni-duesseldorf.de/bmotionstudio/index.php/Main_Page. [Data uzyskania dostępu: 21 01 2014].

[64] C. Snook, M. Poppleton i M. Johnson, „Rigorous engineering of product-line requirements: a case study in failure management,” *Information and Software Technology,* Tomy %1 z %250(1-2), pp. 112-129, 2008.

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

[66] A. Iliasov, „Modularisation Plugin - Event-B,” 06 09 2010. [Online]. Available: http://wiki.event-b.org/index.php/Modularisation_Plug-in. [Data uzyskania dostępu: 05 02 2014].

[67] D. West, T. Grant, M. Gerush i D. D’Silva, „Agile Development: Mainstream Adoption Has Changed Agility. Trends in real-World Adoption of Agile Methods.,” Forrester Research, 2010.

[68] D. Bustard, G. Wilkie i D. Greer, „Towards optimal software engineering: learning from agile practice,” *Innovations in Systems and Software Engineering,* tom 9 (3), pp. 191-200, 2013.

[69] K. Damchoom i M. Butler, „Applying Event and Machine Decomposition to a Flash-Based Filestore in Event-B,” w *Proceedings of SBMF 2009*, Gramado, 2009.

[70] A. Gondal, M. Poppleton i C. Snook, „Feature composition - towards product lines of Event-B models,” w *1st International Workshop on Model-Driven Product Line Engineering*, Twente, 2009.

[71] T. Dybå i T. Dingsøyr, „Empirical studies of agile software development: A systematic review,” *Information and Software Technology,* Tomy %1 z %250(9-10), pp. 833-859, 2008.

[72] O. Hazzan i Y. Dubinsky, Agile software engineering, Springer, 2008.

[73] M. Cohn, Succeeding with Agile: Software Development Using Scrum, Addison-Wesley Professional, 2009.

[74] D. I. Sjøberg, A. Johnsen i J. Solberg, „Quantifying the Effect of Using Kanban versus Scrum: A Case Study,” *IEEE Software,* tom 29, pp. 47-53, 2012.

[75] V. Heikkilä, M. Paasivaara, C. Lassenius i C. Engblom, „Continuous Release Planning in a Large-Scale Scrum Development Organization at Ericsson," w *Proceedings of the 2013 International Conference, XP2013 on Agile Processes in Software Engneering and Extreme Programming*, Vienna, 2013.

[76] V. Rajlich, „Changing the paradigm of software engineering," *Communications of the ACM,* tom 49 (8), pp. 67-70, 2006.

[77] S. Hallerstede, M. Jastram i L. Ladenberger, „A Method and Tool for Tracing Requirements into Specifications," *Science of Computer Programming,* Available online April 2013.

[78] ISO/IEC, „ISO/IEC 25000:2005 - Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE," ISO/IEC, Geneva, 2005.

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Information Technologies

Turku School of Economics
- Institute of Information Systems Sciences

ISBN XXX-XX-XXXX-X

ISSN 1239-1891