Quality Assurance in Scrum Applied to Safety Critical Software

Geir K. Hanssen¹✉, Børge Haugset¹, Tor Stålhane²,
Thor Myklebust¹, and Ingar Kulbrandstad³

¹ SINTEF ICT, Strindveien 4, 7465 Trondheim, Norway
{Geir.K.Hanssen, Borge.Haugset, Thor.Myklebust}@sintef.no

² NTNU, Sem Sælandsvei 9, 7491 Trondheim, Norway
stalhane@idi.ntnu.no

³ Autronica Fire and Security AS, Haakon VII's Gate 4, 7041 Trondheim, Norway
Ingar.Kulbrandstad@autronicafire.no

Abstract. Various agile methods have several quality assurance mechanisms embedded in the process itself, without any explicit QA role. In principle, the team takes care of quality assurance during sprints and as part of daily stand-ups, sprint reviews and retrospectives. We have defined SafeScrum, a variant of Scrum with some additional XP techniques that can be used to develop safety-critical software and have the software certified according to the IEC 61508 standard. This imposes a load of additional requirements on the process. In a recent industrial case, we have experienced that the quality assurance mechanisms in Scrum becomes insufficient. We have therefore analyzed the standard, consulted an independent assessor and worked with the Scrum team to identify necessary additional tasks for a team-internal QA role to be added to the SafeScrum process.

Keywords: Safety critical software · Scrum · Safescrum · IEC61508 · Quality assurance

1 Introduction

Agile software development methods and in particular variants of Scrum, often in combination with XP techniques, has had a large uptake in the software industry over the past decade. One of the many aspects of Scrum and similar approaches is that quality assurance is embedded in the process itself, and not explicitly documented. A Scrum team is supposed to be self-sustained, not having to rely on an external quality management or assurance function like a QA manager or QA department. The latter has been a typical role in line organizations doing plan-based development [19, 20]. First of all, a Scrum project enforces visibility and has frequent evaluation of status, progress and problems, which is used to re-plan and improve the project based on the most recent and updated knowledge. Scrum also has dedicated activities for managing quality issues with both the product under development and the process itself; each short work-period, or sprint, concludes with a sprint review and potentially also a retrospective. The former evaluate the results so far and the latter evaluates the process itself to identify

improvement needs and opportunities. Scrum also strongly emphasizes frequent interaction with the customer or the problem owner, and XP stresses continuous and frequent testing. This is necessary to ensure that the functionality as well as the quality of the system meets requirements and expectations. In short—Scrum can be seen as a combined and self-sustained planning, development, *and* quality assurance process, although lacking traceability.

Scrum was initially designed for small development projects with small self-managed teams, solving small-scale problems. Research from the past decade provides examples that this works well and that Scrum projects are more effective and flexible than plan-driven projects [4]. However, the trend in the software industry today is that Scrum is being used in increasingly more complex settings. We see cases where globally distributed Scrum teams collaborate in developing large software systems [6]. We also see that Scrum is being used for development of safety-critical systems, which have to comply with strict quality and safety standards [5, 14, 18].

This trend of increasing complexity means that the core principles of self-sustained, multi-disciplinary, and self-managed teams are challenged. In this paper, we look into how this development affects the embedded quality assurance function in Scrum. We base our analysis and discussion on an ongoing industry case where a Scrum team develops a high-integrity fire and gas detection system where the goal is to achieve a SIL3 (SIL: Safety Integrity Level) rating according to the IEC61508 standard [7]. The Scrum process used by the team is adjusted and continuously refined to match the requirements of the IEC61508 standard; we name this variant SafeScrum [18] (see Sect. 4 for more details).

In the following, we present some background on the inherent challenges when developing and certifying safety-critical systems, on how SafeScrum can be adapted to support this process and the role of quality management and assurance. We then look into our case to show how this is being done in practical terms before we use our insights from the case to discuss how Scrum can be enforced to manage quality in high complexity settings.

2 Quality Assurance in Agile Software Development

Mnkandla and Duolatzky gives a thorough discussion of the use of the term quality [11]. Most of the definitions identified by the two authors have a production focus and are not relevant for development. The concept of quality—software or otherwise—is defined by ISO 9000 as “the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs”. According to this definition, the main concept of quality is to make the customer happy. Deliver a quality product thus means to deliver a product that is according to the customer’s specified and implied requirements. Sticking with older definitions—e.g., quality is conformance to specifications—ignore the customer and is not a smart move in a competitive industry.

