Marta Olszewska | Sergey Ostroumov | Marina Waldén

Synergising Event-B and Scrum - Experimentation on a Formal Development in an Agile Setting

Turku Centre *for* Computer Science

TUCS Technical Report
No 1152, February 2016

Synergising Event-B and Scrum - Experimentation on a Formal Development in an Agile Setting

Marta Olszewska

Åbo Akademi University, Faculty of Natural Sciences and Engineering

Sergey Ostroumov

Åbo Akademi University, Faculty of Natural Sciences and Engineering

Marina Waldén

Åbo Akademi University, Faculty of Natural Sciences and Engineering

Abstract

This paper explores the opportunities and challenges of the synergy between formal and agile methods, in particular Event-B and Scrum. We fine tune Scrum process in order to fit the specificity of formal development. We then perform formal modelling of a part of the landing gear system within scrum development process. The development serves as hands-on investigation for the quantitative and qualitative analysis of the applicability of such merge.

Our findings show that there is a great potential in this synergy, especially in terms of improving comprehension of requirements and understandability of the system domain, and thus positively impacting the quality and correctness of the system being built. Furthermore, the communication within the team is enhanced, which leads to fine-tuning the development approach and smoothening the modelling process. Finally, the rules and ideas behind formal modelling can be closely associated with agile philosophy, as the latter is flexible enough to handle the rigour necessary to create a correct system.

Keywords: Event-B, Scrum, Formal Methods, Agile Development Process, adaptable development framework, FormAgi

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

Table of contents

1. Introduction 3

1.1. Related Work 4

2. FormAgi Framework 5

2.1. Event-B 5

2.2. Agile Methods, Principles and Practices 7

2.3. Scrum 8

2.3.1. Process 9

2.3.2. Roles 10

2.3.3. Communication – meetings 11

2.3.4. Scrum and Event-B – possibilities of synergy and challenges 12

3. Experimentation 13

3.1. Landing Gear System (LGS) Case Study 13

3.2. Scrum Process for Event-B Development 14

3.3. Experimental Setting 16

3.4. Event-B Development of the LGS Case Study 18

3.4.1. First Iteration (Sprint 1) 19

3.4.2. Second Iteration (Sprint 2) 20

3.4.3. Case Study – Summary of the Development 24

4. Monitoring and Analysis 25

4.1. Development Process 25

4.2. Meetings 26

4.3. Development Effort 27

4.3.1. Proving Effort 29

4.4. Model Measurements 31

4.5. Observations of the Developer 34

5. Conclusions 36

5.1. Validity of Experimentation 36

5.2. Implications of Research and Future Work 37

1. Introduction

Formal methods exist for more than 40 years [1], while agile methods are dated back to Agile Manifesto (2001) [2]. There are many formalisms serving formal development of a system, each suitable for a certain purpose, be it development of computer-based systems, applications that demand safety, security or business integrity. The same applies for agile methods, where particular practices and values can be selected according to the needs of the context the method is supposed to be used in.

Formal methods are known for assuring quality and correctness of critical systems [3], while agile methods became popular due to enabling rapid, flexible and evolutionary development with strong emphasis of its social aspect (team work and communication) [4]. Clearly, the mixture of these methods would create a volatile and adaptable environment that would ensure high quality of the system development process. This combination would benefit from providing transparency in the project by increasing the interaction between team members and improving comprehension of the requirements of the system to be developed.

The merge of these two highly opposing approaches has been discussed on the conceptual level several years after agile methods were proclaimed, for instance in [5]. However, only recently the discussion became more sophisticated and gained visibility 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 potential of such synergy in software development is well described in [6].

We already investigated the merge of agile and formal methods in [7], which resulted in establishing the FormAgi framework. We identified the aspects of agile methods that can act as facilitators for a formal development, as well as determined challenges that can appear when committing to particular agile method. We also determined that for the formal method that we use in our work (Event-B), Scrum development process would be the most suitable agile method. We use Event-B as it supports iterative systems creation in a correct-by-construction manner, i.e., we are able to model software, hardware and environment. Scrum seemed to be a suitable match to Event-B and its idea of stepwise development of a system, as it is time-framed, iterative and incremental, among others. Our goal is to bring all the best from these two and combine it, so that it can lead to development of a high quality and correct system in an adaptive, flexible, continuous and timely way.

We are aware that conceptual study is not sufficient to convince formal modellers and agile enthusiasts that the formal-agile mix is not only possible, but can enhance the development in terms of timeliness and adaptability, on the one hand, and quality and correctness, on the other. To validate our claims, we perform a hands-on experimentation on Event-B development in Scrum setting. We use a case study from the aerospace domain (Landing Gear System – later referred to as LGS) to provide the evidence on how this synergy functions.

This paper is structured as follows: first we present related work. Then in section 2 we give the background for the formal and agile method of our choice. Section 3 describes our experimentation, including the portrayal of the LGS case study, the description of development process and how it is fine-tuned to our setting, as well as explanation of the development itself. In section 4 we present and analyse the qualitative and quantitative data regarding development process and the created model. Finally, we conclude with the discussion on validity of our examination, as well as implications of our work and present our research plans for the future.

1.1. Related Work

The potential of a synergy between formal and agile methods for the development of software was described in [6]. The authors observe that the merge, if applied cautiously, can minimise change-related problems and aid the 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. This paper focuses on the former and provides our vision of possible improvement of this mix.

The LGS case study is well described in [8] and modelled using different formalisms and approaches, just to mention a co-simulation environment for Rodin based on the Functional Mock-up Interface standard and ProB animator for Event-B [9]. Majority of the papers considering the LGS case study focussed on the development methods and how to use them in a smart way, while only a few of them mentioned the process perspective of the development, in particular refinement [10]. In our work we concentrate on the development process and investigate how it can act as a facilitator for the development.

Usually, the V-model [11] is the development process used for critical systems development, as it conforms to the recommendations of standards. The development process that we utilised for our development of the LGS case study, Scrum, is far from being the first choice for any type of rigorous developments. However, with our motivation we follow [12], where it is stated that "Although at first glance, agile and formal methods seem incompatible, we see many opportunities to combine them effectively." The claim in the quoted paper is not supported with any evidence, though. In our work we demonstrate how this synergy works and provide qualitative and quantitative data to support our supposition.

Yet another conceptual solution is given in [13], where authors use XP agile method and integrate it with practices of formal methods, specification and verification (VDM and Z). They perform a formal experiment within academic context and analyse the time of system development phase, error rate and product quality within planning, designing and implementation phases. In our work we use case study as an experimentation technique to evaluate effort, proving-related data and complexity of the model, focusing on the early stage development (requirements, specification and modelling).

To the best of our knowledge no other experimentation on the merge of Event-B and agile methods, specifically Scrum, was performed. Our earlier work [14], concentrated on modelling approach, which starts the development from multiple abstractions and then merges the development into a single refined model of the complete system and by that creates possibility of parallel team work. We have not collected any data during this development, except model metrics, and, therefore, were not able to report any substantial findings from this case study. Here we performed the hands-on experimentation in such a way that we monitored the development and the created model, as well kept a diary with the notes from our meetings. Our goal was to be able to analyse our observations and provide an insight on the realisations and challenges of formal-agile synergy.

The methodology for experimentation in software engineering is very well presented in [15], where authors describe the methods of empirical investigations, guidelines how to choose an appropriate technique and perform experimentation. In our setting we were not able to completely follow the instructions provided there for case studies, as we were not able to compare one situation (method, development, etc.) to another. Instead, we performed a singular small-scale experimentation (a pilot study) with all the steps required for a case study.

The methodology for conducting formal experiments in the field of the agile formal methods is presented in [16]. The authors mainly concentrate on the structure of an experiment, i.e., how to design it. They also emphasise that there are some works on experiments in formal setting; however, there is a need for evidence on the developments regarding the combination of formal and agile approaches. We follow this rationale and consider it as a key-driver for our research.

One of our goals is to make the development with Event-B more flexible and adaptable, as well as "encouraging" and efficient for the (new) users. Our idea is based on merging agile philosophy (process aspect) into the formal development (methodology aspect). Nevertheless, there is a plethora of related work attacking this issue from other perspectives, just to mention visualisation, modularisation or decomposition.

Snook and Butler [17] have proposed an approach to merge visual UML [18] with B [19] by the use of a UML-B profile which provides specialisation of UML entities to support refinement. The idea of this approach is to compensate the lack of precise semantics of the former and to reduce the training effort required to overcome the mathematical barrier of the latter. The approach has been extended to Integrated UML-B [20] which allows the changes in UML mode to be visible in B mode and vice versa.

A modularization mechanism to support scalability of Event-B modelling has been proposed by Iliasov et al. [21]. The authors consider sequential systems whose functionality is distributed among several components. The authors propose to extend Event-B with (atomic) operation calls and introduce the notion of modules (i.e., components) which contain groups of callable operations. According to the authors, their approach can be seen as a special type of the decomposition approach proposed by Abrial [22]. The goal is to split a monolithic model into sub-models, each of which can be further developed separately in parallel. However, once all the modules contain the necessary level of detail, they can be composed back into a system. The composition mechanism is supported by the corresponding proofs.

2. FormAgi Framework

In our previous work we investigated several of agile methods with respect to their feasibility in development of critical systems. [7] We explored the values, principles and practices of agile development methods and placed them in the context of formal, refinement-based developments. We provided a mapping between the characteristics of these two, which established FormAgi [7], a high-level framework consisting of (i) guidelines on what concerns should be tackled before committing to a certain agile method and (ii) pointers in which aspects an agile method can be a facilitator in the formal development.

We chose to use Event-B as a formal method within an agile process. Although Event-B is not considered as a lightweight approach, we want to examine if by conducting the development in small refinement steps [22], and by decomposing the models [21], as well as using component-based visual development, it can be applied in a rapid manner.

In the following subsections we first present short overview of Event-B, then follow with a description of agile methods and illustrate Scrum as an agile method of our choice.

2.1. Event-B

Event-B [23] [24] is a formal method and modelling language for stepwise system-level modelling and analysis, based on the Action Systems formalism [25] [26] [27]. It is dedicated to model complete systems, including hardware, software and environment [28]. It is derived from the B-Method [29], with which it has several commonalities, e.g., set theory and the refinement approach.

Event-B employs refinement to represent systems at different levels of abstraction. It enables us to gradually introduce more details to the constructed system and to represent new levels of a system with more functionality. The consistency between the refinement levels is verified by mathematical proofs. 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 [30], an Eclipse based tool, which is an open source “rich client platform” that is extendable with plug-ins.

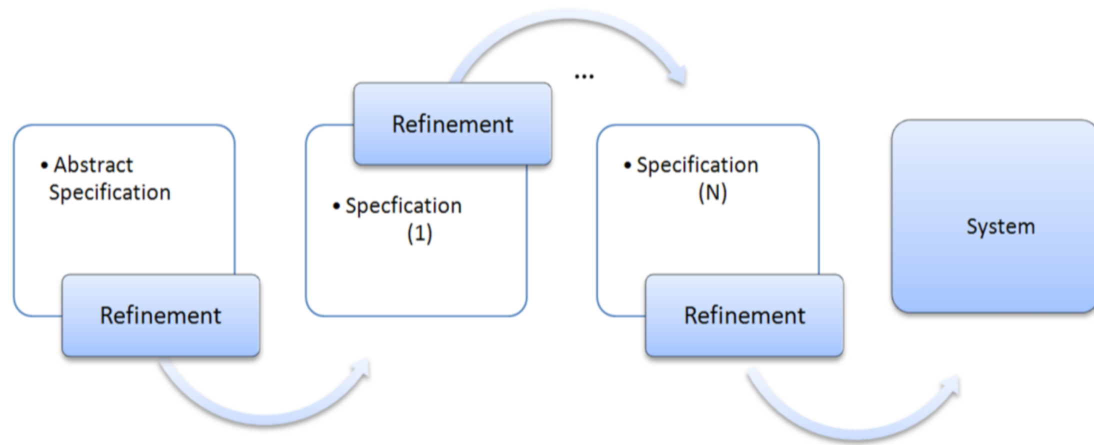


Figure 1 Refinement process

The formal development starts from modelling an abstract specification from a set of requirements and then refining it in a number of steps (as presented in Figure 1). Each consecutive step is called REFINEMENT. It enables tracking and controlling the refinement chain and the modelling process.

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 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. A more abstract machine can be refined by another, more concrete one. The context, on the other hand, encapsulates the sets and constants of the model with their properties given by axioms and theorems. This static part of the specification can also be refined, which is indicated by the EXTENDS clause. The relation between machines and contexts, as well as the refinement relation for these is presented in Figure 2.

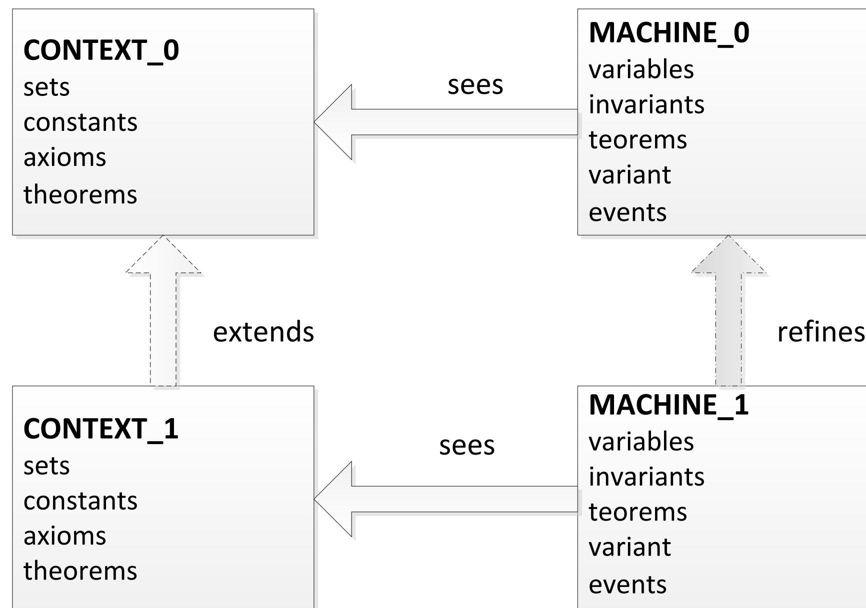


Figure 2 Refinement in Event-B (following [23])

In our work we want to benefit from the way software systems are developed with Event-B. The gradual introduction of properties to the system enabled by refinement allows us to comply with the iterative and incremental nature of agile development. Moreover, we can handle complexity issues more efficiently by decomposing the problems to simpler and smaller ones. Finally, the quality aspect of development is assured by the correct-by-construction approach and strengthening the work on requirements (elicitation). Furthermore, modelling and proving properties of the system contributes to building a well-defined system and diminishing the risk of unnecessary re-work due to misunderstood or not sufficiently described requirements.

