was used on interview transcripts and observations to find out codes, concepts and finally the categories related to our core. Table 3 shows an example of selective coding process.

The other concepts and categories emerged in a similar manner which sheds light on the problems faced by agile teams while adopting test automation. Observations gathered from the two projects were also analyzed and compared to the concepts derived from the interviews. It was found that our observations supported the data provided in the interviews, thereby strengthening our interview data. During our data analysis one more set of concepts emerged that formed the strategies used by agile teams in order to overcome those challenges as described in the present study. Figure 1.a shows different levels of data abstraction using GT and Fig. 1.b explains the emergence of category choosing the right tool from underlying concepts.

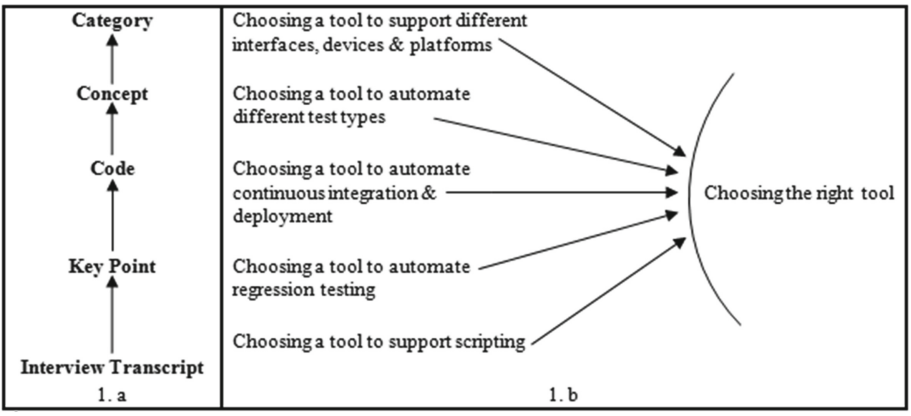

Fig. 1. **a.** Different levels of data abstraction in GT. **b.** Emergence of category *choosing the right tool* from concepts

Determining Theoretical Saturation. The selective coding continues until the researcher has sufficiently integrated the core category and its connections to other relevant categories [20]. On reaching a stage where further data collection and its analysis were leading us to the same categories with no new data, we found out that our categories have reached saturation. Then we started sorting the theoretical memos conceptually and this process is called sorting that forms the theoretical outline of our study.

The last step in GT is generating a theory also know as Theoretical Coding. It involves the conceptualization of how different categories and their associated properties relate to each other as hypothesis so that can be integrated into a theory [19, 26]. We followed Glaser's guidelines and performed theoretical coding at the later stages of analysis [14].

Table 4 shows different concepts and categories that form the challenges and corresponding concepts that form the adopted strategies while practicing test automation on agile projects. Also, the number within the parenthesis indicates the number of interviewees who referred these challenges/strategies. As the codes, concepts, and categories

emerge directly from the data, which is collected from the real world, the resulting theory is grounded within the context of the data [17].

Table 4. Strategies adopted on different agile projects

Challenges	Strategies
Choosing the right tool (26)	• Know your test automation requirements, Know your tool (14) • Cost Benefit Analysis (CBA) (11)
Managing test environment (15)	• Upfront planning for managing test environment (11) • Virtualization (10)
Test script maintenance (18)	• Automation testing framework (12) • Page Object Model (POM) (8)
Mindset toward automation (17)	• Engender automation awareness (12) • ROI evaluation (11)
Effective communication (16)	• One team approach (10)

In the following section, we present the research findings from our study. Selected interview quotations are provided under each category to better explain it in the present context. Our results are grounded further by key points, codes, and concepts from the interviews as well as the observations from two agile projects. It is difficult to describe here in detail due to space reasons.

3 Results: Adopting Test Automation on Agile Projects

In this section, we present our grounded theory: Strategies used by agile practitioners while adopting test automation in their projects. We have selected quotations from our study to explain the challenges faced by agile teams and strategies opted by them.

3.1 Challenge 1: Choosing the Right Tool

Test automation is very important right from the start of any agile project. It is essential to know the project requirements, which tests needs to be automated and what tools are needed. Agile practitioners admitted that while transitioning to scrum and XP, they were still using traditional record and playback tool but results were highly unsatisfactory.

Other associated concerns include choosing a tool for automating continuous integration and deployment, automating acceptance and regression tests and a tool for effective test management.

> *"Output of sprint N has to combine with sprint N + 1, daily defect fixes that continuously check in to the code, this whole process is continuous integration (CI), it also takes lot of time, and only by automating our CI process we could survive our project deadlines."* – P10, Senior Tester

Choice of test automation tool particularly in agile projects is a very crucial decision as if you would end up choosing a wrong tool with the partial or incomplete evaluation; it may

lead to loss of efforts spent in each sprint, loss of licensing fees as well as loss of automation opportunities. In order to prevent these losses, some strategies were used to overcome the problem of choosing the right tool. Two adopted strategies are explained below:

Strategy 1: Know your Test Automation Requirements, Know your Tool. One should be scrupulous while choosing a test automation tool in agile projects. Agile teams should understand their project needs and then decide on test automation tool, it is imperative to first know the exact automation requirements of the projects like test types (unit, acceptance, regression, etc.) needs to be performed, coding languages to be used on the project and suitability of choosing between licensed and open source tools; it is good to choose a tool based upon the compatibility with the application under test (AUT).

> *"A lot of licensed and open source tools are available…You must know that what you want to do with that [Tool] and for what [purpose] as requirements may vary depending on project size, cost and allocated time."*– P16, Test Analyst

Strategy 2: Cost Benefit Analysis (CBA). Cost of the tool is also one of the important deciding factors in most agile projects. Licensed tools have certain benefits over open source tools like good user support, sufficient training material and ease of use but that comes with the cost.

> *"…would be using that [tool], whether it's a licensed or open source it depends on CBA (Cost to benefit analysis) of that tool w.r.t our project."* – P32, Scrum Master

It is always better to know what test types needs to be automated, tools utility with project needs, its ability to integrate with other project and defect management tools.

3.2 Challenge 2: Managing Test Environment

The ultimate aim of any agile project is to deliver quality product and test automation plays an important role in adding that quality to the product in such short sprint durations. Keeping test environment as close as possible to production environment ensures the quality of the test automation. Agile teams were facing difficulties while creating multiple test environments for every different configuration, platforms and workflows.

> *"Why it is worth to have Test Automation in agile projects because it helps you in achieving your quality objectives, test environment should be a replica of your live [production] environment…if you practice this then the code that go into upper [production] environment would meet quality criteria."* – P13, Agile Coach

Strategy 3: Upfront Planning for Managing Test Environment. Testing whether it is automation or manual is only been successful when performed in the proper test environment. In agile, it is very common to have multiple test environments, multiple configurations for the single business application so upfront planning for managing test environment is very important.

"… important to have upfront plan for managing your test environments... by maintaining spreadsheets containing all our test environment related information like different configurations, different test devices and test data used by those devices, any database related information and continuously update it." – P29, Scrum Master

Strategy 4: Virtualization. It serves an important strategy in managing issues related to test environment management. Virtual machine setup provides that additional space to both developers and testers to test their application under test (AUT). It was used to reduce the overhead caused by different OS and hardware configurations.

"…by using virtual machines test environments can be created according to the requirement and the scope of the test… Above all it is scalable and has on demand access which reduces our burden of managing test environment."– P25, Test Analyst

Participants were using a document to gather different test environment requirements to plan for managing their existing environment or building a new. VMware workstations were also used for managing test environments related issues.

3.3 Challenge 3: Test Script Maintenance

For every new addition or modification in feature, test script needs to be modified and maintained for the entire duration of the projects with multiple sprints and this was a challenge for them.

"…The scale of regression testing grows with each sprint and so does the test scripts, so how you would add more test cases to the existing regression test suite? How you maintain those scripts?" – P34, Senior Tester

Maintainability of code was a big issue, many participants worked on web based applications where test script was created by identifying web page elements and their associated properties, so if any page element whether it is a dropdown box or submit button had changed then they needed to track and modify that script.

Strategy 5: Automation Testing Framework. Majority of our participants admitted that having a good automation testing framework solved their test script maintenance problem to the larger extent. Automation testing framework is an engine that runs your automation test scripts with the help of some tool like Selenium or Unified Functional Tester (UFT) to test your application under test. Most commonly used frameworks were: Data driven framework – modular functions are stored in external files and called by test scripts; Keyword driven framework – keyword is assigned to every user action (like button click), stored in a spreadsheet and called by test scripts; Hybrid framework – combination of data and keyword driven frameworks; and Behaviour driven framework – creating examples to describe the user behavior while using the application under test.

Strategy 6: Page Object Model (POM). Another technique used by many agile practitioners to make test script maintenance easier was Page Object Model (POM) approach. Here, each web page element (button, text box) is modeled as an object within the test code and represents as one class.

3.4 Challenge 4: Mindset Toward Automation

Whenever any project is transitioning to agile then it is important to have support from the management so that every team member proactively put up his concern and ask for any assistance that is needed to overcome any constraint regarding implementing test automation. They need to understand that test automation is a long term investment and should support the team by providing enough budget and time.

> *"Transition to agile…need support from your senior management particularly when you embrace test automation in agile…have realistic expectations from the team and…accept initial failures and invest in terms of tools or trainings…only this kind of thinking can encourage use of test automation in any agile project."* – P20, Tester

Strategy 7: Engender Automation Awareness. Agile teams need a shift in their thinking while adopting test automation. They should know the merits and demerits of having test automation in their projects and how to use it [test automation] effectively.

> *"When you wrap test automation around agile…not easy to adapt as your team won't have that thinking that agile demands…to create automation awareness in your team…try to create it by providing coaching, workshops or short trainings on test automation in agile environment."* – P13, Agile Coach

Strategy 8: ROI Evaluation. Senior management should provide the required infrastructure and environment necessary to conduct effective test automation practices. Eleven of our participants used ROI (Return on Investment) evaluation to get their support. ROI calculation is based on evaluating the benefits of test automation with respect to its implementation costs in terms of tool cost, manpower cost, time needed to build required infrastructure for automation.

3.5 Challenge 5: Effective Communication

Many participants admitted that lack of communication in their teams often results in poor automation planning, late feedbacks and wrong automation effort estimates. Test automation is teamwork and should be taken care of by both developers and testers.

> *"…have to consider a lot many things…plan automation, what features to automate in each sprint, when to start automation and one thing is crucial…conversation element - PO talking to developers, testers talking to developers and creating a wonderful coordination with effective communication."* – P38, Product Owner

While implementing test automation, it is very important for developers and testers to collaborate with each other, testers should help developers in designing unit test cases and developers should help testers in automating acceptance tests. The more they communicate more effective test automation would become.

Strategy 9: One Team Approach. One team approach was the key crusader in building effective communication between testers, developers and PO's as mentioned by ten agile practitioners. Many agile teams were giving much emphasis to have proactive communication with each other including both verbal and written communication

so that every team member developed this feeling that they are working together as one single team not as separate entities.

> "*When you automate…expected to not only report defects but also to communicate [defects] effectively to the development team and track it till closure. When you have that [proactive communication] surrounding your team that keeps everyone in one loop then results are more than satisfactory.*" – P32, Scrum Master.

If there is any defect then it should be properly determined whether it is because of script or actually a test case has failed and it can only be possible when testers proactively talk to developers and also send a mail to team's group mail id for better information flow.

4 Discussion and Related Work

Agile projects have daily rounds of unit tests, integration tests, acceptance tests and continuous deployment. The serious effect of not having perfect test automation in place forms the rationale behind our study.

The choice of the right tool from a plethora of available tools is a decisive step towards successful test automation. This is confirmed by studies of Oliveira [27] and Collins [28]. If one tool is not working well for the project, in the next iteration, agile teams should try something new [28]. Yoder [29] discussed the importance of selecting automation tools and when automated tests should be run under "Automate First" pattern.

The implications of managing test environment and test script maintenance revealed by our findings are also supported by a number of studies. Deak [30] highlights a number of negative factors that influence testing like insufficient number of test environments and weak infrastructure. Karhu [31] contributes test environment, test maintenance and implementation time as key concerns about test automation infrastructure. Fewester et al.'s study [32] mentioned negative impact on test automation cost due to improperly managed test script maintenance cost. Bach [33] advocates the benefits of test automation over maintenance cost of constantly changing test scripts suite.

For successful test automation, management should be open to test automation practices and their financial benefits in spite of time constraints. Late testing mindset need to be changed to early testing mindset in agile environment [34] and management support is also desired in terms of having realistic expectations from the test automation [35].

According to [34] efficient communication and interaction between testers and developers improved both testing and development, eventually improving information flow and efficiency in process. Graham [36] suggested active participation of testers in requirement reviews along with developers for performing test planning in parallel. Yoder [29] also reported whole team approach as one of the pattern for agile quality mindset.

5 Limitations

The inherent limitation with grounded theory research study is that the research findings are grounded in the specific contexts that are explored in the research. Data triangulation was used for reducing researcher bias, as we gathered the data from two sources, namely, interviews and observations that may yield more reliable data than using a single data source. The context in this research was governed by our choice of research destinations and the availability and accessibility of agile practitioners to participate in this study. We do not claim that our findings are universally applicable to all the agile projects practicing test automation, however, they accurately characterize the contexts studied.

6 Conclusion

A Grounded Theory study has been conducted over a period of eighteen months that involved 38 agile practitioners from 18 software development organizations in India. This study investigated the test automation adoption from the specific perspective of agile practitioners through their real life project experiences using GT. Unlike most of the participant organizations, some of them were recently transitioned to agile software development methods. However, all of them were striving to build good test automation infrastructure for their projects. During the study, we discovered the various challenges and strategies adopted thereof by agile teams while establishing good test automation practices in their projects. Main contribution of this paper is towards understanding the key challenges while adopting test automation in agile projects and providing some widely used strategies to overcome those challenges. This study can be utilized by agile software development teams to have a plan of action and streamline the test automation to get maximum benefits. We acknowledge this fact that all challenges and strategies adopted by software development organizations practicing test automation in agile projects may not have emerged in this study. This may also serve as the foundation for conducting future studies in the same area.

Acknowledgments. Our big thanks to all agile practitioners for participating in this study. This research is supported by our institute's TRF academic grant. Thanks to Prof. Yogesh Singh for his immense support and guidance.

References

1. Sayed, I.N.: The case of agile testing. White Paper, cognizant 20-20 insights (2016). https://www.cognizant.com/InsightsWhitepaper. Last accessed 08 Jan 2016
2. Gao, J., Tsao, J., Wu, Y.: Testing and Quality Assurance for Component-Based Software. Artech House, Boston (2003)
3. Dustin, E., Rashka, J., Paul, J.: Automated Software Testing: Introduction, Management, and Performance. Addison-Wesley, Boston (1999)
4. Cohn, M.: Succeeding with Agile: Software Development Using Scrum, 1st edn. pp. 314–316. Addison-Wesley Professional, Boston (2009)

5. Gregory, J., Lisa, C.: More Agile Testing. Addison-Wesley, Upper Saddle River (2015)
6. Puleio, M.: How not to do Agile testing. In: Proceedings of the Conference on AGILE 2006 (AGILE 2006), pp. 305–314. IEEE Computer Society, Washington, DC (2006). doi:http://dx.doi.org/10.1109/AGILE.2006.34
7. Collins, E., Lucena Jr., F.: Strategies for agile software testing automation: an industrial experience. In: Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW 2012), pp. 440–445. IEEE Computer Society, Washington, DC (2012)
8. Glaser, B.: Grounded theory institute: methodology of Barney G Glaser (2010). http://groundedtheory.org/. Last accessed 28 Nov 2015
9. Hoda, R., Noble, J., Marshall, S.: Agile undercover: when customers don't collaborate. In: XP 2010, Norway, pp. 73–87 (2010)
10. Goulding, C.: Grounded Theory: A Practical Guide for Management, Business and Market Researchers. Springer, Berlin (2002)
11. Dorairaj, S., Noble, J., Malik, P.: Understanding team dynamics in distributed agile software development. In: Wohlin, C. (ed.) XP 2012. LNBIP, vol. 111, pp. 47–61. Springer, Heidelberg (2012). doi:10.1007/978-3-642-30350-0_4
12. Corbin, J., Strauss, A.: Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, 4th edn. Sage, London (2015)
13. Charmaz, K.: Constructing Grounded Theory, 2nd edn. Sage (2014)
14. Glaser, B.: Basics of Grounded Theory Analysis: Emergence vs. Forcing. Sociology Press, Mill Valley (1992)
15. Dorairaj, S., Noble, J., Malik, P.: Understanding lack of trust in distributed agile teams: a grounded theory study. In: 16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012), pp. 81–90. IET (2012)
16. Hoda, R., Noble, J., Marshall, S.: Organizing self-organizing teams. In: ICSE 2010, pp. 285–294. ACM, South Africa (2010)
17. Martin, A., Biddle, R., Noble, J.: The XP customer team: a grounded theory. In: Proceedings of the AGILE Conference, pp. 57–64 (2009)
18. Whitworth, E., Biddle, R.: The social nature of Agile teams. In: Agile 2007, pp. 26–36. IEEE Computer Society, USA (2007)
19. Glaser, B.: Doing Grounded Theory: Issues and Discussions. Sociology Press, Mill Valley (1998)
20. Glaser, B.: Theoretical Sensitivity: Advances in Methodology of Grounded Theory. Sociology Press, Mill Valley (1978)
21. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Sociology Press, Aldine (1967)
22. Agile Software Community of India. http://www.agileindia.org/. Last accessed 12 June 2016
23. Agile India 2016. http://www.2016.agileindia.org/. Last accessed 10 Feb 2016
24. Urquhart, C., Lehmann, H., Myers, M.D.: Putting the 'theory' back into grounded theory: guidelines for grounded theory studies in information systems. Inf. Syst. J. **20**(4), 357–381 (2010)
25. Georgieva, S., Allan, G.: Best practices in project management through a grounded theory lens. Electron. J. Bus. Res. Methods **6**(1), 43–52 (2008)
26. Glaser, B.: The Grounded Theory Perspective III: Theoretical Coding. Sociology Press, Mill Valley (2005)
27. Oliveira, J.C., Gouveia, C., Filho, R.Q.: A way of improving test automation cost-effectiveness. In: CAST. EUA, Indianapolis (2006)

28. Collins, E., Lucena Jr., F.: Software test automation practices in agile development environment: an industry experience report. In: Proceedings of the 7th International Workshop on Automation of Software Test (AST 2012), pp. 57–63. IEEE Press, Piscataway (2012)
29. Yoder, J.W., Wirfs-Brock, R., Washizaki, H.: QA to AQ part six: being agile at quality "Enabling and Infusing Quality". In: HILLSIDE Proceedings of 23rd Conference on Pattern Languages of Programs, October 2016
30. Deak, A.: A comparative study of testers' motivation in traditional and agile software development. In: Product – Focused Software Process Improvement, pp. 1–16 (2014)
31. Karhu, K., Repo, T., Taipale, O., Smolander, K.: Empirical observations on software testing automation. In: Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation (ICST 2009), Denver, Colo, USA, pp. 201–209 (2009)
32. Fewster, M.: Common Mistakes in Test Automation, Grove Consultants (2001). https://www.stickyminds.com/sites/default/files/presentation/file/2013/01TAU_M5.pdf. Last accessed 02 Feb 2016
33. Bach, J.: Test automation snake oil. Windows Tech. J., 40–44 (1996)
34. Taipale, O., Smolander, K.: Improving software testing by observing practice. In: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering (ISESE 2006), pp. 262–271. ACM, New York (2006). doi:http://dx.doi.org/10.1145/1159733.1159773
35. Kettunen, V., Kasurinen, J., Taipale, O., Smolander, K.: A study on agility and testing processes in software organizations. In: Proceedings of the 19th International Symposium on Software Testing and Analysis, pp. 231–240 (2010)
36. Graham, D.: Requirements: requirements and testing: seven missing-link myths. IEEE Softw. **19**(5), 15–17 (2002). doi:10.1109/MS.2002.1032845

Safety Critical Software

How is Security Testing Done in Agile Teams? A Cross-Case Analysis of Four Software Teams

Daniela Soares Cruzes[1(✉)], Michael Felderer[2], Tosin Daniel Oyetoyan[1], Matthias Gander[2], and Irdin Pekaric[2]

[1] SINTEF Digital, Trondheim, Norway
{danielac,tosin.oyetoyan}@sintef.no
[2] University of Innsbruck, Innsbruck, Austria
{michael.felderer,matthias.gander,irdin.pekaric}@uibk.ac.at

Abstract. Security testing can broadly be described as (1) the testing of security requirements that concerns confidentiality, integrity, availability, authentication, authorization, nonrepudiation and (2) the testing of the software to validate how much it can withstand an attack. Agile testing involves immediately integrating changes into the main system, continuously testing all changes and updating test cases to be able to run a regression test at any time to verify that changes have not broken existing functionality. Software companies have a challenge to systematically apply security testing in their processes nowadays. There is a lack of guidelines in practice as well as empirical studies in real-world projects on agile security testing; industry in general needs a more systematic approach to security. The findings of this research are not surprising, but at the same time are alarming. The lack of knowledge on security by agile teams in general, the large dependency on incidental pen-testers, and the ignorance in static testing for security are indicators that security testing is highly under addressed and that more efforts should be addressed to security testing in agile teams.

Keywords: Security testing · Agile testing · Case study research

1 Introduction

Security testing can broadly be described as (1) the testing of security requirements that concerns confidentiality, integrity, availability, authentication, authorization, non-repudiation [16] and the testing to validate the ability of the software to withstand attack (resiliency) [28]. This process can be performed by showing conformance with the security properties, similar to requirements-based testing; or by trying to address known vulnerabilities, similar to traditional fault-based testing. It is essential to take testing into account in all phases of the secure software development lifecycle, i.e., analysis, design, development, deployment, as well as maintenance. Thus, security testing must be holistic covering the whole secure software development lifecycle. Proper security testing requires a mix of techniques as there is no single testing technique that can be performed to effectively cover all security testing and their application within testing

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 201–216, 2017.
DOI: 10.1007/978-3-319-57633-6_13

activities at unit, integration, and system level [2]. Nevertheless, many companies adopt only one security testing approach, for instance penetration testing.

Agile testing is one approach that is increasingly being adopted by software companies. This approach does not just mean testing on agile projects, but testing an application with a plan to learn about it and let the product information and customer feedback guide the testing. Agile testing involves immediately integrating changes into the main system, continuously testing all changes and updating test cases to be able to run a regression test at any time to verify that changes have not broken existing functionality [18, 23]. In agile software development, there is a focus on the feature implementation and delivery of value to the customer and, as such, non-functional aspects of a system should also be of attention. Non-functional requirements testing is challenging due its cross-functional aspects and lack of clarity of their needs by business in the most part of projects, therefore, although important, the non-functional requirements are often neglected in agile testing for many reasons, such as experience, culture, awareness, priority, cost and time pressure [5].

There is a lack of guidelines in practice as well as empirical studies in real-world projects on security testing; for agile projects in general needs a more systematic approach to security. The main contribution of this paper is to deepen relevant knowledge and experience on the characterization of security testing in an agile context. Based on the "traditional waterfall testing approaches and techniques", we have analyzed four teams and asked about how they perform these in the agile context. We then provide recommendations of ways to improve it based on lessons learned and good practices from the cases. In addition, we provide an improved understanding on how research and practice are aligned.

The remainder of the paper is organized as follows. In Sect. 2, we provide background on software and security testing. It also forms the backbone of the used interview guide. Section 3 presents the research methodology and describes how the studies were conducted. Section 4 presents the main findings of the case studies. Section 5 discusses the cross-case analysis findings. Finally, Sect. 6 concludes the paper and highlights directions of future work.

2 Background on Software and Security Testing

Software testing consists of all software development lifecycle activities, both static and dynamic, concerned with evaluation of software products and related artifacts to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects. Testing can be classified according to the three dimensions objective, scope, and accessibility shown in Fig. 1.

Test objectives are reason or purpose for designing and executing a test. The reason is either to check the functional behavior of the system or its nonfunctional properties. Functional testing is concerned with assessing the functional behavior of an SUT (System under Testing), whereas nonfunctional testing aims at assessing nonfunctional requirements with regard to quality characteristics like security or performance.

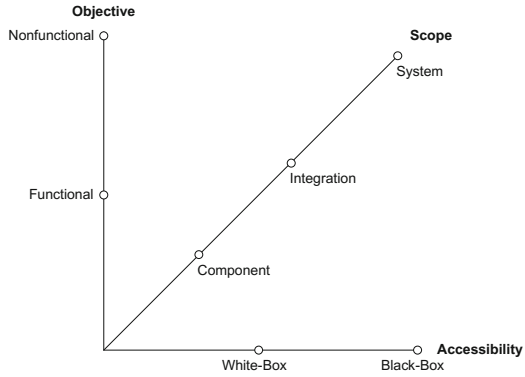

Fig. 1. Software testing dimensions objective, scope and accessibility (adopted from [16]).

The *test scope* describes the granularity of the SUT and can be classified into component, integration and system testing. It also determines the test basis, i.e., the artifacts to derive test cases. Component testing (also referred to as unit testing) checks the smallest testable component in isolation. Integration testing combines components with each other and tests those as a subsystem, that is, not yet a complete system. System testing checks the complete system, including all subsystems. A specific type of system testing is acceptance testing where it is checked whether a solution works for the user of a system. Regression testing is a selective retesting to verify that modifications have not caused side effects and that the SUT still complies with the specified requirements.

In terms of *accessibility* of test design artifacts we can classify testing methods into white-box and black-box testing. In white-box testing, test cases are derived based on information about how the software has been designed or coded. In black-box testing, test cases rely only on the input/output behavior of the software. This classification is especially relevant for security testing, as black-box testing, where no or only basic information about the system under test is provided, enables to mimic external attacks from hackers.

Security testing is testing of security requirements related to security properties like confidentiality, integrity, availability, authentication, authorization, and non-repudiation in addition to testing the resilience of the system against attack. In security testing, there are two principal approaches that can be distinguished, i.e., security functional testing and security vulnerability testing [33]. Security functional testing validates whether the specified security requirements are implemented correctly, both in terms of security properties and security mechanisms. Security vulnerability testing addresses the identification of unintended system vulnerabilities. It uses the simulation of attacks and other kinds of penetration testing attempting to compromise the security of a system by playing the role of a hacker trying to attack the system and exploit its vulnerabilities [1]. Furthermore, security vulnerability testing requires specific expertise, which makes it difficult and hard to automate [21]. By identifying risks in the system and creating tests driven by those risks, security vulnerability testing can focus on specific parts of a system implementation where an attack is likely to succeed.

Figure 2 abstracts from concrete security testing techniques mentioned before, and classifies them according to their test basis within the *secure software development life-cycle*, which takes security aspects into account in each phase of software development, i.e., analysis, design, implementation, deployment, maintenance, and additionally testing.

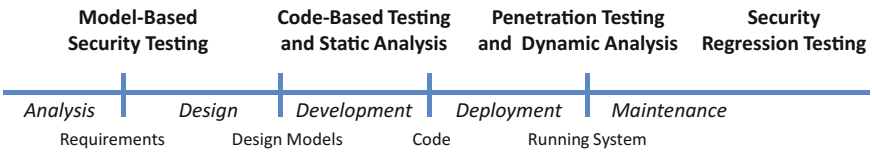

Fig. 2. Process for risk-based test strategy development (adopted from [16]).

Model-based security testing is grounded on requirements and design models created during the analysis and design phase. Examples are misuse cases and threat models. In misuse cases, test cases relating to an attacker's perspective are captured and used to exercise the system [31]. During the design, a threat model can be used to capture security issues and translated into test cases that can be used for security testing [20].

Code-based testing and static analysis is based on source, bytecode, or binary created during development. This testing approach in many cases uses static analysis tools to find code-based defects [6]. There is a range of issues that could be focused by a static analysis tool such as duplications, coding rules, code complexity, unit test coverage, and structural complexity. As regards security testing, specific frameworks exist that provide platform for common enumeration of security defects in the implementation and design. The Common Weakness Enumeration (CWE) [8] provides a formal list of software weaknesses. The OWASP Top-10 provides the list of the most common web application vulnerabilities [26]. The SANS Top-25 list shows the most widespread and critical errors that are applicable to all types of applications [11].