Inherent in the plan-driven approach to software development is the idea that all requirements are known at the start of the project. With faster innovation, more rapid requirement changes, and more volatility allowed in the prioritized tasks, this idea is

void. Plan-driven development creates a significant risk that the users' requirements remain unclear or that important opportunities for innovation are missed. The problem is that up-front requirements become increasingly irrelevant as the pace of innovation quickens and customers' expectancy to fit-for-use rises. Consequently, the discrepancy between software practice, end-users and traditional requirements specification widens. Requirements that are out of sync with real needs has been claimed as a common cause of terminated IT projects [13].

Agile development attempts to allow for frequent updates of the requirements as the customer's needs and problem understanding develop over time, thus increasing the probability of delivering improved product quality.

3 Safety Critical Software Development

A system is defined as safety-critical if a failure may result in death or severe injury to people, loss or severe damage to property, or harm to the environment. Examples of such systems are fire alarm systems (failing to sound an alarm may cause casualties) or railway signaling systems (signal error may lead to collisions etc.). Safety critical systems are found in almost all parts of our daily lives, from transportation, to energy systems, in medical devices etc. Traditionally, such systems have been hardware reliant, but as hardware has become more powerful, flexible and programmable, the trend is that larger parts of the total system are implemented in software, meaning that the software complexity is growing. For example, the top-notch fighter plane of the forties, the Spitfire, had zero lines of code. Today, the F-35 fighter has about 8 million lines of code where most of them comprise what could be defined a safety critical system [10].

Safety critical systems may be classified with a SIL value, defining the level of performance of the safety function of the system, or in other words, how likely the system is to operate as intended. The classification of levels varies between different standards, but for IEC61508, which is relevant to our case, SIL is divided from 1 to 4 where SIL 4 is the highest safety integrity level.

In order to use a safety critical system, the customer needs a certificate. A "software certification demonstrates the reliability and safety of software systems in such a way that it can be checked by an independent authority with minimal trust in the techniques and tools used in the certification process itself" [3]. The certificate is an independent document that ensures that the system operates as specified and according to the safety standard. This introduces the role of the assessor, an independent third party with the responsibility of assessing and eventually certifying that the development process leading to the system is compliant with the requirements in the standard. It is important to understand that a standard like IEC 61508 mainly has requirements for the *process*. As stated in the introduction: "This International Standard sets out a generic approach for all safety lifecycle activities for systems comprised of electrical and/or electronic and/or programmable electronic (E/E/PE) elements that are used to perform safety functions." E.g., the section on architecture contains material on *how* to select architecture but no material on *what* the architecture should look like.

Looking back at the past decades, we see that the development process of safety critical systems is optimized for hardware development, where design decisions have to be made early and locked prior to implementation to avoid late change in design, which may impose very high costs. Normally some variant of the V-model is used to guide design, implementation, testing and validation.

```

graph LR
    A[E/E/PE system safety requirements] --> B[Software safety requirements]
    B -- Validation --> C[Validation testing]
    C --> D[Validated software]
  
```

Fig. 1. The V-model

During software development it is important to control size and complexity. Complexity is obvious – high complexity hinders understanding and can thus lead to errors. However, experience shows that sheer size also will create problems for developers since a large code volume makes it difficult to keep the overview, which again leads to coding errors.

4 SafeScrum – Agile Development of Safety Critical Software

SafeScrum is a variant of the well-known and extensively used Scrum development model [16] where some additional elements are added to be able to fulfill the process requirements from the IEC61508 standard [18]. Based on a thorough investigation of the requirements in the standards part 3 which defines the software part of the total system [14], we propose a set of extensions to make Scrum applicable to development of safety critical software. Firstly, there are two backlogs, one for functional requirements and one for safety requirements. Functional requirements may change frequently whilst safety requirements normally are stable and even reusable between projects and products. Relationships between these are maintained to keep track of which safety requirements that are affected by which functional requirements. Secondly, SafeScrum needs to be a traceable process. All decisions and changes throughout development must be documented, stored and made available to the assessor. The same goes for code reviews where all remarks and how they were resolved needs to be kept track of. Thirdly, each sprint encompasses a validation of the safety of the present system. As part of the sprint review of each sprint, the product backlog may be updated. In cases where a change is considered to affect the safety of the system, a change impact analysis (CIA) [17] is done – and documented. Here the two backlogs come in handy as a mean to identify how a change related to a functional requirement potentially influences a safety requirement. Besides these extensions, common features of agile practices are important, like test-driven development (important to establish high test coverage), regular work iterations, daily stand-ups, and continuous integration.