2.2. Agile Methods, Principles and Practices

Agile software development is a concept that has been on the IT stage already for almost 25 years. Agile manifesto [2], which initiated the agile movement in software systems development, was a mixture of old ideas, new ideas, and transmuted old ideas. It emphasised the social aspect of development, e.g., close collaboration within the development team, as well as between the developers and business experts. It pointed out that the face-to-face communication is more efficient than written documentation. Moreover, it brought up the idea of small, self-organizing teams, where each team member provides his expertise to the development, but does not have to be an expert in all the areas required by the development.

One of the most important aspects identified in the manifesto was the frequent delivery of new deployable business value. Finally, the Manifesto mentioned the issue of volatile requirements, which is inevitable to every development, and ways to handle them so that the risks are mitigated.

Agile process definition

- An evolutionary
- Highly collaborative
- Quality-focused approach
- To software development
- Where potentially shippable working software is produced on a regular basis

Practice

- Work closely with stakeholders, ideally on a daily basis
- Be self-organizing within an appropriate governance framework
- Regularly reflect on how the team works together and then act to improve on its findings
- Produce working software on a regular basis
- Do continuous regression testing (and better yet, take a test-driven development approach)

Figure 3 Definition of agile process and the practical take up on these

In Figure 3 on the left hand side we present the definition of agile process, whereas on the right hand side we show a practical take up on the definition. As presented, major emphasis is placed on the communication and collaboration, as well as the delivered values in a form of working quality code.

The Manifesto is based on 12 principles and recognises certain practices, in order to assist in many areas of development, like requirements, design, modelling, coding, testing, project management, quality assurance etc. All of these serve for facilitating communication and collaboration, boosting team morale, supporting actionability, adaptability, flexibility and quality of development and are oriented towards continuous improvement. We list these principles and practices in Figure 4.

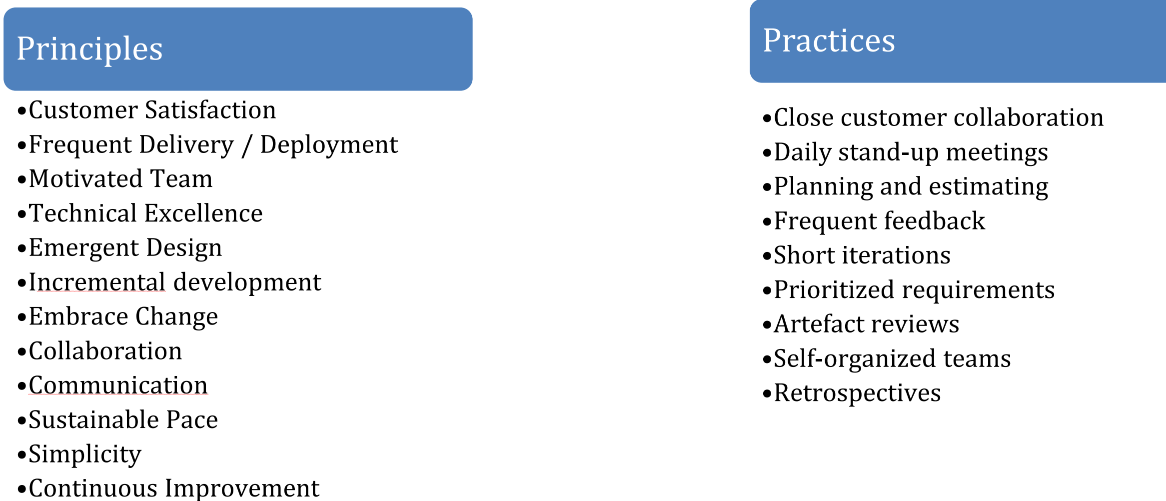


Figure 4 Principles and practices in agile software development

Agile methods are in fact a family of methods, just to mention Scrum [31], Lean [32], Kanban [33] [34], XP [35] [36], DAD [37], DSDM [38] [39], sharing certain characteristics, presented in Figure 5. Agile methods differ with respect to their focus on different aspects of software lifecycle. While some focus on the agile practices, like XP, others focus on managing software projects, e.g., Scrum. There are also methods providing full coverage over the development life cycle, like DSDM, or act as higher-level frameworks for other agile methods, DAD.

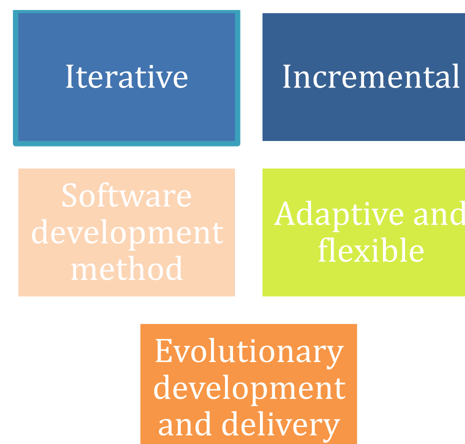


Figure 5 Common characteristics of agile approaches

To summarise, agile methods attempt to provide means for a flexible and transparent development, which is responsive to change and oriented towards customers satisfaction by meeting stakeholders' needs within the given time. Moulding the development process according to agile methods helps in dealing with development complexity and supports social aspects of IT project, i.e., by facilitating collaboration.

2.3. Scrum

Among plethora of agile methods we chose to use Scrum in our work. Not only is Scrum a flexible and complete development strategy, but also it gives a well-described working process with respect to, e.g., roles and interaction between the team members, time limitations of work, supports communication and

collaboration within this process, and it presents some similarities with Event-B. In this section we first illustrate the Scrum process (Section 2.3.1), then depict roles of the team members in the process (Section 2.3.2) and describe how the communication is handled (Section 2.3.3). Finally, we provide reasons why we choose this agile method (Section 2.3.4).

2.3.1. Process

Scrum development process is strongly time-framed, not only with respect to the development, but also with respect to communication. Moreover, advice is given on how to handle the development regarding the requirements: the way they are structured and managed during the development (how they should be described, dividing to “sub-requirements”), as well as when they ought to be implemented (prioritising and their scheduling in the development).

A typical Scrum process is shown in Figure 6, where product backlog and sprint backlog contain requirements (features) to be implemented in the project and during current iteration, respectively. Moreover, two iterations, long and short are present in the diagram, in the given example lasting 30 days and 24 hours, correspondingly. The meetings in Scrum process are marked with green arrows and are described in more detail in Section 2.3.3. At the end of each sprint a working version of software is expected to be shown to the customer.

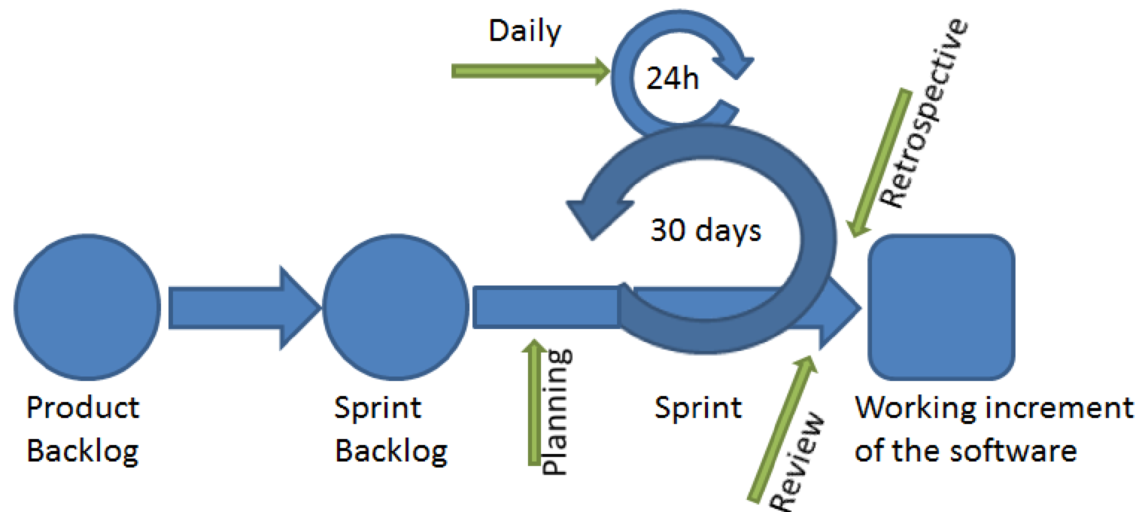


Figure 6 Scrum development process

When choosing Scrum as a development process one needs to answer three questions:

1. What product are we building?
2. What are we doing in this iteration?
3. How well are we doing?

These questions help to keep the project on track, providing the higher-level vision of the product being developed, lower-level viewpoint on the actual work in progress and finally the evaluation of progress itself.

The first question is tackling requirements and their management. Requirements, which are named in agile development as *features*, have certain form of *user stories*: "As a (role), I want (feature), so that (benefit)". A collection of user stories are placed in the *product backlog*. Product backlog essentially describes the features which will enable the product to be of value. Finally, at this stage the planning which user stories go to which release takes place. Note that release means an executable version of project to be shown to the stakeholders.

The second question to a large degree concerns the roles of team members and their tasks in the project. The roles are described in the following subsection. As an answer to this question, the plan for the current iteration is being agreed on. First, the scenarios are chosen and are being moved from product backlog to *release backlog* (task of product owner). Release backlog is a collection of scenarios chosen for a specific iteration. Then, the prioritisation of user stories and estimates for the work for each item is being done by the team. Additionally, larger user stories are broken down to smaller, manageable requirements. Afterwards, it is decided on how much it can be committed to during cycle (team). Finally, the prioritised stories are placed to *sprint*.

Sprints usually take 2-30 days and there are 2-12 sprints in release. The main concept of a sprint is to get a subset of release backlog to ship-ready state. This means that after a sprint the product should be fully tested and all of the features of the sprint ought to be complete. Any items that are left unimplemented in the sprint backlog are returned to the product backlog at the end of the sprint.

While first two questions tackle what needs to be implemented and how to do it, the last question is after the assessment of the actual progress of the project. This is well described with the use of *burndown chart*. This type of chart shows the relation between the time in the sprint and effort estimated for implementing the feature, shown on x and y axis respectively. An example of a burndown chart is given in Figure 7. In the figure we see a descending red line, oscillating closely to the blue line estimating the average progress of the project. The red plot optimally reaches a "Done" status, which means that the work in the current iteration has been completed by implementing all features planned in the sprint backlog. The desirable scenario is when the plot showing work in progress is gradually descending and ultimately getting to the zero point, where all the functionality planned is implemented and tested to an executable state.

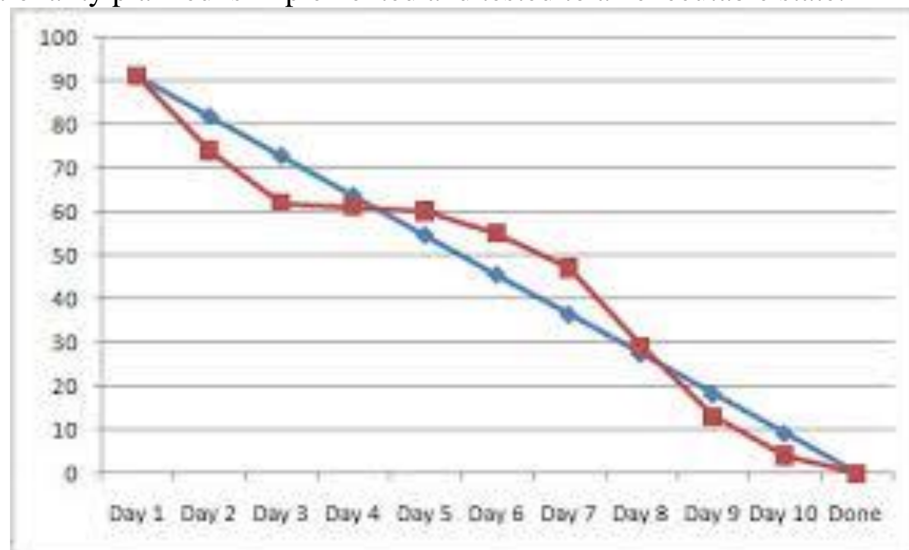


Figure 7. Example of a burndown chart

2.3.2. Roles

Apart from defining time-frames for the development process, Scrum defines also three roles in the project, which are represented by the people involved in the development:

- *Product owner*, who makes sure which features are going into the product backlog; he also represents a user and/or a customer of a product and the business.
- *Scrum master*, whose responsibility is to ensure that the project progresses smoothly and sees to that everybody in the team has all that is necessary to make the job done, e.g., sets up meetings. He can be compared to a manager in traditional development processes.

- *The Team*, which is a cross-functional and self-organising group of people, consisting of developers, testers, executives, etc.

Product Owner's responsibility is to act as the representative of the stakeholders and the customers. It usually is the customer himself, its designated representative, or an executive of the company that produces the software, i.e., a person that finances the development. Product Owner has a final say in negotiations regarding the functionality of the product. The presence of the Product Owner is not required during the development process. This implies assuring other means of contact in case the Product Owner is not constantly available.

Scrum Master, functioning typically as a project manager, is responsible for managing and maintaining the development process and for providing necessary resources for the Team. He is to ensure that there are no hindrances to deliver the product from the development side.

The Product Owner specifies his requirements according to which the Team implements the software. This way, ideally, the cross-functionality in the area of software development can be achieved within the Team. Typically, one or more Teams are formed consisting of, e.g., programmers, designers, architects, product-line managers, and testers. All team members are equally and jointly responsible for delivering the product. Furthermore, the Teams are self-managing, meaning that they solve the management issues, regarding for instance the division of work, internally. The decision about the number of members in each Team is left for the Teams themselves; however, it is advised that the team size should not exceed 6 persons [40] [41].

2.3.3. Communication – meetings

Agile methods emphasise that communication is the cornerstone of smooth development, regardless if it is within the team or between the team and the stakeholder. Communication can be realised in many ways, for instance, using messenger type of applications, e-mails, note-cards while meetings. Due to peoples' perception and ways of acquiring information, meetings (especially face-to-face) seem to be the most informative due to combination of knowledge-carriers, e.g., the body language, facial expressions and the tone of voice.

