ENGINEERING QUALITY REQUIREMENTS IN LARGE SCALE DISTRIBUTED AGILE ENVIRONMENT

Wasim Alsaqaf

University of Twente, the Netherlands,
w.h.a.alsaqaf@utwente.nl

Abstract. Agile software development methods have become increasingly popular in the last years. However, agile methods don't specify explicitly how to deal with the quality requirements. Moreover there is little known about how organizations currently deal with this shortcoming. Based on several case studies this research will investigate real-world large-scale distributed agile projects to understand the challenges agile teams face regarding quality requirements and the approach they are currently using to cope with these challenges. After that a set of good practices will be introduced to explicitly integrate quality requirements in agile processes. Other case studies will be conducted to validate the suggested good practices.

Keywords: Agile requirements engineering, quality requirements, non-functional requirements, empirical research method.

1 Introduction and Motivation

Agile software development methods have become increasingly popular in the last years. The agile paradigm is a reaction to the traditional plan-driven software development methods such as Waterfall. The 2013 report by the Standish group [1] indicates a remarkable increase in agile projects and a decrease in waterfall projects. As literature indicates [2], the traditional approach of developing the software has predefined sequential steps that have to be followed in order to deliver the software. These steps are usually documentation driven and heavyweight software development. Because of this nature, traditional software development methods don't give an appropriate answer to rapid change in business environments [3].

A characterizing aspect of the traditional approach is a significant up-front Requirements Engineering (RE) effort. Usually, this is a process in which the software requirements should be fully elicited and documented prior to proceed to the next phase (the software design process). Although this process might seem logical, it is difficult and nearly impossible [4] to achieve due to the following reasons:

- The requirements tend to change quickly and become outdated even before the project is completed [4][5].

- Pre-specified requirements may become irrelevant as the project progresses. New requirements may arise, and old requirements may become irrelevant with a better understanding of the mission [4].
- The rapid change of the surrounding environment such as stakeholder preferences, technology, and time- to-market [4][5].

Responding to change is one of the agile alliance values stated in the agile manifesto [6]. Based on these values agile practices have been developed to increase the involvement of the customers and to deliver software products faster. The agile practices includes short iterations that end up with delivering a new incremental release, gathering as many requirements as needed to start an iteration, simple incremental design that will be evolved during the iterations and peer reviews [7].

Despite the fact that being agile is a success factor for software projects [1], the agile approach is still facing challenges that might cause the failure of a software project. The study of Ramesh et al. has identified seven challenges posed by agile practices [8], including the neglect of quality requirements (QRs). The neglect of QRs in agile projects may result in deliveries that don't satisfy the user expectations. In small co-located projects, this can be repaired relatively easily by adapting the next batch of requirements and repairing the part of the product already delivered. This is however not possible in large projects where the team is necessarily distributed over space and there is no possible of ad-hoc coordination and communication among team members and with clients. Hence, this doctoral research proposed in this paper addresses the problem of engineering QRs in large scale agile projects in a distributed organizational context and how these organizations cope with it. The research will also suggest a set of good practices as possible solution practices. Our expectation is that solution for large-scale agile projects will also be useful for smaller-scale agile projects. We will check this expectation later on after we identified solutions for large-scale projects.

2 Background on Quality Requirements in Agile Projects and Related Work

QRs are those requirements that describe the qualities of the system [9]. While the functional requirements specify what the system should do in response to specific action from the environment, the QRs describe how the system should perform these actions. Examples of the QRs are: Performance, Security, Maintainability, Testability, Usability, Portability and Reusability. The term non-functional is used in many studies to point to those requirements. However, this term in this case is passive, unfortunate and doesn't give the impression that these requirements are important [9][10]. Boehm used the term "quality requirements (QRs)" instead of non-Functional requirements [11], which in my opinion describes better the nature of those requirements. Therefore, throughout this work the term "quality requirements" is used to refer to the requirements that describe how the system should perform the desired actions.