A series of sprints replace the ‘coding’ part and the ‘evaluation’ part of the V-model. This means that documentation is produced continuously and *as a part of* development and not as a finalizing phase as described by the V-model. SafeScrum simply replaces the bottom and right side of the V-model. This is fundamentally important for *software* development and enables a project to become more flexible with respect to changes and still be able to provide the needed documentation and traceability to the assessor.

Adding these new elements to Scrum is necessary to meet the requirements of the IEC61508 standard. However, this also compromises the concept of a lightweight process as a lot of extra work, checkpoints and especially documentation are added. The most important countermeasure is clever and efficient use of tools to automate as much of the extras as possible. We have identified four classes of tools. Firstly, we need a tool to support process and workflow management, like defining, assigning and following up tasks, their responsibilities and order, etc. Put simply, this is a tool to automate the Scrum board. Secondly, we need tools to establish and maintain traceability of requirements, tests and code. Even a small project will generate large amounts of information, which requires tool support. Thirdly, we need dedicated tools for managing information like design, code and architectural documentation. Fourthly, we need tools to support code quality assurance, which is particularly important with respect to IEC61508. This includes test coverage analysis, static code analysis and test automation. There are plenty of tools to choose from and many of them are flexible and can be combined and linked to create a tool chain to support the SafeScrum process. The IEC61508 standard has

requirements to tools and should be checked (IEC61508-3, Sect. 7.4.4, and table A3). The walkthrough of our case will provide concrete examples of such a tool chain (Fig. 3).

Fig. 2. The SafeScrum model

5 A SafeScrum Case

The authors have been working with Autronica Fire & Security for about two years in order to detail and trial the SafeScrum process in a real SIL3 industrial case. This collaboration is part of a large R&D project, partly funded by the Research Council of Norway. The collaboration is organized as an action research project [2] and the case being described here shows some of the findings, so far. All data are collected, managed and reported according to a joint R&D contract.

Autronica Fire & Security [1], with 380 employees, is an internationally leading provider of fire and gas detection systems. A large part of the business is offshore installations at oilrigs and ships where demands for safety performance are high. Our case is a project developing new software for a fire detection system, SIL 3.

The authors have followed the case project from the start and collected data in the form of 1) observations of sprint review and planning meetings (11), 2) analysis of documentation like project plans and requirements documents, and 3) interviews and discussions with the Scrum team and related roles. This also includes the assessor (a TÜV organization), which has been asked to comment on our development process (SafeScrum). This dialogue has been important to ensure that the development process and the documentation it produces meet the expectations and is aligned with the IEC61508 standard. The product being developed by the case project itself is however not yet certified – the completion is still some years ahead. In addition to this participative role in the shaping of SafeScrum, the researchers also made an analysis of the standard

to identify any issues with respect to using an agile method [18]. Such issues were discussed with the Scrum team to ensure compliance. The SafeScrum process has emerged through practice. It started out by using only a few fundamental principles, like short iterations and daily standups. Based on growing experience and dialogue between researchers, team members, the product owner and the external assessor, the SafeScrum process and related tools are continuously being refined and extended.

The project started development in January 2014 with a Scrum team of five experienced and co-located developers, one of them also acting as Scrum Master. Previous experience and education in Scrum were low – only one of the developers had some experience with Scrum from his previous workplace. Other than that the team read a basic introduction [8], and in addition the researchers had a few seminars at the case company, addressing agile principles, research and examples.

This was the first attempt at applying Scrum in the development department and started out with a simplistic process. Sprint duration was set to four weeks. The team shared workspace with separate offices and a common area in between and used a whiteboard with stickers to track work items and their flow from planned, in progress to done. The product owner role was managed by a company internal with extensive knowledge of the market, the requirements, and the technology. The team focused on producing working code and unit tests from early on and a put emphasis on improving the Scrum process continuously. Each sprint ended with a sprint retrospective where the process was evaluated by the team, adjustments were made and new tools were trialed and added/removed as needed.