Penetration testing and dynamic analysis are based on running systems, either in a test or production environment. It is referred to as a black-box testing approach because the tester has no access to the source code of the system under test. Penetration testing seeks to break into running software but from ethical point of view. As a result, the rule of engagement must always be defined before such a test is carried out [28].

Refactoring and feature implementation may break existing security controls, increase the attack surface, and introduce new vulnerabilities into the system. In the agile context, it would be an activity that would need to be continuously performed to validate that the security properties of the system is not compromised.

2.1 Four Quadrants of Agile Testing

Crispin and Gregory [9] discuss the Agile Testing quadrants that are widely adopted in practice. Each quadrant in Fig. 3 reflects different reasons to test. Traditionally, software testing is involved late in the development process to detect failures, but typically not to prevent them. Companies focus almost exclusively on the right hand side (Q3 and

Q4), criticizing the product, but not playing a productive part in supporting the creation and guidance of the product (Q1 and Q2). In agile testing, the testers are not only involved in identifying, but also in preventing failures by continuous interaction with developers and customers. Automation is an important enabler for agile testing. Automation of the tests in Q1 is usually easiest to implement, and at the same time has a big impact on the process effectiveness. Tests in Q3 are usually performed manually. Tests in Q4 are heavily dependent on tools and specialized skill sets. But, manual exploratory testing by a knowledgeable security tester is indispensable to detect issues that automated tests can miss.

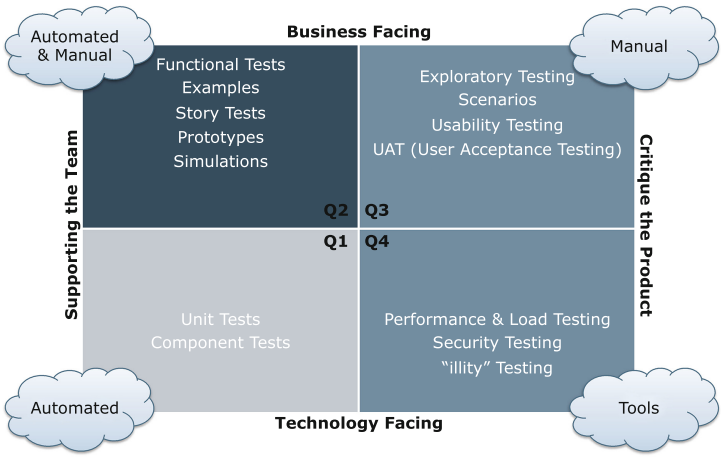

Fig. 3. Agile test quadrants [9]

Agile testing increases the need for improved communication and coordination between testers and developers, in addition to a new mind-set at the personal and organizational levels. In the rush to deliver functionality, most agile teams lack to think about security [5]. Authorization is often the only aspect of security testing that the agile teams consider as part of business functionality.

During the last years there have been several efforts to reconcile software security with the conflicting premises imposed by agile methodologies [4, 19, 24]. In a systematic review of agile challenges for secure software development Queslati et al. [24] conclude that the reported security assurance challenges are as follows: security assessment favors detailed documentation; tests are, in general, insufficient to ensure the implementation of security requirements; tests do not cover in general, all vulnerability cases; security tests are in general difficult to automate; and continuous changing of the development processes conflicts with audit needs of uniform stable processes.

Probably, the most widely known software security methodology is Microsoft's framework, which is integrated into the Microsoft Agile Security Development Lifecycle [22]. Other approaches also exist. Recently, Baca et al. [3] demonstrate how security features can be integrated into an agile software development method process at Ericsson AB. The approach focuses on risk management. Chólis et al. [7] describe a

case study of a software security testing process based on the Microsoft Software Development Lifecycle for Agile. The case company moves their software engineering teams from waterfall to agile. The case shows that a synchronization between the tasks of agile software engineering teams and the independent security team is possible. Türpe et al. [34] report on a one-year study of penetration testing and its aftermath at a major software vendor, and show how an agile development team managed to incorporate the test findings. Rindel et al. [30] describes a case of building a secure identity management system and its management processes. The project's steering group required the use of Scrum. In the implementations of this model the security testing, reviews and audits are viewed as normal stories in the sprint backlog and executed as part of the daily scrum.

Furthermore, security testing approaches for agile projects have especially been proposed for web applications [12, 32] and service-oriented systems [15]. These cases show how it is possible to integrate security testing into agile software development for specific system types. Our research comprises an independent study on the state of practice in security testing in agile teams.

3 Research Methodology

The overall goal of this paper is to investigate the role of security testing in agile teams, process-wise. For this purpose, we present the synthesis of the results of the four cases in security testing, highlighting the security engineering process, testing phases and techniques. The results of the interviews and context mapping provide insights into the recommended practices and lessons learned in the context of agile testing. The following three research questions (RQs) were investigated:

(**RQ 1**) How is the traditional security engineering process managed/organized in the agile teams?
(**RQ 2**) How does the agile teams perform security testing in each testing phase?
(**RQ 3**) How are traditional security testing techniques generally used in the agile software development lifecycle?

This study is carried out in four teams in two countries, i.e., Austria and Norway, within three organizations and denoted as 1, 2, 3-Team1, and 3-Team2, as shown in Table 2. Organizations 1 and 2 are located in the same country while organization 3 is located in another country. Organization 3 is a company with roughly 90 engineers. The team setup are both co-located and distributed. 3-Team1 has teams distributed in separate locations while 3-Team2 has the core development teams (frontend and backend) in the same location and interacts with a QA team that sits in a separate location. 3-team1 develops identity management APIs that are mainly consumed by other teams within the organization. They do not interact with external users. 3-Team2 on the other hand, develops solution for storage and processing of end user images and videos.

We prepared semi-structured interview guide (see Table 1) using a qualitative data collection approach that is based on in-depth literature review of the state-of-the-art in security testing. The interviews were compared with the collected information about the organizational contexts and interactions with the companies. The resulting interview

audios were then analyzed using the thematic analysis approach [10] to crosscheck and compare the answers in order to find behavioral confirmation and disconfirmation as well. The transcripts and recordings of the interviews were categorized, tabulated, and also analyzed by coding of the interviews. All the transcriptions and coding were validated with other researchers before analysis. By doing so, another researcher independently double-checked the codes and data to tag the key words, phrases and paragraphs. It is important to note that basic information on each context was considered (see Table 2). This information served as a context to better understand the points of view of each participant connected to the results. In this analysis, we considered in which areas the cases suggest the same points, where they differ, and where the cases conflict.

Table 1. Semi-structured interview guide

#	Questions	
1	Can you briefly describe the kind of system you develop? Back-end or Front-End?	
2	Can you give us a brief introduction of how your development team is organized? (Developers, Testers, Architects, CSOs, etc.), (Distributed, Co-located, etc.)	
3	How is your agile software development process? Which practices do you adopt? (Fill in the table with agile and lean practices)	
4	How is your security engineering process (for example, security requirements, secure design, secure coding, security testing) organized/managed in your team? Can you describe how you organize your security testing along these axes of the Fig. 1?	
5	Can you describe the kind of security testing that you perform in each testing phase listed below?	
	Phases of testing	**Components**
	Unit Testing	Classes, functions, statements, data
	Integration Testing	Modules, packages, etc.
	System Testing	System
	Regression Testing	Classes, Modules, System
	UAT Testing	System
	Production/Configuration Testing	System
6	Figure 2 shows the security testing techniques generally used in secure software development lifecycle. Could you talk about how you perform these activities in your agile software development? How often are security testing or security related activities done in your agile cycles? How do you decide when to perform them? How do you decide when not to perform them?	
7	Do you see benefits of performing security testing?	
8	On the test automation and continuous integration. Do you automate your testing activities? To what extent? How do you incorporate security testing in this process?	
9	Anything you would like to add?	

Table 2. Teams under study

Team	Team Size	Type of software	Other context information
1	20 Frontend and backend developers divided in teams of 5	Medical Information System	Applies a Scrum-based agile process; the software is certified according to medical standards
2	6 developers	Security service tools	Scrum-based agile process
3–A	21 developers (UI, Backend, Mobile, and Infrastructure)	Identity Management APIs that are consumed by other business units and teams	A mix of Agile Practices. Not specifically scrum by the book. DevOps approach is also spread used
3–B	22 developers (Frontend (web/mobile) and backend teams)	Mobile client and backend system for close storage and processing of images and videos	A mix of Agile Practices. Not specifically scrum by the book. DevOps approach is also spread used

4 Results

We collected our main findings in a mind map shown in Figs. 4, 5 and 6. These results are then discussed in more detail in the next subsections.

4.1 RQ 1: How Is the Traditional Security Engineering Process Managed/Organized in the Agile Teams?

We found three main themes from the interviews in relation to the roles and responsibility (Fig. 4). The first observation is that larger companies have their own chief security officer, who is not part of the teams to not interfere with any daily team activities. Sometimes the responsibility of the chief security officer overlaps with the project owner in order to ensure that the applications being developed do not impose security risks. One team mentioned that their project owner (PO) or project manager (PM) has domain-specific security knowledge, which is not the case for the other teams. In fact, for the smaller companies, there is no such chief security officer role. One problem that the teams experienced with involving the security officer is that it is hard to identify when to include him in the activities.

The second observation is that external experts are normally hired for penetration testing. However, a problem experienced by one of the companies is that external consultants do not have sufficient domain knowledge needed for security testing. Therefore, some domain-specific vulnerabilities are left undiscovered. The periodicity of the

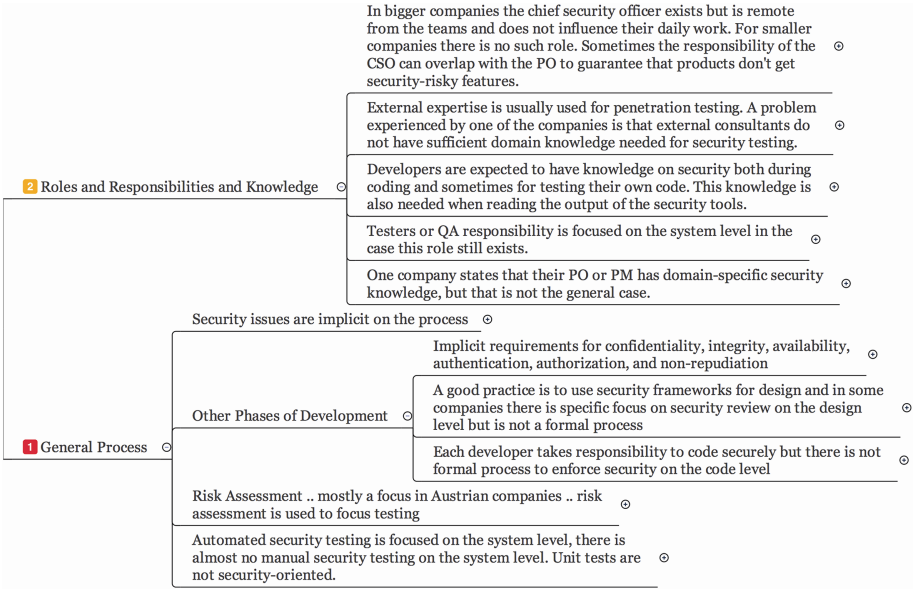

Fig. 4. Mind map: security software engineering process.

execution of these tests is quite ad-hoc, sometimes linked to big deliveries or when there are too many changes in the source code. The results of the tests are not completely integrated in the development process and almost never get into the planning of the activities of the sprint.

The third observation is that testers or QA personnel focus on the system level in the case this role still exists and the developers take care of the daily activities and developers are expected to have knowledge on security both during coding and sometimes for testing their own code. This knowledge is also needed when reading the output of the security tools. One interviewee said: "*We generally organize mainly as software developers, we generally have a software engineering role and we are expected to be with a broad knowledge, and skill set, computer science engineering and security and safe programming*". But there is no specific validation of this stated 'broad knowledge and skill set'. Another interviewee stated on some tool output: "*Normally, the errors are quite readable. From technician level, the developer that develops component should also understand the message of the tool. For instance, if the tool says, open API C# token found, hopefully developers also know what it says. The tools check very huge part, but they cannot check all. This is the responsibility that developer has while developing.*" It was clear that this knowledge was not something systematically evaluated or externalized, just assumed, as the agile mindset brings the focus to people instead of process and tools the teams are not completely sure of how much knowledge on secure coding was in the teams.

Automated unit testing is not security-oriented at all. Risk assessment is performed mostly by the Austrian teams (Team 1 and Team 2), and is applied to focus testing. One interviewer said: "*Yes, we are using risk assessment, it is a kind of matrix where we have*

on one hand probability occurrence and on the other hand importance of that stuff or if it can occur. We have this matrix and we are using it for small tools".

4.2 RQ 2: How Does the Agile Teams Perform Security Testing in Each Testing Phase?

To answer how security testing is performed in each testing phase, we analyzed the scope, objective and accessibility of the security testing, as shown in Fig. 5. With regard to the scope, unit tests are commonly used in agile teams, but typically not with a specific security focus. With some approaches for example testing positive and negative cases one team specifically mentions security focus for unit tests. Only one team highlights that security aspects are considered when negative unit tests, which are intended to fail, are executed.

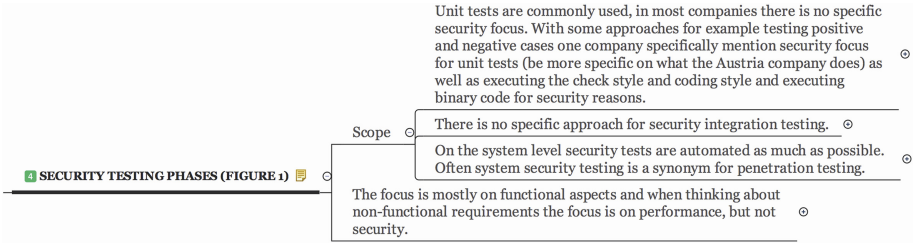

Fig. 5. Mind map security phases.

Static source and binary code analysis is performed for security reasons on the unit level. All teams stated that no specific security aspects are considered during integration testing. Security testing is most prominent on the system level. On this level security tests are typically a synonym for penetration testing, typically performed as black box testing. Security tests on the system level are to a large extent automated and there is almost no manual security testing on this level. White-box aspects are typically only considered during static source or binary code analysis.

When testing non-functional requirements, the focus in the interviewed teams is typically on performance. One interviewee said: "*We usually have unit test. And those are trying to exercise the happy path, which should already catch a many of basic the problems. We don't have much integration tests. We have also some performance tests. And that may go to the non-functional category, but we do not have much. We worry mostly on if the code works as it is supposed to work*".

4.3 RQ 3: How Are Traditional Security Testing Techniques Generally Used in the Agile Software Development Lifecycle?

For this question, the interviewees were asked to analyze Fig. 2. It shows the security testing techniques generally used in traditional secure software development lifecycle, i.e., model-based security testing, code-based testing and static analysis, penetration

testing and dynamic analysis as well as security regression testing. The interviewees were asked to talk about how they perform these activities in their agile software development and how often security testing or security related activities are done in their agile cycles. An overview of the results is shown in Fig. 6.

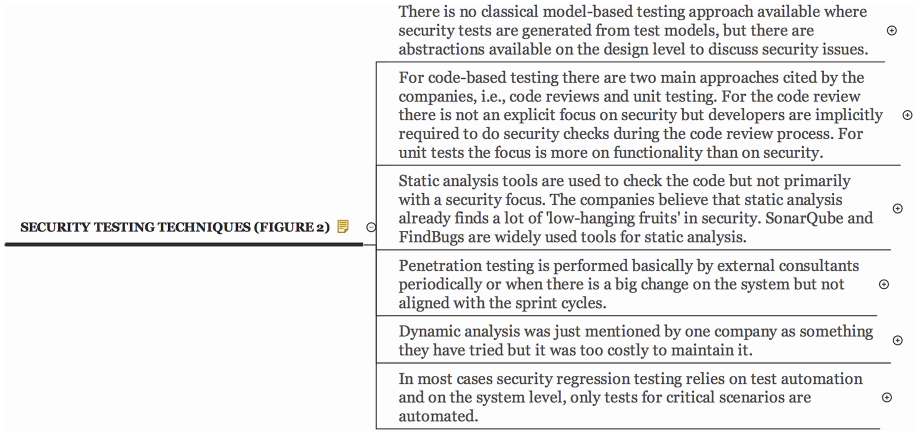

Fig. 6. Mind map security testing techniques findings.

In general, there is no classical model-based testing approach available where security tests are generated from test models, but there are abstractions available on the design level to discuss security issues. One interviewee said: *"We don't do any model-based testing. We consider security aspects as part of design and we don't try to buy a formal model around that. During development as we said we do code-based testing and static analysis. And that is probably on where most of our focus is. We have done some dynamic security tests in the past. As I said, those took a lot of manual effort and it was very unstable, it broke up often with some UI changes and it was hard to keep up".*

For code-based testing there are two main approaches referred to by the companies, i.e., code reviews and unit testing. When it comes to code review, there is no explicit emphasis on security, but developers are implicitly required to do security checks during the code review process. As for unit tests the focus is more on functionality than on security.

Static analysis tools are used to check the code but not primarily with a security focus. The interviewed teams believe that static analysis already finds the most important 'low-hanging fruits' in security. SonarQube and FindBugs are widely used tools for static analysis for the teams interviewed.

Penetration testing is performed basically by external consultants periodically or when there is a big change on the system but not aligned with the sprint cycles. One interviewee stated: "*We do penetration testing from external testers from the company, this was done together with the University of Innsbruck and plus our customers are doing against software. They are completely independent and we are not informed, we offer our aid only if there is a problem, and if we take Austrian medical network for*

example, it is not allowed to go live without testing from external company and that does not only involve our software but the whole system."

Dynamic analysis was only mentioned by one interviewee as something they have tried but it was too costly to maintain it. He said: "*It was taking too much time to keep it for us. And it requires a lot of manual integration and once that the scenario broke because of an UI change or something and then we would have again manual effort to fix that. For me, what makes code review and static analysis to work so well is that every time you compile the code you can see the feedback on it. On the dynamic tests, you cant do that very easily at that point you have to wait, and there is a lag between you writing your code and you receiving some feedback on it. Even if it is part of the development process, it doesn't happen right away. In my experience the further away from your commit, it less likely that you will either notice or be able to change it".*

In most cases security regression testing relies on test automation and on the system level only tests for critical scenarios are automated, but not a specific regression testing for security. One interviewee said: "*So what is working well is, I think our development processes are well structured and the biggest problem is, that we have frequent changes of user stories and that is very challenging on the one hand side on the development process and on the other side testing process. You have to adopt everything. The user stories are not from our customers, the problem the changing part is more about our c-level changes, on time this and one time that. So this is very big problem which also is very big problem for agile software development because it is very big problem".*

5 Discussion

Based on the results, we discuss recommendations for practice and research as well as limitations of this work.

5.1 Recommendations for Practice

With regard to the security engineering process, it is evident that the teams assume that developers have some security knowledge, but the issue is that they did not state how they conduct security engineering processes as well as what they need. For this reason, there is a demand for better use of guidelines for secure coding and testing practices like the OWASP guidelines [25]. Moreover, there should be a more systematic approach of spreading knowledge in security inside the teams. In a recent survey, Oyetoyan et al. [27] found that the developers' confidence in their software security knowledge is low, and therefore more efforts should be spend on getting the level of security knowledge higher at the companies. This is stronger in agile setting context because there is a strong dependency on people and not on process and tools. In addition code review and static analysis are used more and more in software projects, but without specific focus on security [27]. For this reason, processes of code reviews and static analysis should be more focused on security.

Even though the teams rely on penetration testing performed by externals, there is a danger of external penetration testers not having domain knowledge to catch important

vulnerabilities. While independent penetration testing is possible, there is a need that the penetration testing feedback is well integrated with the whole development process lifecycle [7, 34]. Chóliz et al. [7] have focused their study on the security testing activities, with the clear objective of synchronizing the tests from the independent security team with the agile rhythm of sprints, with frequent deliveries, of the software engineering teams, showing that the rate of found security vulnerabilities increased gradually. The results of Türpe et al. [34] suggest that penetration tests improve developers' security awareness, but long-lasting change of development practices is hampered if security is not properly reflected in the communicative and collaborative structures of the organization, e.g. by a dedicated stakeholder.

POs should have more security awareness because they are the only one responsible for maximizing the return on investment (ROI) of the development effort. In addition, the PO is responsible for product vision and constantly re-prioritizes the Product Backlog, thereby adjusting any long-term expectations such as release plans and making sure the team considers the stakeholders interests. The main issue with the explicit functional security requirements is that, most of the time stakeholders do not explicitly state them as requirements, and neither do the product owners. On the other hand, the non-functional security requirements are not features, which mean they never become a user story. In other words, they are not inserted into the product backlog. From the performed study, we see that security issues are implicitly handled on the process, but there is need for a more systematic approach to handle security issues in the development process. As shown by Rindel et al. [30] it is possible to have the security user stories as part of the product backlog.

5.2 Recommendations for Research

Research can help to increase knowledge and application of security testing in several respects. First, knowledge can be increased by the development of suitable courses and guidelines based on empirical evidence showing which approaches work in which context. Good efforts have been done in the last years [3, 7, 30, 34]. Therefore more empirical studies are needed which investigate challenges of security testing and derive respective evidence-based guidelines to address them.

With regard to model-based security testing, lightweight approaches are needed, which support the model creation, for instance, by learning of domain language concepts, based on design-level abstracts that are available also in agile teams. Also, a general understanding of the return of investment of model-based security testing approaches, which has already been highlighted as a challenge in [17], would help to apply such approaches efficiently. The issue of efficiently applying model-based testing approaches becomes even more critical when agile teams develop systems where the connection between safety and security is essential as in modern Internet-of-Things applications.

As seen in the results, system testing is often limited to penetration testing and testing of functional security requirements is often neglected. As automation is difficult to achieve fully, but at the same time, important for successful application in agile teams, suitable automation support and innovative techniques are required [29].

So far, security testing in agile teams makes little use of security risk assessments, which typically exist in an implicit or explicit for in other organization units. Risk assessment can be used to develop risk-based testing approaches [14], which can guide decisions during testing, and for instance help to select and prioritize security regression tests [13]. Baca et al. [3] shows that using a risk analysis approach, it s possible to find more severe risks, besides, more advanced skills and a deeper awareness of the problems become available. More research needs to be done in order to understand the best way to apply risk management in agile projects and especially on security.

5.3 Work Limitations

Common criticisms to a case study also apply to this study, among them one may list: uniqueness, difficulty to generalize the results, and the introduction of bias by participants and researchers. In our study, we generalized the findings from empirical statements to theoretical statements, which involved generalizing data from interviews and perceptions by discussing them in accordance with the literature. Interview data were though our primary source of information.

Qualitative findings are highly context and case-dependent. Our findings apply to software projects teams within four participating teams. However, all the participants were professionals using typical development technologies in a typical working environment, e.g., the natural setting demanded by the case study approach. We described the main characteristics of each case and company, including context and settings, data collection, analysis, and analysis process, as well as quotations with our major findings. This makes the results easier to generalize.

As commonly done in in-depth qualitative studies, we also had to do a trade-off between the number of participants, the duration and the cost of this study. The number of subjects interviewed in this context is not quantitatively significant, but gives deeper insights on the issues investigated in this work.

6 Conclusion

In this paper, we investigated by a cross-case analysis of four teams, two from Austria and two from Norway, how security testing is performed in agile teams. We investigated how the security engineering process is managed/organized in agile teams, how security testing is performed in each testing phase, and how security testing techniques are generally used in the secure software development lifecycle.

Although the study is based only on the results of a limited amount of agile teams, i.e., four, agile teams, we could derive recommendations for research and practice. The findings of this research are not surprising, but at the same time are alarming. The lack of knowledge on security by agile teams in general, the large dependency on incidental penetration testers, and the ignorance in static testing for security are indicators that security testing is highly under addressed and that more efforts should be addressed to security testing in agile teams.

In the future, we plan to replicate this study and to develop and evaluate suitable security testing approaches to support the adoption of security testing in agile teams through action research studies with industry.

Acknowledgments. This work was partially supported by the SoS-Agile (247678/070) project funded by the Research Council of Norway, and by MOBSTECO (FWF P 26194-N15) funded by the Austrian Science Fund. The authors are grateful to all involved in this study, specially the interviewees for their insights and cooperation and to the software companies for supporting this work.

References

1. Arkin, B., Stender, S., McGraw, G.: Software penetration testing. IEEE Secur. Priv. **3**(1), 84–87 (2005)
2. Austin, A., Williams, L.: One technique is not enough: a comparison of vulnerability discovery techniques. In: ESEM 2011, pp. 97–106 (2011)
3. Baca, D., Boldt, M., Carlsson B., Jacobsson, A.: A novel security-enhanced agile software development process applied in an industrial setting. In: ARES 2015, pp. 11–19 (2015)
4. Beznosov, K., Kruchten, P.: Towards agile security assurance. In: NSPW 2004, pp. 47–54 (2004)
5. Camacho, C.R., Marczak, S., Cruzes, D.S.: Agile team members perceptions on non-functional testing: influencing factors from an empirical study. In: ARES 2016, pp. 582–589 (2016)
6. Chess, B., McGraw, G.: Static analysis for security. IEEE Secur. Priv. **2**(6), 76–79 (2004)
7. Choliz, J., Vilas, J., Moreira, J.: Independent security testing on agile software development: a case study in a software company. In: ARES 2015, pp. 522–531 (2015)
8. Common Weakness Enumeration (CWE), 5 March, 2017. https://cwe.mitre.org/index.html
9. Crispin, L., Gregory, J.: Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley Professional, Boston (2009)
10. Cruzes, D., Dybå, T.: Recommended steps for thematic synthesis in software engineering. In: ESEM 2011, pp. 275–284 (2011)
11. CWE/SANS TOP 25 Most Dangerous Software Errors, 5 March 2017. https://www.sans.org/top25-software-errors/
12. Erdogan, G., Meland, P.H., Mathieson, D.: Security testing in agile web application development - a case study using the EAST methodology. In: Sillitti, A., Martin, A., Wang, X., Whitworth, E. (eds.) XP 2010. LNBIP, vol. 48, pp. 14–27. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13054-0_2
13. Felderer, M., Fourneret, E.: A systematic classification of security regression testing approaches. Int. J. Soft Tools Technol. Transf. **17**(3), 305–319 (2015)
14. Felderer, M., Schieferdecker, I.: A taxonomy of risk-based testing. Int. J. Softw. Tools Technol. Transf. **16**(5), 559–568 (2014)
15. Felderer, M., Agreiter, B., Breu, R., Armenteros, A.: Security Testing by Telling Test Stories. Modellierung **161**, 195–202 (2011)
16. Felderer, M., Büchler, M., Johns, M., Brucker, A.D., Breu, R., Pretschner, A.: Chapter one-security testing: a survey. Adv. Comput. **101**, 1–51 (2016)
17. Felderer, M., Zech, P., Breu, R., Büchler, M., Pretschner, A.: Model-based security testing: a taxonomy and systematic classification. Softw. Test. Verification Reliab. **26**(2), 119–148 (2016)

18. Fitzgerald, B., Stol, K.-J.: Continuous software engineering: a roadmap and agenda. JSS **123**, 176–189 (2017)
19. Keramati, H., Mirian-Hosseinabadi, S.: Integrating software development security activities with agile methodologies. In: AICCSA 2008 (2008)
20. Marback, A., Do, H., He, K., Kondamarri, S., Xu, D.: A threat model-based approach to security testing. Softw. Pract. Experience **43**(2), 241–258 (2013)
21. McGraw, G., Potter, B.: Software security testing. IEEE Secur. Priv. **2**(5), 81–85 (2004)
22. Microsoft, Agile Development Using Microsoft Security Development Lifecycle 5 March 2017. http://www.microsoft.com/en-us/sdl/discover/sdlagile.aspx
23. Moe, N.B., Cruzes, D., Dybå, T., Mikkelsen, E.M.: Continuous software testing in a globally distributed project. In: ICGSE 2015, pp. 130–134 (2015)
24. Oueslati, H., Rahman, M.M., Othmane, L., Ghani, I., Arbain, A.F.: Evaluation of the challenges of developing secure software using the agile approach. Int. J. Secure Softw. Eng. **7**, 17 (2016)
25. OWASP Foundation: OWASP Testing Guide v4. 5 March, 2017. https://www.owasp.org/index.php/OWASP_Testing_Project
26. OWASP Top 10. 5 March 2017. https://www.owasp.org/index.php/Top_10_2013-Top_10
27. Oyetoyan, T.D., Cruzes, D.S., Jaatun, M.G.: An empirical study on the relationship between software security skills, usage and training needs in agile settings. In: ARES 2016, pp. 548–555 (2016)
28. Paul, M.: Official (ISC)2 Guide to the CSSLP CBK, 2nd edn. (ISC)2 Press (2014)
29. Peischl, B., Felderer, M., Beer, A.: Testing security requirements with non-experts: approaches and empirical investigations. In: QRS 2016, pp. 254–261 (2016)
30. Rindell, K., Hyrynsalmi, S., Leppänen, V.: Case study of security development in an agile environment: building identity management for a government agency. In: ARES 2016, pp. 556–563 (2016)
31. Sindre, G., Opdahl, A.L.: Eliciting security requirements with misuse cases. Requirements Eng. **10**(1), 34–44 (2005)
32. Tappenden, A., et al.: Agile security testing of web-based systems via HTTP unit. In: Proceedings of Agile Conference. IEEE (2005)
33. Tian-yang, G., Yin-sheng, S., You-yuan, F.: Research on software security testing. World Acad. Sci. Eng. Technol. **70**, 647–651 (2010)
34. Türpe, S., Kocksch, L., Poller, A.: Penetration tests a turning point in security practices? In: Organizational Challenges and Implications in a Software Development Team, WSIW@SOUPS 2016 (2016)

An Assessment of Avionics Software Development Practice: Justifications for an Agile Development Process

Geir K. Hanssen[1(✉)], Gosse Wedzinga[2], and Martijn Stuip[2]

[1] SINTEF, Trondheim, Norway
geir.k.hanssen@sintef.no
[2] NLR, Amsterdam, The Netherlands
{gosse.wedzinga,martijn.stuip}@nlr.nl

Abstract. Avionic systems for communication, navigation, and flight control, and many other functions are complex and crucial components of any modern aircraft. Present day avionic systems are increasingly based on computers and a growing percentage of system complexity can be attributed to software. An error in the software of a safety-critical avionic system could lead to a catastrophic event, such as multiple deaths and loss of the aircraft. To demonstrate compliance with airworthiness requirements, certification agencies accept the use of RTCA document DO-178 for the software development. Avionics software development is typically complex and is traditionally reliant on a strict plan-driven development process, characterized by early fixture of detailed requirements and late production of working software. In this process, requirement changes and solving software errors can lead to much rework, and create a risk of budget and schedule overruns. This raises the question whether avionics software development could benefit from the application of agile approaches. Based on the results of three activities: (1) a literature study on industrial experience with the use of agile methods in a DO-178 context, (2) an expert assessment of the DO-178 objectives, and (3) a survey conducted among European avionics industry, an outline is presented of an agile development process, where Scrum is extended to achieve the DO-178 objectives. The application of agile methods is expected to support frequent delivery of working software and ability to respond to changes, resulting in reduced risk of budget and schedule overruns.