Scrum has very well defined set of meetings, i.e., sprint planning meeting, daily scrum, sprint review and sprint retrospective (see Figure 8). Each meeting aims to provide better information flow and supports the agile ideas of improvement. Moreover, the meetings can be used as a way of controlling the development and its process in a less formal and demanding manner. In the rest of this subsection we describe the meetings in Scrum, their purpose and the people involved.

Figure 8. Meetings in Scrum

Every long iteration (sprint) starts with a *sprint planning meeting*, where it is decided what items from the product backlog should be included in the sprint backlog. It is also agreed on how the work should be organised within the team. During the meeting the Product Owner assigns priorities to the items in the product backlog. Then, the Team decides which items are to be implemented during the sprint and by that commits to a certain amount of work. It should be noted that the priorities and the selection of items can be changed during the meeting as a result of discussion between the Product Owner and the Team.

In order to manage the work in progress and timely react to the difficulties in the development, *daily meetings* are set up. They are quite short, up to 15 minutes, held by the Team every day, approximately at the same time of the day. They are also called *stand-ups* and are a check on how the development is advancing. There are three basic questions to be answered, which aim at checking the work completed since the last meeting, planning the work for the day, as well as identifying impediments and challenges for the development:

1. What was done yesterday?
2. What will be done today?
3. What are the problems / obstacles that prevent you / make it harder for you to execute your plan for today?

The meeting that focuses on the work that has been carried out during the sprint is called *sprint review meeting*. During this meeting the Team discusses sprint backlog items that have been completed (status: *done*) or not. Moreover, the Team is expected to present an executable version of the system to the stakeholders.

Scrum, as an agile method, is oriented towards continuous improvement. An inner-team meeting dedicated for identifying possibilities for improvements in the development process is called *retrospective*. The goal is to examine the current state of the development from a higher-level viewpoint and elicit the guidelines for fine-tuning the process. Retrospective should last 15-60 minutes and is supposed to answer 3 questions:

1. What went well during the sprint cycle?
2. What went wrong during the sprint cycle?
3. What could we do differently to improve?

Meetings are not only the basis for collaboration, but also support self-organisation of teams. Furthermore, they facilitate the idea of a cross-functional team, particularly when planning the work and discussing the challenges in the development. Not every team member has to be an expert in all problem domains – it suffices that the team knows the expertise of other team members, so that the problem can be shifted to the most knowledgeable person.

2.3.4. Scrum and Event-B – possibilities of synergy and challenges

It might seem that formal modelling cannot be combined with agile development processes. However, we found particular characteristics that Event-B – a formal method of our choice, and Scrum – an agile method selected after conducting research for FormAgi framework, have in common. Moreover, there are properties that enrich and smoothen Event-B developments.

We have chosen an agile method that has well-defined rules, but at the same time is flexible enough to handle the rigour imposed on the development by a formal method. Introducing Scrum into formal development with Event-B meant first of all emphasising that every development needs a process to better manage the development activities, and second of all highlighting that every development process needs continuous improvement based on its progress. Scrum, as an agile method, by definition very well supports process improvement. It can be achieved not only through the retrospectives, which were set up specifically for the purpose of improvement, but also indirectly by the organisation of work.

The work arrangement within Scrum strongly supports communication, which seemed to lack in developments using formal methods. A set of meetings to be held during the development process helps not only to control the progress of development, but also timely discover and act upon the challenges and problems that occur. Since Scrum is a team-based process, it assists the team members with the knowledge exchange and eases the acquisition of "second opinion" or model review, whenever needed. The idea of having teams with multifaceted know-how reinforces the development by allowing people with various levels of expertise in different domains complement each other. Finally, Scrum puts emphasis on human factor in development underlining that it is a very much developer-centred activity. Although formal development is strongly relying on mathematical foundations, it is the developer who decides to apply certain modelling strategy, which is only later reinforced by the mathematical know-how.

The sprints in Scrum resemble the refinement steps in Event-B. The short development cycles (long iterations aka sprints) to some degree correspond to the gradual construction of system (refinement steps). Division of features according to their feasibility within the time of sprint is a similar mechanism to decomposing certain properties of a system to few smaller ones. All of these efforts serve to lessen the complexity of a problem to be developed or modelled.

Scrum has a clear definition of time frames for development, specifying long and short iterations (sprints and dailies) and associated meetings. This aids in managing the progress of the development. It also supports the steady pace of work, minimising the risk of having too many features to be implemented at the end of the development. Event-B development relies on the continuous and gradual introduction of properties to the system, since it is based on the refinement process and refinement rules. Thus, the aforementioned characteristics of Scrum and Event-B harmonise with each other.

3. Experimentation

In FormAgi we investigated a possibility of a merge of formal and agile development approaches. As a result, we proposed a framework which provides guidelines on what concerns should be tackled before committing to a certain agile method. Moreover, we gave pointers in which aspects an agile method can be a facilitator in the formal development. Then, we chose to utilise Event-B as a formal method. Furthermore, following the advices presented in FormAgi, we selected Scrum as (seemingly) most suitable method that would correspond to characteristics of Event-B development. In this section we explore our claims and examine how feasible it is to use Event-B within Scrum development process. We experiment with a case study originating from aerospace industry to bring up strong points of such a merge and highlight the challenges that need some further attention.

3.1. Landing Gear System (LGS) Case Study

We demonstrate the proposed approach using the LGS [8]. The focus of this paper is on the Event-B development within agile process, in particular Scrum. Therefore, we omit the construction details of the formal model and focus on a high-level description of the modelling process. The details about the formal development of the case study can be found in [42].

The system consists of a digital controller and a few actuators. The function of the system is to operate the landing gears and associated doors. Depending on the reactions from the pilot, the digital controller manipulates the mechanical part. The mechanical part, in its turn, consists of front, left and right landing sets. Each set includes a door, a landing gear and hydraulic cylinders that are attached to and move the

corresponding doors and gears. In addition, the system has an analogical switch, which purpose is to prevent an abnormal behaviour of the digital part. The architecture of the system is shown in Figure 9.

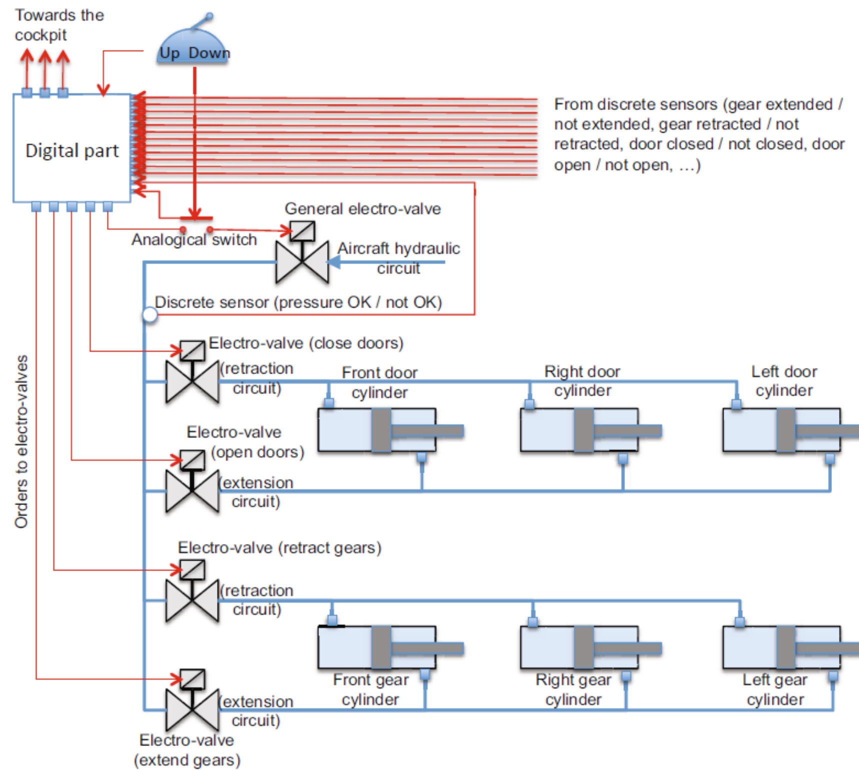


Figure 9 Architecture of the LGS [8]

The general electro-valve provides hydraulic power to the specific electro-valves from the aircraft hydraulic system. There are 4 specific electro-valves which set the pressure to the cylinders opening/closing the doors as well as to the cylinders extending/retracting the gears. Clearly, the position of the piston of a cylinder coincides with the position of the corresponding controlling component. For instance, if the front door cylinder is extended, the front door is open.

We develop the part that consists of the general electro-valve, the specific doors and gears electro-valves, the cylinders as well as the analogical switch. We start by introducing the general electro-valve. We then add specific electro-valves and cylinders. Finally, we extend the specification with the analogical switch. To ease the development and distinguish between different valves and cylinders, we number them from 0 (topmost) to 3 (bottommost) for the specific electro-valves and from 0 (left-top) to 5 (right-bottom) for the cylinders in addition to the name. For example, the general electro-valve is named GEV_0 whereas the electro-valve used to close the doors (retraction circuit) has a name “evalve_0”.

3.2. Scrum Process for Event-B Development

One of the major advantages of agile methods is their flexibility, i.e. the ability to be tailored to fit the characteristics of the environment they are utilized in. The possibility of adjustments helps to benefit the most from the used methodologies and tools. For Event-B developments we aim at making the development more proactive and smoothen it by enabling shorter iterations. Moreover, we want to facilitate the intra-project communication, as well as support the communication between the team and the stakeholders, as we believe they are crucial to obtain software of high quality.

Formal development (herein modelling activity) differs from traditional development (consisting mostly of coding activity) not only in the rigour of the development, but also how the progress of the development can be seen and measured. This has its reflection in the *artefacts* being created during the development. Namely, instead of recognising executable code as the main measure of development, we consider requirements, specifications, model on specific abstraction levels or implementation of a feature as factors determining the progress of the development.

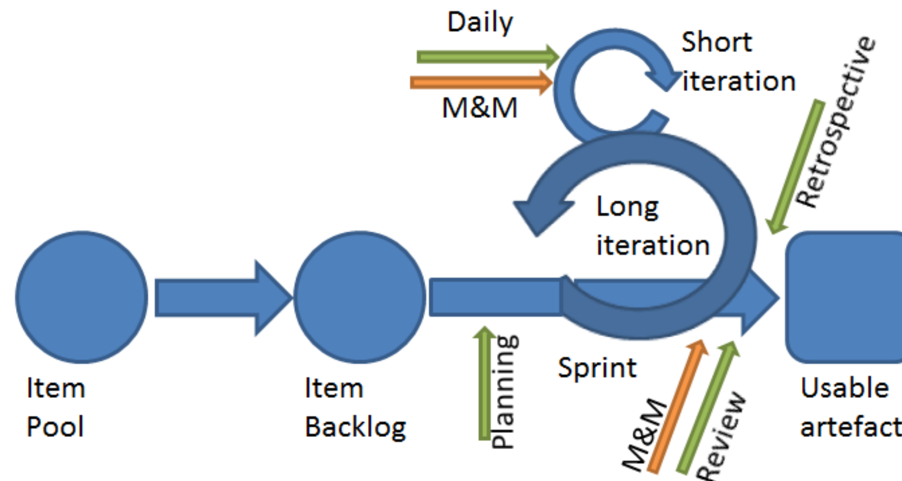


Figure 10 Scrum adapted for Event-B development

We adjust Scrum to suit the specifics of Event-B development, which is depicted in Figure 10. We use the term *item*, to denote that we understand the term "features" in a slightly different way. Not only we transform features, and in our case requirements, into models and afterwards prove the models and their properties to be correct, but also we work on elicitation and modelling of requirements themselves. Thus, formal development has much broader implications for the development process already at the conceptual stage.

Item pool consists of a set of requirements and acts as product backlog. It not only contains high-level requirements, but also lower-level requirements, safety cases, environmental and context descriptions. A subset of the item pool comprising of requirements chosen for the current sprint is called an *item backlog*. The requirements in item backlog are not prioritised, since prioritization may take more time than what is scheduled for regular Scrum process. Therefore the prioritisation is done within sprints. The reasoning is twofold: (i) we do not want to rush decisions which would lead to a complex and hard to prove model and (ii) the work on the requirements and their structuring with respect to the modelling strategy will pay off later, when the model needs to be extended. Therefore, a sprint includes modelling of the requirements, as well as developing and proving a model. Finally, model animation and simulation, verification mechanisms well supported by the Rodin platform, can also be a part of the sprint.

The duration of *long* and *short iterations* should be decided before the development starts and then fine-tuned, if necessary. There is a risk that some requirement or property is too complex to be processed within a short iteration. In this case it should be discussed during the short *daily* meeting so that the team is informed, and in consequence it is decided that it either spreads over two or more sprints or is possibly decomposed to few smaller problems. The sprint *review* resembles the discussions in a regular sprint. However, some issues like model walkthroughs, or demonstrating the results to stakeholder as model simulation or animation should also be included at this stage. The *retrospective* is meant to reflect upon the sprint and highlight the areas of the sprint for future improvement from the process perspective.

We believe that formal development and development process are not equivalent; however, we want to examine it. Explicitly, we will study the relation between refinement step and development process iteration and check if there exists a one-to-one mapping. We suppose that there can be several refinement steps within a single iteration. Moreover, in case a requirement is not defined precisely (so that refinement needs to be revised), a refinement step might occur to be too large and complex for a single short iteration. In this scenario, the problem to be modelled needs to be decomposed into sub-problems and only as such placed into the item backlog.

Finally, although not present in original Scrum, a feedback system is included in the sprint via the *Monitoring and Metrics* mechanisms *(M&M)*, which is to raise understanding on what is being done (short iterations), as well as to facilitate the process improvement and provide evidence on the development (long iterations). We have a tool support within Rodin platform for M&M, which provides us with the measurements regarding the number of proof obligations and the time that is used for modelling (proactive metrics), as well as an external metric tool that evaluates the size and complexity of a model (status metrics). We are aware that metrics and measurements within agile developments are sometimes considered as harmful to the team morale and against agile philosophy. However, we use them for informational purposes rather than "plunger of blame".

3.3. Experimental Setting

We fine-tune Scrum for formal development in Event-B, in particular for modelling of a LGS case study from the aircraft industry. We perform our experiment in academic setting, where three people are involved of various backgrounds and expertise domains: formal methods, formal modelling of systems, quality assurance and quality measurements. The standard roles in the Scrum process were fine-tuned in order to enable knowledge exchange, facilitate the adaptation of Scrum to Event-B modelling activities and accelerate the comprehension of the concepts of the case study to be modelled (domain knowledge). The roles were to some extent shared and in our context they were as follows:

- product owner: role shared by the modeller, due to the familiarity with the requirements given in [8], and the senior expert,
- scrum master: quality assurance expert and agile expert,
- team: role shared by the modeller, Event-B senior expert and quality assurance expert.