After a few sprints, Jira was introduced to manage the workflow and thus replaced the manual scrum board. RMsis, a plug-in for Jira, was used to establish traceability of the requirements management process. Confluence was used to support team collaboration and to document the sprints, e.g. by storing memos from sprint reviews. Stash and Git was used to manage software version control and code reviews and Bamboo was used for continuous builds, tests and release management. Doxygen was used for maintaining design and code documentation. In addition, a set of tools was used for additional quality assurance; Gtest and Gmoc were used to manage unit tests, Squish Coco was used for code coverage analysis and QAC/QACPP for static code analysis.

In total, these tools constitute a tool-chain where some of the tools are linked and operate as a greater whole. In particular, Jira serves as a hub in the tool chain. There are many alternative tools and ways of composing them, but in our case, this setup enables the Scrum team to be both agile and to automate many of the additional requirements imposed by the IEC61508 standard. Figure 3 shows how the tools are inter-related.

Fig. 3. The tool-chain supporting the SafeScrum process

6 The Need for Extra Attention to Quality Management

The shaping and introduction of SafeScrum at the case company has been done step by step and by building on growing experience. The described process extensions and the tool chain made the team able to produce and maintain the information required to achieve a SIL3 certificate. However, after a few months, and sprints, of operation we saw that this also puts a burden on the team, and in particular on the Scrum master who spent an increasingly large amount of time making sure that development was compliant with IEC61508. Some corrective actions were made but there was a concern that some glitches happened. It was also not clear what the most important considerations to make were. In short, the project needed some clarification on quality assurance, and asked the researchers for assistance.

This led us to consult the assessor to clarify his expectations for QA on three topics:

1. *Question:* Traceability of safety related requirements: Is it sufficient to have a trace between documents or should it be possible to trace issues down to sections, pages, or lines in the text?

Answer: The assessor requires a link between requirements and tests, e.g. by referring to unique requirements ID in test cases.

Response: This level of trace is handled by a dedicated requirements management tool, RMsis, linking requirements to tests that validates them, as well as linking requirements and tests to design and code. However, we have identified a need to manually verify that this is done correctly and to make necessary corrections. The QA role shall continuously verify that traceability is kept up to date and verify that all steps of the process are done.

2. *Question:* The standard directs ‘limited use of interrupts, pointers and recursion’ – are there exact limits or does it mean that we should avoid this as much as possible?
Answer: You should make a table with use of interrupts, pointers etc. and a simple explanation for why this must be used.

Response: Someone needs to check for “bad code” in every sprint and decide whether it should be corrected (refactored) or whether the table of reasoned exceptions should be updated. This can be assisted by checking for pointers during static code analysis.

3. *Question:* Control of size and complexity: should we aim for specific limits for module complexity?

Answer: You need to define something that is useful to you, and argue why – the standard does not specify this.

Response: We have defined a set of metrics and their values that are important for this specific project. Checking code is done by tools (QAC and QACPP) but there is a need to verify that this is done in every sprint and that any deviations are resolved, typically by refactoring or improving code in the following sprint.

In addition to these issues raised from the dialogue with the assessor, we also saw that quality of documentation and test coverage had become unsatisfactory. All in all, it became obvious that the self-regulating quality mechanisms in Scrum were overloaded and that there was a need to strengthen the QA function. A dedicated QA function is a necessary part of SafeScrum, but it is important to remember that the entire team still has a large responsibility for QA as well. The team plays an important role e.g. in retrospectives to continuously improve the development process and in resolving issues that are identified by the QA-role.

7 Shaping an Embedded QA Role in SafeScrum

Using the insights described above, the researchers, the Scrum team and the product owner had a series of meetings to define a specific QA role in SafeScrum to alleviate the problems. The traditional approach in this and similar organizations would be to place the QA role as a specialized function in the line organization, outside the project. We decided, however, to add the QA role to the Scrum team to be close to the activities and to the information needed to execute quality assurance. This adds to the principle of cross functional teams [12] in Scrum. We are also considering making this a rotating role to make it a shared responsibility and to share the workload. We need a QA log to trace findings, decisions, corrective actions, and the follow up/results of these. In our case, Confluence is a good tool to establish this log. We have identified four tasks:

QA Task 1: Check Code Metric Values for New or Changed Code: Neither the IEC61508 standard nor the assessor provide directions on specific metrics and limits to monitor at the component level. We have consulted the research literature [9] and used Minitab to analyze data from code from previous projects at the case company to define the following metrics and limits: (1) number of static paths – STPTH: 250, (2) McCabe’s cyclomatic number – STCYC: 15, (3) number of parameters – STPAR: 5, (4) function

call count – STSUB: 13, (5) maximum nesting of control structures – STMIF: 5, (6) number of executable lines – STXLN: 70, and (7) Myer's value – (STMCC): 10.