Keywords: Avionics · Certification · Safety critical software · DO-178 · Software Life-Cycle · Agile · Scrum

1 Introduction

Avionic systems play a crucial role aboard modern aircraft. These systems offer pilots operational support in areas such as communications, navigation, and control of the aircraft during all phases of flight and in all weather conditions. A system is safety-critical when its failure could result in loss of life, significant property damage, or damage to the environment [11]. An example of a safety-critical avionic system is the flight control system, which governs the attitude of an aircraft and, as a result, the flight path it follows. Safety-critical systems are not limited to the avionics domain only,

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 217–231, 2017.
DOI: 10.1007/978-3-319-57633-6_14

examples of other important domains include, process control [20], medical equipment [17], and automotive [9].

Present day avionic systems are increasingly based on computers and more functions are implemented as software. Certification agencies, like the European Aviation Safety Agency (EASA), accept the use of RTCA document DO-178 [18] for the development of avionics software to provide assurance of compliance with airworthiness requirements. Document DO-178 requires the achievement of many safety objectives, which is generally costly and time consuming [4, 10].

The avionics industry traditionally uses the V-model, or a variant thereof, as life-cycle model for software development. This matches DO-178 well when looking at the life-cycle data items that have to be produced. There are, however, also disadvantages. For example, no working software is produced until late in the development life-cycle. Errors detected in this stage can lead to much rework of earlier performed activities, and increase the risk of budget and schedule overruns [4]. In the same way, changes in requirements in a late stage can also lead to much rework with similar consequences.

The application of agile methods could be a solution for these problems. The difficulty lies, however, in the fact that the looseness of an agile process does not seem to be reconcilable with the rigour imposed by DO-178. For example, agile development considers responding to change more important than following a plan, while DO-178 is strictly plan driven. The main question addressed by this research is how agile methods can be adapted to be usable in an avionics development process that is governed by DO-178.

The following of the paper describes our research method (Sect. 2), an analysis of DO-178C (Sect. 3), an overview of research and industry experience (Sect. 4), a survey of present practice (Sect. 5), and an outline of an agile process aligned with DO-178 (Sect. 6). Conclusions and further work are presented in Sect. 7.

2 Research Method

In order to answer our research question, three complimentary activities have been carried out and used to propose a DO-178-aligned agile process.

(1) An assessment of DO-178 has been performed to indicate how an agile strategy for meeting the objectives could look like and whether there are potential conflicts by using an agile method (Sect. 3.2). Annex A of DO-178 contains 10 summary tables with 71 objectives. The information provided for each objective includes: (a) a brief description, (b) its applicability for each software criticality level, (c) the requirement for independent achievement, and (d) the data items in which the results are collected. Each objective has been assessed to determine how the objective can be met using an agile approach like Scrum and whether there is a need for extensions beyond what can be considered a plain agile approach. The work performed by K. Coetzee[1] was taken as a starting point.

[1] http://www.embeddedfool.net/blog/2015/04/08/a-more-agile-do-178/ (last accessed, Dec. 5, 2016).

(2) Relevant literature addressing the application of agile methods in the avionics domain has been reviewed and main findings about opportunities and limitations of using agile methods for development of avionics software were summarized (Sect. 4). In order to build an understanding of the status of research and reported industrial experience on the use and effects of agile methods in development of safety-critical avionics software, a search for relevant literature has been conducted with Google Scholar. We applied search phrases based on relevant terms such as 'agile', 'avionic', and 'DO-178'. To strengthen the search, we applied snowballing, meaning that relevant work referenced in identified publications was checked for relevance and potentially included if the focus and quality was found sufficient. From this search, 11 publications were found that potentially could offer insight into industrial experience.

(3) A survey was done as an online questionnaire to establish a better overview of the state–including challenges and potential points of improvement–of software development and certification in the avionics industry, and to map the current status of using or plans to use agile methods. As part of the ASHLEY[2] EU-project, we selected professionals believed to have sufficient knowledge about their own organization and about how software is developed and certified. 29 contact persons were selected, each representing a unique ASHLEY partner organization. 10 contact persons completed the questionnaire fully or partially.

Our study has some limitations. Firstly, the literature review identified a relatively low number of relevant studies providing industrial experience. This is however a valuable insight as it nevertheless summarizes the present state of research within this specific domain. Secondly, the survey has a relatively low number of respondents. This is due to resource priorities, but is somewhat compensated by selecting qualified respondents, each representing a major avionic system provider in Europe. The results present the most comprehensive overview of this industry so far.

3 Certification Aspects of Avionics Software Development

3.1 Overview of Document DO-178C

Document DO-178C, "Software considerations in airborne systems and equipment certification" [18] governs the approval of software for avionic systems by certification authorities, such as EASA. In this paper, we simply write DO-178 when referring to revision C of the document.

DO-178 distinguishes five software levels (A–E) based upon the failure condition that may result from erroneous behaviour of the software. Software is classified as (the highest) level A, if erroneous software behaviour can cause or contribute to a catastrophic failure condition of the aircraft, which would result in multiple fatalities, usually

[2] Avionics Systems Hosted on a distributed modular electronics Large scale dEmonstrator for multiple tYpe of aircraft, http://www.ashleyproject.eu (last accessed, Dec. 9 2016).

with loss of aircraft. For lower software levels, the consequence of erroneous software behaviour gradually reduces to no effect on safety (level E).

DO-178 is a process-based standard relying on evidence that the various activities associated with software development have been performed successfully. DO-178 categorizes processes into three types: (1) the software planning process, which defines and coordinates the activities of all processes (2) the software development processes, which produce the software product, and (3) the integral processes, which ensure the correctness of the software product and confidence in the software development processes and their outputs. DO-178 does not address system life-cycle processes, but it does describe the interaction with system processes, including system safety assessment.

Table 1. Assessment of objectives for the software development processes.

DO-178 Objective	Agile Strategy	Remarks
1. High-Level Requirements (HLRs) are developed	A system is divided into features. Features are divided into stories. Stories consist of HLRs (and their test cases)	Features are client-valued functions. At the end of each Sprint, the implemented user stories are used to update the HLRs
2. Derived HLRs are defined and provided to the system processes, including system safety assessment process	Derived HLRs are not directly traceable to system requirements. They are developed in the same way as HLRs (see objective 1)	Derived HLRs are provided to the system processes to determine if there is any impact on the system safety assessment and system requirements
3. Software architecture is developed	Start with a high-level architecture and update/refine it at each software release	Closure activities include a review of the software architecture to make sure it is consistent with the source code
4. Low-Level Requirements (LLRs) are developed	Develop LLRs by defining conditions and associated actions [13]	LLRs can be contained in the source code or the unit tests (embedded in the source code)
5. Derived LLRs are defined and provided to the system processes, including system safety assessment process	Derived LLRs are not directly traceable to HLRs. They are developed in the same way as LLRs (see objective 4)	Derived LLRs are provided to the system processes to determine if there is any impact on the system safety assessment and system requirements
6. Source Code is developed	Develop source code by applying Test-Driven Development (TDD)	Stories are implemented during Sprints
7. Executable Object Code and Parameter Data Item Files, if any, are produced and loaded in the target computer	Develop object code by applying Continuous Integration (CI) and Continuous Delivery (CD)	When a defined set of features is completed, a release will follow

DO-178 provides guidance by (1) stating objectives for software life-cycle processes, (2) describing activities that provide a means for satisfying the objectives, and (3) describing evidence in the form of data items to demonstrate that the objectives have been satisfied. DO-178 does not prescribe a particular software life-cycle or methodology. A software development project defines its software life-cycle by specifying a set of processes and their sequence. The usual sequence through the software development processes is requirements, design, coding, and integration.

3.2 Assessment of Document DO-178C

The assessment revealed that objectives for the software development processes (DO-178, Table A-2) and testing (DO-178, Table A-6) can be achieved by applying agile techniques. The remaining objectives are either outside the agile process or there are no suitable agile techniques to achieve them. These objectives can be achieved using traditional methods (inspections, reviews, analyses, management records).

Table 1 presents the assessment of the 7 objectives for the software development processes (DO-178, Table A-2).

In conclusion, agile methods can be used to achieve a subset of the DO-178 objectives. No prohibitive conflicts have been identified.

4 Overview of Existing Research and Industry Experience

Most of the 11 reviewed publications provide discussions at a conceptual level without any empirical data, indicating that this is a relatively new and immature–but growing–concept within the avionics domain. Some empirical data is presented in only three of the papers. Wils et al. [22] provide some minor insights from the Barco company, Paige et al. [16] present a very small-scale experiment, and Carlson and Turner [1] make a review of five case studies.

This lack of empirical data from industry is in contrast to non-safety-critical domains where the use of agile methods has become common, with correspondingly more empirical research available [6]. One comparable domain, the process control domain, where the IEC 61508 standard applies, is a bit more advanced, but in general it seems that the application of agile methods and techniques to safety-critical software is in its early stages [8]. However, the emergence of literature presenting ideas over the past few years means that the industry is seeking new opportunities for improving their software development processes inspired by other domains.

4.1 Why This Interest in Agile Methods?

The common background and motivation for nearly all reviewed publications is the need for improving the software development process, including certification based on DO-178B/C. The trend seems to be that avionic system complexity is increasing [5]. Requirements tend to be more volatile (even late in the development process), calling for better approaches to manage requirements and their changes in more flexible ways

[5, 15, 16]. We also see an increased customer orientation where industry wants to listen more closely to customers [1, 3, 16, 21, 22], opening up for a more flexible development process with less emphasis on complete and detailed up-front design. Experience also indicates that cost and schedule overruns are happening too frequently [1, 4].

4.2 Evidence and Documentation

Regardless of the process framework, e.g., V-model or an agile process, there is a set of formal data items that has to be produced [5], but an agile process may allow for doing this more efficiently as well as data items may be updated more often. However, if an agile approach is to be used, it calls for some extensions [16], as agile methods, such as Scrum, do not specify such documentation at all. Examples of such data items that are required by certification authorities are the Plan for Software Aspects of Certification (PSAC) and the Software Accomplishment Summary (SAS) [18]. These documents, together with the plans that concern the definition of the life-cycle processes may best be kept outside the agile process.

4.3 More Flexible Management of Requirements and Change

One of the main characteristics of the established practice and application of the V-model is that development of avionics software may be characterized as document driven and sequential [16]. This may become challenging in cases where requirements change throughout a development project, even despite there have been made very detailed plans and design up-front. Change may come from several sources, like design revisions, review of safety analysis, and verification [16]. Recent figures indicate that requirements change can be quite extensive, from 25% in typical projects to 35% in large and complex projects [21], and discovering problems and dealing with changes late in the process may become very costly [4]. According to Wils et al., agile methods may lower the change effort as compared to traditional development [22]. This does not mean that up-front plans are to be avoided, as that would conflict seriously with the process objectives in DO-178. However, the role of agile requirements management is to detail high-level requirements per iteration, not to create new high-level requirements [5]. New high-level requirements could be added after the Sprint, as part of the Sprint review. Up-front requirements may not be complete or even in conflict (and need to be refined) [5].

However, there is a potential conflict here–that flexible requirements management negatively affects the software verification process. If previously verified components of a system are changed, the verification results need to be updated. This requires strict configuration management and relentless testing of the software under development [2].

4.4 Applicability and Obstacles

In general, the consensus seems to be that there is no conflict per se for using agile methods in development of avionics software [2, 3, 13, 21, 22]. In fact XP/Agile is claimed to be particularly suitable [3] to deal with the increasing complexity and

requirements volatility in safety-critical software projects. As changes inevitably do happen, we could make use of better strategies to manage changes.

However, agile methods, such as Scrum, were not designed to support development of large and complex systems like safety-critical avionic systems and there is a lack of techniques and practices to meet the objectives of DO-178. E.g., the requirements for data items and traceability have to be met by setting up a well-functioning framework of tools to support and automate the process to a large extent [3, 22]. An agile process, with short iterations of work, frequent feedback, and evaluation of status and incremental development of the software supports the production of some of the needed data items as part of the development itself. Instead of explicitly producing separate documents, some of the information may be extracted from tools and logs. One of the core objectives of agile methods is to minimize the effort for producing documentation [16, 21]. There is work going on to extend Scrum to make it applicable to regulated domains, for example the SafeScrum framework [14] and R-Scrum [7], which seek to meet requirements mentioned above.

Besides practical aspects of setting up an agile process and a chain of supporting tools, we also need to clarify such a change with the certification authority. A more or less radical change in process will affect the work to be done by this stakeholder and it is of course important that the certification authority representative gets all requested information and eventually gets confidence that the applied approach has led to a safe product without extra problems and in an efficient way.

Besides the core principle of incremental and iterative development, agile methods may also be seen as a collection of practices and techniques. From Chenu [3] and Paige et al. [16], we extracted the following set that may be particularly relevant to safety-critical systems development:

- Test-driven development (need some adaptation, see also [12]).
- Coding standards (already mandatory for DO-178 levels A–C).
- Design improvement/refactoring (creates some challenges with respect to safety analysis [5]).
- The planning game (from XP).
- Emphasis on communication (other than through extensive documentation).

4.5 Team Efficiency and Motivation

One of the main aspects of agile methods is how people work together. As a contrast to plan-based methods where developers take on specialized roles, following detailed plans, agile methods rely on multi-disciplinary teams, with the idea that this better enforces learning and motivation [3]. Furthermore co-located teams are also believed to improve design flexibility and a shared vision of the system under development [1]. A team may also have Designated Engineering Representatives (DERs), who are embedded representatives of the certification authorities within the development team [5].[3]

[3] Under EASA regulation, Certification Verification Engineers (CVEs) perform equivalent tasks as DERs.

4.6 Testing

Extensive testing and full traceability is fundamental in development of avionics software and implementation of all requirements has to be verified by tests [3]. Testing is also strongly emphasized in agile methods, which focus on test-driven development and high test-coverage. However, for avionics software development purposes, agile methods need to extend testing activities–e.g. by having more thorough acceptance testing (not (only) relying on customer feedback) [2, 16]. Carlson and Turner argue that incremental testing increases iteration pace and enables issues to be revealed and dispatched [1]; they also argue that testers should be part of the development team (provided that any independency requirements are guaranteed).

4.7 Adoption of New Software Process Models

Experience (e.g. from object-oriented development) shows that uptake and acceptance of a new practice takes time–we should expect the same for agile methods as well [21, 22]. The avionics domain relies on well-established and well-proven practices and processes and it is natural to be careful with new ideas, like agile methods, as they may seem to impose more challenges than benefits. However, as this literature in sum shows, there seems to be a growing interest at least.

4.8 Relating Findings to Other Domains

The literature review done here has focused explicitly on the avionics domain. However, we find that the main challenges and approaches clearly coincide with other domains where safety-critical software is essential. Other studies show that the same type of challenges are being addressed, e.g., for process control systems [20], medical equipment [17], and automotive [9], and that agile methods may be applicable to other safety standards and frameworks like IEC 61508, SPICE, and IEC 62304.

5 Survey to Assess Present Practice

A questionnaire was used to gain insights into the organizations' profiles, their maturity, their relationship to safety standards and authorities, various life-cycle aspects, and perceived challenges and problems.

5.1 Respondents' and Organizations' Profiles

Respondents have a great variety in profiles, from developers and testers to managers. Their organizations also have a wide range of business models, target markets (civil passenger aircrafts on the top), and type of software applications (real-time embedded systems being the most common).

5.2 Maturity

The avionics domain/industry is mature and professional with established system providers having decades of experience. There is a wide range of methods for requirements analysis and architectural and detailed design in use. There is also a wide range of testing approaches in use (white/black-box–unit/module/system/hardware-in-the-loop). All practice extensive testing and inspection. Customer involvement is extensive. There is extensive use of DOORS® from IBM Rational for requirements analysis and management, but half of the respondents also use typical office tools.

5.3 Relationship to Safety Standards and Authorities

DO-178 is clearly the most relevant standard for all organizations. Applications are developed at all levels of DO-178, where level C is the most common (60% of the respondents). Consequently, there is a very high coverage of data items. When asked about the level of interaction with the external assessor, 50% report that they collaborate with the assessor in all phases of the project. The rest report a lower level. The average estimate of costs related to verification and certification (including all reviews and testing) is 40% of the total project budget.

5.4 Life-Cycle Aspects

There are a wide variety of software life-cycle models in use. The V-model is in use in some form by all organizations, while 25% use incremental/iterative methods in some form. Customers are involved to a very high degree. Testing (in general) and code inspection/analysis are used by all respondents. Formal methods are applied by about a third of the respondents.

5.5 Perceived Challenges and Problems

The top challenges with respect to verification and certification include: (1) having sufficient resources, infrastructure, and competency/staff, (2) having sufficient quality of customer communication, including requirements specification and feedback, and (3) demonstrating compliance with DO-178 requirements to certification authority. The top-rated problems with the software development process are requirements management (frequent changes, insufficient requirements, ambiguous requirements, and addition of new requirements), late discovery of problems/defects, and project cost overruns.

6 Towards an DO-178-Aligned Agile Approach

As mentioned in Sect. 3.1, document DO-178 [18] does not prescribe a particular software life-cycle model. This makes it possible to define software life-cycles, such as, waterfall, V-model, incremental, and spiral, but also to apply agile methods. Scrum is considered to be a suitable (non-safety) agile framework that could be used as a baseline.

It is the most commonly used agile framework in the software industry, in general, with a large number of training resources, industrial experience, and available research literature. Scrum will have to be extended for the development of avionics software to enable delivery of all required data items in compliance with DO-178.

6.1 Scrum Phases

In his seminal paper [19] on the Scrum development process, K. Schwaber made a distinction into the phases Pregame, Game, and Postgame. In this paper, we use the terms Preparation, Development, and Closure, which are also frequently used, e.g., [13]. Applying the Scrum phases to the software development and software verification processes of DO-178, as depicted in Fig. 1, allows the mapping of agile methods to these processes.[4]

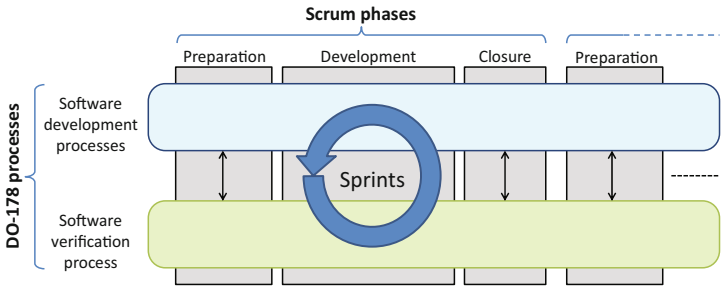

Fig. 1. Application of Scrum phases to DO-178 processes.

During the Preparation phase, planning and architecture activities are performed. Scrum's concept of planning is somewhat broader than that of DO-178. Scrum includes the definition of the next software release based on the currently known backlog, analysis of system requirements, and development of user stories. The architecture activities establish (or update) the software structure. During the Development phase, the functionality of a new release is developed as well as tests for new or changed code. The software is designed, and source code is implemented, integrated, and tested during a sequence of Sprints. In the Closure phase, the software release is prepared, including system testing, final documentation, and release. The sequence of Preparation, Development, and Closure is repeated until the final software release has been completed. In the next sections, the activities in each phase are described in more detail.

6.2 Preparation Phase Activities

During the Preparation phase, the allocated system requirements, or a subset thereof, are taken and high-level requirements (HLRs) are produced in the form of features that

[4] For simplicity, the DO-178 planning process and integral processes other than software verification are not shown in Fig. 1.

are further divided into user stories. A software architecture is established (or refined), which, together with the prioritized HLRs, as part of the product backlog, is provided to the Development phase. As required by DO-178, outputs of all processes are verified, e.g., by means of review or analysis. Further details are presented in Table 2.

Table 2. Activities during Preparation phase.

DO-178 Process	Inputs	Activities	**Outputs**
Software requirements	Allocated system requirements, software level	Define system features and prepare user stories. A story consists of HLRs	HLRs, trace data
Software design	HLRs	Establish or refine software architecture, including partitioning concept	Software architecture, trace data
Software verification	HLRs, software architecture, trace data	Define test cases for HLRs. Verify all outputs	HLR test cases, verification results

The planning process of DO-178 is kept outside the agile process. It is responsible for establishing and updating all plans, including the Software Development Plan, the Configuration Management Plan, and the Plan for Software Aspects of Certification. The latter document is used for communication with the authorities.

6.3 Development Phase Activities

The Development phase consists of a sequence of Sprints, all with preferably the same fixed duration (from 1 to 4 weeks). The number of Sprints is not fixed. The result of a Sprint is a set of implemented and tested user stories that are integrated into a working application. In addition, a Sprint produces information for the assessor (the data items). The application can be demonstrated to stakeholders, but not all features may be complete and hence it is not releasable. Further details are presented in Table 3.

Agile development promotes the Test-Driven Development (TDD) technique. A cyclic process is performed whereby first LLRs are established together with their test cases. Next, test code is produced and all tests are executed to verify that they fail. Then, source code is produced that just passes the tests. Finally, the code is refactored and tests are re-executed. This cycle repeats until all LLRs have been implemented. In practise, the TDD technique implies that software development activities will be performed in conjunction with software verification activities.

6.4 Closure Phase Activities

Upon start of the Closure phase, a sufficient number of features should be completed to warrant release of the application. During Closure, all data items that already exist in some form (see outputs in Tables 2 and 3) are brought up to date. The remaining data

Table 3. Activities during Development phase.

DO-178 Process	Inputs	Activities	**Outputs**
Software design	HLRs, software architecture, trace data	Define Low-level requirements (LLRs) by conditions and associated actions [13]	LLRs, trace data
Software coding	LLRs	Produce code for the LLRs	Source code
Integration	Source code	Perform continuous integration	Executable object code
Software verification	HLRs, HLR test cases, software architecture, LLRs, source code, executable object code, trace data	Establish test cases for LLRs. Produce test code for HLRs and LLRs. Execute (automated) tests. Verify all outputs	HLR test procedures, HLR test results, LLR test cases, LLR test procedures, LLR test results, verification results

items required for compliance with DO-178 are produced by other processes than software development and software verification. For example, the software configuration process produces the Software Configuration Index and the certification liaison process produces the Software Accomplishment Summary.

6.5 Remarks and Potential Issues

The proposed process aims to address some of the key challenges we identified in the survey, in particular challenges related to requirements management. Breaking work down in shorter iterations, including planning (Preparation) and evaluation (Closure) means that planning may be done using updated information from previous Sprints, and that each Sprint provides information needed to meet the requirements of DO-178 (in the form of data items). From related research we know that such a process needs to be supported by tools to automate test-driven development and documentation creation as much as possible in order to save time and to ensure quality and consistency [8].

Including agile approaches in the development process for avionics software promises the usually cited benefits such as frequent delivery of working software, including all data items required by DO-178, and the ability to deal with frequent changes in requirements. There are, however, also a number of potential issues.

Contrary to the waterfall model, or the V-model, HLRs are defined in batches; each time that the Preparation phase is entered, a sufficient number of HLRs are defined for the subsequent sequence of Sprints. Having no overview of the complete set of HLRs in an early phase of the development could lead to an inadequate software architecture that may need drastic (and therefore costly) revision during subsequent Preparation phases. This means that also agile projects needs to invest in a sufficient level of detail of HLRs and overall system architecture early. An agile process though may create better opportunities to manage changes when they occur.

Another issue is that the definition of derived HLRs late in the development, e.g., after several cycles of Preparation, Development, and Closure have taken place, may have consequences for the safety analysis [21]. For example, if derived HLRs imply new interfaces that falsify earlier independence claims, a higher software level could be required, creating additional (verification) work that could have been done more efficiently when known beforehand.

7 Conclusions and Further Work

The development of safety-critical software by the avionics industry is governed by RTCA document DO-178. The document places much emphasis on documented and traceable verification to achieve an acceptable level of confidence that the software development activities have been performed successfully. Indeed, our survey, among major players in the European avionics industry, confirmed that verification and certification constitutes a large portion of the total costs of development (estimated 40%). The survey also revealed other challenges perceived by this industry, including requirements volatility, late discovery of problems/defects, and project cost overruns.

The adoption of an agile framework could be a solution for these challenges; this is in line with other related safety-industry oriented research [7, 8]. At present, the lifecycle model mostly used by the avionics industry to organize software development is the V-model, or variants thereof. DO-178, however, does not preclude the use of any particular model, and in general, there seem to be no obstacles for adopting an agile framework. It is clear that agile methods, like Scrum, need to be adapted to fit in the development and certification of avionics software. In particular, such methods need to be extended to fulfil requirements of traceability and documentation. Some of these may be enabled by use of proper tools that provide a high level of automation.

Using Scrum as a basis, an approach has been outlined that benefits from agile methods and can also satisfy the objectives of DO-178. Some DO-178 objectives are achieved in an agile way, while others, in particular a subset of the verification objectives, are achieved by traditional means (management plans, reviews, and analyses). Benefits expected from the agile approach include reduction of risks, adaptability to changing requirements, and overall a reduction of development cost.