Due to some other work-related commitments of the team members, we have set up a two-week restriction time for the development. Also, note that the effective worktime is not a complete workday. Thus, it is visible that the development process heavily depends on human factors.

We have divided our development into two sprints, each a week long (long sprint), within which every day was tackled as a short sprint. We kept the planning and daily meetings, retrospectives and reviews, just as it is defined in the original Scrum. We also held an introductory planning meeting, "sprint 0", to familiarise the non-agile members with concepts of Scrum, as well as to ensure that the goals of the project are clear for all of the team members ("what product are we building?"). In addition, functionality of a case study and vocabulary used were explained, so that their comprehension is the same for all of the participants of the experiment. We used various communication means throughout the development; however, we relied mostly and benefitted from the direct and active communication (face-to-face or internet communicators).

Our item pool and item backlog was managed in Excel form, which was constructed as a simplified tracking system and is exemplified by Table 1. The table contains *ID* field – a unique number assigned to a requirement, which enables accurate classification and discussions over the requirement. Furthermore, the *Name of a feature / title* is a short term describing the requirement and bound to ID. The requirements are *prioritised* with natural numbers 0 to 3, designating 0 as requirement of low relevancy and 3 as the

requirement of utmost significance. A *Description* is a simplified version of a requirement (taken after analysing the requirements document); Additional field, *Remarks*, describes to which iteration the requirement is assigned to; it can also provide some further comments. The *Done* field depicts whether the requirement is modelled and proved; if a requirement is only modelled or partially proved it has the “not done” status. Finally, there are two optional fields in the table (denoted with an asterisk) that we thought could be of use: (i) *Category*, assigning the requirement to a product or a model, depending on the activity in the development (implementation/code generation or modelling and proving) and (ii) *Complexity/story points* showing the difficulty level according to Fibonacci sequence, where the lower value means the easier it is to model and prove a requirement. It is an estimation given by the developer according to his/hers experience and is not time related; rather it is related to the difficulty of the problem. Note that we have not utilised the two latter columns due to several factors: small scale of the development, keeping our main focus on modelling and proving (no code generation or implementation involved) and the way the requirements were provided (creating use cases would require additional effort and not bring much benefit to the modelling and proving activity).

Table 1. Item pool (backlog) of features in the LGS case study (result of Sprint “0”)

ID	Name of a feature / title	Priority	Description	Remarks
1	Developing a valve	3	Can be done in parallel with 2 & 11	
2	Developing a cylinder	3	Can be done in parallel with 1 & 11	
3	Developing analogical switch	2	-	
4	Introducing general electro-valve	3	general electro-valve supplies the specific electro-valves with hydraulic power from the aircraft hydraulic circuit	
5	Refining generic component into valves of doors and gears	3	electro-valves set pressure on the portion of the hydraulic circuit related to door opening/closing and landing gear extending/retracting	
6	Introducing generic component for valves of doors and gears	3	Can be done when 11 is done	
7	Refining generic component into cylinders	3	cylinders open/close doors as well as retract/extend the landing gear	
8	Introducing generic component for cylinders	3	Can be done when 11 is done	
9	Introducing analogical switch	2	switch tolerates an abnormal behaviour of the digital part	
10	Developing a pump	1	-	
11	Developing a generic component	3	Can be done in parallel with 2 & 1	

Figure 11 Backlog of the case study (result of sprint 0)

The backlog presented in Table 1 is a result of the so called "sprint 0", where we planned the modelling by first prioritising the features listed in the item pool (assigning priorities 0-3) and then scheduling them for

certain iterations (1st or 2nd iteration). We deliberately left out columns *Category* and *Complexity*, as either they were not yet relevant or at all not needed in this project.

As a technical support for our development we used subversion control system (svn) in order to be able to better control the development and its progress, as well as run the metrics tool on the versions of the model in the retrospective manner (if it occurs to be not feasible during the development).

The safety, functional, equipment and other requirements were provided in [8] and added to the item pool/backlog. However, more requirements can be included to the item pool/backlog by the team and product owner / stakeholder later on, since during the development, some additional properties might be revealed; these can be initially overseen, for instance, by the stakeholder.

The Excel sheet that we use contains also the documentation for tracking the time spent on the development. It is based on the time recorded by the Rodin tool when the developer is actively constructing and proving the model. Naturally, some additional resources are put for, e.g., elicitation of requirements, getting familiar with technical documentation or inner-team discussions. Note that the development was not a sole activity of the development Team during the regular work-days, therefore one cannot relate the collected data to the complete work-day.

3.4. Event-B Development of the LGS Case Study

The initial backlog of the features to be developed together with their prioritisation is captured by Table 10. Clearly, the features that have the highest priority (3 in this case) need to be developed first. This is because the other features typically dependent on them, so that the development cannot proceed forward, if they are not done. Observe that the features with the same priority can also have interdependencies, so that some features may need to be implemented first. Note also that the development of components (features with IDs 1, 2, 3 and 11) can be done in parallel with the system development, even though we chose to develop some of them beforehand. Table 2 is a result of transformation of Table 1 according to the priorities and related iterations in which the modelling was planned.

Table 2 Item pool (backlog) consisting of features ordered according to the priorities and planned iterations

ID	Name of a feature / title	Priority	Description	Remarks
1	Developing a valve	3	Can be done in parallel with 2 & 11	1st iteration
2	Developing a cylinder	3	Can be done in parallel with 1 & 11	1st iteration
11	Developing a generic component	3	Can be done in parallel with 2 & 1	1st iteration
4	Introducing general electro-valve	3	general electro-valve supplies the specific electro-valves with hydraulic power from the aircraft hydraulic circuit	1st iteration
6	Introducing generic component for valves of doors and gears	3	Can be done when 11 is done	1st iteration
5	Refining generic component into valves of doors and gears	3	electro-valves set pressure on the portion of the hydraulic circuit related to door opening/closing and landing gear extending/retracting	2nd iteration
8	Introducing generic component for cylinders	3	Can be done when 11 is done	2nd iteration

7	Refining generic component into cylinders	3	cylinders open/close doors as well as retract/extend the landing gear	2nd iteration
3	Developing analogical switch	2		2nd iteration
9	Introducing analogical switch	2	switch tolerates an abnormal behaviour of the digital part	2nd iteration
10	Developing a pump	1		2nd iteration

The analysis of Table 20 provides us with the following refinement strategy for the system development. Once, the components are developed and added to the library, we first instantiate the formal library component, namely the electro-valve (see [42]), into the general electro-valve. Then, we refine this model by adding a connector between the general electro-valve and the other electro-valves controlling the doors and the gears. However, instead of instantiating these valves directly, we will first add a generic component as the placeholder and only then replace it with the specific electro-valves in a separate refinement step. This development sequence is done deliberately to show the development when not all the components have assigned priorities, but the overall architecture has been established. After specific electro-valves are introduced, we perform several more refinement steps, in which we gradually add connections between the components, as well as add cylinders using the generic components in a similar manner as when adding specific electro-valves. As a final step, we introduce the analogical switch component.

3.4.1. First Iteration (Sprint 1)

During the first iteration (sprint), we develop the necessary components (task ID 1, 2, 11, Table 3), namely the electro-valves and cylinders in Event-B (see [43] for details on the development). The components are made parameterized, so that the development is performed once and the components can be reused later (library of components). In addition, we start the development of the LGS by introducing the general electro-valve (task ID 4, Table 3). Note, however, that the development is not restricted to the sequence described here. We opt for this sequence due to the component-based rigorous development using the library of visual reusable components.

Table 3 Features to be modelled in first sprint

ID	Name of a feature / title	Priority	Description	Remarks
1	Developing a valve	3	Can be done in parallel with 2 & 11	1st iteration
2	Developing a cylinder	3	Can be done in parallel with 1 & 11	1st iteration
11	Developing a generic component	3	Can be done in parallel with 2 & 1	1st iteration
4	Introducing general electro-valve	3	general electro-valve supplies the specific electro-valves with hydraulic power from the aircraft hydraulic circuit	1st iteration
6	Introducing generic component for valves of doors and gears	3	Can be done when 11 is done	1st iteration
5	Refining generic component into valves of doors and gears	3	electro-valves set pressure on the portion of the hydraulic circuit related to door opening/closing and landing gear	1st iteration

			extending/retracting	
8	Introducing generic component for cylinders	3	Can be done when 11 is done	1st iteration
7	Refining generic component into cylinders	3	cylinders open/close doors as well as retract/extend the landing gear	1st iteration

At the abstract level, the system specification corresponds to the model of the general electro-valve whose graphical representation is shown in Figure 12. Notice that the system development can be initiated from the components already available in the library. The components that may be missing from the library can be developed and added to the library in parallel with the system development whenever needed.

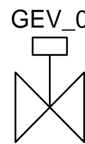


Figure 12 LGS, specification 0: General electro-valve

After deriving the abstract specification, we refine it by first adding a connector and then the generic component, i.e., we implement the task ID 6 (Table 3). Consequently, in two refinements, we obtain the specification visually illustrated in Figure 13.

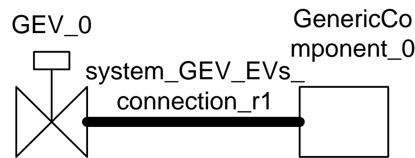


Figure 13 LGS, refinement 2: GEV connected with generic component

Although there were several tasks left in the item backlog (tasks ID 5, 7 and 8, Table 3), they were moved to the second iteration. This was due to our over estimation of tasks that could have been developed within the first iteration.

3.4.2. Second Iteration (Sprint 2)

The second iteration consists of tasks that were shifted from the first sprint, and the ones originally planned for the second sprint (Table 4).

Table 4 Features to be modelled in second sprint

ID	Name of a feature / title	Priority	Description	Remarks
5	Refining generic component into valves of doors and gears	3	electro-valves set pressure on the portion of the hydraulic circuit related to door opening/closing and landing gear extending/retracting	2nd iteration
8	Introducing generic component for cylinders	3	Can be done when 11 is done	2nd iteration
7	Refining generic component into cylinders	3	cylinders open/close doors as well as retract/extend the landing gear	2nd iteration
3	Developing analogical switch	2	-	2nd iteration
9	Introducing analogical switch	2	switch tolerates an abnormal behaviour of the digital part	2nd iteration
10	Developing a pump	1	-	2nd iteration

The second iteration starts with the refinement of the generic component (see Figure 13) into the set of specific electro-valves (task ID 5, Table 4). Specifically, we introduce all four specific electro-valves which control the doors and the gears. The development proceeds according to the pattern described in [42]. Therefore, we obtain the system model illustrated by Figure 14.

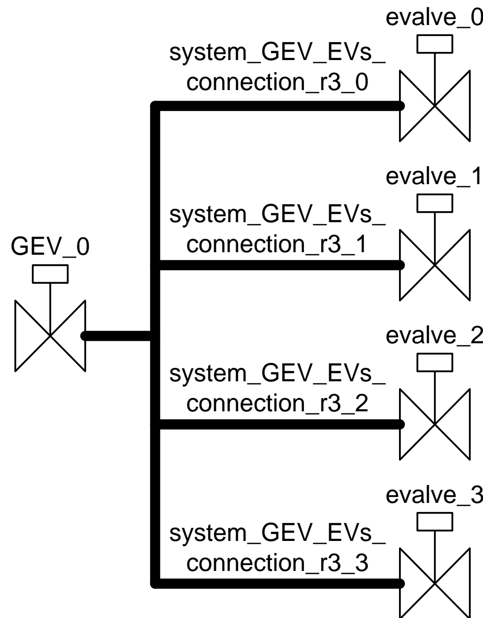


Figure 14 LGS, refinement 3: general electro-valve connected with specific electro-valves

Next, we gradually add all the cylinders starting from the ones that open/close the doors. Clearly, the valves and the cylinders have to be connected in order for the system to function properly. The cylinders require two connections: one for the head and the other one for the cap. Since the specification refinement with connectors is rather simple [42], we add both connectors at the same refinement step. This leads to the model visualised in Figure 15.

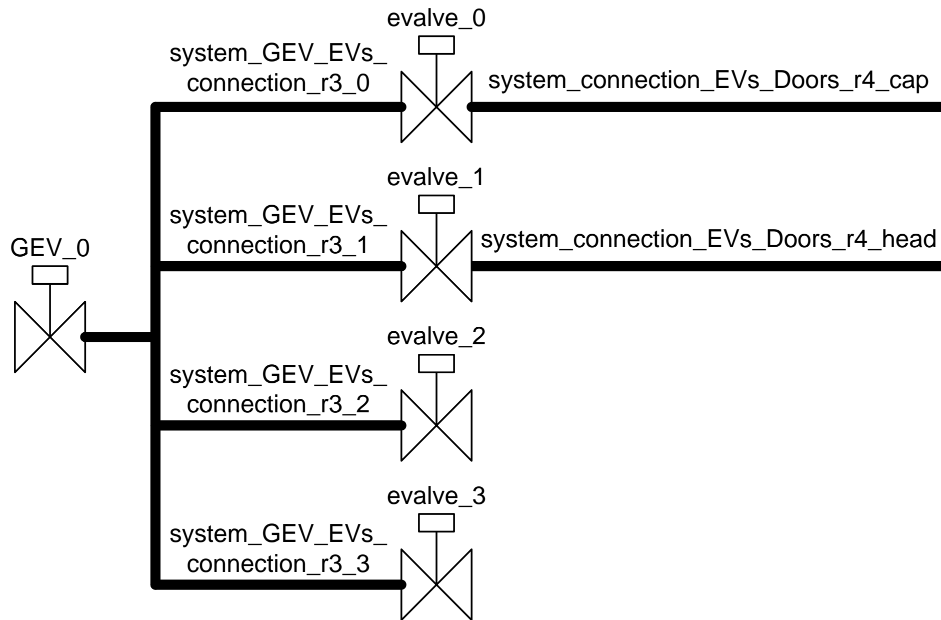


Figure 15 LGS, refinement 4: all the valves with connectors between them and cylinders of doors

Instead of directly extending the system specification with the specific cylinders, we first add another generic component (task ID 8, Table 4). While adding the generic component, we connect it to the corresponding valves (Figure 16).