QAC is used to analyze new and changed code to produce values for the metrics. This is done at the end of each sprint. The metrics are displayed, together with their defined maximum values in a radar plot. It is thus easy to see if there are metrics that are exceeding their defined limits. If the values are inside their limits, QA will accept the code. If one or more values are outside their limits, the code is presented by QA in the sprint review meeting where the team decides to either accept the violation or plan refactoring. If the violation is accepted, a brief explanation must be added to the log and potentially also in the code (required by the standard). If the violation is unacceptable, the team needs to define a new task in Jira to refactor the code.

QA Task 2: Check Documentation Coverage: Check new/changed code to ensure proper inline documentation and documents in Doxygen. This has to be done manually. In case of missing or poor documentation, the QA-log should be updated and the findings should be discussed in the sprint review meeting to decide how to resolve it (giving a task to someone in the team). This check could be done at the end of each sprint.

QA Task 3: Check Test Coverage: Check for code coverage using Squish Coco. The QA log should be updated with references to uncovered code. This could be checked by the end of each sprint. Uncovered code should be discussed at the sprint review meeting and the team should define corrective actions, like defining tasks to produce tests. According to the standard, the coverage should be at least 99%.

QA Task 4: Check Requirements-Task-Code Traceability: For new requirements, tasks, and code check that 1) requirements (RMs) is linked to issues (e.g. in Jira), and 2) that code (e.g. Stash) is linked to issues (e.g. Jira). The QA role should control consistency at the end of the sprint and the team should resolve any identified issues immediately. The IEC61508 standard provides a set of explicit requirements for traceability; see table A.4 – Software design and development – detailed design, and table A.5 – Software design and development – software module testing and integration. We consulted the assessor about a definition of 'module' and its size (LOC). He referred to part 7, chapter C.2.9: *"a software module should have a single well-defined task or function to fulfill."* The assessor recommended 1000 LOC as an upper limit. In cases where the limit is exceeded it should be explained and documented.

The tasks have been defined to be as simple and inexpensive as possible, partly by using the tool chain that already is in place. The goal is to not add more work to the process ceremony than strictly needed, but to use the QA role to simplify the sprint reviews so that they are not bogged down with unimportant details. So far, this is done by letting the QA close issues that have low risk and complexity on his own in advance of the sprint retrospective, where the whole team participates. This reduces the time spent on unimportant decisions and helps the team focus on difficult tasks where a joint evaluation and decision is needed.

There should be a list of which criteria's the team members shall have done before the issue can be set as resolved and the QA needs to check that they have been fulfilled (not do them himself). The project should also create a list of which criteria's where the QA can close an issue without any further investigation. Only QA or the team can move an issue from resolved status to closed status. Closed issues cannot be changed.

8 Conclusions

After around two years of shaping and using SafeScrum we see that the inherent quality assurance mechanisms in Scrum are not sufficient to meet the demands imposed by the IEC61508 standard. We have consulted the assessor to ensure a proper match with the standard and shaped a new role to Scrum.

Developing safety critical solutions using SafeScrum, calls for a lot of extra attention, ceremony and documentation, which initially may be seen as a threat to the ability to be agile [13]. However, we see that the iterative nature of Scrum with frequent breaks between the sprints in addition to the tool chain we have put into use makes it possible to manage quality assurance internally in the team without adding too much extra work.

As part of further work we will look into opportunities to streamline and perfect this new role as it is vital to maintain an efficient SafeScrum process and to meet the requirements of the IEC61508 standard and the assessors expectations. One viable step would be to add tool support to assist the QA role in order to collect and analyze quality information with less effort and with more precision. The authors have been involved in the U-QASAR FP7 EU project, which has developed a tool that serves as a central quality dashboard consolidating quality information in a unified overview. We will look into integrating this with the tool chain as a new tool for the team and the QA role.

Acknowledgements. This work was partially funded by the Norwegian research council under grant #228431 (the SUSS project) and the EU FP7 project U-QASAR (grant agreement no. 318082). Research has been done in collaboration with Autronica Fire & Security AS. We are grateful for valuable input from the external assessor.