There are, however, issues that need further investigation. One of these is that software requirements are defined in batches; each time, sufficient software requirements are defined for the subsequent sequence of Sprints. Having no overview of the complete set of software requirements in an early phase of the development could lead to an inadequate software architecture that would need thorough revision later on.

To conclude, agile methods may promise to resolve some of the specific challenges in the avionics domain, but there is still a clear need for more research and industrial experimentation to verify applicability and to demonstrate improvement effects.

Acknowledgments. The authors would like to thank the anonymous contributors to the survey and Rob Udo from NLR for his contributions to this research. Also the insightful comments from the reviewers are much appreciated. The research leading to these results has received funding

from the European Community's Seventh Framework Programme FP7/2012-2016 under grant agreement no. ACP2-GA-2013-605442.

References

1. Carlson, R., Turner, R.: Review of agile case studies for applicability to aircraft systems integration. Procedia Comput. Sci. **16**, 469–474 (2013)
2. Cawley, O., Wang, X., Richardson, I.: Lean/Agile software development methodologies in regulated environments – state of the art. In: Abrahamsson, P., Oza, N. (eds.) LESS 2010. LNBIP, vol. 65, pp. 31–36. Springer, Heidelberg (2010). doi:10.1007/978-3-642-16416-3_4
3. Chenu, E.: Agility and lean for avionics. In: Lean, Agile Approach to High-Integrity Software Conference, Paris (2009)
4. Chenu, E.: Agile and Lean software development for avionic software. Whitepaper, Thales Avionics (2011)
5. Coe, D.J., Kulick, J.H.: A model-based agile process for DO-178C certification. In: Proceedings of 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing, Las Vegas (2013)
6. Dingsøyr, T., Nerur, S., Balijepally, V., Moe, N.B.: A decade of agile methodologies: towards explaining agile software development. J. Syst. Softw. **85**(6), 1213–1221 (2012)
7. 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)
8. Hanssen, Geir K., Haugset, B., Stålhane, T., Myklebust, T., Kulbrandstad, I.: Quality assurance in scrum applied to safety critical software. In: Sharp, H., Hall, T. (eds.) XP 2016. LNBIP, vol. 251, pp. 92–103. Springer, Cham (2016). doi:10.1007/978-3-319-33515-5_8
9. Hantke, D.: An approach for combining spice and scrum in software development projects. In: Rout, T., O'Connor, R.V., Dorling, A. (eds.) SPICE 2015. CCIS, vol. 526, pp. 233–238. Springer, Cham (2015). doi:10.1007/978-3-319-19860-6_18
10. Hilderman, V.: DO-178B Costs Versus Benefits. HighRely Inc., HighRely Whitepaper (2009)
11. Knight, J.C.: Safety critical systems: challenges and directions. In: Proceedings of the 24rd International Conference on Software Engineering, ICSE 2002. IEEE (2002)
12. Lambourg, J., Comar, C.: Methodology: agile development of safety critical systems. OpenCoss Framework 7 project (2012)
13. Meunier, V., Destouesse, M., Cros, T.: How to "take credit" of agile principles within a certification context? (2008) (Presentation)
14. 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 Assesment and Management (PSAM). PSAM, Hawaii (2014)
15. Paige, Richard F., Charalambous, R., Ge, X., Brooke, Phillip J.: Towards agile engineering of high-integrity systems. In: Harrison, Michael D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 30–43. Springer, Heidelberg (2008). doi:10.1007/978-3-540-87698-4_6
16. Paige, R.F., Galloway, A., Charalambous, R., Ge, X.: High-integrity agile processes for the development of safety critical software. Int. J. Crit. Comput.-Based Syst. **2**(2), 181–216 (2011)
17. Rottier, P.A., Rodrigues, V: Agile development in a medical device company. In: AGILE 2008 Conference (2008)
18. RTCA, DO-178C: Software considerations in airborne systems and equipment certification (2011)

19. Schwaber K.: SCRUM development process. In: Sutherland, J., Casanave, C., Miller, J., Patel, P., Hollowell, G. (eds.) Business Object Design and Implementation, pp. 117–134. Springer, London (1997). ISBN 978-3-540-76096-2
20. Stålhane, T., Myklebust, T., Hanssen, G.K.: The application of Scrum IEC 61508 certifiable software. In Proceedings of ESREL, Helsinki, Finland
21. VanderLeest, S.H., Buter, A.: Escape the waterfall: agile for aerospace. In: Proceedings of IEEE/AIAA 28th Digital Avionics Systems Conference, DASC 2009, p. 6, (6D3). IEEE (2009). doi:10.1109/DASC.2009.5347438
22. Wils, A., Baelen, S., Holvoet, T., Vlaminck, K.: Agility in the avionics software world. In: Abrahamsson, P., Marchesi, M., Succi, G. (eds.) XP 2006. LNCS, vol. 4044, pp. 123–132. Springer, Heidelberg (2006). doi:10.1007/11774129_13

Short Research Papers

Inoculating an Agile Company with User-Centred Design: An Empirical Study

Silvia Bordin[1(✉)] and Antonella De Angeli[1,2]

[1] Department of Information Engineering and Computer Science, University of Trento, via Sommarive 9, 38123 Trento, Italy
{Silvia.bordin,antonella.deangeli}@unitn.it
[2] School of Computer Science, University of Lincoln, Brayford Pool, Lincoln, LN6 7TS, UK

Abstract. We present an empirical study on facilitating the adoption of user-centred design (UCD) in small Agile companies. To this end, we introduced a curated set of qualitative design practices in an Agile organisation, engaging developers in a lightweight series of workshops. Our results suggest that the approach followed enhanced internal communication and promoted a concrete shift towards a more user-centred perspective. However, the presence of a predominant non-Agile customer seems to have limited potential benefits.

Keywords: Qualitative research · Training developers · User-centred mindset

1 Introduction

Still in 2013, Moreno et al. stated that "the integration of usability engineering methods into software development life cycles is seldom realized in industrial settings" [11]. One reason for this is the "sheer lack of usability specialists in the industry" [5], which results in insufficient knowledge about the work of the end user [8] and in the so-called "developer mindset" [1], overly focused on technological aspects. Another issue relates to the limited suitability of most usability and UX methods for the Agile setting [15], with several authors [2, 4, 7] reporting a particular scarcity of lightweight practices for user involvement in development projects despite the benefits induced by the ability to perform usability and UX work in an agile context [4, 15]. In addition, even companies realising a need to increase the usability of their products may be unable to invest in the resources needed to achieve this [5], and this is particularly true in the case of small enterprises [1, 5].

To facilitate the adoption of user-centred design (UCD) in small Agile companies, we curated the identification of a small set of design techniques; we then planned an action research intervention for presenting them to developers and assessing the impact of these techniques on their working practices. A first iteration has been reported in [3], while a second iteration is reported here. Our results suggest that even such a lightweight approach may support the enactment of a user-centred mindset. However, the impact of the intervention has been limited by the relationship with a dominant customer resistant both to Agile and UCD: we conclude by pointing out the need both for researchers and

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 235–242, 2017.
DOI: 10.1007/978-3-319-57633-6_15

practitioners to investigate more effective ways to communicate the business benefits that the two approaches may bring.

2 Related Work

The term "user-centred design" denotes a broad set of techniques, methods, procedures and processes that places the user at the centre of an iterative design process [17]. The acknowledged benefits of involving users in systems design [e.g. 1] include improved quality and acceptance of the system, and cost saving [12]. Although promising to support "the execution of software development projects targeting the delivery of useful *and* usable software" [4], the integration of UCD and Agile development is however not trivial to achieve [e.g. 2] and limited empirical research exists on the topic [4, 7]. One of the ways to enact this integration, particularly in the limited-resource context of small enterprises, is "to use the software developers as a UX work resource by enhancing their qualifications within the field of usability and UX" [14], or in other words to train developers on usability techniques. Advantages include "the potential of easing problems regarding the lack of usability specialists in the industry" [5]; the chance for small companies to lessen "the need to staff usability specialists, which cannot be funded" [5]; a good fit with the Agile feature of team members being able to perform every given work task [13]. This is where we place our contribution.

We also point out, however, that also the customer needs to be supportive of the integration of UCD and Agile, allowing for a suitable design to be researched [2] and for adequate access to users. Scepticism is more frequent in large customers [10] and may result in a lack of customer engagement, which can be a big challenge for development teams [10] especially given the relevance placed on the customer by the Agile philosophy. The solution may require the capability of demonstrating business value, management support, and nurturing a change of mindset and culture in the customer [9]: how to effectively communicate this has however remained largely an open point to date.

3 Action Research Intervention

The activities described here were carried out in "the Company", a branch of a large Italian IT group providing cyber security and network configuration services to the largest telecommunication operators of the country. The Company had long adopted Agile successfully, and had one main customer, a large telecommunication provider that we will call "the Telco". Being the Telco a much larger venture, the power relationship between the two parties was naturally asymmetrical, although generally warm. However, the Telco is also a highly structured corporate, whose constrained workflow prevents a full implementation of Agile in the projects followed by the Company, and where some representatives seem to oppose contacts between the Company and final users of their software. While trying to reduce their dependency from the Telco, the Company realised that they were lacking sufficient skills in usability and interface design, and that this could be an issue in proposing their products to fresh customers; therefore, they asked for our help.

3.1 Method

We followed the Cooperative Method Development approach, a "domain specific adaptation of action research" that moves from an ethnographically-inspired understanding of the "existing practice of software development in concrete industrial settings" and aims at improving such practice by cooperating with practitioners [6].

Design techniques presented to developers were chosen to overcome potential communication breakdowns in the integration of UCD and Agile [2], and to reflect surveys on the usability techniques most used in industry [e.g. 14], particularly accounting for their feasibility of integration into an Agile environment and of teaching non-UX professionals. These methods include low-fidelity prototyping, usability testing, personas, expert evaluations, and user task analyses. We remark that this intervention is meant for "supporting developers during ongoing day-to-day product development" [13] and that "we do not discard the need for a usability expert" [8].

3.2 Preliminary Understanding

The first author interviewed developers in June 2016 about their perception of the working environment, their current working practices, and their attitude towards UCD-related themes. The interview study lasted two days and involved 7 people. For what concerns the organisational setting, the Company employs about 20 people, mostly young graduate developers, and exhibits a pretty flat hierarchy. The environment is predominantly technical, yet with a positive and rather curious attitude towards UCD-related themes, to the point that employees explicitly argued in favour of the collaboration with our University in front of the group managers, who tend to adopt a more "command and control" approach instead.

The Company proposed to focus on what we will call the Software (a desktop application used to configure and monitor networking devices for corporate customers of Telco) as a running example during the intervention, for a variety of reasons: it is entirely developed within the Company for Telco; it has evolved over several years as the juxtaposition of different parts, and would now need a refactoring; being one of the main projects of the Company, it is sufficiently well known to all employees.

3.3 Implementation

In August 2016 a series of four workshops, each lasting a whole day, was carried out at the Company site. The agenda of workshops is outlined in Fig. 1 and was grounded on different elements: on a practical level, we accounted for the results of preliminary interviews and for the feedback from our previous iteration of a similar series of workshops [3]; on a more theoretical level, we accounted for the stages of the traditional UCD lifecycle, and the set of focal points to consider in the integration of UCD and Agile development [2], namely the extent of user and customer involvement, the role of documentation, the synchronisation of design and development iterations, and ownership over UX tasks.

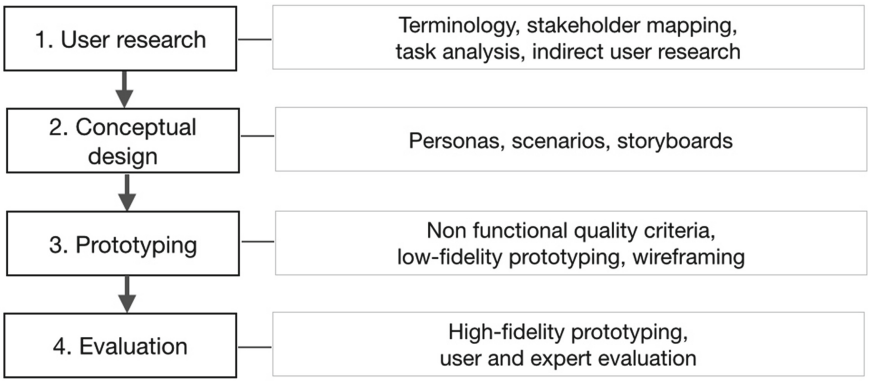

Fig. 1. Workshops overview.

Three developers (who will be referenced as D1 to D3) were appointed to attend the whole workshop cycle, led by the first author in the role of a facilitator. All of them expressed great interest in user-related themes: D1 was self-taught on UCD techniques, while D2 and D3 were not familiar at all with them.

"One doesn't even know where to start from, without knowing any basics" (D3)
"More than once [design choices] have been a stab in the dark" (D1)
"If there's one skill in the Company we are really lacking it is interface design … we try to do what we can" (D2)

Workshops started by motivating more formally the advantages of adopting UCD, that is by presenting well-known reports from industry [e.g. 18] highlighting user involvement as a key factor for project success, and in contrast the lack of it as one of the most common reasons for failure. We then considered the workflow supported by the Software, illustrated in a very technology-centred way in its user manual, and guided participants in re-elaborating it focusing on the perspective of users. In collaboration with the facilitator, participants then outlined the stakeholder network related to the Software, which confirmed how the needs of actual users were generally mediated when reported to the Company, if collected at all. A task analysis was then performed for actors most likely to interact significantly with the Software, and was represented through use-case diagrams. The project manager was chosen as the reference user: information on the characteristics of Software users in this role was retrieved indirectly, i.e. through LinkedIn and narratives of other Company employees, and then expressed through a couple of personas representing different levels of expertise; these in turn inspired scenarios and storyboards.

Once a reference persona was chosen, participants rated the dimensions of usability listed in [12] through poker planning, regarding them as non-functional quality criteria for the Software. Participants then elaborated different low-fidelity prototypes for a specific interface of the Software; however, a later inspection revealed that these alternatives could not support the same workflow articulation as the existing interface. Hence, since the latter was anyway rather complex, participants asked for support in wireframing a more logical re-grouping of its functionalities.

Finally, the different purposes of low- and high-fidelity prototypes and how to communicate them were illustrated, since D1 repeatedly pointed out that Telco would not accept discussing over a "non serious" low-fidelity prototype and that previous attempts at doing this had failed. In addition, a session of user testing was simulated on the ERP system in use at the Company for demonstrative purposes. After the end of the intervention, participants organised a wrap-up session and, a few weeks later, a dissemination seminar for their colleagues.

3.4 Evaluation

In December 2016, an external researcher interviewed participants about what they remembered of proposed techniques after a few months and whether they felt that their approach to design and development had changed. Interviews were loosely transcribed and thematically analysed. Overall, participants positively welcomed our intervention, regarding it as a chance of professional growth. They appreciated having learnt concrete techniques, and remembered them correctly:

"I enjoyed wireframing a lot. It really gave me a different point of view" (D1)
"We should organise the info with wireframes, the poor user will be scared" (D3)

In addition, they expressed appreciation also for the presence of a trainer, reiterating the effectiveness of scaffolding [19]:

"In terms of common sense, this is what every developer should do. Yet having someone explaining you the steps to follow is something different" (D2)
"Now I have a method"(D3)

The training seems in fact to have contributed to enacting a shift from a technology-focused mindset to a more user-centred one:

"Before we used to say – [the user] will have to get over it" "The interface as the means to achieve an objective from the user's point of view" "The goal is to remove the need for a manual – even for us as developers!" (D3)
"We'll surely follow this approach rather than – bah, let's just do something" "I have been assigned to a project where the interface is set in stone [by Telco], BUT [developers and management] all agree that we are going to apply UCD techniques at the first suitable occasion" (D2)

D2 and D3 in fact claimed to have applied proposed techniques as much as possible to the improvement of minor parts of the Software interface that had been assigned to them, and to have used them to support communication with colleagues:

"Prototypes and scenarios can be used internally to understand how to design something […] I proposed some prototypes to my colleagues and this simplified the discussion" "In my opinion personas should be shared by the whole team… to raise awareness among colleagues" (D3)

Participants also commented on the positive attitude shown by the rest of the Company at the end of the dissemination seminar they led:

"We reported to the rest of the Company and the reaction was – let's hope we will soon have projects where to apply this approach" (D2)

Despite the satisfaction and interest shown, participants did not believe the approach would prove fully applicable in the interaction with their customer due to the strong "developer mindset" [1] of Telco's representatives, even harder to overcome due to the unbalanced power relationship with the Company:

> *"Personas cannot be used with Telco: our customer is very much feature-oriented and in a dominant position […] it does not want to see the prototype, it wants to see the product" "Some techniques will be more applicable than others, because it is impossible to access users […] We have no [user] feedback. Clearly we miss it"* (D1)
> *"I guess the customer would be disappointed by storyboards on paper […] it may think we did not put too much effort into such a proposal"* (D3)

4 Discussion

In terms of the applicability of the presented approach to other small enterprises, we suggest that, together with the Company's "culture receptive to UCD" [2], developers' consolidated familiarity with Agile (including being used to change and flexibility, iteration, and frequent interaction with the customer) may have allowed a deeper appropriation of UX techniques, resulting in a potentially sustained impact over working practices. Furthermore, participants demonstrated an accurate recall of techniques and of their rationale, and reported a spontaneous sharing of their learning with colleagues, applying proposed techniques whenever possible to support interface design and internal discussion. This reflects claims in [16], where Agile and UCD-inspired practices are considered to have a positive impact on mutual understanding and communication; moreover, these factors suggest that even a lightweight intervention such as the one described in this paper may support the enactment of a more user-centred mindset. This can constitute a first step for the organisation towards the awareness of the benefits of integrating UCD, providing elements to decide whether to proceed in developers' UX training or to hire a specialist designer.

The impact of proposed techniques seems however to have been limited by the Company doing Agile in a non-Agile environment, where this label includes both the culture of the parent group and of the Telco. We argue that the same challenges encountered in this setting [9, 10] can be found when introducing UCD in an environment not accustomed to it. We envision as future work the evaluation of the set of UCD techniques in an Agile company whose customer is also Agile: this would be the most favourable context. In conclusion, we point out to the research and practitioners' community that there is still a lack of suitable ways to clearly communicate to reluctant customers the potential benefits of Agile and UCD [10].

Acknowledgments. We thank Angela Di Fiore for collaborating in the evaluation interviews, and our participants and the whole Company for their welcoming and kind support. This work has been possible thanks to the funding granted by the Italian Ministry of Education, University and Research (MIUR) through the project "Città Educante", project code CTN01_00034_393801.

References

1. Ardito, C., Buono, P., Caivano, D., Costabile, M.F., Lanzilotti, R.: Investigating and promoting UX practice in industry: an experimental study. Int. J. Hum. Comput. Stud. **72**(6), 542–551 (2014)
2. Bordin, S., De Angeli, A.: Focal points for a more user-centred agile development. In: Sharp, H., Hall, T. (eds.) XP 2016. LNBIP, vol. 251, pp. 3–15. Springer, Cham (2016). doi: 10.1007/978-3-319-33515-5_1
3. Bordin, S., De Angeli, A.: Supporting cooperative work by integrating user-centred design and agile development. Submitted at the European Conference on Computer-Supported Cooperative Work (2016)
4. Brhel, M., Meth, H., Maedche, A., Werder, K.: Exploring principles of user-centered agile software development: a literature review. Inf. Softw. Technol. **61**, 163–181 (2015)
5. Bruun, A.: Training software developers in usability engineering: a literature review. In: Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries. ACM (2010)
6. Dittrich, Y., Rönkkö, K., Eriksson, J., Hansson, C., Lindeberg, O.: Cooperative method development. Empirical Softw. Eng. **13**(3), 231–260 (2008)
7. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: a systematic review. Inf. Softw. Technol. **50**(9), 833–859 (2008)
8. Eriksson, E., Cajander, Å., Gulliksen, J.: Hello world! – experiencing usability methods without usability expertise. In: Gross, T., Gulliksen, J., Kotzé, P., Oestreicher, L., Palanque, P., Prates, R.O., Winckler, M. (eds.) INTERACT 2009. LNCS, vol. 5727, pp. 550–565. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03658-3_60
9. Gregory, P., Barroca, L., Sharp, H., Deshpande, A., Taylor, K.: The challenges that challenge: engaging with agile practitioners' concerns. Inf. Softw. Technol. **77**, 92–104 (2016)
10. Hoda, R., Noble, J., Marshall, S.: The impact of inadequate customer collaboration on self-organizing agile teams. Inf. Softw. Technol. **53**(5), 521–534 (2011)
11. Moreno, A.M., Seffah, A., Capilla, R., Sanchez-Segura, M.-I.: HCI practices for building usable software. Computer **46**(4), 100–102 (2013)
12. Nielsen, J.: Usability Engineering. Morgan Kaufmann, San Francisco (1994)
13. Øvad, T., Bornoe, N., Larsen, L.B., Stage, J.: Teaching software developers to perform UX tasks. In: Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction, pp. 397–406. ACM (2015)
14. Øvad, T., Larsen, L.B.: The prevalence of UX design in agile development processes in industry. In: Agile Conference (AGILE), pp. 40–49. IEEE (2015)
15. Øvad, T., Larsen, L.B.: How to reduce the UX bottleneck–train your software developers. Behav. Inf. Technol. **35**(12), 1080–1090 (2016)
16. Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., Still, J.: The impact of agile practices on communication in software development. Empirical Softw. Eng. **13**(3), 303–337 (2008)
17. Rogers, Y., Sharp, H., Preece, J.: Interaction Design: Beyond Human-Computer Interaction. John Wiley & Sons, Hoboken (2011)
18. The Standish Group CHAOS Report (2014). https://www.projectsmart.co.uk/white-papers/chaos-report.pdf
19. Vygotsky, L.S.: Mind in Society: The Development of Higher Psychological Processes. Harvard University Press, Cambridge (1980)

On the Usage and Benefits of Agile Methods & Practices

A Case Study at Bosch Chassis Systems Control

Philipp Diebold[1(✉)] and Udo Mayer[2]

[1] Fraunhofer Institute for Experimental Software Engineering, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
philipp.diebold@iese.fraunhofer.de

[2] Bosch Chassis Systems Control, Robert-Bosch-Allee 1, 74232 Abstatt, Germany
udo.mayer@de.bosch.com

Abstract. Since software became a major part of the car, we were interested in identifying which agile practices are used and adapted at Bosch automotive. Therefore, we conducted a multi-case study with nine interviews from five Bosch projects. Our results showed a strong focus on Scrum. Most of the Scrum practices are adapted due to the specific project context. Practices from other agile methods, e.g. XP, are used and adapted as well. We further collected the benefits of the practices, most often resulting in improved transparency and planning. The results are used to support automotive projects in selecting and applying agile practices according to their specific process improvement goals.

Keywords: Automotive · Agile practices · Scrum deviations · Case study

1 Introduction

Within software engineering, agile development has shown to be a commonly used approach [8, 9, 12]. Since embedded domains struggle with the integration of different disciplines, e.g. hardware and electronics, there is a lot more communication necessary [11]. These domains are currently using agile only to some extent in order to profit from the benefits of agile development, e.g. shorter time-to-market. This was one of the major reasons for Bosch Chassis Systems Control (CC) to become more agile. Thus, the state of agile in projects is interesting. Knowledge about usage, adaptations of Agile Methods and Practices, and their benefits should help spreading of agile.

2 Chassis Systems Control of the Bosch Group

The division CC is part of the business sector Mobility Solutions of the Bosch Group. The business sector Mobility Solution generated sales of 41.7 billion euros in 2015 (Bosch Group in total: 70.6 billion euros) which makes the Bosch Group to one of the world's largest automotive suppliers. The division CC develops and manufactures innovative components, functions and systems that are designed to make driving a safe and

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 243–250, 2017.
DOI: 10.1007/978-3-319-57633-6_16

comfortable experience. The projects of this study are developing systems for (highly) automated driving, e.g. a highway pilot.

3 Study Design

Research Questions. Our objective is to better understand the usage, reasons and adaptions as well as benefits of agile methods and practices within Bosch CC. Thus, we ended up with the following research questions: RQ1 - Which agile elements (incl. agile methods and agile practices [3]) are used? RQ2 - How is the usage of the agile practice deviating from the textbooks? RQ3 - What are the benefits of the agile practices?

Case and Subject Selection. Possible cases (Bosch CC projects) and subjects (interviewees) were identified by Bosch based on the organizational scope and their usage of agility. The latter was mandatory for participation. We identified 15 candidates representing five projects. Nine of the 15 candidates participated, covering all five projects.

Data Collection Procedure. We conducted structured interviews with the participants, seven face-to-face and two via phone. The questions were categorized into (1) introduction, (2) usage and (3) benefits of agile elements (first methods, then practices), and (4) project context. The principal author conducted all interviews as follows: He explained the idea and reason behind this study, the data collection, and the guaranteed anonymity. He asked for used agile methods and agile practices. He discussed about the experienced impacts, mainly benefits of the different elements. During the gathering of the benefits, we were open to any mentioned benefits. If they had no idea, we named potential benefits as triggers. For not biasing them, the interviewees had to give examples for these benefit triggers. We ended up in a coded list of the benefits (cf. Table 2).

Analysis Procedure. We analyzed the notes of the interviews qualitatively. First, we extracted the data from each interview (usage of agile elements, where and how, and their benefits). Second, we compared and aggregated information from interviews related to the same project. Third, we compared the project-aggregated results among each other and with our experience and literature.

4 Results

We covered different team sizes and developed functions and considered different roles of the interviewees: group leaders, department leaders, project leaders/managers (cf. Table 1). Two participants performed the Scrum Master role.

4.1 RQ1: Usage of Agile Methods and Practices

Four Agile Methods were used by the projects: Scrum (n = 4 projects), Flow (n = 2), iPeP (n = 1), and SAFe (n = 1). The practices that were used in **all five projects** belong to Scrum, namely Sprint, Backlog, Sprint Planning, and Daily Stand-Ups, although one

Table 1. Project characteristics

Project	Size	Locations	Project phase [6]	Participating roles
P1	60 persons, 7 sub-teams	2 in Germany	Pre-development	system project lead
P2	8 teams	1 in Germany, Europe, Asia each	Pre-development	technical project lead & project lead
P3	33 persons	2 in Germany	Series development	project lead & team lead
P4	8 persons	1 in Germany	Pre-development	Group lead
P5	70 persons	1 in Germany, Europe, Asia each	Pre-development	project lead, SW project lead & department lead

project reported that it is not using Scrum. The practices that were used in **most projects** were Sprint Review, Sprint Retrospective, Burndown-charts, Definition of Done, Scrum Master, and Product Owner (PO) (all Scrum), and User Stories and Epics (both XP). Continuous Integration, Release Planning and Scrum-of-Scrums completed the set. The practices that were used in **few projects** were: Planning Poker and Pair Programming (both XP), 80%-rule and Pull-system (both Lean/Flow), and Backlog Grooming (Scrum). In Table 2, we show the use of agile practices.

Table 2. Benefits of used agile practices on goals (numbers indicate how many projects mentioned a benefit)

(numbers indicate how many projects mentioned a benefit)

Agile Practice	understandability	knowledge transfer	communication	transparency	feedback	satisfaction	team empowerment	focusing	structuredness	planning	time-to-market	risk management	# adressed benefits	# using projects
Daily Stand-Up			3	5						2			**3**	**4**
Sprint				2				1	2	1		1	**5**	**5**
Sprint Review	1		2	1						1			**4**	**4**
Retrospective		1			1	1							**3**	**4**
Sprint Planning						1				4		1	**3**	**5**
Backlog	1			2					1	1			**4**	**5**
Burndown-Charts				3									**1**	**4**
Scrum Master						1	1						**2**	**4**
Product Owner				1			1			1			**3**	**4**
80%-Rule									1	2		2	**3**	**2**
Planning Poker							1			2			**2**	**1**
User Stories								1					**1**	**4**
Epics										3			**1**	**3**
Cont. Integration				2							1		**2**	**3**
Scrum-of-Scrums			4	1					1	1			**4**	**3**
# practices	**2**	**1**	**3**	**8**	**1**	**3**	**3**	**2**	**4**	**10**	**1**	**3**		

4.2 RQ2: Deviations of Agile Practices

The used practices are now discussed regarding their deviations from existing guidelines, e.g. the Scrum Guide:

The major issue of the **Scrum Master** was the manifestation as "caretaker", e.g. in one team the Scrum Master was only inviting for the different Scrum meetings. Further, often the Scrum Master was not the only role. Being a project leader or a PO in combination might result in a conflict of interests. The **PO** also deals with the aspect of several roles by one person without problems. But some teams defined an additional "feature responsible", which covered POs tasks, e.g. breaking down the features into Stories. Finally, within one project, they struggled with the issue of having a Chief-PO communicating with the customer high-level, but not knowing the requirements in detail.