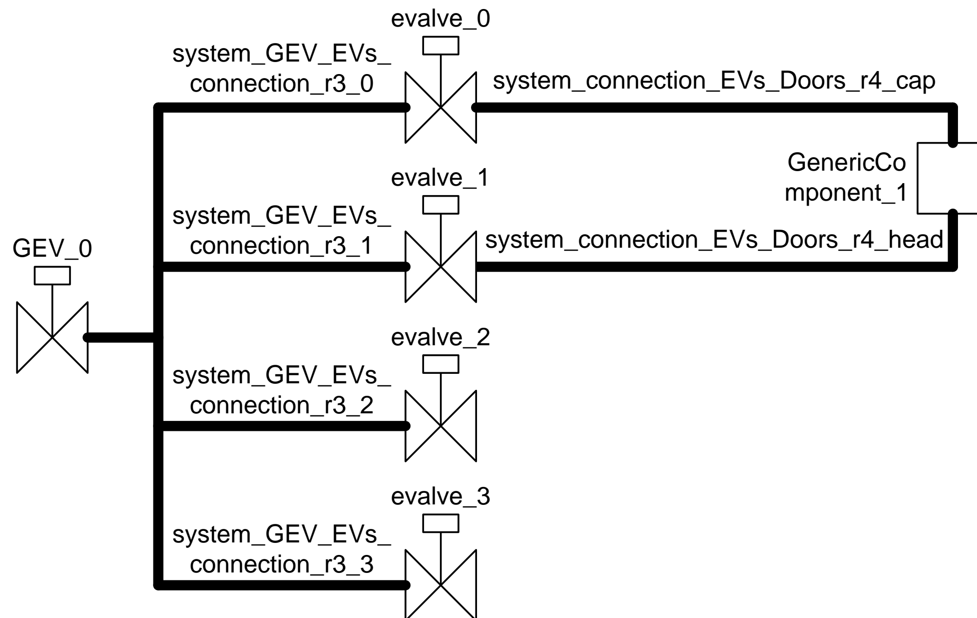


Figure 16 LGS, refinement 5: introduction of the generic component for cylinders of doors

The generic component is then replaced by the specific cylinders (task ID 7, Table 4) at the subsequent refinement step following the same approach as for the valves. At this point, the specification of the system contains all the electro-valves and three cylinders, as well as connections between them (Figure 17).

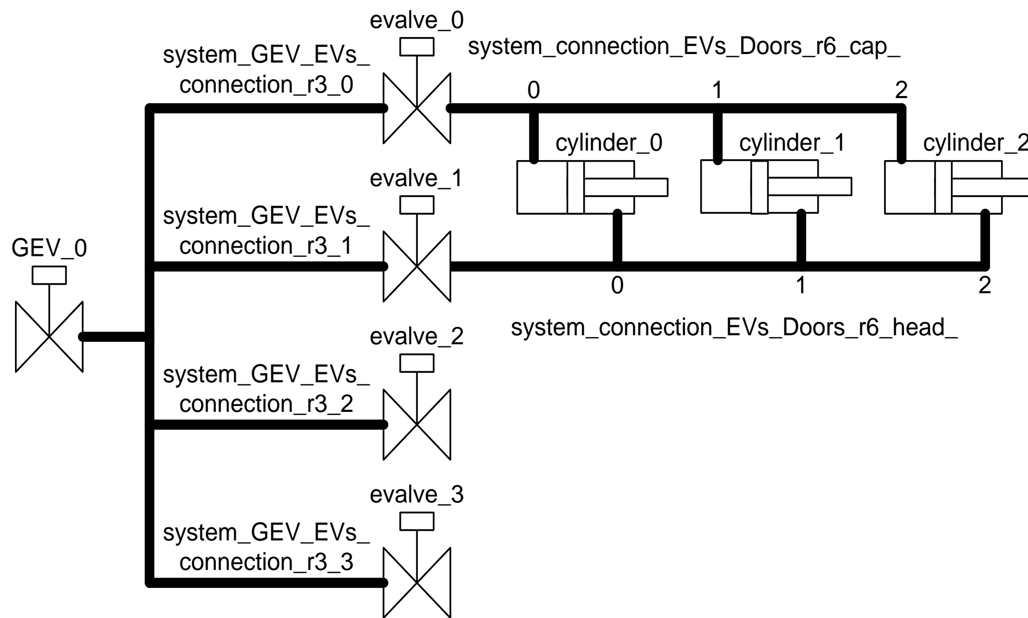


Figure 17 LGS, refinement 6: cylinders of doors

Following the same approach as for the addition of the cylinders manoeuvring the doors, we add the remaining cylinders that extend/retract the gears (repeat tasks ID 8, 7, Table 4). Eventually, we derive the complete specification of the actuators part. It consists of the general electro-valve, 4 specific electro-valves and 6 cylinders. All components are interconnected as specified by the requirements (Figure 18)

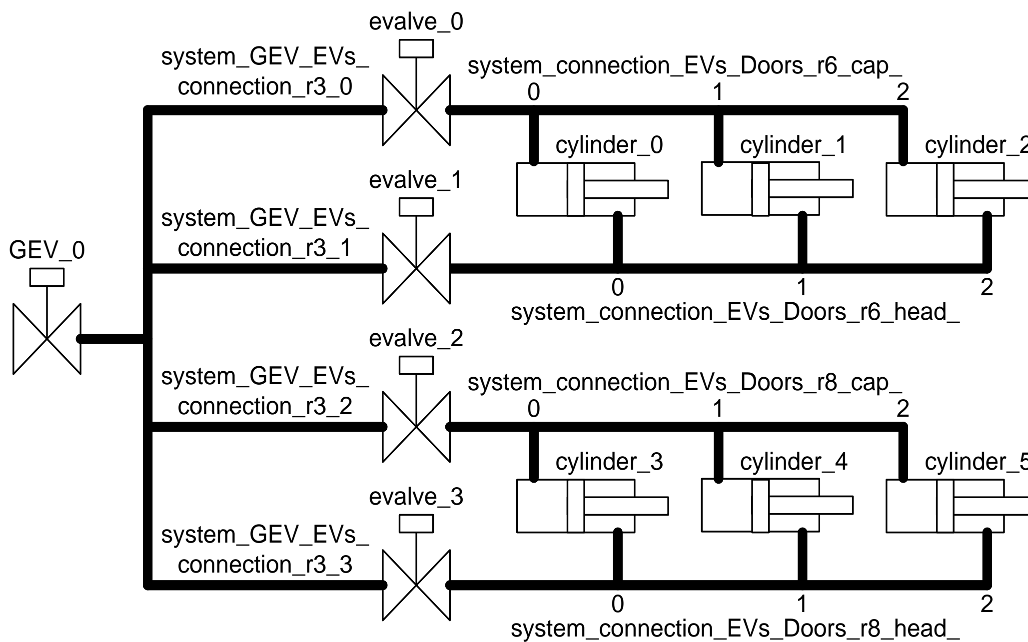


Figure 18 LGS, refinement 9: interconnected electro-valves and cylinders

As the last step of our development, we have specified the model of the analogical switch component (task ID 3, Table 4) and added it to the system specification (task ID 9, Table 4). Consequently, we obtain

the system as depicted in Figure 19, where analogical switch is placed on the left-hand side of the system (component denoted as "as_0").

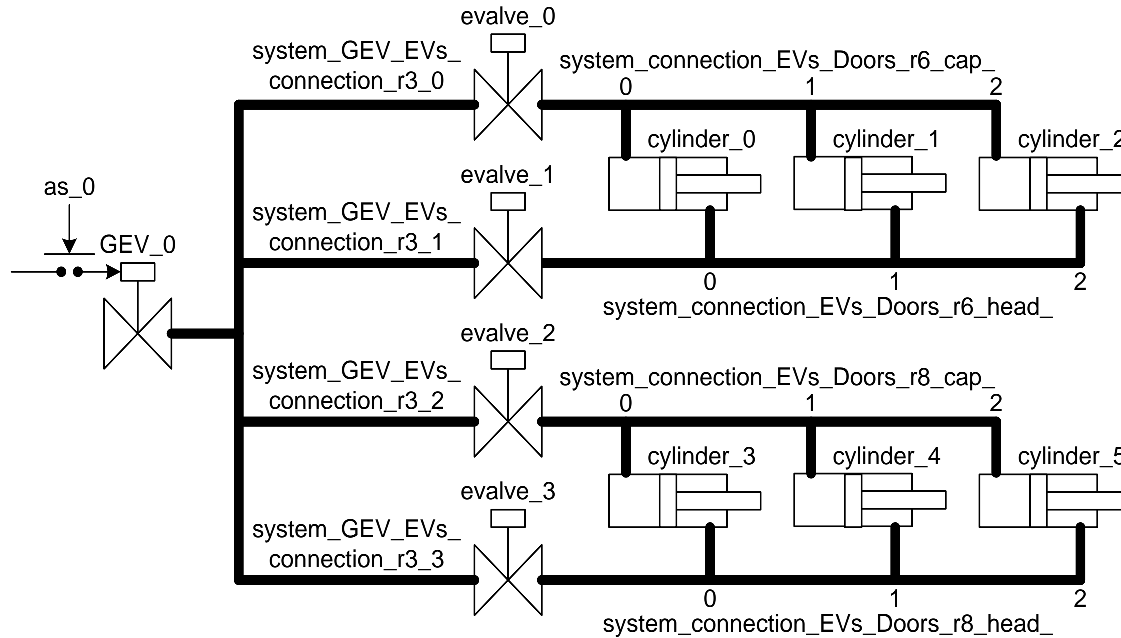


Figure 19 LGS, refinement 10: Analogical switch

Developing a pump (task ID 10, Table 4) was not completed within the second sprint due to the lack of time. In real life, this task would be moved to the next sprint. However, we were limited to only two weeks in our experiment, thus the task is not included in the development.

3.4.3.Case Study – Summary of the Development

We completely developed the part of the case study comprised of the valves, cylinders and the analogical switch. However, we left out the development of the pump due to the time limitation. In total it took ten refinement steps to complete the development (see Table 5).

The summary of the proof statistics for the LGS case study is shown in Table 5. The numbers in the table reflect the sum of POs of a context and a machine of the corresponding refinement step. Most proof obligations were automatically proven by the tool. A large number of the manual proof obligations were derived from the specifications of the library components and can be simply reproduced. Note that the asterisk (*) in refinement steps 1, 4 and 7 denotes a requirement that we have not considered when planning the development. Despite this, these were naturally needed to interconnect the components and were rather simple. In addition, the development of the library of components is not shown due to the fact that the focus is solely on the LGS. Therefore, the requirements with IDs 1, 2, 3 and 11 were modelled and proven for generic purpose (library of components) and only utilised in this case study, although included in the sprints.

Table 5 Case study proof statistics

Ref. Step	Req. ID	Name	Total POs	Auto
0	4	General electro-valve	24	21
1	*	Connection between general electro-valve and the other valves	7	7
2	6	Generic component for specific electro-valves of doors and gears	26	26
3	5	Electro-valves of doors and gears	143	131
4	*	Connection between electro-valves of doors and cylinders of doors	14	14
5	8	Generic component for cylinders of doors	41	41
6	7	Cylinders of doors	73	73
7	*	Connection between electro-valves of gears and cylinders of gears	14	14
8	8	Generic component for cylinders of gears	41	41
9	7	Cylinders of gears	73	73
10	9	Analogical switch	48	47
		Summary	*504*	*478*

4. Monitoring and Analysis

In this section we include our comments on the development, containing not only analysis of the Excel trac document (development timeline and effort), but also we provide information on the proof statistics. Furthermore, we investigate the measurements we took from our models during the development. The measurements are based on the versions of the models submitted to the repository, meaning that the models needed to be modelled and proved. Moreover, we study the relation between the iterations and refinement steps, which was one of the interest points raised in one of our recent papers [44]. Finally, we present the remarks of our developer on the development process and its feasibility for the formal developments.

4.1. Development Process

Our development consisted of two sprints. In addition, we held sprint '0' whose purpose was to clarify the goals of the project, plan the work and familiarise the non-agile team members with the principles and practices behind the scrum development process. Moreover, some fine-tuning of scrum was needed for the setting of the experiment. We found it particularly useful to have the planning sprint, so that the case study can be discussed and preliminary strategy for the formal development can be agreed upon. Moreover, the setup for the development (process viewpoint) is made transparent for all the team members.

In Table 6 we present the final snapshot of our requirements document that we used for tracking and managing the development and iterations. Similarly like in traditional agile developments, where only the tested and executable version of a system can be committed to the repository, it was essential in our setting that a requirement should be modelled and proven in order to be submitted to the version control system (and thus obtain status "Done"). As can be seen, almost all the requirements are modelled and proved (status

"Done"), except the one regarding the pump, which was moved for next iteration. Note that the timetable for our experiment was limited to two weeks, meaning that the development of pump needed to be continued beyond the experimental setting.

Table 6 Requirements table (Excel trac document) captured at the end of the development

ID	Name of a feature / title	Priority	Description	Remarks	Done
1	Developing a valve	3	Can be done in parallel with 2 & 11	1st iteration	V
2	Developing a cylinder	3	Can be done in parallel with 1 & 11	1st iteration	V
11	Developing a generic component	3	Can be done in parallel with 2 & 1	1st iteration	V
4	Introducing general electro-valve	3	general electro-valve supplies the specific electro-valves with hydraulic power from the aircraft hydraulic circuit	1st iteration	V
6	Introducing generic component for valves of doors and gears	3	Can be done when 11 is done	1st iteration	V
5	Refining generic component into valves of doors	3	electro-valves set pressure on the portion of the hydraulic circuit related to door opening/closing and landing gear extending/retracting	2nd iteration	V
8	Introducing generic component for cylinders	3	Can be done when 11 is done	2nd iteration	V
7	Refining generic component into cylinders	3	cylinders open/close doors as well as retract/extend the landing gear	2nd iteration	V
3	Developing analogical switch	2	-	2nd iteration	V
9	Introducing analogical switch	2	switch tolerates an abnormal behaviour of the digital part	2nd iteration	V
10	Developing a pump	1	-	3rd iteration	-

4.2. Meetings

By introducing agile principles and values to our development, in particular scrum practices, we wanted to support the internal communication in the project. Based on our experience, we claimed it was a weak point

when applying formal methods [44], which can certainly be improved. As in scrum, we kept basic four meetings, i.e. planning, daily, review and retrospective meetings and reinforced them with other communication means (e-mails, live chats and calls with shared screens).