Open Access. This chapter is distributed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (<http://creativecommons.org/licenses/by-nc/4.0/>), which permits any noncommercial use, duplication, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, a link is provided to the Creative Commons license and any changes made are indicated.

The images or other third party material in this chapter are included in the work's Creative Commons license, unless indicated otherwise in the credit line; if such material is not included in the work's Creative Commons license and the respective action is not permitted by statutory regulation, users will need to obtain permission from the license holder to duplicate, adapt or reproduce the material.

References

1. Camarinha-Matos, L.M., Afsarmanesh, H. (eds.): Collaborative Networked Organizations: A research agenda for emerging business models, 1st edn, p. 346. Springer, Heidelberg (2004)
2. Davison, R.M., Martinsons, M.G., Kock, N.: Principles of canonical action research. *Inf. Syst. J.* **14**(1), 65–86 (2004)
3. Denney, E., Fischer, B.: Software certification and software certificate management systems. In: NASA Ames Research Center (2005)
4. Dingsøyr, T., Dybå, T., Moe, N.B.: Agile Software Development: An Introduction and Overview, in Agile Software Development. Springer-Verlag, Heidelberg (2010)
5. Fitzgerald, B., Stol, K.-J., O’Sullivan, R., O’Brien, D.: Scaling agile methods to regulated environments: an industry case study. In: Proceedings of the 2013 International Conference on Software Engineering. IEEE Press (2013)
6. Hanssen, G.K., Smite, D., Moe, N.B.: Signs of agile trends in global software engineering research: A tertiary study. In: Proceedings of International Conference on Global Software Engineering (ICGSE). IEEE, Helsinki (2011)
7. IEC, 61508: 2010 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (E/E/PE, or E/E/PES)
8. Kniberg, H.: Scrum and XP from the Trenches - How we do Scrum. Enterprise Software Development Series, Plesa, D. (ed.) InfoQ. 131 (2007)
9. Krusko, A.: Complexity analysis of real time software – using software complexity metrics to improve the quality of real time software. In: Department of Numerical Analysis and Computer Science, Royal Institute of Technology (KTH): Stockholm, Sweden, p. 97 (2004)
10. Martin, L.: A Digital Jet for the Modern Battlespace (2015). <http://www.f35.com/about/life-cycle/software>
11. Mnkanla, E., Dwolatzky, B.: Defining agile software quality assurance. In: International Conference on Software Engineering Advances, IEEE (2006)
12. Moe, N.B., Dingsøyr, T., Dybå, T.: A teamwork model for understanding an agile team: A case study of a Scrum project. *Inf. Softw. Technol.* **52**(5), 480–491 (2010)
13. Myklebust, T., Stålhane, T., Hanssen, G., Wien, T., Haugset, B.: Scrum, documentation and the IEC 61508-3:2010 software standard. In: International Conference on Probabilistic Safety Assessment and Management (PSAM). PSAM, Hawaii (2014)
14. Myklebust, T., Stålhane, T., Hanssen, G.K., Wien, T., Haugset, B.: Scrum, documentation and the IEC 61508-3:2010 software standard. In: Proceedings of Probabilistic Safety Assessment & Management conference (PSAM12). Self-published, Oahu, USA (2014)
15. Myklebust, T., Stålhane, T., Haugset, B., Hanssen, G.K.: Using a goal-based approach to improve the IEC 61508-3 software safety standard. In: Proceedings of the Twenty-Third Safety-Critical System Symposium, Bristol, UK, 3rd-5th February 2015
16. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice Hall, New Jersey (2001)
17. Stålhane, T., Hanssen, G.K., Myklebust, T., Haugset, B.: Agile change impact analysis of safety critical software. In: Bondavalli, A., Ceccarelli, A., Ortmeier, F. (eds.) SAFECOMP 2014. LNCS, vol. 8696, pp. 444–454. Springer, Heidelberg (2014)
18. Stålhane, T., Myklebust, T., Hanssen, G.K.: The application of Scrum IEC 61508 certifiable software. In: Proceedings of ESREL. Helsinki, Finland (2012)
19. Talby, D., Keren, A., Hazzan, O., Dubinsky, Y.: Agile software testing in a large-scale project. *IEEE Softw.* **23**(4), 30–37 (2006)
20. Vaibmu. QA role in Agile Teams (2013). <http://www.uqasar.eu/qa-role-in-agile-teams/>