The most deviating aspect of **sprints** was their length. For one project (needing four weeks) more than two weeks were necessary for the integration of hardware aspects. Another one worked with varying lengths over time.

The **Scrum-of-Scrums** meetings conducted weekly by the larger teams or projects was in one case a team-meeting in which all project members participated. In another case, it was conducted by the project lead to keep the offshore team on track.

The **Daily Stand-Ups** deviated in three different aspects: (1) Usage only for status-tracking purpose, (2) Meetings lasting up to one hour, (3) Frequency of the meeting: Instead of every day, teams conducted it every other day, every three days, once a week, or unregularly. An extended duration of the meeting (one hour) was the consequence.

Sprint Planning: Within the sprint shift (Review, Retrospective, and upcoming Planning), most of the teams performed all three meetings en-block in about three days. In one of the larger teams, they found a solution to break down the Backlog to the teams by the PO. The planning itself was conducted within the single teams without the PO.

Sprint Review: The composition of meeting participants ranged from meetings without the customer and only with the team leaders and functional owners up to an "open event". Some meetings were unstructured without agenda or open topic list. Others were not conducted as an "acceptance meeting" with stakeholders or PO.

Sprint Retrospective: This meeting varied within the timing: One team performed it every other sprint, because two weeks were too short to resolve impediments. Another team directly resolved smaller impediments during the meetings. One interviewee mentioned that the retrospective was only weakly defined and valued within their team.

Within the **Backlog**, two major variations existed: Some included prioritization, whereas other did not. Different backlogs were used, from team-, project- or overall backlogs up to release backlogs. These kinds also contained various backlog items, e.g. User Stories vs. Epics. Two teams did not use **User Stories** as prescribed by the given templates, since they were not used to it and needed more information.

Planning Poker: One team used this practice for estimating the granularity-level to refine the story or not. Furthermore, not all teams used "story points" as values, but e.g. person hours or days. Finally, one deviation was that in one team just the "key player" (a senior developer or feature responsible) decided on the estimation value.

Within the **80%-Rule**, one interviewee mentioned that they are not considering it for the workload, but on their throughput. In the only case using the **pull-principle**, the

responsibility regarding the different functions was clear such that it was "obvious who is going to pull what". Thus, sometimes one team member just "assigned" the tasks.

4.3 RQ3: Benefits of Agile Practices

Most of the considered agile practices are beneficial for *planning* (10 practices) and *transparency* (8), followed by *structuredness* (4), *communication*, *risk management, satisfaction,* and *team empowerment* (each 3). Checking the overall number of addressed benefits per practice (cf. Table 2), e.g. the highest with five is the sprint, three yield four benefits, and five yield three benefits.

Except for the Definition of Done and the Backlog Grooming, we could gather at least one impact for each of the Scrum practices. Considering the Scrum meetings, except the retrospective, all impact *planning*. Sprint Review as well as the Stand-Ups also influence *communication* and *transparency*. The Sprint Retrospective was the only practice dealing with *feedback* and *knowledge transfer*. The Burndown-Chart is a quite good mechanism for *transparency*. Both Scrum roles impact the *team empowerment*. The improved *satisfaction* resulting from the Scrum Master correspond with that. The PO provides *transparency* as well as *planning* aspects. Considering the Non-Scrum practices, there is information about the impact of six practices: *Risk management, planning* and *structuredness* were impacted by the 80%-Rule. Planning Poker increases *planning* and *team empowerment*. User Stories and Epics influence *planning* (Epics) or *focusing* (User Stories). Continuous Integration improves *transparency* and *time-to-market*. Finally, Scrum-of-Scrums was quite good for *communication* and it affects *transparency, structuredness,* and *planning*.

5 Related Works and Discussion

Distribution/Frequency of Agile Practices. Within VersionOne [12], the most used agile practices were Daily Stand-Ups, prioritized Backlogs, short Iterations (=Sprints), Retrospective, Iteration Planning, and Release Planning. All of them were used by at least one project, except for the Release Planning. For the other practices mentioned by [12], there are some differences: The Sprint Reviews are used less often than in our study, whereas, Continuous Integration is more common. Focusing on agile practices, Kumos [9] reports that almost all of the top six used agile practices are Scrum practices, all used by the Bosch projects. Scrum is also the prevalent agile method in automotive [8]. Similar to our results, the Planning, Daily Stand-Ups, Review, and Retrospective are used. We cannot confirm that the Sprint Review is used less often. In our study it was the Retrospective.

Deviations of Agile Practices. A prior study on Scrum variations [1] showed similar patterns, with the Scrum Master and PO given to people already owing a role, e.g. developer or team lead. However, within our cases most of the Scrum roles were staffed. Considering the 15–30 min of the Daily Stand-Ups, this timeframe is exceeded by some cases and extended up to one hour. The frequency deviation of the Stand-Up is also

common [1]. For the sprint length variance from two to four weeks could be confirmed. Only one project reasonably decided to have four weeks due to synchronizations of teams and disciplines such as hardware and software.

Benefits of Agile Practices. Considering the benefits reported by [12], we see some similarities: The benefits *increased team productivity, improved project visibility, increased team moral/motivation, better delivery predictability*, and *reduced project risk* can directly be mapped to the ones reported to us. In contrast, the benefits dealing with engineering and quality, such as *enhancing software quality, software maintainability*, or *improving engineering discipline*, were not mentioned within our interviews. Compared to [9], five of nine benefits were also mentioned by our participants: *transparency, customer orientation, timing*, *teamwork*, and *employee motivation*. The results of [9] (similar to [12]) additionally showed *quality* as highly impacted, not indicated by our results.

Besides these studies, there are some practices studied in detail with their different impacts, e.g. User Stories [4], Planning Poker [5] or Pair Programming [7]. Furthermore, some studies analyzed which agile practice influence one specific benefit, e.g. [10] dealing with communication: Daily Stand-Ups improve *communication* and *transparency*, due to keeping developers, project leaders, and customers aware of the status. Iteration Planning created awareness of the *project plan* and iteration goals, whereas the Retrospective was a good way for working on *process improvement*. Even if we consider the benefits on a more fine-granular level, the results confirm each other.

6 Validity of the Results

An interview guideline and data collection sheet eased aggregation and comparisons. The guideline reduced the risk of misinterpretation and increased the objectivity. We could not recognize any misunderstanding, since all interview participants were aware of the common concepts, methods and practices. Additionally, we experienced that assuring anonymity led the interviewees to answer openly. Regarding the interviewed people within a project, we selected independent ones (not from the same team). Thus, our data is a representative sample of agile development in the area of autonomous driving. Within the data analysis, the aggregation of interview data within one project was the most difficult and error-prone, because of considering different project roles and teams with their viewpoints into one data set. The IESE and Bosch team discussed the aggregated results involving colleagues for an external point of view. We provided a summary report of the results via e-mail and gathered the feedback. We also performed a presentation and discussion session with all participants. That our qualitative results from Bosch CC are in line with most of the related work is another strong indicator for their validity.

7 Conclusions and Future Work

Within this paper, we present the results of an interview series covering five different automotive projects at Bosch CC with overall nine interviews on the topic of usage, deviation, and benefits of single agile practices.

The usage of agile methods as well as the underlying agile practices shows a similar picture as common studies. The most commonly used method is Scrum, which is adapted and extended by other practices. There seem to be similar variations of agile practices in the automotive domain as in information systems. Our study could confirm some of the benefits mentioned by other agile surveys, and could provide further answers to the question, which agile practice provides what specific benefits, and furthermore, that agile practices overall are most beneficial for *transparency* and *planning*.

Within future work, we intend to use the elicited data to instantiate the Agile Capability Analysis [2] for Bosch CC, a goal-oriented SPI approach using agile practices. The next step will be the integration of A-SPICE® and connection to the agile practices.

Acknowledgements. We thank all interviewees for their time, participation, and openness. We also thank A. Schmitt, T. Zehler, S. Theobald and Dr. P. Fröhlich for providing feedback.

References

1. Diebold, P., Ostberg, J.-P., Wagner, S., Zendler, U.: What do practitioners vary in using scrum? In: Lassenius, C., Dingsøyr, T., Paasivaara, M. (eds.) XP 2015. LNBIP, vol. 212, pp. 40–51. Springer, Cham (2015). doi:10.1007/978-3-319-18612-2_4
2. Diebold, P., Zehler. T.: The agile practice impact model. In: Proceedings of ICSSP 2015. ACM (2015)
3. Diebold, P., Zehler, T.: The right degree of agility in rich processes. In: Kuhrmann, M., Münch, J., Richardson, I., Rausch, A., Zhang, H. (eds.) Managing Software Process Evolution, pp. 15–37. Springer, Cham (2016). doi:10.1007/978-3-319-31545-4_2
4. O'hEocha, C., Conboy, K.: The role of the user story agile practice in innovation. In: Abrahamsson, P., Oza, N. (eds.) LESS 2010. LNBIP, vol. 65, pp. 20–30. Springer, Heidelberg (2010). doi:10.1007/978-3-642-16416-3_3
5. Haugen, N.: An empirical study of using planning poker for user story estimation. In: Proceedings of AGILE 2006, pp. 23–34. IEEE (2006)
6. Hirz, M., Dietrich, W., Gfrerrer, A., Lang, J.: Overview of virtual product development. In: Hirz, M., Dietrich, W., Gfrerrer, A., Lang, J. (eds.) Integrated Computer-Aided Design in Automotive Development, pp. 25–50. Springer, Heidelberg (2013)
7. Hulkko, H., Abrahamsson, P.: A multiple case study on the impact of pair programming on product quality. In: Proceedings of ICSE 2005, pp 495–504. ACM (2005)
8. Kugler Maag CIE. Agile in Automotive – State of the Practice (2015)
9. Kumos, A.: Status Quo Agile 2014. University of Applied Science Koblenz (2014)
10. Pikkarainen, M., Haikara, J., Salo, O., Abrahamson, P., Still, J.: The impact of agile practices on communication in SW development. ESEJ **13**(3), 303–337 (2008). Springer
11. Shen, M., Yang, W., Rong, G., Shao, D.: Applying agile methods to embedded software development: a systematic review. In: Proceedings of SEES 2012, pp. 30–36. IEEE (2012)
12. VersionOne: The 10th annual State of Agile Report (2016)

Checklists to Support Test Charter Design in Exploratory Testing

Ahmad Nauman Ghazi(✉), Ratna Pranathi Garigapati, and Kai Petersen

Blekinge Institute of Technology, Karlskrona, Sweden
{nauman.ghazi,kai.petersen}@bth.se, pranathi.r8@gmail.com

Abstract. During exploratory testing sessions the tester simultaneously learns, designs and executes tests. The activity is iterative and utilizes the skills of the tester and provides flexibility and creativity. Test charters are used as a vehicle to support the testers during the testing. The aim of this study is to support practitioners in the design of test charters through checklists. We aimed to identify factors allowing practitioners to critically reflect on their designs and contents of test charters to support practitioners in making informed decisions of what to include in test charters. The factors and contents have been elicited through interviews. Overall, 30 factors and 35 content elements have been elicited.

Keywords: Exploratory testing · Session-based test management · Test charter · Test mission

1 Introduction

James Bach defines exploratory testing as simultaneous learning, test design and test execution [3]. Existing literature reflects that ET is widely used for testing complex systems as well and is perceived to be flexible in all types of test levels, activities and phases [7,13]. In the context of quality, ET has amassed a good amount of evidence on overall defect detection effectiveness, cost effectiveness and high performance for detecting critical defects [1,9–11,13]. Session-based test management (SBTM) is an enhancement to ET. SBTM incorporates planning, structuring, guiding and tracking the test effort with good tool support when conducting ET [4].

A test charter is a clear mission for the test session and a high level plan that determines what should be tested, how it should be tested and the associated limitations. A tester interacts with the product to accomplish a test mission or charter and further reports the results [3]. The charter does not pre-specify the detailed test cases which are executed in each session. But, a total set of charters for an entire project generally include everything that is reasonably testable. The metrics gathered during the session are used to track down the testing process more closely and to make instant reports to management [11]. Specific charters demand more effort in their design whilst providing better focus. A test session often begins with a charter which forms the first part of the scannable session

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 251–258, 2017.
DOI: 10.1007/978-3-319-57633-6_17

sheet or the reviewable result. Normally, a test charter includes the mission statement and the areas to be tested in its design.

Overall, the empirical evidence of how test charters are designed and how to achieve high quality test charters are designed are scarce. High quality test charters are useful, accurate, efficient, adaptable, clear, usable, compliant, and feasible [4]. In this study we make a first step towards understanding test charter design by exploring the factors influencing the design choices, and the elements that could be included in a test charter. This provides the foundation for further studies investigating which elements actually lead to the quality criteria described by Bach [4]. We make the following contributions:

C1: Identify and categorize the influential factors that practitioners consider when designing test charters.
C2: Identify and categorize the possible elements of a test charter.

The remainder of the paper is structured as follows: Sect. 2 presents the related work. Section 3 outlines the research method, followed by the results in Sect. 4. Finally, in Sect. 5, we present the conclusions of this study.

2 Related Work

Test charters, which are an SBTM element plays a major role in guiding inexperienced testers. The charter is a test plan which is usually generated from a test strategy. The charters include ideas that guide the testers as they test. These ideas are partially documented and are subject to change as the project evolves [4]. SBTM echoes the actions of testers who are well experienced in testing and charters play a key role in guiding the inexperienced testers by providing them with details regarding the aspects and actions involved in the particular test session [2].

The context of the test session plays a great role in determining the design of test plan or the charter [4]. Key steps to achieve context awareness are, for example, understanding the project members and the way they are affected by the charter, and understanding work constraints and resources. When designing charters Bach [4] formulated specific goals, in particular finding significant tests quicker, improving quality, and increasing testing efficiency.

The sources that inspire the design of test charters are manifold (cf. [4,8,12]), such as risks, product analysis, requirements, and questions raised by stakeholders. Mission statements, test priorities, risk areas, test logistics, and how to test are example elements of a test charter design identified from the literature review and their description [1,4,6]. Our study will further complement the contents of test charters as they are used in practice.

3 Research Method

Study Purpose and Research Questions: The goal of this study is to investigate the design of test charters and the factors influencing the design of these charters and their contents.

RQ1: What are the factors influencing the design of test charters? The factors provide the contextual information that is important to consider when designing test charters, and complements the research on context aware testing [4].

RQ2: What do practitioners include in their test charters? The checklist of contents supports practitioners to make informed decisions about which contents to include without overlooking relevant ones.

Interviews: Interviews (three face-to-face and six through Skype) were conducted with a total of nine industry practitioners through convenience sampling combined with choosing experienced subjects who are visible in the communities discussing ET (see Table 1).

Table 1. Profile of the Interviewees

Interview ID	Role	Experience in testing	Organizational size
1	Senior systems test engineer	4 years	More than 500
2	Test quality architect	10 years	50–500
3	Test specialist	10 years	50–500
4	Test consultant	12 years	More than 500
5	Test strategist	3 years	Less than 50
6	CEO, Test consultant	30 years	More than 500
7	Test manager	20 years	More than 500
8	CEO, Test lead	4 years	50–500
9	Test quality manager	13 years	50–500

The interviews were semi-structured, following the structure outlined below:

1. *Introduction to research and researcher:* The researchers provide a brief introduction about themselves, followed by a brief description on the research objectives.
2. *Collection of general information:* In this stage, the information related to the interviewee is collected.
3. *Collection of research related information:* This is the last stage where the factors and contents of test charters have been elicited.

Data analysis: All the interviews were recorded by consent of the interviewees and later transcribed manually. The qualitative data collected using literature review and interviews was later analyzed using thematic analysis [5]. After thoroughly studying the coded data, similar codes have been grouped to converge their meaning to form a single definite code.

Validity: The potential bias introduced by interviewing thought leaders and experienced people in the area who are favorable towards exploratory testing may bias the results, and hence may not be fully generalizable. Though, we have not put any value on the factors and contents elicited, and they may be utilized differently depending on context. That is, identifying the potential elements to

include in test charters is the first step needed. To reduce the threat multiple interviews have been used. Using a systematic approach to data analysis (thematic analysis) also aids in reducing this threat.

4 Results

RQ1: What are the factors influencing the design of test charters? Based on interviews with test practitioners, 30 different factors have been identified (see Table 2). The table provides the name of the factors as well as a short description of what the factor means.

We categorized the factors and identified the following emerging categories, namely:

- *Customer and requirements factors:* These factors characterize the customer and their requirements. They include: F01: Client Requirements, F10: Business Usecase, F15: Quality requirements, F27: Client location, and F30: User Journey Map.
- *Process factors:* Process factors characterize the context of the testing in regard to the development process. They include: F21: Process Maturity Level and F25: SDLC Phase.
- *Product factors:* Product factors describe the attributes of the product under test, they include: "F08: Functional flows, F09: Product Purpose, F14: Product Characteristics, F19: General Software design, F20: System Architecture, F22: Product Design Effects, and F28: Heterogeneous Dimensions.
- *Project management factors:* These factors concern the planning and leadership aspects of the project in which the testing takes place. They include: F05: Timeframe, F06: Project Purpose, F12: Effort estimation, F17: Test Team Communication, F18: Project Plan, and F29: Project Revenue.
- *Testing:* Testing factors include contextual information relevant for the planning, design and execution of the tests. They include: F02: Test Strategy, F03: Knowledge of Previous Bugs, F04: Risk Areas, F07: Test Function Complexity, F11: Test Equipment Availability, F13: Test Planning Checklist, F16: Test coverage areas, F23: Feedback and Consolidation, F24: Session Notes, and F26: Tester.

RQ2: What do practitioners include in their test charters? The interviews revealed 35 different contents that may be included in a test charter. Table 3 states the content types and their descriptions.

Similar to the factors we categorized the contents as well. Seven categories have been identified, namely testing scope, testing goals, test management, infrastructure, historical information, product-related information, and constraints, risks and issues.

- *Testing scope:* The testing scope describes what to focus the testing on, be it the parts of the system or the level of the testing. It may also describe what not to focus on and set the priorities. It includes: C02: Test Focus, C03: Test Level, C04: Test Techniques, C10: Exit Criteria, C14: Specific Areas of Interest, C19: Priorities, C28: Coverage, and C33: Omitted Things.

Table 2. Factors influencing test charter design

Charter influence factors	Description
F01: Client requirements	Requirements elicited from clients
F02: Test strategy	Set of ideas that guide the test plan
F03: Knowledge of previous bugs	Knowledge regarding system related bugs that occurred in the past
F04: Risk areas	Results of product risk analysis
F05: Time-frame	Time needed for test mission execution, time constraints
F06: Project purpose	Purpose of the project
F07: Test function complexity	Complexity of the tested functions
F08: Functional flows	Flow of data and functions
F09: Product purpose	Principle goal(s) of the product
F10: Business use-case	Business use-case for the system
F11: Test equipment availability	Accessibility to tools and equipment needed for the software tests
F12: Effort estimation	Effort needed to carry out the test mission
F13: Test planning checklist	Testing heuristics appointed for the particular test charter
F14: Product characteristics	Features of the product
F15: Quality requirements	Quality requirements of the product
F16: Test coverage areas	Parts of the system to be tested
F17: Test team communication	Means of communication between the testing team members
F18: Project plan	Plan for the project prior to its execution
F19: General software design	Design of the system software
F20: System architecture	Structure, interfaces and platforms of the system
F21: Process maturity level	Maturity of the process (e.g. CMMI levels)
F22: Product design effects	Impact of product design and features on other modules
F23: Feedback and consolidation	Feedback and consolidation of the test plan based on the comments of previous testers and clients
F24: Session notes	Notes filled during previous test sessions
F25: SDLC phase	Phase involved in the system development life-cycle
F26: Tester	Testers and their experience level
F27: Client location	Location of the client, local or global
F28: System heterogeneity	Differences between interacting systems (different programming languages, platforms, system configuration)
F29: Project revenue	Business returns for project
F30: User journey map	User interaction with the product over time

Table 3. Contents of test charters

Content type	Description
C01: Test setup	Description of the test environment
C02: Test focus	Part of the system to be tested
C03: Test level	Unit, Function, System test, etc.
C04: Test techniques	Test techniques used to carry out the tests
C05: Risks	Product risk analysis
C06: Bugs found	Bugs found previously
C07: Purpose	Motivation why the test is being carried out
C08: System definition	Type of system (e.g. simple/ complex)
C09: Client requirements	Requirements specification of the client
C10: Exit criteria	Defines the "done" criteria for the test
C11: Limitations	It tells of what the product must never do, e.g. data sent as plain text is strictly forbidden
C12: Test logs	Test logs to record the session results
C13: Data and functional flows	Data and work flow among components
C14: Specific areas of interest	Where to put extra focus on during the testing
C15: Issues	Charter specific issues or concerns to be investigated
C16: Compatibility issues	Hardware and software compatibility and interoperability issues
C17: Current open questions	Existing questions that refer to the known unknowns
C18: Information sources	Documents and guidelines that hold information regarding the features, functions and systems being tested
C19: Priorities	Determines what the tester spends most and least time on
C20: Quality characteristics	Quality objectives for the project
C21: Test results location	Test results location for developers to verify
C22: Mission statement	One liner describing the mission of the test charter
C23: Existing tools	Existing software testing tools that would aid the tests
C24: Target	What is to be achieved by each test
C25: Reporting	Test session notes
C26: Models and visualizations	People, mind maps, pictures related to the function to be tested
C27: General fault	Test related failure patterns of the past
C28: Coverage	Charter's boundary in relation to what it is supposed to cover
C29: Engineering standards	Regulations, rules and standards used, if any
C30: Oracles	Expected behavior of the system (either based on requirements or a person)
C31: Logistics	How and when resources are used to execute the test strategy, e.g. how people in projects are coordinated and assigned to testing tasks.
C32: Stakeholders	Stakeholders of the project and how their conflicting interests would be handled
C33: Omitted things	Specifies what will not be tested
C34: Difficulties	The biggest challenges for the test project
C35: System architecture	Structure, interfaces and platforms concerning the system, and its impact on system integration

- *Testing goals:* The testing goals set the mission and purpose of the test session. They include: C07: Purpose, C22: Mission Statement, and C24: Target.
- *Test management:* Test management is concerned with the planning, resource management, and the definition of how to record the tests. Test management includes: C12: Test Logs, C18: Information Sources, C21: Test Results Location, C25: Reporting, C26: Models and Visualizations, C31: Logistics, C32: Stakeholders, and C34: Difficulties.
- *Infrastructure:* Infrastructure comprises of tools and setups needed to conduct the testing. It includes: C01: Test Setup and C23: Existing Tools.
- *Historical information:* As exploratory testing focuses on learning, past information may be of importance. Thus, the historical information includes: C06: Bugs Found, C16: Compatibility Issues, C17: Current Open Questions, and C27: General Fault.
- *Product-related information:* Here contextual product information is captured, including: C08: System Definition, C13: Data and Functional Flows, and C35: System Architecture.
- *Constraints, risks and issues:* Constraints, risks and issues to testing comprise of the items: C05: Risks, C15: Issues, and C29: Engineering Standards.

5 Conclusion

In this study two checklists for test charter design were developed. The checklists were based on nine interviews. The interviews were utilized to gather a checklist for factors influencing test charter design and one to describe the possible contents of test charters. Overall, 30 factors and 35 content types have been identified and categorized.

The factors may be used in a similar manner and should be used to question the design choices of the test charter. For example:

- Should the test focus of the charter be influenced by previous bugs (F03)? How/why?
- Are the product's goals (F09) reflected in the charter?
- Is it possible to achieve the test charter mission in the given time for the test session (F12)?
- etc.

With regard to the content a wide range of possible contents to be included have been presented. For example, only stating the testing goals (C22) provides much room for exploration, while adding the techniques to be used (C04) may constrain the tester. Thus, the more information is included in the test charter the exploration space is reduced. Thus, when deciding what to include from the checklist (Table 3) the possibility to explore should be taken into consideration.

In future work we need to empirically understand (a) which are the most influential factors and how they affect the test charter design, and (b) which of the identified contents should be included to make exploratory testing effective and efficient.

References

1. Afzal, W., Ghazi, A.N., Itkonen, J., Torkar, R., Andrews, A., Bhatti, K.: An experiment on the effectiveness and efficiency of exploratory testing. Empirical Softw. Eng. **20**(3), 844–878 (2015)
2. Bach, J.: Session-based test management. Softw. Testing Qual. Eng. Mag. **2**(6) (2000)
3. Bach, J.: Exploratory testing explained (2003)
4. Bach, J., Bolton, M.: Rapid software testing. Version (1.3. 2) (2007). www.satisficc.com
5. Christ, R.E.: Review and analysis of color coding research for visual displays. Hum. Factors J. Hum. Factors Ergonomics Soc. **17**(6), 542–570 (1975)
6. Ghazi, A.N.: Testing of heterogeneous systems. Blekinge Inst. Technol. Licentiate Dissertion Ser. **2014**(03), 1–153 (2014)
7. Ghazi, A.N., Petersen, K., Börstler, J.: Heterogeneous systems testing techniques: an exploratory survey. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2015. LNBIP, vol. 200, pp. 67–85. Springer, Cham (2015). doi:10.1007/978-3-319-13251-8_5
8. Hendrickson, E.: Explore it! The Pragmatic Programmers (2014)
9. Itkonen, J., et al.: Empirical studies on exploratory software testing (2011)
10. Itkonen, J., Mäntylä, M.V.: Are test cases needed? Replicated comparison between exploratory and test-case-based software testing. Empirical Softw. Eng. **19**(2), 303–342 (2014)
11. Itkonen, J., Rautiainen, K.: Exploratory testing: a multiple case study. In: 2005 International Symposium on Empirical Software Engineering, p. 10. IEEE (2005)
12. Kaner, C., Bach, J., Pettichord, B.: Lessons Learned in Software Testing. Wiley, New York (2008)
13. Pfahl, D., Yin, H., Mäntylä, M.V., Münch, J., et al.: How is exploratory testing used? In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2014 (2014)

Discovering Software Process Deviations Using Visualizations

Anna-Liisa Mattila[1(✉)], Kari Systä[1], Outi Sievi-Korte[1], Marko Leppänen[1], and Tommi Mikkonen[2]

[1] Tampere University of Technology, Tampere, Finland
{anna-liisa.mattila,kari.systa,outi.sievi-korte,marko.leppanen}@tut.fi
[2] University of Helsinki, Helsinki, Finland
tommi.mikkonen@helsinki.fi

Abstract. Modern software development is supported by a rich set of tools that accumulate data from the software process automatically. That data can be used for understanding and improving software processes without any manual data collection. In this paper we introduce an industrial case where data visualization of issue management system was used to investigate software projects. The results of the study show that visualization of issue management system data can really reveal deviations between planned process and executed process.

Keywords: Software visualization · Mining software repositories

1 Introduction

Various business information systems are focal for corporate management, and often companies utilize metrics as critical success indicators for their business [1]. So, in management of any process, both access to valid process data and the ability to understand the meaning of the data are essential.

Building automated data collection frameworks requires time and effort and is an investment for the company [2], but collecting data manually from the employees is a tedious and error prone effort. Fortunately in software development the effort required to access the data can be reduced significantly as many tools, such as version control and issue management systems, already automatically collect some data [3]. Thus, utilization of this ready-at-hand data could make process analysis more feasible for software companies.

Raw data items or numbers can rarely illuminate the analyst. Therefore, visualization methods are used to get a better overall picture of the organization and its business. When visualizations are available, various stakeholders can enjoy improved transparency to the actual status and react to possible issues faster. The usefulness of these visualizations is not limited to managers only, as everybody can benefit from good visualizations of the progress and properties of the project. This follows the spirit of Andon boards that are used to notify

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 259–266, 2017.
DOI: 10.1007/978-3-319-57633-6_18

management, maintenance, and other workers of a quality or process problems in the Toyota Production System [4].