We held sprint "0", when we discussed the scrum development process and fine-tuned it for our setting, as well as we agreed on the technologies used and introduced some additional development facilitators (e.g., the times we communicate, how the development will be documented). Afterwards, we started component-based formal development governed by the adjusted scrum process, with two sprints, each having 5 days (see Figure 20).

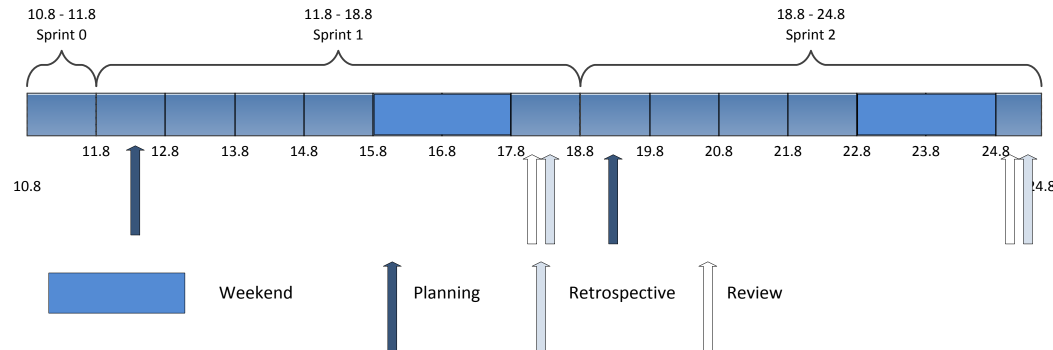


Figure 20 meetings within development

Every day we held a daily meeting, according to the scrum-by-the-book instructions, i.e., having 3 questions to be shortly answered within 15 minutes. We noticed that sometimes the discussions became quite technical; however, we found it beneficial for the progress of the development, as we could jointly come up with a solution to a bottleneck. On the first day of each sprint we had a planning meeting, where we decided which features from item pool will be moved to the item backlog and what are their priorities. Moreover, for sprint 2, we determined if the features which were not handled in sprint 1 will be processed in sprint 2 and in which sequence (according to priorities and the latency fact). The review meeting and retrospective were held on the fifth day of each sprint and served as checkpoint for the development and a chance for process improvement, respectively. Some comments regarding the latter are presented in Section 4.5.

4.3. Development Effort

It was essential for us to be able to show evidence on the effort spent on particular tasks in the development, so that somebody interested in applying the method gets the feeling about what it involves in terms of time and effort. Table 7 presents the report on the effort spent by our developer on particular activities (including modelling and proving).

Table 7 Report on effort and activities recorded by the developer

Date	Description
10.08	Introductory planning meeting - Sprint 0
11.08	Checked components: generic, valve, cylinder; took ~18min. Introduced generic component for the valves, had troubles with a variant; took ~70 min. Due to variant issue had to split the refinement into steps: first introduced connection, then generic component. Total time spent: ~1.5 hours
12.08	Instantiated valves with specific values, took ~22min. Refined generic component into valves; took ~180 min. The issue is with gluing invariants and proper data structures for refinement (replacing one generic component with 4 parallel valves)
13.08	Fixed the second refinement with the control variables (instead of two, there is one now); took ~5min. Tried to fix the issue with the refinement took ~35min
14.08	Worked on the refinement took ~136min
15.08	-
16.08	-
17.08	Experimented with the case study without generic components took ~40min. Continued working on the refinement took ~62min
18.08	Worked on the refinement took ~111min. Done with the refinement took ~524min (total 8 hours 42 min). Worked on task id 8 took ~34min
19.08	Worked on task id 8 took ~3min. Worked on task id 7 took ~42min
20.08	Worked on the connections of the valves with cylinders for gears took ~16min. Introduced and refined generic component into cylinders for gears took ~60min
21.08	Started the development of the analogical switch took ~11min
22.08	-
23.08	-
24.08	Developed analogical switch including timeput for opening took ~11min. Total took ~22min; Accomplished task id 9 took ~23min. Updated analogicalswitch component took ~2min.

The report is quite detailed for the purpose of our experiment and demonstration of the development process. We believe that for a development, which is not within experimentation, the documentation does not need to be so thorough. Note that the excerpt is in the form of an Excel table. Rodin tool offers reporting of effort spent on active modelling and proving; however, it does not support additional time needed in the development for e.g., the requirement analysis, checking the model, working on assumptions for the model, reworking the modelling strategy, etc. Naturally, reading the documentation or spending time on the inner project communication is not reported there either. However, more complex tracking system may be employed, if it occurs to be necessary.

In Figure 21 we present the pre-processed data described by Table 7. Note that the developer was not having the complete work days devoted for the experiment (due to other work-related obligations). Sprint "0" is marked with yellow colour, whereas sprints one and two are given with green and violet colours respectively. The lack of data for days 15, 16, 22 and 23 signify weekends. The brown arrows below the dates signify the days when there was a commit to the repository.

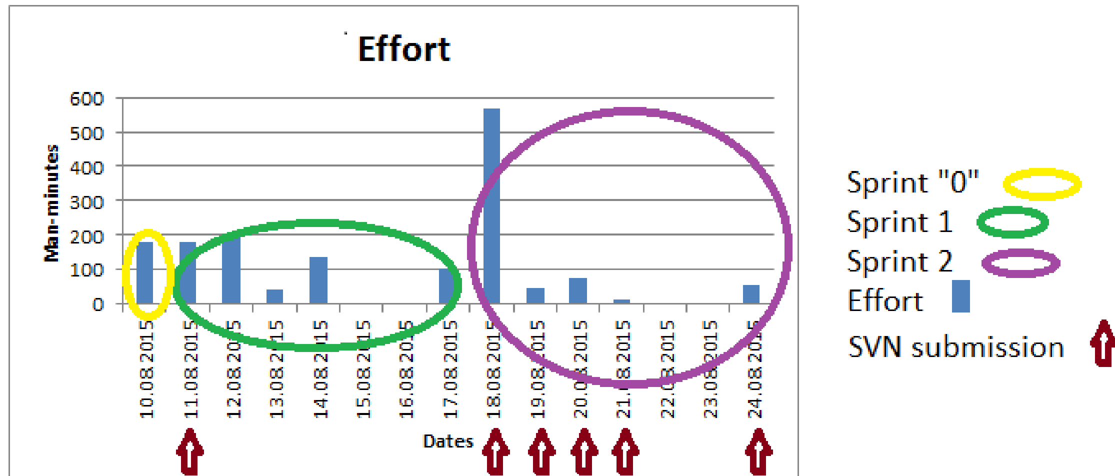


Figure 21 Distribution of effort (in minutes) within the development

When analysing Figure 21 one can observe the preparatory activities involved at the beginning of the development. Like in most of rigorous developments, there is an overhead that needs to be taken into the consideration, when the system is being studied and the modelling is planned. It should be emphasised that the choice of modelling strategy and the sequence of modelling artefacts and properties has an impact on how easy it will be to prove the model. Therefore, we believe that investing some time at the beginning of the development (sprint “0” and part of sprint “1”) is beneficial in the long run and does not contradict the idea of agile development process. On the opposite, any progress, also involving preparatory actions as eliciting and reworking the requirements conforms to the idea of agile by supporting value creation (and indirectly eliminating waste by constructing system in such a way that it does not need to be remodelled).

We observed that the time necessary to model and prove requirements very much varied from case to case. In order for some requirements to obtain status “Done” the developer needed to spend more than one day of work per requirement (e.g., requirement ID 5, for which it was necessary to adjust the modelling approach in order to ease proving; submitted to svn 18th August); whereas for several other requirements it was sufficient to be completely modelled and proven within one day (e.g., requirements ID 1, 2, 4 and 11; submitted to svn 11th August).

4.3.1. Proving Effort

Whenever analysing some formal development, it is particularly interesting to investigate the proof statistics, which may shed some light on the development complexity, required effort or even give some indirect information on the experience of the modeller. Proof statistics for our experiment (Table 5) are displayed in Figure 22 and consist of data representing manual, automatic and total number of Proof Obligations (POs) for each of the machine and context combined for each refinement step. The analysis of Figure 22 shows that the modelling and refinement of electro-valves between doors and gears, as well as modelling of analogical switch entailed more manual proving than any other machine. This is because the number of manually proved POs for a single valve equals to 3. Since we introduced 4 valves at the third refinement step, the total number of manually proved POs equals 4x3 = 12. Note however, that since these components have been instantiated from the library, the corresponding proofs can be reproduced from there as well. The only manual PO for the analogical switch regarded well-definedness condition which sometimes requires user assistance due to necessary substitutions.

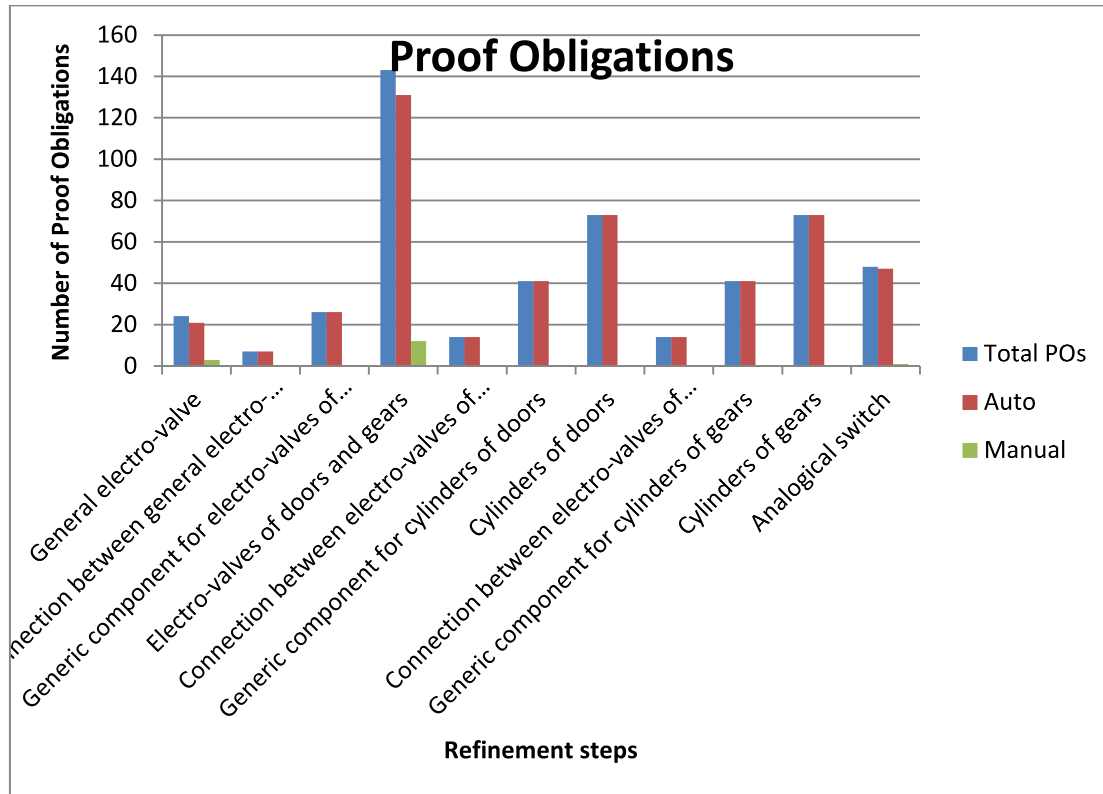


Figure 22 Proof statistics

Some interesting finding emerges when the machines and contexts are treated separately, like in Figure 23. It is quite apparent that the development of machines required more effort with respect to proving, than the development of contexts. This is due to the fact that a machine models the behaviour of the system (events), as well as incorporates invariant properties. Since events change the state of the system, they are required to show consistency with the invariants. Clearly, the greater the number of events and invariants, the more proof obligations generated and the more proving is needed.

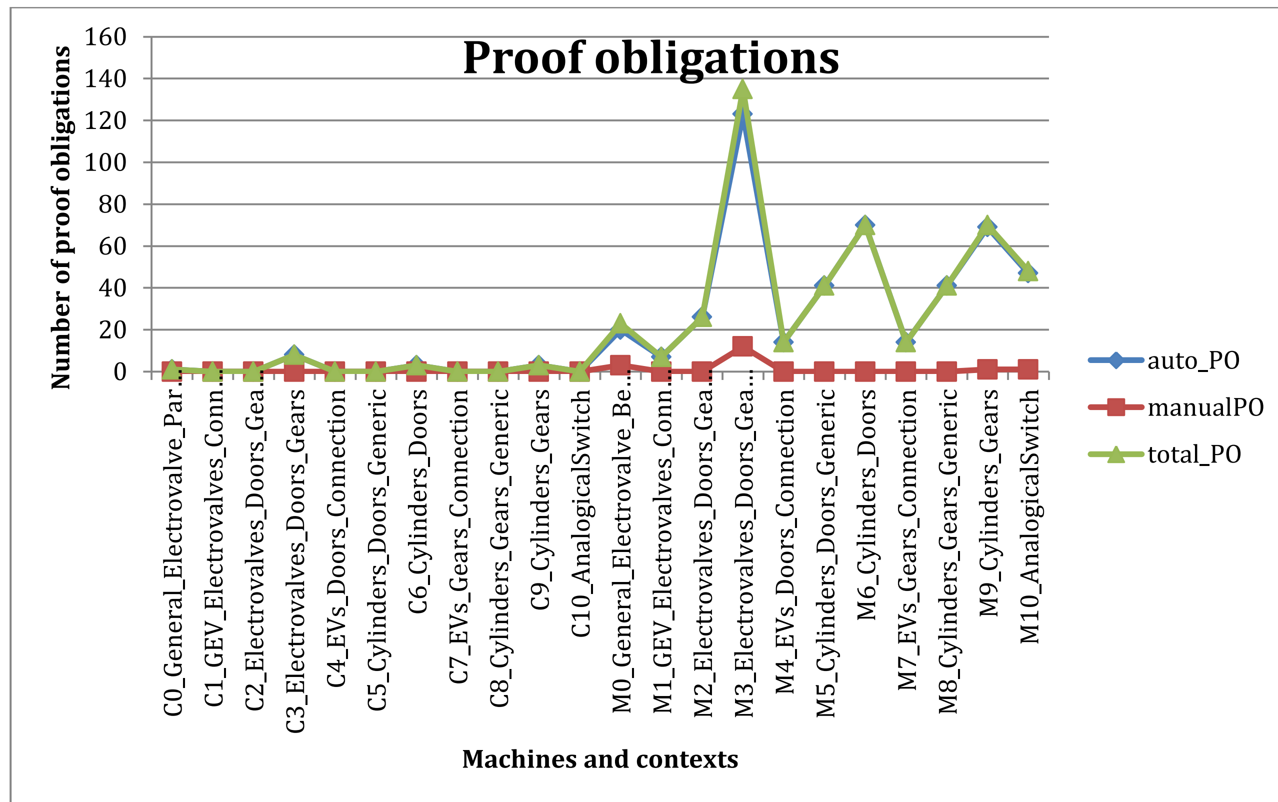


Figure 23 Proof obligations – a detailed view

4.4. Model Measurements

We are also interested in the size and complexity of created models, as it is claimed that application of formal methods and refinement may add complexity to the development, not only due to a steep learning curve for the formal method to be used, but also due to multiple refinements. However, the use of modelling strategies, herein abstractions and decomposition [45], application of patterns (see related work section) and simple monitoring of the development [46], as well as providing a feasible development process [44] can aid in controlling complexity.