Many kinds of visualizations are used to show different aspects of software engineering process. Standard visualisation methods in project planning, such as *Gantt charts* [5] and Scrum *burndown charts* [6] can be used as well as workflow visualizations such as *Kanban board* [7,8]. The current state of the project can be communicated to the developer team using *radiators* and *dashboards* [8,9]. When in software process management and improvement, it is important to know what happened in the past and for this purpose various timeline based visualizations have been developed [10,11]. The idea in these methods is to show what happened in the past based on data. This kind of visualizations can be used as a tool in retrospective meetings [10], and during the development to spot abnormalities in the process [11].

In this paper we present experiences on visualizing data from software repositories. We explore the software process in two industrial projects. The paramount goal of our work is to help stakeholders, especially project managers, to observe the execution of software process to find deviations from the planned process, and to detect possible problems in the projects.

2 Research Process

The main research questions of the study are: (1) *Can we show deviations from the assumed software process by visualizing data gathered from software repositories?* (2) *Is the visualization of project data helpful for keeping track of the projects?* To answer these questions, we decided to study software projects where we could access the data starting from the beginning of the project. Issue management system was chosen as our data source as it is used for managing and reporting the software projects. The research process is presented in detail in Fig. 1.

Selected Cases. The cases studied are two industrial projects of a Finland-based multinational large-sized company involved in software R&D. We selected the company based on their interest towards the research. The company representatives selected the studied projects with the following constraints: suitable

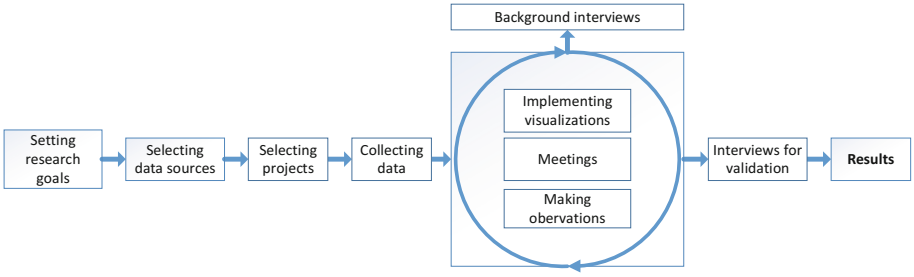

Fig. 1. The research process.

data are available from the beginning of the project, the project is currently in the development phase, and selected projects are comparable with each other.

Both of the studied projects are sub-projects of a larger software entity. In this paper, we refer to the projects as project A and project B. In project A, a software platform is developed whereas in project B a user interface for the software is developed. Both projects have 5–10 team members; the team composition varies based on the current need. Most of the team members are developers, but in team B there are also dedicated persons for testing and user experience design.

The projects use JIRA[1] for issue tracking. The guidelines for using JIRA are the same in both projects. The projects follow the same software development process, namely Scrumban [12], which is a hybrid of Scrum and Kanban. As the projects have uniform practices and processes, we assume that the project data are comparable between the projects.

The data collection period was from the start of the projects till the beginning of January 2015. The projects had started in 2013 – project A in May and B in August. The data sets we used were anonymized by the company representatives. The data were delivered in text format and contained only the information necessary for visualization and analysis – for example person names, JIRA comments or issue names were not visible to us.

Participants. We had eight participants from the case company. A manager of the larger project entity which the studied projects are part of (P1), a person responsible of the realization of agile ways of working in both projects and who was also a former developer in project B (P2), three developers from project A (P3, P4, P5), and three developers from project B (P6, P7, P8).

Four researchers participated to the research by studying the project data, developing the visualization tool, and participating the meetings with the case company.

The visualization tool. To empirically examine the relationship between project data and the perceived state of the project we built a software visualization tool. We chose to utilize timeline as the visualization format because it enables us to easily explore how projects evolved over time and it is used for similar purposes in other studies as well [11,13]. We held several meetings with participants P1, P2, and P3 from the case company to receive feedback from the visualization. The visualization was developed in an iterative manner where we fine-tuned the visualizations based on the received feedback.

The main element of the visualization is to show lifespans and state changes of issues reported in JIRA. Through the lifespan visualization we can observe which issues have been open for a long time and through which states the issue is finally resolved. Detailed figures of the visualization are provided with other additional material on https://github.com/pervcomp/DSPDUV.

Interviews. We held two interview sessions for developers in the projects studied. The first interviews were held at the beginning of the research process to

[1] JIRA – Project management system, https://www.atlassian.com/software/jira.

gain feedback from the initial version of the visualization and get deeper knowledge of the case projects. In the first interview we had three participants: P2, P3, and P6. The second interviews were held four months later to validate our observations made from the visualizations and to gather feedback from the visualization. In the second interview we had seven participants: all the three people interviewed in the first round (P2, P3, and P6) were interviewed again along with two more developers from both projects (P4, P5, P7, and P8). We selected the themes in a fashion that allowed us to (i) validate the assumptions considering the visual observations, and to (ii) reveal the ways of working in the projects as well as possible problems in the team and project.

The interviewing sessions were conducted as follows. Each interview began by discussing the background of the interviewee and continued to the discussion about ways of working, challenging issues in the team work and the project's current status. We showed the visualization to the interviewee during the last part of the interviewing session and asked the interviewee to observe and interpret the visualization. Finally, we discussed the observations made by researchers together with the interviewee to identify potential misinterpretations and to determine causes of the observed issues. Interviewees were also asked to give feedback from the visualization and tell if they thought the visualization is a useful tool for managing projects. The duration of the interviews varied from 30 to 60 min. All interviews were recorded and written notes were made. The interviews were conducted by one researcher.

3 Results

The results are based on studying the visualizations of project data and interviews. The data visualized from the projects were *bug reports*, *epics*, and *stories*. We made assumptions of status and ways of working from the visualizations. The table of assumptions made is available on https://github.com/pervcomp/DSPDUV with the visualizations and other additional material.

Bugs. When comparing the views that show the lifespans and resolution rate of bug reports we noticed that the resolution rate of bug reports was higher in project A than in project B. When interviewing the participants, we found out that in project A bug fixes were prioritized over implementing new features. Prioritizing the bug fixes over new features was not an actual policy of the software process but an agreement within the team thus in project B similar convention was not applied.

The long life spans and increasing amounts of bug reports in project B could be a sign of technical debt or bad architectural decisions but also relate to problems in organizing and reporting work. Based on the interviews we learned that in project B there was technical debt as they had built the project directly on top of their initial prototype, which should have been just a throw away prototype. In project A the initial prototype was discarded. There were also problems in organizing and reporting work in project B.

Epics. The projects differed in how they used epics to plan greater entities. In project B only three epics were closed during the data collection period and all open epics were in their initial state. In project A epics were closed and opened in a more regular pattern. Based on this we could assume that in project B the role of epics in planning was not clear and they were not used systematically, which was also proven to be the case based on the interviews. Based on the process and instructions given to the teams they were supposed to use epics similarly when planning work.

Stories. When looking at the projects individually we assumed that both projects had problems in organizing and reporting work. The assumption was made based on long lifespans and increasing amounts of open stories that were visible in the visualizations. Also the long lifespans and high amount of open bug reports in project B supported this assumption for project B. Based on the interviews there were problems in organizing work. In both projects the product owner's role was not clear. In project A the product owner was not committed to organize the backlog, and in project B there was no product owner.

Usefulness of the visualization. To get feedback on use of the visualization for tracking the projects, we asked if the interviewees considered the visualization useful. All of the interviewees agreed that the visualization we presented is practical in tracking the projects as it shows clearly the issues which have been open for a long time. Most of the interviewees mentioned that the visualization would be especially useful for the project managers but also for them selfs.

4 Threats to Validity

Wohlin et al. [14] state four different categories when considering threats to validity - conclusion validity, internal validity, construct validity and external validity. We will deal with those that are particularly relevant to our study.

Threats to conclusion validity are concerned with issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment. The threats most concerning our study have to do with "fishing" for a particular result and reliability of measures. In the start of the research process we did not expect any results, but were simply curious about what could be learned by visualizing software project data. Thus, all the observations made are purely drawn from what could be seen and without prejudice. Furthermore, the visualizations were interpreted together with company representatives, who would correct false assumptions. Additionally, the interviews were designed to reveal possible overlooked information from the visualizations.

Reliability of measures, in turn, involves the measured data (from the repositories) and verbal information (interviews). The data itself is visualized "*as is*", without any human involvement required in between, so it is valid. The interview questions, in turn, were designed in a way that would allow as open answers as possible and for the interviewer to also perform follow-up questions. Naturally, the wording of the questions is still always critical, and for example a pilot study of the interviews could have been beneficial.

Threats to internal validity concern causality and threats to conclusions about relationships between treatment and outcome. In our experiment the most relevant threat regards *selection*, i.e., selecting the subjects, in our case the interviewees from the company, and how volunteering might affect the results. The interviewees were selected so that we had at least one developer and one person in charge of the process for both projects, who answered questions on both interview rounds, thus ensuring a versatile perspective of the project. For the second round the subjects were selected among developers based on who had the time. Thus there was no direct volunteering, which might affect results, and also selection was not made on any other criteria than having different roles, which should ensure a true view of the project. However, there was also no means to control the backgrounds of interviewees either.

Finally, construct validity concerns generalizing the result of the experiment. The most relevant threat to this study are *hypothesis guessing* and *evaluation apprehension*, both having to do with whether the interviews can be trusted. We argue that evaluation apprehension is not a concern, as all interviewees were willing to discuss the problems in their projects, and did not attempt to hide them from the researcher. As for guessing the hypothesis, we did not show the visualization to the interviewees until in the end of the interview, so all answers were purely given based on the questions.

The final category of threats given by [14] relates to external validity are conditions that limit the ability to generalize the results of the experiment to industrial practice. This does not concern us, as our cases were from the industry, and thus we can argue that the results already reflect industrial practice. However, more research need to be done for generalization of results.

5 Discussion and Conclusions

The first research question we addressed in Sect. 2 was *"Can we show deviations from the assumed software process by visualizing data gathered from software repositories?"*. Using the visualization we could note differences in practices between projects that should have had the same practices. We were also able to interpret from the visualization that there were problems in the software process. We learned that problems related to organizing work shows well in the visualization of issue management data. We also learned that different problems show differently. The problems in planning and reporting are visible in long lifespans of issues as well as different kinds of patterns in creating issues where as technical debt may be visible in bug report lifespans and creation rate.

Our second research question was: *"Is the visualization of project data helpful for keeping track of the projects?"*. Based on the feedback we can conclude that the visualization is a useful tool for project managers. Furthermore, we noticed that the visualization raised questions and interest in participants to discuss about the state of the projects. The visualization creates a good common ground for such discussion as it shows empirical evidence.

We have developed the visualization tool further based on the feedback received from the case company. We have also done first experiments using the

visualizations in teaching software engineering. As a future work we will investigate the use of the tool for other industrial projects as well as for open source projects to validate our findings and evaluate the tool further.

References

1. Menzies, T., Zimmermann, T.: Software analytics: so what? IEEE Softw. **30**(4), 31–37 (2013)
2. Robbes, R., Vidal, R., Bastarrica, M.: Are software analytics efforts worthwhile for small companies? The case of Amisoft. IEEE Softw. **30**(5), 46–53 (2013)
3. Mäkinen, S., Leppänen, M., Kilamo, T., Mattila, A.L., Laukkanen, E., Pagels, M., Männistö, T.: Improving the delivery cycle: a multiple-case study of the toolchains in Finnish software intensive enterprises. Inf. Softw. Technol. **80**, 175–194 (2016)
4. Liker, J.K.: The Toyota Way. Esensi (2004)
5. Gantt, H.: Work, Wages and Profit. The Engineering Magazine (1910)
6. Schwaber, K., Beedle, M.: Agile Software Development with Scrum, 1st edn. Prentice Hall PTR, Upper Saddle River, NJ (2001)
7. Kerzazi, N., Robillard, P.N.: Kanbanize the Release Engineering Process. In: 2013 1st International Workshop on Release Engineering (RELENG), pp. 9–12. IEEE (2013)
8. Paredes, J., Anslow, C., Maurer, F.: Information visualization for agile software development. In: 2014 Second IEEE Working Conference on Software Visualization (VISSOFT), pp. 157–166. IEEE (2014)
9. Baysal, O., Holmes, R., Godfrey, M.W.: Developer dashboards: the need for qualitative analytics. IEEE Softw. **30**(4), 46–52 (2013)
10. Bjarnason, E., Hess, A., Doerr, J., Regnell, B.: Variations on the evidence-based timeline retrospective method: a comparison of two cases. In: 2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 37–44. IEEE (2013)
11. Lehtonen, T., Eloranta, V.P., Leppänen, M., Isohanni, E.: Visualizations as a basis for agile software process improvement. In: 2013 20th Asia-Pacific Software Engineering Conference (APSEC), vol. 1, pp. 495–502. IEEE (2013)
12. Ladas, C.: Scrumban - Essays on Kanban Systems for Lean Software Development. Modus Cooperandi Press, USA (2009)
13. Bjarnason, E., Svensson, R.B., Regnell, B.: Evidence-based timelines for project retrospectives - a method for assessing requirements engineering in context. In: 2012 IEEE Second International Workshop on Empirical Requirements Engineering (EmpiRE), pp. 17–24. IEEE (2012)
14. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Norwell (2000)

Exploring Workflow Mechanisms and Task Allocation Strategies in Agile Software Teams

Zainab Masood(✉), Rashina Hoda, and Kelly Blincoe

SEPTA Research, Department of Electrical and Computer Engineering, The University of Auckland, Auckland, New Zealand
zmas690@aucklanduni.ac.nz, {r.hoda,k.blincoe}@auckland.ac.nz

Abstract. Task allocation is considered an important activity in software project management. However, the process of allocating tasks in agile software development teams has not received much attention in empirical research. Through a pilot study involving mixed open-ended and closed-ended interviews questions with 11 agile software practitioners working within a software development organization in India, we explain the process of task allocation as including three different mechanisms of workflow across teams: team-independent, team-dependent, and hybrid workflow; and five types of task allocation strategies: manager-driven, team-driven, individual-driven, manager-assisted and team-assisted. Knowing these workflow mechanisms and task allocation strategies will help software teams and project managers make more effective decisions around workflow and task allocation.

Keywords: Task allocation · Workflow · Allocation mechanism · Agile software teams · Task allocation strategies

1 Introduction

Successful project completion depends on how well and effectively the project activities are planned and managed throughout [1]. Primary project management activities include managing resources, task allocation, and tracking time and budget in the best possible way [2]. Several studies have researched task allocation in global and distributed software development using traditional or agile methods [3–6]. A limited number of studies have assessed task allocation mechanisms practiced by Free/Libre Open Source Software (FLOSS) development teams; however, they did not cover commercial projects [7]. Overall, task allocation in agile software teams, which are meant to be self-organizing [9, 12], has not be studied.

We conducted a pilot study involving face-to-face interviews with 11 agile practitioners from three teams in a software organization in India. Thematic analysis [8] was performed to derive the different types of workflow mechanisms and task allocation strategies from the interview data. We identified three workflow mechanisms: team-independent, team-dependent, and hybrid workflow. We also identified five types of task allocation strategies: manager-driven, team-driven, individual-driven, manager-assisted and team-assisted. Identifying these mechanisms and strategies helped understand the

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 267–273, 2017.
DOI: 10.1007/978-3-319-57633-6_19

flow and forms in which tasks arrives to the team and the basis on which tasks are classified and allocated.

2 Related Work

In traditional software development, the project manager plays a key role in task allocation and management and overall decision making. With the evolution of agile methods, software teams are meant to be self-organizing with high levels of autonomy, teams empowerment and mutual decision making in their everyday work [10, 12] including project management activities such as task allocation [11, 12]. In practice, however, agile teams are seen to display varying levels of autonomy as they gain experience of functioning in a self-organizing way [11]. How the varying levels of autonomy influence task allocation is not well understood. In particular, it is unclear how work flows to and within the team, how tasks are allocated on an individual level, and what are the different types and autonomy levels of task allocation in agile teams.

The research on task allocation in software teams has been largely dominated by distributed contexts in global software development. Imtiaz et al. in their recent survey-based study identified "functional area of expertise and phase-based" task allocation as the most common way of allocating tasks global software development [5]. Other studies, e.g. [4, 6], explored task allocation in distributed agile software development contexts through literature review and proposed models indicating further studies as a promising area of research. Crowston et al. 2007 [7] demonstrated the possible mechanisms of tasks allocation in community-based Free/Libre Open Source Software (FLOSS) development in self-organized volunteer teams. Their findings support self-assignment as one of the common ways of assigning tasks adopted by FLOSS teams. However, not much has been explored in the literature about task allocation mechanisms outside the FLOSS domain and specifically for commercial software development. Overall, much remains to be understood about how work flows to and within agile teams and how they practice task allocation.

3 Research Method

Our pilot study involved mixed open- and closed-ended interview questions with 11 agile practitioners. The overarching research questions were:

RQ1: How does work flow in agile teams?
RQ2: How does task allocation happen in agile teams?

3.1 Participant Selection and Description

An invitation to participate was sent out to members of the Agile software community of India. The company willing to offer a maximum number of teams and participants was selected. Eleven software practitioners from three agile teams working in this digital technology company were included (one additional participant was later dropped since

they were the sole representative of a fourth team). Participants were experienced software practitioners and were using agile methods, either Scrum or Kanban, including key agile practices such as Daily Team Meetings, Release and Iteration planning, Pair Programming, Review meetings and Retrospectives. Teams were collaborating with off-shored customers or product teams in the USA through Google Hangout, Skype or Webex. The project management tool used by all teams was Jira. Team, project and participants' details are profiled in Table 1.

Table 1. Team, project contexts and participants demographics (TS: Team Size, SP#: Participants; TX: total experience in years; X: agile experience in years; ATL: Assoc.Tech Lead; TL: Tech Lead; SSE: Senior Software Engineer)

Team	TS	Software method	Project Area/ Context	SP#	Role	Age group	TX	AX
T1	10–15	Scrum	Digital Marketing/ Features & Maintenance	SP1	TL	31–35	10–11	6–7
				SP2	SE	21–25	2–2.5	1
				SP3	ATL	26–30	4–5	4–5
				SP4	SE	21–25	2.5	2.5
T2	5–10	Scrum	Analytics/ Features	SP5	TL	36–40	7	7
				SP6	SSE	26–30	4	2
				SP7	TL	31–35	7.5	7.5
T3	15–20	Kanban	Cloud Services/ Migration & Enhancement	SP8	TL	31–35	5.5	5–6
				SP9	ATL	26–30	4	2
				SP10	SSE	21–25	3.5	1
				SP11	ATL	26–30	4.5	2

3.2 Data Collection and Analysis

We conducted face-to-face interviews lasting 30–40 min with each participant using a combination of open- and close-ended questions about their current projects applying agile methods. Initial questions gathered participants' demographical data, details related to the project, team and the agile methods used. Most other questions focused on task allocation process e.g. how, when and from whom the teams receive the tasks and how the tasks are allocated among the teams and the individuals. These were mostly open-ended questions to allow a range of answers, with some choices being given to facilitate the interviewees during the interview.

All the interviews were recorded with detailed notes taken during the interview. Interview data was transcribed and analyzed manually using thematic analysis [8] to derive the common themes, i.e. patterns of workflow mechanisms and task allocation strategies common across the participants. This was led by one of the authors and supported by the other two through careful reviews and discussions.

4 Findings

In answer to RQ1, we identified three distinct workflow mechanisms (illustrated in Fig. 1) that describe how the teams receive the work from the relevant stakeholders: team-independent, team-dependent, and hybrid workflow. Additionally, in answer to RQ2, we found five different task allocation strategies based on how tasks were allocated within the team: manager-driven, team-driven, individual-driven, manager-assisted and team-assisted.

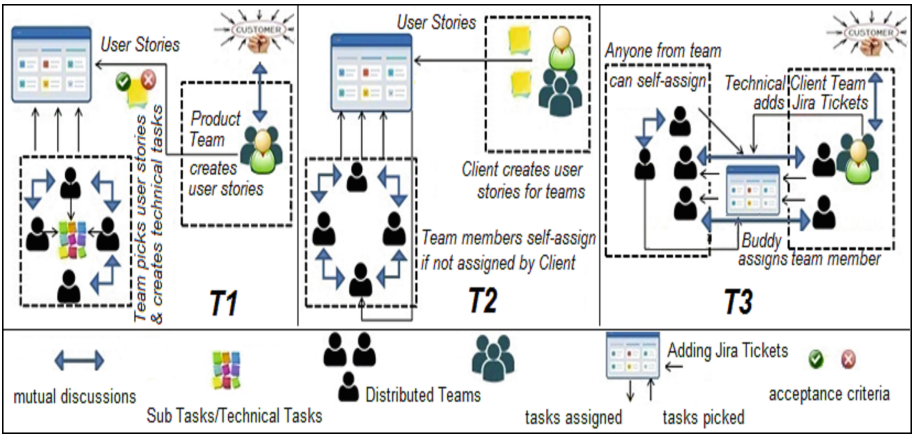

Fig. 1. Teamwise task allocation mechanisms (T1: team independent workflow; T2: team dependent workflow; T3: hybrid workflow)

4.1 Team Workflow Mechanisms

Team Independent Workflow: In this workflow, the tasks are defined irrespective of the team location (US, India). Tasks comes to both teams from Product Owner mostly in form of user stories during sprint planning meeting. Members of all teams individually pick and break user stories into technical tasks. The work allocation is done by volunteering for tasks through mutual discussions. For example, one participant explained:

> *"They[Product Team] bring whole description of the ticket[user story]…Everyone is in sprint planning meeting, every developer I should say and then ticket by ticket we volunteer, they do not assign any name." SP1, Tech Lead.*

Team Dependent Workflow: Client defines the tasks for respective teams (US, India) separately as user stories during fortnightly iteration planning meeting. Before sprint planning meeting, the team (T2) go through their stories and team members allocate the tasks either individually or through mutual consensus. SP7 described the workflow as follows:

> *"Client creates user stories then one day before sprint planning we [T2] go through stories which are meant for India team and we pick whatever we want to do." SP7, Tech Lead*

Hybrid Workflow: Team T3 was seen to follow multiple workflow mechanisms, but tasks are typically allocated during a monthly release from the USA technical team, who collaborates with the client. For a few members of the team, the USA team creates Jira tickets with a set priority and complexity level. As specified by SP9:

"Now that teams have been divided so they have to work according to the tasks that are assigned to those particular teams only so it's not like that X team can work on team Y cards." SP9, Associate. Tech Lead

For other team members, work comes as features with a defined priority and release date from the USA team. These features are selected by the Tech Lead in USA, who breaks them into tasks and sub-tasks and allocates them to their 'buddy' programmer in India.

"So the client decides the criticality and to which release these [cards] will belong so once the lead has decided that then pair [buddy] can pick up." SP10, Sr. Software Engineer

4.2 Task Allocation Strategies

In **Manager-driven Task Allocation**, the manager/client/technical-lead allocates tasks to the team members with names against the tasks as stated by a participant, where the 'buddy' was a senior Tech Lead in the USA:

"Nowadays I am given task by my buddy." SP11, Assoc. Tech Lead

In **Team-driven Task Allocation**, the team discusses and mutually decides who will perform which task, for example:

"We are three people [in the team] so mutually decide who will do [what]." SP6, Sr. Software Engineer

In **Individual-driven Task Allocation**, tasks are self-assigned i.e. selected and managed individually without any assistance from others. For example, SP4 quoted practicing self-driven allocation:

"Mostly we volunteer it." SP4, Software Engineer

In **Manager-assisted Task Allocation**, tasks are allocated with some assistance from the manager/client/technical-lead to the team members. As a technical lead, SP1 mentioned assisting team member with picking tasks:

"'Hey [name] you should do this [task]', let say he is new and he doesn't know [so] I help him, 'pick this one because this is lesser complex'." SP1, Tech Lead

In **Team-assisted Task Allocation,** every team member self-assigns tasks with some assistance from fellow team members, for example:

"So any of the pair[s] can pick up [a task]." SP10, Sr. Software Engineer

5 Discussion

We identified five task allocation strategies. Four of these strategies involve either the team as a whole or the manager/client in the task allocation process, making it evident

that the task allocation mostly takes place through assistance or mutual discussions. In other words, task allocation strategies rely on collective decision making. A prior study [13] has shown that agile teams make effective decisions collectively compared to individual decisions, benefitting from collective knowledge and experiences.

Another aspect is that for high priority tasks all mechanisms agree on a common allocation method, i.e. tasks are directly allocated to a skilled and experienced person, an aspect supported by previous research [7].

Our study supports the different levels of autonomy evident on agile teams [11] as we found evidence of varying management approaches: manager-driven, manager-assisted and team-driven. Additionally, we also identified a new level: individual-driven task allocation.

With respect to the effectiveness of their current strategy, all the teams reported being satisfied, but some participants shared a few challenges, e.g. vagueness or missing clarity on tasks was the most commonly reported challenge. One participant (SP10) mentioned that with their current task allocation strategy (Team-assisted), work at times is not evenly distributed. Another participant (SP1) revealed drawbacks of picking tasks remotely. Since their client and the USA team are co-located they were perceived to have an advantage in picking tasks over SP1's India team. However, these challenges are not directly related to task allocation, rather, they are also linked to requirements clarity issues and the distributed nature of the team. This illustrates that task allocation is impacted by many factors.

This research study can serve as a basis for exploring other task allocation strategies and internal workflow mechanisms of agile teams. This pilot study included only 11 interviews from the same organization which signifies a limited dataset and context. Our larger study will interview more software teams and individuals representing different roles. Future work can focus on evaluating the effectiveness of the strategies.

6 Conclusion

This study presents a preliminary understanding of workflow mechanisms and task allocation strategies in agile teams. Clients typically provide high-level requirements as features or user stories to the agile teams who then break them down into technical tasks or sub-tasks by themselves or directly allocate them to team members. The team members then select them individually or through mutual discussions within the team. Allocation of tasks usually takes place during iteration or release planning. The findings of this study demonstrate that there are multiple types of task allocation strategies practiced by agile teams based on what suits the completion of the work in the best possible way. A common mechanism found in a majority of the teams is that if the priority of the task is high, then the task is allocated to the most suitable person directly. Also on average, the practice most commonly followed is that the team members collaborate with each other and with their manager/client when assistance is needed.

References

1. Pinto, J.K., Slevin, D.P.: Critical success factors across the project life cycle. Proj. Manag. J. **19**(3), 67–75 (1988)
2. Hoda, R., Noble, J., Marshall, S.: Agile project management. In: New Zealand Computer Science Research Student Conference, vol. 6, pp. 218–221 (2008)
3. Lamersdorf, A., Munch, J., Rombach, D.: A survey on the state of the practice in distributed software development: criteria for task allocation. In: 2009 Fourth IEEE International Conference on Global Software Engineering, ICGSE 2009, pp. 41–50 (2009)
4. Filho, M.S., Pinheiro, P.R., Albuquerque, A.B.: Task allocation approaches in distributed agile software development: a quasi-systematic review. In: Silhavy, R., Senkerik, R., Oplatkova, Z.K., Prokopova, Z., Silhavy, P. (eds.) Software Engineering in Intelligent Systems. AISC, vol. 349, pp. 243–252. Springer, Cham (2015). doi: 10.1007/978-3-319-18473-9_24
5. Imtiaz, S., Ikram, N.: Dynamics of task allocation in global software development. J. Softw. Evol. Process **29**(1) (2016). doi:10.1002/smr.1832
6. Mak, D.K., Kruchten, P.B.: Task coordination in an agile distributed software development environment. In: 2006 Canadian Conference on Electrical and Computer Engineering, CCECE 2006, pp. 606–611. IEEE (2006)
7. Crowston, K., Li, Q., Wei, K., Eseryel, U.Y., Howison, J.: Self-organization of teams for free/libre open source software development. Inf. Softw. Technol. **49**(6), 564–575 (2007)
8. Clarke, V., Braun, V.: Thematic analysis. In: Encyclopedia of Critical Psychology, pp. 1947–1952. Springer, New York (2014)
9. Moe, N.B., Dingsøyr, T., Dybå, T.: Understanding self-organizing teams in agile software development. In: 2008 19th Australian Conference on Software Engineering, ASWEC 2008, pp. 76–85. IEEE (2008)
10. Highsmith, J.: Agile Project Management: Creating Innovative Products. Pearson Education, Upper Saddle River, NJ (2009)
11. Hoda, R., Noble, J.: Becoming agile: a grounded theory of agile transitions in practice. In: IEEE International Conference on Software Engineering (ICSE2017) (2017)
12. Hoda, R., Murugesan, L.K.: Multi-level agile project management challenges: A self-organizing team perspective. J. Syst. Softw. **117**, 245–257 (2016)
13. Drury, M., Conboy, K., Power, K.: Obstacles to decision making in Agile software development teams. J. Syst. Softw. **85**(6), 1239–1254 (2012)

Are Daily Stand-up Meetings Valuable? A Survey of Developers in Software Teams

Viktoria Stray[1(✉)], Nils Brede Moe[2], and Gunnar R. Bergersen[1]

[1] University of Oslo, Gaustadalléen 23B, 0374 Oslo, Norway
{stray, gunnab}@ifi.uio.no
[2] SINTEF, Strindveien 4, 7465 Trondheim, Norway
nils.b.moe@sintef.no

Abstract. The daily stand-up meeting is a widely used practice. However, what is more uncertain is how valuable the practice is to team members. We invited professional developers of a programming forum to a survey and obtained 221 responses. Results show that the daily stand-up meeting was used by 87% of those who employ agile methods. We found that even though the respondents on average were neutral towards the practice, the majority were either positive or negative. Junior developers were most positive and senior developers and members of large teams most negative. We argue that the value of the practice should be evaluated according to the team needs. Further, more work is needed to understand why senior developers do not perceive the meetings as valuable and how to apply the practice successfully in large teams.

Keywords: Daily meetings · Stand up meeting · Daily scrum · Communication · Coordination · Teamwork · Team size · Agile adoption · Agile practices

1 Introduction

Agile methods introduced the daily stand-up meeting (DSM) as a practice to improve communication in software development projects. In Scrum, the meeting is mandatory, time-boxed to 15 min and team members address: (1) what they have done the previous work day, (2) what they will do today and (3) what obstacles are preventing them from making progress [1]. Scrum recommends that the DSM should not be used for discussing solutions to obstacles raised. However, empirical studies have found that spending time in the short meeting on discussing and solving problems is valuable [2, 3].

DSMs are task oriented, generally unrecorded, and members gather to focus on a narrow organizational goal. According to Boden [4], such meetings can be characterized as informal. The practice gives team members an overview of what other team members are doing and is therefore an important mechanism to increase information sharing and team awareness [5]. The meeting is often conducted standing up to keep it brief and avoid lengthy discussions, hence the term "stand-up meeting". The practice is also called "frequent short meetings" [6], "morning roll call" [7], and "daily Scrum meeting" [1]. The DSM is an important practice for agile teams because it helps the team in monitoring and managing its performance, which is important for the team to

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 274–281, 2017.
DOI: 10.1007/978-3-319-57633-6_20

self-manage [8]. Further, such meetings improve access to information that foster employee empowerment [9].

While DSM is a relatively straightforward practice to adopt, it is challenging to implement it successfully. Challenges include finding a suitable time of day, keeping the time limit and whether it should be held daily and standing up [10]. We have previously found DSMs to last 63% longer when team members sit rather than stand during the meeting [5]. Another challenge is members reporting their status to the team leader, resulting in team members not paying attention to each other [8].

Although the DSM is one of the most popular agile practices [11, 12] and the only daily team-based coordination mechanism, the practice has received little research attention. Further, because meeting satisfaction is part of overall job satisfaction [13], it is important to understand what makes this meeting valuable for team members. In a recent, qualitative study of thirteen teams (in Norway, Poland, UK and Malaysia) we found that the attitudes towards DSMs were slightly more positive than negative [5]. However, the level of satisfaction varied within the teams. Therefore, to understand how to implement DSM, it is important to explore satisfaction with the practice on an individual level. This leads to the following research question: "*What are the characteristics of developers perceiving the daily stand-up meeting to be valuable compared to those who do not?*"

Our work also answers a call for more empirical studies on the adoption rate of agile software development methods [14].

2 Method

The target population for this study was professional software developers. Accordingly, we posted the survey on Reddit, which is a social media website that allows scientists to recruit a targeted population [15]. We chose two programming-related subreddits (subforums) that provide news and discussions about computer programming (r/programming, 710 000 subscribers) and web development (r/webdev, 130 000 subscribers). The survey was administered using the Qualtrics software which prevents the survey to be completed more than once by the same respondent. Participation was voluntary. Further, no compensation was offered, which increase the quality of the data because the incentive to cheat is largely reduced [15]. The survey (available from https://figshare.com/s/a10006dd8f5f26141511) took about three minutes to complete.

We received 316 responses, of which 243 contained data that could be analyzed. Because we were interested in the opinions of software professionals currently working in teams, we removed students and those not working or not working in a team. In total, 221 responses were used for the reported results. The majority of responses were from the programming forum (n = 165). Nearly all the respondents were male (96.8%) with a mean age of 31 years (n = 204, sd = 6.86). Among the respondents who answered whether their team was distributed (n = 168), about two-thirds of the respondents (63.1%) reported working in co-located teams, whereas the remaining had team members distributed across sites (36.9%).

All Likert questions used a five-point scale. All nominal-scale questions were presented with a randomized order of categories because the order of response alternatives can influence results [16]. Some questions were not compulsory, which resulted in missing data for the reported variables. Analyses are reported using the R statistical software [17]. To err on the side of caution, we use two-tailed analysis and chose non-parametric statistical tests. The one-sample Wilcoxon test is used to check for statistically significant differences in distributions. When comparing frequencies between two dichotomous variables that contain count data (i.e., frequencies) we used Fisher's exact test which reports the odds ratio (OR) effect size.

3 Results

In our study, the average number of DSMs conducted per week was five, which suggests that it is a daily meeting. Table 1 shows descriptive statistics. We found no difference regarding the frequency of meetings when it comes to being part of a distributed team or not, or to team size. Among all the respondents, one-third reported to work in teams with two to five members, one-third in teams with six to eight members and one-third in teams with nine or more members. We found a difference of 52% points with an odds ratio of 12.3 for agile teams using DSMs over non-agile teams ($p < 0.001$, 95% confidence interval for OR: 5.4–29.5).

Table 1. Descriptive statistics

	Unit	n	Mean (M)	sd	median
Meetings	Frequency per day, DSMs included	166	1.8	1.2	2
Time in meetings	Hours per day; DSMs included	187	1.4	2.0	1
Time programming	Hours per day	196	6.2	2.3	7
Team size	Members including self	168	7.3	3.9	6
DSM valuable	Likert: Negative (1)–Positive (5)	149	3.0	1.2	3
Programming skill self	Likert: Novice (1)–Expert (5)	177	3.7	0.8	4
Programming skill peers	Likert: Novice (1)–Expert (5)	177	3.6	0.8	4

Overall, 70.6% report that they attend DSMs (n = 221). Those who attend and those who do not attend DSMs spend the same amount of hours in meetings (DSMs included) and report similar values for programming skills. However, those who attend DSMs spend almost one hour more each workday on programming ($p = 0.046$, attend: M = 6.5 h, sd = 2.1; not attend: M = 5.6 h, sd = 2.7). Further, those who attend DSMs work in larger teams ($p = 0.03$, attend: M = 6.90 members, sd = 4.7; M = 7.44 members, sd = 3.52); the median difference was 2 team members. Moreover, the practice is regarded as more valuable by those who attend DSMs than those who do not ($p = 0.002$, attend: M = 3.1, n = 123, not attend: M = 2.3, n = 29).

We now report on only those respondents who attend DSMs. While the mean perceived value by these respondents towards the practice was neutral (3.1), only 18.7% chose this middle category on the Likert scale. Most respondents were either positive (44.7%) or negative (36.6%). We coded responses of 4 and 5 as "positive", responses of 1 and 2 as "negative", and removed those who responded neutral to be able to better understand differences between these two groups. We found no relation between working in a co-located or distributed team and the perceived value of DSM. However, those positive were significantly younger ($p = 0.008$, positive: $M = 29.6$ years, $n = 49$; negative: $M = 33.5$ years, $n = 42$).

Figure 1 shows the characteristics of the respondents who attend DSMs according to whether they are positive (green, $n = 55$) or negative (red, $n = 45$) towards the meetings they attend. The left part of the figure shows that those positive and negative towards their DSMs spent about the same amount of time in meetings: 83 min for those positive versus 77 min for those negative. However, there was a significant difference in meeting frequency; those positive attended fewer meetings per day (DSMs included) than those negative. Those positive towards DSM report somewhat more time spent on programming per day (24 min) than those negative. Being positive towards DSMs was, to some extent, associated with working in smaller teams. As a post hoc analysis, we investigated differences in attitudes further and found that teams with 12 or more members were most strongly associated with negative attitudes towards DSMs.

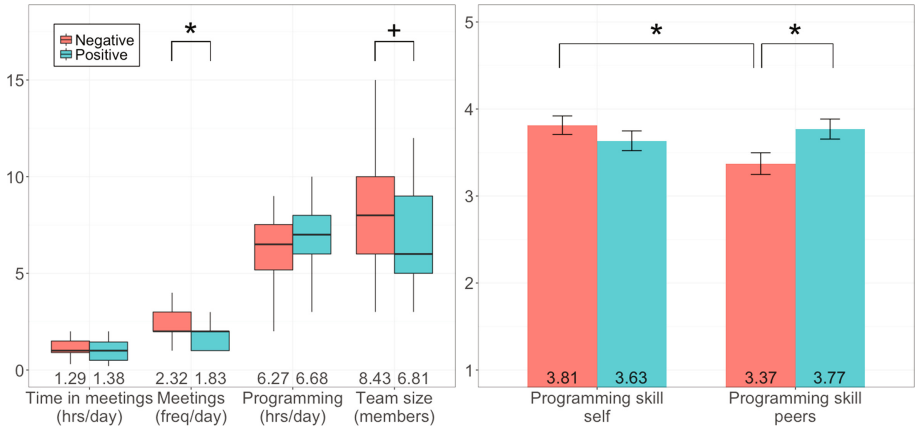

Fig. 1. Characteristics of those positive and negative towards their DSMs being valuable. Significant differences are shown at the top and means are shown at the bottom of the figure (as numbers). Outliers are omitted. Error bars represent the standard errors of the mean. + is $p < 0.10$ and * is $p < 0.05$ (two-sided). (Color figure online)

The right part of Fig. 1 shows a minor difference between how those positive and negative towards DSM rated their own programming skills. However, those positive rated the programming skills of their peers as significantly higher compared to how the

negative rated their peers. Further, those negative also rated their own skills as significantly higher than that of their peers, whereas it was, to some extent, the opposite for those positive.

4 Discussion

The main explanation of the widespread use of DSM (70,6%) is the high adoption rate of agile development methods among our respondents. Table 2 shows that the agile adoption rate in our survey is higher than what was found by Rodríguez et al. [11]. Rodríguez et al. did not report the adoption rate of DSM but concluded that it was one of the most widely used practices. The last column in Table 2 shows the adoption rate of DSM in both agile and non-agile teams in our study. VersionOne [12] report the DSM to be the most employed agile practice with an adoption rate of 83%. VersionOne's sample mostly consisted of agile practitioners. In comparison, our DSM adoption rate among those using agile or agile in combination with Lean was 87.3%. Our results indicate that the practice has spread to companies not using agile methods because 35.4% of the respondents who work in non-agile teams also report using DSM. Thus, being agile implies that DSMs are used to a large extent which supports that DSM is a practice that distinguishes agile from non-agile teams [17].

Table 2. Usage rates of agile methods and DSM adoption according to development method

Development method	Agile adoption in our survey	Agile adoption in Rodríguez et al. [11]	DSM adoption in our survey
Agile and/or Lean	**73.6%**	**57.8%**	87.3%
Only agile	*54.9%*	*33.6%*	*89.0%*
Agile and Lean	*18.7%*	*21.6%*	*82.4%*
Only Lean	*0.0%*	*2.7%*	*0.0%*
Neither agile nor Lean	**26.4%**	**42.2%**	35.4%
Total	**100.0%**	**100.0%**	

For our research question, "What are the characteristics of developers perceiving the daily stand-up meeting to be valuable compared to those who do not?", our results indicate that those positive towards DSM are more junior developers. This inference is supported by age, how they rate their own programming skills and their self-reported skills compared to the perceived skill of their peers. Those positive towards DSM also participate in fewer meetings than those negative. The same variables also indicate that those negative towards DSM are more senior developers. One explanation for why a senior developer regards DSM as less valuable is because seniors may already know what goes on in the team and does not get any new information in the meeting. The personal gain from the meeting is thus reduced. Moreover, being able to have quick problem-solving discussions in the DSM make developers perceive the DSM as more valuable [5]. Senior developers often work on more complex tasks, and it might be that high complexity problems are seldom discussed at the meeting because they require too

much time. It is more likely that the problems a junior developer encounter are more easily solved in a DSM.

A second explanation is that senior developers attend more meetings than junior developers. The DSM then becomes an additional daily interruption, which reduces the satisfaction with such meetings. Perceiving the meeting to have too high frequency negatively affects the attitude towards DSMs [5]. Moreover, meeting load affects employees well-being [18] so companies should be sensitive to the number of meetings the developers have to attend. While it has been claimed that DSMs eliminate the need for other meetings [1], we found no difference between hours spent in meetings for those who attend or do not attend DSMs.

In a self-managing team, the team goal should be more important than the individual goal, and then a developer should rate the DSM value depending on the team needs. One respondent commented: *"I think some people need the daily stand-up format. So even though I personally don't feel like I need it, I feel it benefits us all to do it because of the different personalities."* Because we do not know the perspective of the respondent we do not know if the respondents are considering the value from an individual or team perspective, or a mixture of the two views.

We found that larger teams are more likely to have DSMs. Paradoxically, the larger the team, the less is the satisfaction with DSM. Large teams using DSMs should therefore pay special attention to improving the quality of these meetings. In particular, developers were negative towards DSM when teams consisted of 12 or more team members. Previous research also found a negative correlation between the number of meeting participants and the attitude towards DSM [5].

The main limitation of this study concerns the representativeness. Although the distribution of self-reported programming skill in this study is nearly identical to our earlier study of programming skill of developers [19], the sample and target population may differ. For example, it is possible that only those who knew or had a strong (polarized) opinion of DSMs responded to the survey. This may bias results in favor of more respondents reporting using DSM and more variability in opinions than is actually present in the target population. Another potential concern is that we had subjects from two different programming forums, but the results we report still hold when analyzing the data from the two forums separately.

5 Conclusion and Future Work

The present study investigated the perceived value of daily stand-up meetings (DSMs) and reports the adoption rate of the practice. Among those who use agile methods, the majority conducts DSMs. Although it is a common practice, the perceived value of the meeting varies with junior developers being more positive and senior developers more negative towards the DSMs they attend. A possible explanation is that junior developers receive more relevant information and assistance in solving problems during the meeting. In contrast, senior developers often work with larger, more complex and independent tasks that are more difficult to share with team members on a daily basis. Agile teams are expected to be self-managed, and the need of the team should be more important than that of the individual. The value of the practice should, therefore,

be evaluated according to the team needs. Consequently, senior developers should be made more aware that DSMs are beneficial for the junior developers as well as the team as a whole. Another result was that developers in larger teams see the meeting as less valuable than developers in smaller teams. Because the work in large teams is often loosely coupled, the information shared during the meeting may be less relevant for the individuals. Consequently, large teams in particular need to invest resources in improving the practice to make it valuable.

Future work should investigate other criteria of the participants, such as role and domain. Because the perceived value of meetings affects job satisfaction, there is a need to understand why senior developers and large teams do not perceive the meeting as more valuable. The DSM is a widely adopted practice and is an important mechanism for information sharing and team awareness, thus, how to apply the practice successfully in large teams should also be studied.

Acknowledgments. We are grateful to the survey respondents and to the reviewers. This work was supported by the Smiglo project, which is partly funded by the Research Council of Norway under the grant 235359/O30.

References

1. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice Hall, Upper Saddle River (2002)
2. Stray, V.G., Moe, N.B., Aurum, A.: Investigating daily team meetings in agile software projects. In: The 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2012), Cesme, Turkey, 17 August 2012
3. Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., Still, J.: The impact of agile practices on communication in software development. Empirical Softw. Eng. **13**, 303–337 (2008)
4. Boden, D.: The Business of Talk: Organizations in Action. Polity Press, Cambridge (1994)
5. Stray, V., Sjøberg, D., Dybå, T.: The daily stand-up meeting: a grounded theory study. J. Syst. Softw. **114**, 101–124 (2016)
6. Rising, L.: Agile meetings. STQE, pp. 42–46 (2002)
7. Anderson, D.J.: Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results. Prentice Hall, Upper Saddle River (2003)
8. 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**, 480–491 (2010)
9. Allen, J.A., Lehmann-Willenbrock, N., Sands, S.J.: Meetings as a positive boost? How and when meeting satisfaction impacts employee empowerment. J. Bus. Res. **69**, 1–8 (2016)
10. Stray, V.G., Lindsjørn, Y., Sjøberg, D.: Obstacles to efficient daily meetings in agile development projects: a case study. In: The ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2013), Baltimore, USA, 13 September 2013
11. Rodríguez, P., Markkula, J., Oivo, M., Turula, K.: Survey on Agile and Lean Usage in Finnish Software Industry. ACM, New York (2012)
12. VersionOne: VersionOne 10th Annual State of Agile Report. https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf

13. Rogelberg, S.G., Allen, J.A., Shanock, L., Scott, C., Shuffler, M.: Employee satisfaction with meetings: a contemporary facet of job satisfaction. Hum. Resour. Manag. **49**, 149–172 (2010)
14. Stavru, S.: A critical examination of recent industrial surveys on agile method usage. J. Syst. Softw. **94**, 87–97 (2014)
15. Shatz, I.: Fast, free, and targeted: reddit as a source for recruiting participants online. Soc. Sci. Comput. Rev., pp. 1–13 (2016)
16. Schwarz, N., Hippler, H.J.: Response alternatives: the impact of their choice and presentation order (1991)
17. Murphy, B., Bird, C., Zimmermann, T., Williams, L.: Have agile techniques been the silver bullet for software development at Microsoft? In: The Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2013), Baltimore, USA, 7 July 2013
18. Luong, A., Rogelberg, S.G.: Meetings and more meetings: the relationship between meeting load and the daily well-being of employees. Group Dyn. Theor. Res. Pract. **9**, 58–67 (2005)
19. Bergersen, G.R., Sjøberg, D., Dybå, T.: Construction and validation of an instrument for measuring programming skill. IEEE Trans. Softw. Eng. **40**, 1163–1184 (2014)

Doctoral Symposium Papers

Knowledge Management and Reflective Practice in Daily Stand-Up and Retrospective Meetings

Yanti Andriyani(✉)

SEPTA Research, Department of Electrical and Computer Engineering,
The University of Auckland, Building 903, 386 Khyber Pass, New Market Auckland,
Auckland 1023, New Zealand
yand610@aucklanduni.ac.nz

Abstract. Knowledge management and reflection are important aspects in daily stand-up and retrospective meetings, which contribute to agile teams continuous improvement. Research in knowledge management in agile software development has shown knowledge classifications which do not seem closely related with agile practitioners and current research has not treated agile reflective practice in detail. This research, which will focus on daily stand-up and retrospective meetings, addresses two objectives: (i) to investigate specific knowledge types (i.e. product, project and process knowledge) in everyday agile practice and knowledge management strategies applied by agile teams; (ii) to explore the actual knowledge involved in the meetings, which helps agile teams to perform reflection and use that knowledge for reflection. Case studies will be applied for this research to analyse both meeting practices. It is expected that the research results will provide a framework for agile teams to manage knowledge and perform reflection, which would be useful for team and process improvement.

Keywords: Agile software development · Knowledge management · Reflective practice · Agile retrospective meeting · Daily stand-up meeting

1 Introduction

Agile Software Development (ASD) is a group of software methods that use an "inspect and adapt" process as a part of regular reflection for continuous improvement [1]. Daily stand-up and retrospective meetings are practices that are meant to help evaluate team progress, impediments, and plans and find ways to improve [2]. In the daily stand-up meetings, agile teams share progress, discuss the impediments that occur in the team and share their plans daily [3]. The retrospective meeting enables agile teams to inspect the feedback shared and discuss the ways to improve.

Supervisor: Dr. Rashina Hoda, Department of Electrical and Computer Engineering, The University of Auckland, email: r.hoda@auckland.ac.nz.
Co-Supervisor: Prof. Robert Amor, Department of Computer Science, The University of Auckland, email: trebor@cs.auckland.ac.nz.

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 285–291, 2017.
DOI: 10.1007/978-3-319-57633-6_21

Knowledge management is an important aspect for agile team creativity, which can lead to improving the agile process [4]. By knowing how to manage knowledge, which is useful for team learning, agile teams would be able to reflect and find ways to improve the process [4]. While the daily stand-up and retrospective meetings are meant to be used to share knowledge and perform reflection, in practice what specific knowledge (e.g. contextual information, understanding, insight, experience), knowledge types (e.g. product, process and project) and knowledge management strategies help agile teams perform reflection are not well understood.

With the motivation to address these research problems of knowledge management and reflective practice in ASD, this paper contains some initial findings of our research, which include a concept of knowledge management in ASD and a reflection framework in retrospective meetings (Sect. 5). However, there are some issues that are still unclear on how to correlate these findings. We hope that the consortium can provide suggestions and feedback on:

1. The content and presentation of our preliminary theoretical models.
2. Best practice in cross-team comparison and analysis of data and combined presentation.
3. Recommendations on known or hypothesized relationship(s) between knowledge management and reflective practice.

2 Relevant Prior Work

2.1 Knowledge Management in ASD

Knowledge is the combination of content from more than one categories of information, which taken from documents, practices and norms [5]. Relevant prior research shows several classifications of knowledge management in ASD. Several reviews focus on knowledge management school classification [6] and knowledge management concept in ASD [7].

There are three categories of knowledge management [8] (i.e. technocratic, economic and behavioural) of which two of categories (technocratic and behavioural) are associated with ASD [6] and the third category (economic) is not related to ASD. The *technocratic* category emphasizes on explicating knowledge and its flows. The technocratic school is further subdivided into three schools: system, engineering and cartographic schools. The system school refers to knowledge management strategies that use technology, such as JIRA, Wiki and GitHub; the engineering school focuses on the business context of software processes; and the cartographic school focuses on experts in a team as a centre of knowledge for the team. The behavioral category is further subdivided into three schools: organizational, spatial and strategic schools. This school focuses on collaboration and communication as knowledge management strategies. Developing the network among teams and using office space to support team communication are included in the behavioural school.

Another research is about knowledge management concept map in ASD [7]. Yanzer et al. [7] present several concepts map, such as ways of communication, human and

social factors, tools for knowledge management and knowledge representation forms. The human and social factor concept covers knowledge management adoption in agile projects. Other concepts, which include the ways of communication, tools for knowledge management and knowledge representation forms, focus more on techniques and tools to manage the discussion.

Specific explanation about knowledge classification in software engineering [9] is explained by Ebert & De Man [9], which classifies knowledge types into three types, such as product, project, and process knowledge. Product knowledge is the knowledge that consists of product features, which is related to other product features, protocols, products and standards. Project knowledge is the knowledge about project resources, such as work products, budget, milestones, team performance and targets achieved. Process knowledge is the knowledge about the workflow related to business process, supporting technologies and how teams integrate their work with others.

Referring to the aforementioned knowledge classification, this research intends to investigate the knowledge types (product, project and process knowledge) involved in ASD. By referring to these knowledge types, the explanation of knowledge involved in ASD would be more detailed and closely related with agile practices.

2.2 Reflective Practice in ASD

Most studies in the topic of reflective practice in ASD have only focused on how to perform retrospective meetings with the broad explanation on the reflective practice. One of the techniques introduced to be implemented in retrospective meeting is Post Iteration Workshop (PIW) [10]. PIW is performed in retrospective meeting by collecting obstacles and generating tasks and decisions. Postmortem review is another technique that is applied in retrospective meeting [11]. This review is useful to highlight five important issues during a two-week sprint that need to be focused on.

In addition, reflective practice is also explained in Babb et al. [12]. In their study, they investigate reflection in agile practices by introducing the Reflective Agile Learning Model (REALM). REALM classifies some agile practices based on Argyris and Schön's [13] classification, which embody reflection-in-action and reflection-on-action. Although reflection in each agile practice is captured in REALM, the specific knowledge used by agile teams to perform reflection has not been investigated.

To fill this gap, this research attempts to investigate what knowledge is managed by agile teams in performing reflection and how the reflection occurs in daily stand-up and retrospective meetings. By referring to Bain [14] about level of reflection (i.e. reporting, responding, relating, reasoning and reconstructing), the explanation about reflective practice in those practices would be more specific.

3 Research Objectives

This research attempts to answer the following research questions:

RQ1. What specific knowledge types (i.e. product, project and process knowledge) are involved in daily stand-up and retrospective meetings and how do agile teams manage that knowledge?

RQ2. What actual knowledge helps agile teams perform reflection and how agile teams use that knowledge for reflection in daily stand-up and retrospective meeting?

The aims of this research are to explore knowledge types based on three knowledge types (i.e. product, project and process), the strategies in managing that knowledge in daily stand-up and retrospective meetings for agile team's reflection.

4 Research Design

In order to answer the research questions, this research will apply Yin's case study research methodology [15], which is classified into three phases: (a) Define and design, (b) Collect, prepare and analyse (i.e. data), (c) Analyse (i.e. findings) and conclude. Figure 1 summarises the structure of Yin's case study and shows some phases in this research. The colours indicate the research progress. Green refers to the tasks that are "done", yellow indicates the tasks that are "in progress" and red refers to "to do" tasks. The current research focuses on phase b which is to analyse collected data (interviews and observations).

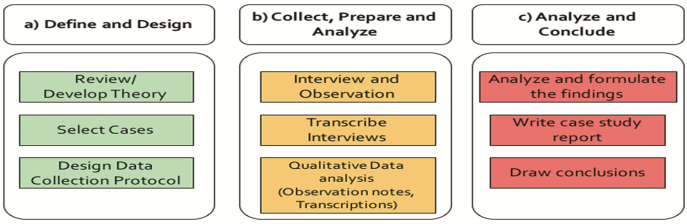

Fig. 1. Case study method [15] (Color figure online)

Firstly, in the define and design phase, a concept about knowledge management in ASD was generated through the Systematic Literature Review (SLR-'review/develop theory'). The next phase in the case study is to collect, prepare and analyze (phase b). Data collection was started by conducting interviews (individual and group) and meeting observations, which aims to gain specific explanation from the participants and understand the situation and actual knowledge managed in the meetings.

The next steps after observations and interviews are transcribing the interviews and analyzing them. The interviews transcripts were analyzed by using a qualitative data analysis technique called thematic analysis [16] by generating initial codes, searching for themes, defining and naming themes and finally producing the report that will be integrated on the next phase. Lastly, in the analyse and conclude (phase c), the analysis results of each team will be compared with those from other teams to formulate the findings and conclude the research. The results will be analysed comprehensively and followed by formulating the findings (phase c).

5 Current Research Progress

This research has two initial findings, which emerged knowledge management and reflection in daily stand-up and retrospective meeting. Initial findings were generated from SLR and case studies are described in the section below.

5.1 Initial Findings on Knowledge Management in ASD

A Systematic Literature Review was performed that reviewed 46 empirical studies focused on knowledge management in ASD selected from an initial pool of 2317 papers from reputed databases such as Springer, Scopus, and IEEE Xplore. Using a combination of thematic analysis [16] to analyse the primary studies and a Grounded Theory [17] approach to synthesise the results, it was discovered that:

1. Agile practices were found to be associated with the three types of software engineering knowledge proposed by Ebert & De Man [9]: timelines, team progress, and plans representing *project knowledge*; requirements and designs representing *product knowledge*; and coding techniques and synchronised teamwork representing *process knowledge*.
2. To manage the knowledge, agile teams use three specific knowledge management strategies: *discussions* (e.g. sharing requirements), *artefacts* (e.g. user stories) and *visualisations* (e.g. burn down charts).