Model measurements were automatically collected from all the versions of the model that were submitted to the repository. Therefore, all the measurements are investigated according to the "Done" status, and not (as in most formal developments) according to the refinement steps. Since there are minor differences between the contexts or machines of the same level between versions, for the purpose of analysis we only chose the final version of the model, dated 24th August. The model consists of eleven machines and contexts, each numbered from 0 to 10.

We used a set of metrics we established in [47], which is based on the syntactical metrics created for the programs on the code level by Halstead [48]. The metrics we use are adapted for the Event-B setting and the syntax of Event-B language (well identified operators, such as "and", "or" and "implication", as well as operands, such as sets, constants and variables). They have been previously verified on a large and complex system from the aerospace domain. The metrics are applied separately to each machine and context, and comprise of:

- size of a vocabulary (n) of a specification (machine or context); defined as a sum of distinct operators and operands;

- a size of a specification (N); it is a sum of operator and operand occurrences;
- the information contents of the program volume (V); it relates the size of the vocabulary (n) and the size of the specification (N);
- the difficulty level (D), representing the difficulty experienced during writing a specification, meaning modelling and proving; it is proportional to the number of distinct operators and occurrences of operands, and inversely proportional to the number of distinct operands (4). One should note that in practice, since there is a possibility that no operators are used in a machine (empty events with skip) or in a context (either no axioms or theorems are present or sets are given without their properties), the result of D after computation could be undefined;
- the effort (E) of modelling and proving a specification; defined as dependant on the number of proof obligations (automatic and interactive), the volume V and difficulty D.

In Figure 24 and Figure 26 we present the measurements and computed metrics for the dynamic and static parts of our model. Note that we use a logarithmic scale for plotting data, since there are quite big differences between the values of particular groups of data (e.g., for machines values of n are roughly within a hundred, whereas values of N within thousand, values of V are up to 8000, values of D are, again, lower – below hundred and values of E are very high – up to 350 thousand). Additionally, negative or zero values cannot be plotted correctly on log charts. Only positive values can be interpreted on a logarithmic scale. Division by zero handled by substituting zero to the result.

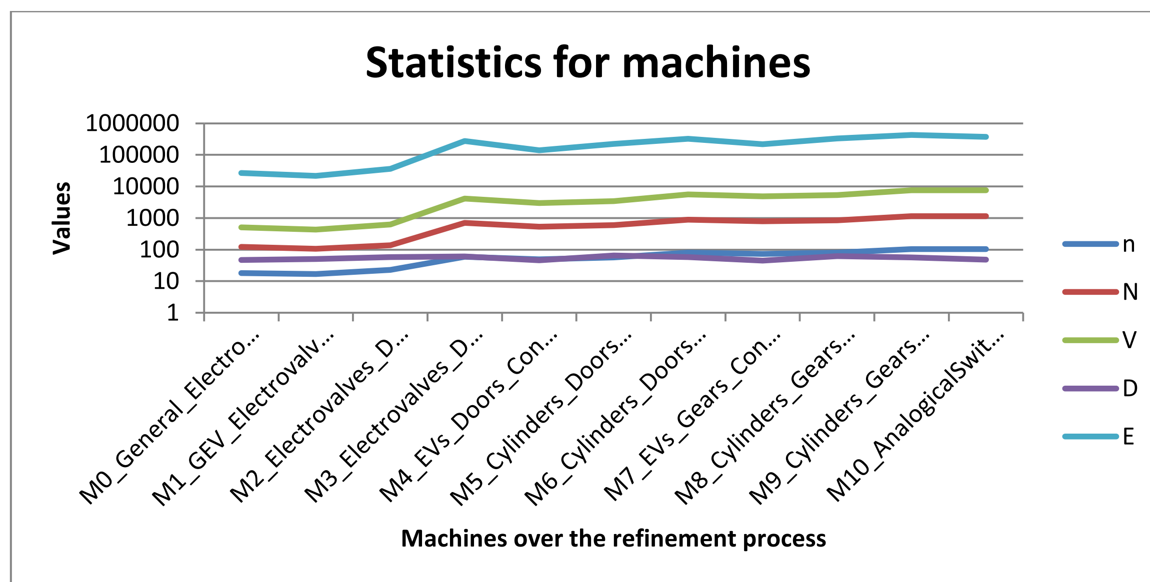


Figure 24 Statistics for the dynamic part of the model

In Figure 24 we observed that each machine and its refinement have steady pace of development denoted by no sudden peaks in the chart for the size indicators (n, N and V). This signifies that the system requirements were well decomposed into features and that they were modelled in a stepwise way following the refinement process. The difficulty and effort results (D and E) are also confirming the even development growth, D showing the difficulty in modelling and E the effort required form modelling and proving the machines. Most of the plots are displayed in parallel to each other, meaning the measurements are correlated (n, N, V, E). The only exception is the difficulty plot (D), which is almost "flat". It suggests that the difficulty of modelling was kept on the same level throughout the development, regardless of the size of the machine.

Additionally, in Figure 25 we display the effort distribution that was computed from the dynamic part of our model (machines). We can compare it with the development effort reported by the developer in Figure 21. It can be observed that the former solely represents the modelling and proving activities, whereas the latter also shows other activities related to the development (including modelling and proving). Note also that the effort slightly rises while the development continues. We attribute it to the fact that the model gets larger and, thus, requires more effort to comprehend all the relations between the system functionalities and properties.

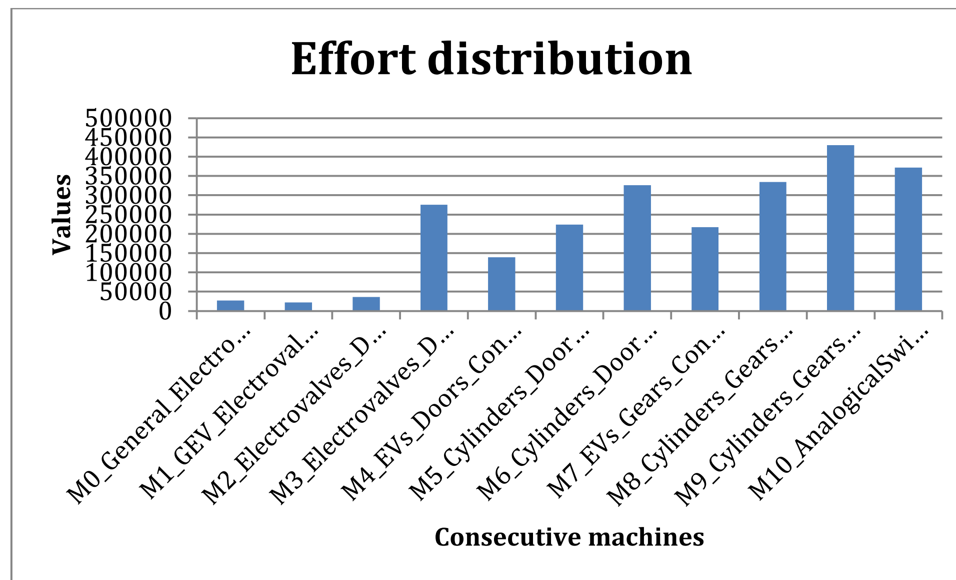


Figure 25 Distribution of effort based on the model measurements

In Figure 26 we observe the measurements for the static part of the model throughout the refinement process displayed on the logarithmic scale. One can see that there are regular peaks in the chart, which are regarding contexts C0 (general electro-valve), C3 (specific electro-valves for doors and gears), C6 (cylinders of doors) and C9 (cylinders of gears). These contexts include the parameters of the components being introduced at these refinement steps. The C0 is introducing one component, whilst C3, C6 and C9 "peaks" are caused by introducing several components in one step.

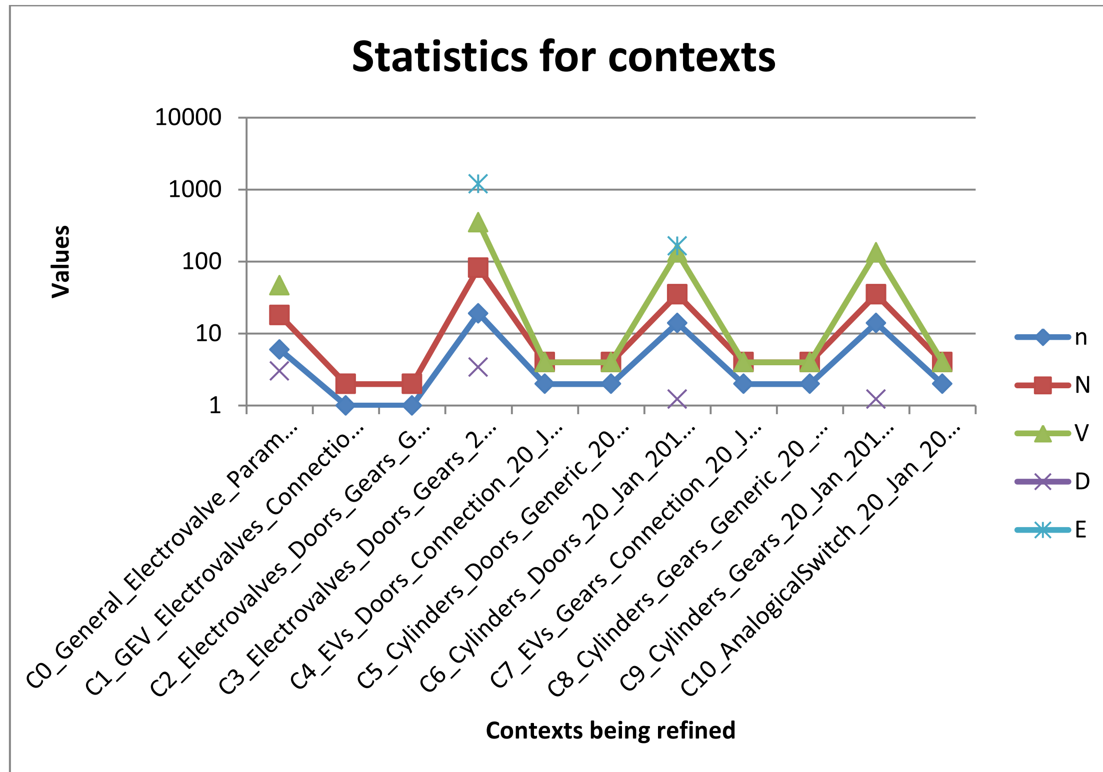


Figure 26 Statistics for the static part of the model

4.5. Observations of the Developer

In this section we present the remarks made by the developer during the review and retrospective meetings of the experiment, as well as some generic comments that were raised during the development. We first describe the technical issues that appeared during the experiment, as we believe they can re-appear in other developments using a reusable approach (parameterised components); only afterwards we present some observations related to process improvement.

We aggregated our comments from both sprints, as there was certain degree of overlap in our observations regarding iterations one and two. We provide the problem followed by our reasoning on what could have been the cause and (possibly) a suggestion on how to handle such situation.

We found the following statements regarding the technical side of the development especially interesting:

- Refinement from generic component (placeholder) to several different components (e.g., valves) working in parallel occurred to be problematic. There were issues between how the generic component was defined and how it could be refined (refinement restrictions). It was implicitly caused by the Event-B limitations (see future work).
- Finer granularity (better connectivity between the components) was needed in refinement strategy, which led to the development of reusable refinement patterns [42]. However, stepwise development and instantiations were working well, due to the prior knowledge of the developer regarding such a refinement strategy – the basic specifications of the components were available, only some refactoring was needed.
- Some tasks turn out to be particularly time consuming, although they did not seem to be, when planned. For instance the refinement of generic component (e.g., into valves) was not at all

straightforward and required a lot of effort to model and prove. Therefore, more careful planning and better monitoring combined with some estimation mechanisms would be valuable when scheduling the requirements within iterations. We suppose that larger projects would benefit from having the column "Complexity / story points" in the trac document, as it would facilitate project management and reduce the risk for requirements to be pushed to the following iteration due to lack of time in current iteration.

As for the development process, we noticed that *a more thorough plan would be needed* (item pool and/or item backlog), especially if the project is within an unfamiliar domain or the development team has little experience. The detail level of such plan is mostly related to technology (Rodin tool) and/or formal method (Event-B) restrictions, which are only encountered while actively modelling properties (not when working with requirements and planning the modelling). In our case these involved for instance proving the connections of refined components, which were not considered when creating item pool, or refining generic components to specific ones. In both cases there were some constraints on logical side of refinement, which required choosing a more detailed refinement strategy ("softer refinement").

Although agile processes stand against plan driven developments, there should be some planning and anticipating involved in the project due to the fact that the sequence of modelling matters and it impacts the difficulty of proving. The order of modelling depends on how the requirements are processed and prioritised, as well as refinement rules. In our case the order of introducing components did not matter, as the development was based on library of components that needed to be linked.

Moreover, we observed that the *daily meetings could* actually *take place less frequently*, as there are situations, where the progress of the development is not visible and there is not much to present or discuss (e.g., planning the development in more detail or decomposing a problem). However, the meetings should be held regularly, for instance once per two days. The daily meetings should be more tightly related to the item backlog and priorities of requirements to be modelled, so that they follow the sequential character of stepwise refinement. With this respect sequential development following refinement rules does not fully conform to rapid and iterative development suggested by agile methodologies. Finally, the dailies should be treated as safeguards for the development, specifically help in monitoring the actual direction of a development and its original plan. This is especially important when working in a team.

We also noticed that the *frequency of meetings should be dependent on the complexity of a problem*, i.e. the more complex the problem, the more frequent meetings. These meetings should not be under the "dailies" banner, as they might be too technical for this purpose. Nevertheless, they should facilitate communication, knowledge exchange and decision making. The main idea behind it is not to get stuck with the development and by that support the agile philosophy of a continuous progress.

Formal modelling requires mathematical background and application of formal methods differs from using traditional development methods and languages. In order to efficiently model and prove a system involves not only (mathematical) knowledge, but also largely benefits from experience. Gathering knowledge on how to formally model a system is in fact a complex, and often long, process and involves steep learning curve. Therefore, having an *expert* on site (or on-call) *for consultancy and monitoring* purposes would facilitate and smoothen the development. Presence of such a specialist in the project would mainly concern obtaining advice on the development strategy, handling the restrictions of Event-B language and refinement relations. Finally, *model reviews* done by such a skilled person (or a person of some experience in formal modelling) would be beneficial for the development and serve as a mechanism for assuring that the chosen design decision is "good enough". The reviews could also help to decrease the risk of subjective thinking (biased decisions of a modeller). The model could then be shown to the expert (or stakeholder) in two ways – statically (proofs) or dynamically (using ProB [49]).

Our final remark regards the tool support and multiple plugins that are optional for the Rodin platform. In our project we noticed that tweaking the tool, e.g., by extending the time for the provers to work, increases the possibility to automatically discharge some of the proof obligations.

5. Conclusions

The experimentation on using Event-B in an agile context, in particular Scrum development process, was necessary to investigate our claims regarding their synergy (given in our previous work [44]). We are aware that the setting for the experiment was small-scale. However, we believe that the analysis and our observations will shed some light on what kind of issues need special attention when choosing Event-B as a modelling language in an agile setting, as well as what kind of fine-tuning of an agile process is needed to benefit the most from the agile and formal merge.

Juxtaposing the development iterations (see Table 6) and the data regarding submissions to the repository lead us to the following remarks:

- One refinement step and iteration are not equal. This is especially the case when a refinement strategy involves decomposing a problem into several smaller ones. This leads to subsequent comment:
- There can be several refinement steps in a single iteration. In fact, we would recommend this manner of developing a system, since it is easier to backtrack and pinpoint some erroneous or inaccurate assumptions, or make changes to the model, as the complexity of a problem is being divided to tasks with smaller complexities.
- Submissions to the svn system are often not daily, since the "Done" status might not necessarily be obtained at the end of the working day. For instance, in case there is more proving involved or a change in the modelling approach is needed. Therefore, one refinement step does not equal to one commit to the repository.
- Set of meetings proposed by Scrum is facilitating the communication within and transparency of the development. However, the daily meetings can be less frequent, depending on the nature of a specific development.
- Sequential nature of refinement does not fully conform to rapid and iterative development suggested by agile methodologies. This is especially the case when a new feature request is made, which means that the created model needs to be adjusted to fit the new properties (reengineering the model).
- Scrum process helps to monitor and manage the development with respect to the planned modelling. The item pool and item backlog supports organisation of requirements and, specifically the latter, requirements prioritisation. Furthermore, the sustainable pace and continuous progress is controlled in a twofold manner: by the meetings and the backlog.

5.1. Validity of Experimentation

The validity discussion shows how reliable the results of the experimentation are. It also illustrates the truthfulness of work and discusses the risks of bias that could potentially be injected in the investigation by the researcher. We sequentially consider *construct, internal, external and reliability (conclusion) validity* [15].

The *construct validity* is a check whether the problem being investigated truly reflects the observations drawn from the experimentation. Here we managed to keep the artefacts being studied and the context of our study transparent for all parties involved; everybody had the same level of familiarity of the terminology

used and concepts being studied. The experimentation was supposed to validate the claims we made in our previous paper, regardless whether they would be confirmed or refuted. Therefore, we were not biasing the experimentation with our expectations. Finally, we avoided the mono-method bias, as we support our observations with qualitative and quantitative means.

The *internal validity* discussion involves identifying the influences that impacted the studied problem and thus could have caused the observed effect. To our knowledge there were no threats to internal validity that we would not describe in the context of the experiment (see Section 3.3).

We held a small-scale experimentation (a pilot study), so that we cannot generalise our findings. In this respect the *external validity* is threatened. However, we believe that our findings are of interest to other people outside the investigated case (as motivated in the introductory section of this paper). Our intention was to show hands-on research, which is based on the concepts theoretically investigated in our previous work. We believe that our results are relevant when extended to other cases with characteristics common with our experimentation. Since we described the environmental characteristics of our work (methods, tools, experimental setting), the others can themselves assess the applicability of presented approach to their specific context.

We believe that there is a threat to *conclusion validity*, as we have not statistically checked the soundness of our results. There are many variables that can impact the experimentation, for instance, experience of the developer with a formal method or the tool that supports it; familiarity to agile processes; domain knowledge of the problem to be modelled, etc. However, they were well described in the section about experimental setting (see Section 3.3). Furthermore, we did not bias the investigation, as were not fishing for a certain result (we were quite curious about the results of the study). Finally, for the quantitative analysis we used the metrics that we verified in our previous work [47].

5.2. Implications of Research and Future Work

The synergy of formal method and agile philosophy has a potential, since the well-defined development methods are complemented with efficient and flexible development process, respectively. We believe that our results can be, to some degree, transferable to other formalisms and different agile methods. We suppose that further advances both on formal method end (Event-B) and agile method end (Scrum) will foster the development of systems even more.

We are currently investigating guidelines for modelling in Event-B, which will aid the developer with modelling decisions. The guidelines would be beneficial for the experienced formal method users with developing their modelling strategy and fine-tuning of modelling approach. Moreover, they could serve as recommendations for the less experienced ones. Finally, they could be used in the training sessions, such as the university courses.

As for Scrum, we would like to continue our work on fine-tuning the Scrum process to the specifics of Event-B modelling. Therefore, as future work, we plan to perform experimentation in academic setting, which would involve students working on a project course. Formal methods, specifically Event-B would be used as part of the development (modelling some logical relationships), whereas all the development process would be of agile type, preferably scrum. We are aware that the future experimental setting differs from the one described in this paper. For instance, having students as modellers might mean that the formal part will be more difficult for them due to their minor experience. Moreover, some learning process for the use of formal method and associated toolset might be necessary when preparing for the formal part of the development.

The idea of experimentation with the synergy of formal and agile approaches was born when we came across the DevOps philosophy [44]. We were wondering if formal methods are at all suitable in DevOps

context [50] [51]. Now, we are aware what challenges and opportunities there are in such a mix for the development part of DevOps and want to continue our study on its operational part.

Acknowledgements

This work was carried out within the project ADVICeS, funded by Academy of Finland, grant No. 266373.

References

1. Butler, R.: What is Formal Methods? In : NASA LaRC Formal Methods Program. (2001)
2. Manifesto for Agile Software Development. (Accessed February 11-13, 2001) Available at: http://agilemanifesto.org/
3. Holloway, M.: Why Engineers Should Consider Formal Methods. In : AIAA/IEEE16th Digital Avionics Systems Conference (1997)
4. Alliance, A.: What is Agile? In: Agile Alliance. Available at: https://www.agilealliance.org/agile101/what-is-agile/
5. Larsen, P., Fitzgerald, J., Wolff, S.: Are Formal Methods Ready for Agility? A Reality Check. In : Second International Workshop on Formal Methods and Agile Methods, Pisa (2010)
6. Bowen, J., Hinchey, M., Janicke, H., Ward, M., Zedan, H.: Agility, Security, and Evolution in Software Development., 86-89 (2014)
7. Olszewska, M., Waldén, M.: FormAgi – A Concept for More Flexible Formal Developments. TUCS TR, Åbo Akademi University, Turku (2014)
8. Boniol, F., Wiels, V.: ABZ 2014: The Landing Gear Case Study. In Boniol, F., Wiels, V., Ameur, Y., Schewe, K.-D., eds. : Proceedings of Case Study Track, held at the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z, Toulouse, France, pp.1-18 (2014)
9. Savicks, V., Butler, M., Colley, J.: Co-simulation Environment for Rodin: Landing Gear Case Study. In Boniol, F., Wiels, V., Ait Ameur, Y., Schewe, K.-D., eds. : ABZ 2014: The Landing Gear Case Study, Toulouse, France, vol. 433 of the series Communications in Computer and Information Science, pp.148-153 (2014)
10. Arcaini, P., Gargantini, A., Riccobene, E.: Modeling and Analyzing Using ASMs: The Landing Gear System Case Study. In Boniol, F., Wiels, V., Ait Ameur, Y., Schewe, K.-D., eds. : International Conference on Abstract Statemachines, Alloy, B, TLA, VDM, and Z, Toulouse, France, vol. 433 CCIS, pp.36-51 (2014)
11. Pressmann, R.: Software Engineering: Apractitioner's Approach 8th edn. The McGraw-Hill Companies (2014)
12. Jaspan, C., Keeling, M., Maccherone, L., Zenarosa, G., Shaw, M.: Software Mythbusters Explore Formal Methods., 60-63 (2009)
13. Shafiq, S., Minhas, N.: Integrating Formal Methods in XP — A Conceptual Solution., 299-310 (2014)
14. Parsa, M., Snook, C., Olszewska, M., Waldén, M.: Parallel Development of Event-B Systems with Agile Methods. In Mousavi, M., Taha, W., eds. : Proceedings of 26th Nordic Workshop on Programming Theory, NWPT '14. Halmstad University, Halmstad (2014)
15. Wohlin, C., Ruenson, P., Höst, M., Ohlsson, M., Regnell, , Wesslén, A.: Experimentation in Software Engineering. Springer (2012)
16. Sfetsos, P., Stamelos, I.: Formal Experimentation for Agile Formal Methods. In : Proceedings of the 1st South-East European Wokshop on Formal Methods, Thessaloniki, Greece, pp.48-56 (2003)
17. Snook, C., Butler, M.: UML-B: Formal Modelling and Design Aided by UML., 92-122 (2006)
18. Rumbaugh, J., Jacobson, I., Booch, G.: Unified Modeling Language - A Reference Manual 2nd edn. Pearson Higher Education (2004)
19. Schneider, S.: The B-method: An Introduction. Palgrave Macmillan (2001)

20. Snook, C., Butler, M.: UML-B and Event-B: an integration of languages and tools. In : Proceedings of the IASTED International Conference on Software Engineering, Innsbruck, Austria, pp.336--341 (2008)
21. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilic, D., Latvala, T.: Supporting Reuse in Event B Development: Modularisation Approach. In : Abstract State Machines, Alloy, B and Z: Second International Conference (ABZ) (2010)
22. Abrial, J.-R.: Event Model Decomposition. Available at: http://wiki.event-b.org/images/Event_Model_Decomposition-1.3.pdf
23. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
24. Abrial, J.-R.: "Extending B without Changing it (for Developing Distributed Systems). In : Proceesings of 1st Conference on the B Method, Nantes (1996)
25. Back, R.-J.: Refinement Calculus, Part II: Parallel and reactive programs. Stepwise Refinement of Distributed Systems. Åbo Akademi (1990)
26. Back, R.-J., Kurki-Suonio, R.: Decentralization of process nets with centralized control. 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, 131-142 (1983)
27. Back, R.-J., Sere, K.: From modular systems to action systems. Software - Concepts and Tools 17, 26-39 (1996)
28. Event-B: Home of Event-B and the Rodin Platform. http://www.event-b.org/index.html (2008)
29. Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (1996)
30. Platform, R.: http://www.event-b.org/platform.html. (2006)
31. Takeuchi, H., Nonaka , I.: New New Product Development Game. Harvard Business Review 86116, 137-146 (1986)
32. Poppendieck, M., Poppendieck, T.: Lean Software Development: An Agile Toolkit. Addison-Wesley Professional, Boston, MA, USA (2003)
33. Anderson, D., Reinertsen, D.: Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press (2010)
34. Klipp, P.: Getting Started with Kanban. Amazon Digital Services (2014)
35. Beck, K.: Extreme Programming Explained. Addison-Wesley (2000)
36. Beck, K.: Extreme Programming Explained: Embrace Change, 2nd edition. Addison-Wesley Professional (2004)
37. Ambler, S., Lines, M.: Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise 1st edn. IBM Press (2012)
38. Tudor, D., Tudor, I.: The DSDM Atern Student Workbook: A Guide to the Definitive Agile Framework. Galatea Training Services Ltd (2010)
39. Consortium, D.: The DSDM Atern Handbook. DSDM Consortium (2013)
40. Jones, C.: Applied software measurement: assuring productivity and quality 3rd edn. McGraw-Hill, Inc., New York (1996)
41. Sutherland, J.: ScrumInc. In: SCRUM: Keep Team Size Under 7! Available at: http://www.scruminc.com/scrum-keep-team-size-under-7/
42. Ostroumov, S., Waldén, M.: Facilitating Formal Event-B Development by Visual Component-based Design. TUCS Technical Report 1148, Turku Centre for Computer Science, Turku (2015)
43. Ostroumov, S., Waldén, M.: Formal Library of Visual Components. TUCS Technical Report 1147, Turku Centre for Computer Science, Turku (2015)
44. Olszewska, M., Waldén, M.: DevOps Meets Formal Modelling in High-Criticality Complex Systems. In : 1st International Workshop on Quality-Aware DevOps (QUDOS 2015), collocated with 10th Joint Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Bergamo (2015)
45. Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., Deardeuff, M.: Use of Formal Methods at Amazon Web Services., 66-73 (2015)
46. Olszewska, M.: On the Impact of Rigorous Approaches on the Quality of Development. Turku Centre for Computer Science (2011)
47. Olszewska (Pląska), M., Sere, K.: Specification Metrics for Event-B Developments. In : 13th International Conference on Quality Engineering in Software Technology (CONQUEST 2010), Dresden (2010)
48. Halstead, M.: Elements of Software Science Operating and programming systems series edn. Elsevier. Elsevier Science

Inc., New York, NY, USA (1977)

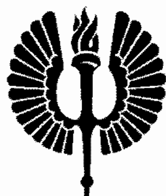
49. ProB: The ProB Animator and Model Checker. (Accessed November 18, 2013) Available at: http://www.stups.uni-duesseldorf.de/ProB/index.php5/Main_Page

50. Loukides, M.: What is DevOps? O'Reilly Media, Sebastopol (2012)

51. Loukides, M.: What is DevOps (yet again)? Radar(radar.oreilly.com) (2015)

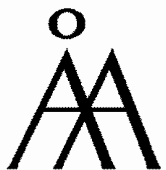
Turku Centre *for* Computer Science

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 978-952-12-3342-5
ISSN 1239-1891