A theoretical model was generated from the results (see Fig. 2), which explains that the three knowledge types are managed by performing agile practices and knowledge management strategies. This result was submitted currently under review.

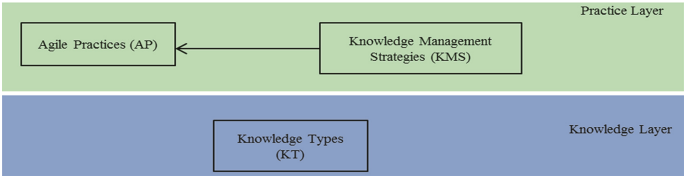

Fig. 2. A theoretical model of knowledge management in ASD

5.2 Initial Findings on Reflection in Agile Retrospective Meeting

A case study was conducted using data collected from interviews of sixteen software practitioners from four agile teams and observations of their retrospective meetings. Collected data was analyzed by applying thematic data analysis [16]. By transcribing data, generating codes, searching for themes, reviewing themes, defining and naming themes, the findings of this case study were formulated in the form of a paper, which has been accepted in XP 2017 conference ("Reflection in Agile Retrospective").

This case study aims to investigate what aspects are focused on during the retrospective meeting and how reflection occurs in the retrospective meeting. By applying

thematic analysis to analyze the interviews, it was discovered that identifying and discussing *obstacles, discussing feelings, analyzing previous action points, identifying background reasons, identifying future action points and generating a plan* are important aspects involved in the retrospective meeting, which is useful for agile team reflection. These aspects are associated with five (grouped to three) levels of reflection from education [14]. The levels of reflection from education appear related to the answer of how reflection occurs in the retrospective meeting, which can be classified into three levels of reflection [14], *reporting and responding, relating and reasoning, and reconstructing*.

According to these findings, a reflection framework for agile retrospective meeting was presented on Fig. 3. The framework combines five steps of the standard agile retrospective – set the stage, gather data, generate insight and decide what to do, close the retrospective – and the levels of reflection – reporting and responding, relating and reasoning, and reconstructing [14] include the aspects involved on each step.

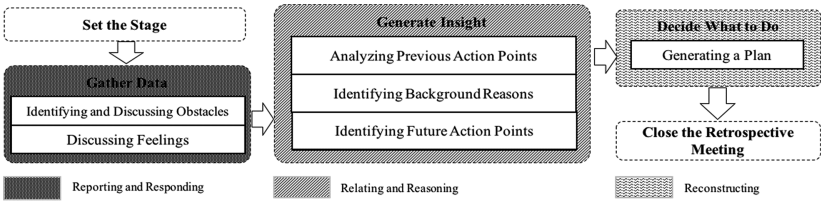

Fig. 3. A reflection framework for agile retrospective meeting

There are some research tasks remains, as can be seen in Fig. 1 (phase b and phase c), in which some tasks are in progress. Next steps include writing up the results of the case study pertaining to the daily stand-up practice, conducting further observations, analysing the transcription of software teams and formulating the findings.

References

1. Fowler, M., Highsmith, J.: The agile manifesto. Softw. Dev. **9**, 29 (2001)
2. Ringstad, M.A., Dingsøyr, T., Brede Moe, N.: Agile process improvement: diagnosis and planning to improve teamwork. In: O'Connor, R.V., Pries-Heje, J., Messnarz, R. (eds.) EuroSPI 2011. CCIS, vol. 172, pp. 167–178. Springer, Heidelberg (2011). doi: 10.1007/978-3-642-22206-1_15
3. Santos, V., Goldman, A., de Souza, C.R.B.: Fostering effective inter-team knowledge sharing in agile software development. Empirical Softw. Eng. **20**(4), 1–46 (2014)
4. Crawford, B., De La Barra, C.L., Soto, R., Misra, S., Monfroy, E.: Knowledge management and creativity practices in software engineering. In: Proceedings of the International Conference on Knowledge Management and Information Sharing, KMIS 2012, pp. 277–280 (2012)
5. Davenport, T.H., Prusak, L.: Working Knowledge-How Organizations Manage What They Know, vol. 5, pp. 193–211. Harvard Business School Press, Boston (1998)

6. Bjørnson, F.O., Dingsøyr, T.: Knowledge management in software engineering: a systematic review of studied concepts, findings and research methods used. Inf. Softw. Technol. **50**, 1055–1068 (2008)
7. Yanzer Cabral, A.R., Ribeiro, M.B., Noll, R.P.: Knowledge management in agile software projects: a systematic review. J. Inf. Knowl. Manage. **13**, 1450010 (2014)
8. Earl, M.: Knowledge management strategies: Toward a taxonomy. J. Manage. Inf. Syst. **18**(1), 215–233 (2001)
9. Ebert, C., De Man, J.: Effectively utilizing project, product and process knowledge. Inf. Softw. Technol. **50**(6), 579–594 (2008)
10. Cockburn, A., Highsmith, J.: Agile software development: the people factor. Computer **34**, 131–133 (2001)
11. Dingsöyr, T., Hanssen, G.: Extending agile methods: postmortem reviews as extended feedback. Adv. Learn. Softw. Organ. **2640**, 4–12 (2003)
12. Babb, J., Hoda, R., Nørbjerg, J.: Embedding reflection and learning into agile software development. IEEE Softw. **31**, 51–57 (2014)
13. Argyris, C., Schon, D.A.: Organisational Learning II: Theory, Method and Practice, Organisation Development Series. Adisson Wesley, Reading (1996)
14. Bain, J.D., Ballantyne, R., Packer, J., Mills, C.: Using journal writing to enhance student teachers' reflectivity during field experience placements. Teach. Teach.: Theo. Pract. **5**, 51–73 (1999)
15. Yin, R.K.: Case Study Research: Design and Methods, vol. 5, p. 11. Sage Publications, Inc, Thousand Oaks (2003)
16. Braun, V., Clarke, V.: Using thematic analysis in psychology. Qual. Res. Psychol. **3**, 77–101 (2006)
17. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine Pub. Co., Chicago (1967)

Self-Assignment: Task Allocation Practice in Agile Software Development

Zainab Masood(✉)

Department of Electrical and Computer Engineering, The University of Auckland, Building 903, 386 Khyber Pass, Newmarket Auckland, Auckland 1023, New Zealand
zmas690@aucklanduni.ac.nz

Abstract. Self-assignment is a self-directed way of task allocation commonly practiced by members of agile teams. However, not much is known about different aspects of self-assignment in literature. This research focuses on two objectives with respect to self-assignment. The first objective is to explore what strategies agile practitioners follow to self-assign tasks of different nature (i.e. new feature, enhancement, and bug-fix). The second objective is to identify the challenges associated with self-assignment and investigate how agile practitioners overcome these challenges to achieve project outcomes. Grounded theory is chosen as the research methodology for this study with data collection through interviewing agile practitioners and observing teams practicing self-assignment. Based on the results, we would propose a theory for self-assignment as a task allocation practice and a set of context-driven guidelines. Knowing the proposed theory and guidelines will help the agile practitioners and companies to make self-assignment a valuable practice in their settings.

1 Introduction

Agile development methodology emerged as an alternative to conventional, sequential and phase-based development. It follows an iterative and incremental approach to development and is open to changes throughout the project [1]. In contrast to traditional development processes, agile offers a different approach to managing the software development cycle. Agile software development constitutes a set of methods and practices based on twelve principles formulated in the Agile Manifesto [2]. The leading agile methodologies (Scrum, XP, Kanban) suggest different strategies and practices to ensure smooth development to achieve project outcomes.

An agile team is a cross-functional group of people who brings a different set of skills to the team. The essence to successful agile teams is their capability to self-organize accompanied by ownership. We find many contributions by researchers made

Supervisor: Dr. Rashina Hoda, The University of Auckland, email: r.hoda@auckland.ac.nz.
Co-Supervisor: Dr. Kelly Blincoe, The University of Auckland, email: k.blincoe@auckland.ac.nz.

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 292–297, 2017.
DOI: 10.1007/978-3-319-57633-6_22

exclusively on self-organization and self- organizing nature of the teams [3]. However, there is a dearth of research on how task allocation is done in self-organizing agile teams and what are the common practices followed by agile practitioners to achieve their goals.

Agile methodology uses self-assignment method for the allocation of tasks among team members [4]. However, we do not have enough studies and evidences regarding how software engineers tend to choose these tasks for themselves. There are certain factors that tend to motivate the engineers and developers to prioritize while self-assignment of tasks. During the process of self-assignment, they also have to face issues that need to be addressed for proper allocation and self-assignment. This study will be focusing on mainly these two aspects of self-assignment i.e. strategies and challenges for self-assignment in agile methodology. The contribution will be twofold. Firstly, it will add theoretical knowledge about self-assignment as a way of task allocation. Secondly, the results of this study will benefit developers and managers to overcome challenges during the process of task allocation.

2 Feedback or Areas Seeking Advice

At this time, we are seeking feedback and advice for the following things:

- What is the best way to compare findings from different sources and present overall findings in a way that data integrity is not compromised?
- Advice on reaching theoretical saturation for different task allocation strategies under different contexts.

3 Related Work

The success of a software development project depends heavily on the way the related project management activities are executed [5]. These activities primarily include managing the resources, organizing the software teams, allocating tasks to relevant stakeholders, monitoring time, budget, and resources [4]. These activities are carried out differently depending on the project management approach followed. In traditional software development, a project manager plays a key role in task allocation. The main duty of a project manager is to assign tasks to the project teams. This work is assigned keeping in mind the knowledge, skills, expertise, experience, proficiency and technical competence of the team member [6].

The benefits of agile methodologies include but are not limited to teams empowerment, collaborative atmosphere, shared decision-making and a transparency with a client [4, 7]. In addition, the concepts of ‘light touch’ management and self-organizing teams are the essence of agile teams [1]. These benefits have taken many software firms by a storm, as a result adopting many of these practices in their everyday project management activities including task allocation [4, 8]. This has affected the way the tasks allocation takes place in agile teams. Instead of manager directing or assisting the tasks, these teams are meant to practice picking up tasks or volunteering for tasks [7].

Self-directed task allocation or self-assignment is an attribute of agile teams [4, 8]. In theory, every member of the agile team is meant to assign a task or user story to themself [4]. This method of assigning tasks has also been observed in open source software (OSS) development in both commercial and non-commercial projects [9, 10]. Research on industry practices gives some evidence to support this method of task allocation but how this takes place is not very deeply investigated. For this reason, it is potentially a promising area for study leading to both academic and practical implications.

4 Research Basis

In this study, we intend to explore self-assignment of tasks in agile software development teams. The main research questions governing the research are:

RQ1: How agile practitioners practice self-assignment of tasks? What are the best strategies for self-assigning different types of tasks (new feature, enhancement, bug fixation) in agile software teams?

RQ2: What are the challenges associated with self-assignment of tasks? How agile practitioners overcome these challenges?

5 Research Plans

To answer RQ1 and RQ2, we will focus on how task allocation is played out in agile teams. In particular, this will center on the strategies that teams and individuals undergo in practice using different agile methodologies and for different projects including challenges associated with using these practices. We also plan to study self-assignment as task allocation in different scenarios and contexts and the study will not be limited to a single domain.

- Identifying strategies for tasks of different nature(New feature, Bug Fix, Enhancement)
- Classification of common strategies
- Strengths and weaknesses of the common strategies
- Identifying best strategies in their settings
- Factors affecting self-assignment of tasks
- Comparing self-assignment to alternative methods of task allocation
- Challenges faced with different strategies(threats to autonomy and cross-functionality, complexity and dependency dimensions)
- Identifying the areas of improvements with these strategies
- Evaluating the generated theory using GT guidelines
- Proposing a context-driven set of guidelines which agile practitioners may take into account while self-assigning tasks to get the best out of it.
- If time permits, evaluating the effectiveness of these guidelines through survey based feedback.

6 Research Method

We studied few research methodologies [13] and selected Grounded Theory (GT) for our study [11, 12, 14]. Grounded theory was developed in the early 1960's by Glaser and Strauss. It is chosen as the research methodology mainly due to listed reasons.

- Interest of the researcher towards generating theory explaining how self-assignment is practiced by agile practitioners
- GT is suitable for research areas which have not been explored thoroughly before.
- GT is extensively used for studying agile software teams, human and social aspects of software engineering, and many project management issues [3].
- GT *treats everything as data* giving researcher the freedom to use quantitative data, qualitative data, video, diagrams, and existing theories [14].

Initially, literature and related work are explored generally on identifying how and when tasks are allocated using traditional and non-traditional software methods for software projects. As recommended by Glaser, a minor literature review is conducted in the area of research [12] i.e. self-assignment as a practice of agile teams and individuals. Additionally, we went through articles describing grounded theory in other areas which helped to understand the research methodology and the emergence of the theory from the data [15–17].

As the research is mainly qualitative in nature, the intended data source is semi-structured interviews with agile practitioners of the relevant industry. In terms of data collection, we intend interviewing a total of 40–50 agile practitioners with team observations. But for some parts of the study e.g. factors affecting self-assignment, survey-based data collection will be pursued. Ongoing data analysis and synthesis procedures will be employed on collected data leading to findings of the research. In later stages, when the findings will be sufficiently developed latest and previous related literature will be reviewed again. We intend to assess the generated theory on the basis of four criteria: fit, work, relevance, and modifiability as recommended by Glaser [11].

The main components as adopted by some of the researchers are listed below [14]:

- Data Selection and Collection: Theoretical Sampling (Recruiting participants; Interviews; Observations; Surveys; Questionnaires);
- Data Analysis: Open Coding; Selective Coding; Theoretical Coding; Constant Comparison; Memoing; Sorting; Theoretical Saturation; Generating Theory

7 Validity Threats and Control

The most relevant validity threats to the research along with some checks to be taken to minimize them are given below.

- To reduce researcher bias, we intend to collect data from different sources interviews, observed meetings, and questionnaire. Such data triangulation will help us to generate more substantial data.

- Additionally, to collect multiple perspectives we plan to collect data from different contexts so that we do not limit this study to a particular setting, also we will be interviewing different roles belonging to variant sized organizations working on different software types.
- The supervisor and the co-supervisor, have strong expertise in empirical methods, especially GT and will keep a constant check to make sure that the researcher is not inclined to some side at some point during the study.

8 Current Status

- We have completed an initial round of literature review of related work on task allocation from a pool of papers published between 1990 to 2016 and gathered related work on task allocation generally in software projects and explicitly for agile projects.
- We have also explored some research methodologies to analyze and synthesize the data. After studying few research methodologies we decided to use grounded theory.
- Additionally, we conducted a pilot study to explore self-assignment in agile teams and investigated few aspects associated with it on a relatively small number of agile practitioners. During this study, we found self-assignment to be a potential area for further research as it has not been addressed extensively in the literature. The findings of this study are formulated and submitted to XP 2017 conference and accepted as a short paper ("Exploring Workflow Mechanisms and Task Allocation Strategies in Agile Software Teams") and the social aspects of the study are formulated, submitted to CHASE2017 and accepted as notes paper ("Motivation for Self-Assignment: Factors Agile Developers Consider").
- At present, we are collecting data. We have been successful in gaining some agile practitioner participants and continue to approach others.

References

1. Hoda, R., Noble, J., Marshall, S.: Agile project management. In: New Zealand Computer Science Research Student Conference, vol. 6, pp. 218–221 (2008)
2. Manifesto for agile software development (2001). http://agilemanifesto.org. Accessed 7 Jan 2017
3. Hoda, R., Noble, J., Marshall, S.: Developing a grounded theory to explain the practices of self-organizing Agile teams. Empirical Softw. Eng. **17**(6), 609–639 (2012)
4. Hoda, R., Murugesan, L.K.: Multi-level agile project management challenges: a self-organizing team perspective. J. Syst. Softw. **117**, 245–257 (2016)
5. Pinto, J.K., Slevin, D.P.: Critical success factors across the project life cycle. Proj. Manage. J. **19**(3), 67–75 (1988)
6. Acuna, S.T., Juristo, N., Moreno, A.M.: Emphasizing human capabilities in software development. IEEE Softw. **23**(2), 94–101 (2006)
7. Deemer, P., Benefield, G., Larman, C., Vodde, B.: A lightweight guide to the theory and practice of scrum. Version 2 (2012)
8. Hoda, R., Noble, J.: Becoming agile: a grounded theory of agile transitions in practice. In: IEEE International Conference on Software Engineering (ICSE 2017) (2017)

9. Crowston, K., Li, Q., Wei, K., Eseryel, U.Y., Howison, J.: Self-organization of teams for free/libre open source software development. Inf. Softw. Technol. **49**(6), 564–575 (2007)
10. Kalliamvakou, E., Damian, D., Blincoe, K., Singer, L., German, D.M.: Open source-style collaborative development practices in commercial projects using github. In: Proceedings of the 37th International Conference on Software Engineering, vol. 1, pp. 574–585. IEEE Press (2015)
11. Glaser, B.G.: Basics of Grounded Theory Analysis: Emergence vs. Forcing. Sociology Press, Mill Valley (1992)
12. Glaser, B.G.: Theoretical Sensitivity: Advances in the Methodology of Grounded Theory. Sociology Pr., Mill Valley (1978)
13. Myers, M.D.: Qualitative research in information systems. Manage. Inf. Syst. Q. **21**(2), 241–242 (1997)
14. Stol, K.J., Ralph, P., Fitzgerald, B.: Grounded theory in software engineering research: a critical review and guidelines. In: Proceedings of the 38th International Conference on Software Engineering, pp. 120–131. ACM (2016)
15. Hoda, R., Noble, J., Marshall, S.: Self-organizing roles on agile software development teams. IEEE Trans. Softw. Eng. **39**(3), 422–444 (2013)
16. Adolph, S., Kruchten, P., Hall, W.: Reconciling perspectives: a grounded theory of how people manage the process of software development. J. Syst. Softw. **85**(6), 1269–1286 (2012)
17. Stray, V., Sjøberg, D.I., Dybå, T.: The daily stand-up meeting: a grounded theory study. J. Syst. Softw. **114**, 101–124 (2016)

Software Development Practices Patterns

Herez Moise Kattan(✉) and Alfredo Goldman

Department of Computer Science, University of São Paulo (IME-USP),
São Paulo, Brazil
{herez,gold}@ime.usp.br

Abstract. Our ultimate goal is to propose a catalog with recommendations on how to organize the work of programmers. In this research we intend to provide experiments to explore the most suitable forms to allow programmers to develop software, either alone, in pair programming or in group. We also explore other approaches like code review. Our goal is not only to reduce the software development cost, but also to improve programmers life quality.

Keywords: Mob Programming · Swarming · Pair programming · Pair and review simultaneous in pairs · Code review · Coding Dojo

1 Introduction

The motivation of our research is to find better ways to organize the programmers work to develop quality software in a productive way suitable to their current context. Our goal is not only to reduce the software development cost, but also to improve the programming experience. Toward to do this a set of unanswered questions related on how many programmers should implement a task emerged:

- When Pair programming should be used?
- When it is interesting to perform Mob programming?
- What are the situations where it is better to do simultaneous work?
- What's the influence of the context and of the team?

2 Description of Points on Which We Would Like to Get the Most Advice on

We would like to have initial hints on when is better to use each one of the techniques and when alternating among them is a good idea. Our research is based upon the process of the Illuminated Arrow (see below).

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 298–303, 2017.
DOI: 10.1007/978-3-319-57633-6_23

3 Relevant Prior Work

Herez [9] did an extensive work on when to apply pair programming on several teams. The main conclusions were that pair programming should be applied when the task being developed is more complex or when there is a large gap on the programmers experience. On other situations, other more light techniques like code review can be applied without any drawback.

More recently we started to study also the benefits of Mob Programming.

There are points of convergence in the literature about the advantages of the use of Mob Programming over other techniques [9–13]. On a first experiment we figured out that Mob Programming was not very useful when no one in the team knew the language/framework being used [10].

4 Research Objective

Elaborate a catalog with suggestions on how the programmers should organize their work concerning pair programming and related techniques.

5 Research Approach, Study Design and Arrangements

The interpretation made in an interpretive case study is frequently impossible to be auditing posteriorly and, is very difficult to conduct controlled experiments. For this reason, Kattan [2] suggests to conduct application examples to produce raw data. After, to analysis this data, is suggested the use of the Grounded Theory techniques, to looking for one auditable Theory to explain the findings [5].

There are no silver bullets [6], but maybe together we could build illuminated arrows that somehow inspire the correct path to innovators. Figure 1 show the phases of this research method, that reduces the gap between software developers and academic researchers and, thus, produce more ready to use knowledge. The Illuminated Arrow [2] proposed application examples to deepen impartially the initial work of an action research, supported by systematic and tertiary revisions [4].

In Software Engineering it is very difficult to conduct controlled experiments or make convincing Double Blind experiments [3]. Furthermore, human expertise and human subjectivity interfere with the result of experiments. The types of software are very different, each software is unique, it depends on the problem it solves, so is different from medical research, where every human being has blood, lung, heart, brain, etc [2].

The reason to start with an action research is to fix the initial mistakes of the research and to be sure about the benefits and limitations of it. If the result of the initial work, is considered positive, the next step suggest by the Illuminated Arrow is systematically review the literature, making it easier to audit.

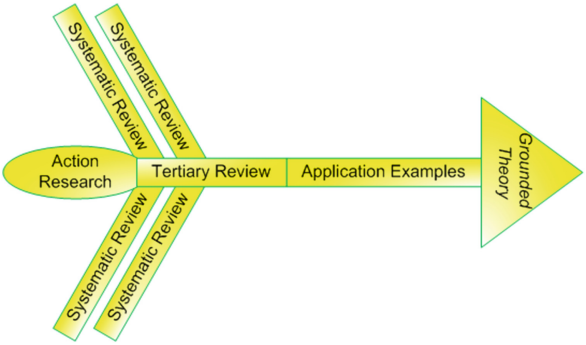

Fig. 1. Phases of illuminated arrow, starting from left and finishing in the right [2].

6 Action Research and Application Examples

The Empirical Study occurs twice. The first is in the beginning of the research as suggested by illuminated arrow, because start with an action research helps to deep the knowledge on this theme. Thus, makes easier the identification of some aspect possible to be improved and will guide the systematics reviews.

The second time, occurs after the literature review and is the empirical study by application examples. Thus, makes easier audits compared with interpretative case studies usely used. The applications examples will be careful design based at the literature reviews and action research.

These application examples will produce raw data about what we observe, toward to confirm and validated some aspects, provide new ideas and these raw data produced we hope that permit emerge one Theory in the way of one recommendation system to software developers about the better set of practices based on a specific context.

7 Data Analysis Methods and Techniques

The use of grounded theory is founded on the premise that the generation of theory at various levels is indispensable for a deep understanding of social phenomena [7,8]. The techniques of data analysis in grounded theory are:

- coding data (that comprises open, axial and selective):

  Open coding, to find categories;

  Axial coding, to find links between the themes/categories;

  Selective coding, to find the core category.

- memo writing;
- theoretical sampling.

8 Summary of the Current Status of the Research and Planned Next Steps

This proposal research is the continuation of Kattan [9] master's thesis. The technique is called Programming and review simultaneous in Pairs, is one extension to the pair programming. It's concluded when the goal is to reduce the time-to-benefit suggest use the Programming and review simultaneous in Pairs, when the pair is compose by professionals with the follows experience levels: intermediate and senior, or senior and senior, or junior and junior. The complexity of these tasks were classified as: low, medium and high.

Kattan reviewed the Mob Programming literature too in his master's dissertation and also applied Mob Programming in one application example.

Figure 2 illustrates the extension to pair programming, was used aspects of Simultaneous Engineering [9] to create one alternative to pair programming. The phases 1, 2, 3, 4, 5 and 6 are illustrated in Fig. 2. Phase 7 is illustrated in the form of the team with the work, because is the reflective rest and conflict resolution, is unformatted due to the miscellaneous possibilities for reflective/productive rest and conflict resolution.

The current status of the research and planned next steps are:

- We are conducting in companies experiments on Mob Programming, Programming and review simultaneous in Pairs, Pair Programming, Code Review and Coding Dojo [1].
- We are continuously reading the live science of this theme in literature in a frequently updating process.
- Beyond the use of questionnaire, we are analysing possible metrics [9].
- Based on feedback of international community we will rock the research and start the data collection.
- After conducting field studies, called here of application examples, we will analyse the data using Grounded Theory techniques.

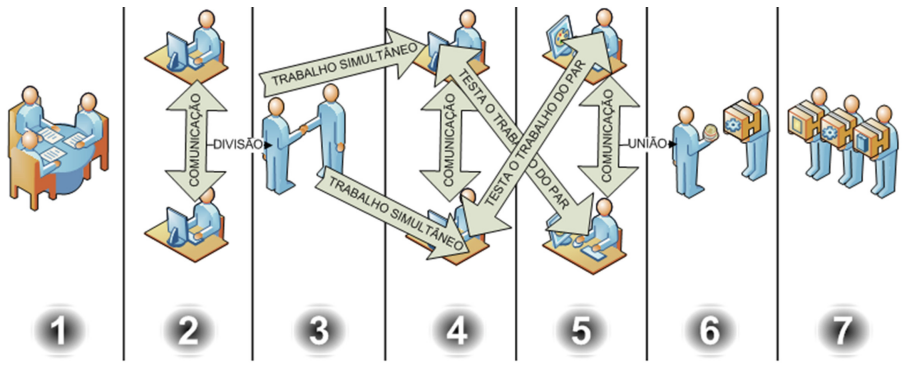

Fig. 2. Programming and review simultaneous in Pairs

References

1. Rooksby, J., Hunt, J., Wang, X.: The theory and practice of randori coding dojos. In: Agile Processes in Software Engineering and Extreme Programming: Proceedings of the 15th International Conference, XP 2014, Rome, Italy, vol. 179, pp. 251–259, 26–30 May 2014
2. Kattan, H.M.: Illuminated Arrow: a research method to software engineering based on action research, systematic review and grounded theory. In: CONTECSI - International Conference on Information Systems and Technology Management 2016, pp. 1971–1978, 21 July 2016
3. Budgen, D., Charters, S., Turner, M., Brereton, P., Kitchenham, B., Linkman, S.: Investigating the applicability of the evidence-based paradigm to software engineering. In: Proceedings of WISER Workshop, ICSE 2006, pp. 7–13. ACM Press, May 2006
4. Kitchenham, B., Charters, S., Budgen, D., Brereton, P., Turner, M., Linkman, S., JØrgensen, M., Mendes, E.: Guidelines for performing systematic literature reviews in software engineering, version 2.3. EBSE Technical Report EBSE-2007-01, Software Engineering Group, School of Computer Science and Mathematics, Keele University Keele, Staffs ST5 5BG, UK and Department of Computer Science, University of Durham, Durham, UK, 9 July 2007
5. Allan, G.: The legitimacy of grounded theory. In: Proceedings of Fifth European Conference on Business Research Methods, pp. 1–8 (2006)
6. Brooks, F.: The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition, 322 pages. Addison-Wesley, Reading (1995)
7. Glaser, E.G.: Advances in the Methodology of Grounded Theory: Theoretical Sensitivity. Sociology Press, Mill Valley (1978)
8. Glaser, E.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research (1967)
9. Kattan, H. M.: Programming and review simultaneous in Pairs: a pair programming extension. Master dissertation, Institute for Technological Research of the State of São Paulo (IPT) (2015). http://aleph.ipt.br/F or http://ipt.br, click on: Online Consultations, then click on: Library
10. Questionnaire. http://ccsl.ime.usp.br/wiki/SwarmQuestionnaire
11. Wilson, A.: Mob programming - what's works, what's doesn't. In: Agile Processes in Software Engineering and Extreme Programming: Proceedings of the 16th International Conference on Agile Software Development, XP 2015, Helsinki, Finland, pp. 319–325, 25–29 May 2015
12. Griffith, A.: Mob programming for the introverted. Experience report, Agile (2016)
13. Hohman, M., Slocum, A.: Mob Programming and the Transition to XP (2001)

Author Index