In the RE literature, there is a consensus that the success or failure of a system is not only decided by the correct implementation of the right functional requirements. If the response time of the system for example doesn't meet the customer expectations, we can't say that the system deliver quality [10]. Besides that, high quality software architecture is intimately connected to the achievement of the QRs [12]. A software architecture that provides a high security and an acceptable performance is not the same as the software architecture that maximizes the performance and guarantees a certain level of security. However, since it is impossible to maximize the implementation of all QRs [13], it is necessary to define the desired level of each quality requirement to implement the most appropriate software architecture that delivers the expected business value.

A recent systematic literature review on agile requirements engineering practices and challenges [14] reported that neglecting QRs is a problem for agile requirements engineering processes. The same review reported that until 2013 there was just one study that comes up with a literature solution proposal (NORMATIC) [15]. This study has addressed the problem of the QRs in agile practices and developed a Java-based simulation tool for modelling QRs for semi-automatic agile processes. Although this study is directly related to our study, it differentiates form our study in the following aspects:

- NORMATIC's problem investigation is a literature research while our investigation will be done empirically.
- NORMATIC hadn't the focus on distributed agile environments and due to this nature; the challenges of being distributed and their effect on the QRs weren't taken in consideration.
- The results of the NORMATIC research weren't validated in real-world agile development projects and the effectiveness of the suggested tool in modelling QRs in agile projects weren't measured.

In 2015, Darshan Domah and Frank Mitropoulos developed the NERV Methodology [16]. Similarly to NORMATIC, NERV addresses the non-functional requirements in agile software development based on literature study while our research as mentioned before is an empirical research.

3 Research Goal and Research Questions

The main goal of this research is to define a framework to identify, model and test QRs in a large scale project of a distributed agile environment. Based on this objective, the central research question (RQ) of this study is:

How to identify, implement and test the QRs in a large scale distributed agile project?

To meet the research goal, the main RQ is elaborated in the following sub-questions:

1. What are the primary challenges to agile teams when working with QRs in a distributed agile environment?
2. How do agile teams currently handle those challenges in a distributed agile environment?
3. Are there any gaps between the practices the agile teams use and the expected outcomes of deploying these practices?
4. What is important for agile teams to improve in their current way of handling QRs?
5. What improvements could be done?
6. Do the improvements work in practice?

4 Research Method

A lack of empirical research on agile software development and QRs was observed in recent studies [17]. Particularly, in agile literature, much of the work is focused on validating the applicability and usefulness of practices in agile development methods. Beyond a few high visibility case studies the current literature does not provide much insight on actual practices used in agile and their effectiveness in supporting RE activities [8]. To respond to the lack of empirical evidence, this project will adopt case study research method [18] for the purpose of problem analysis, solution design and solution evaluation (or validation).

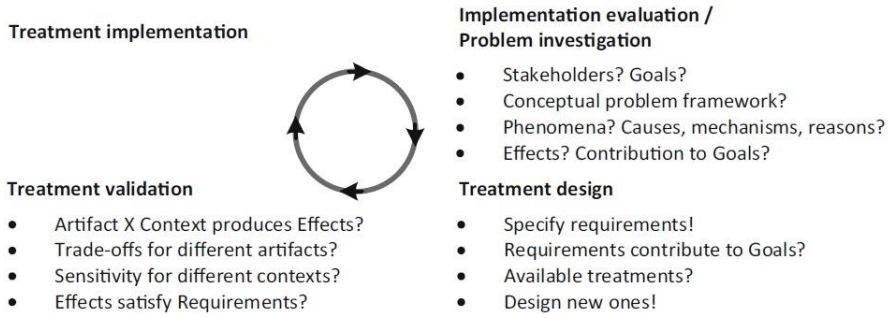

Fig. 1. Design and engineering cycle used in this research (Wieringa, 2014)

The design and engineering cycles consist of two mainly stages, each of which has its own set of tasks (Fig.1). The first stage "the design cycle" is decomposed into three tasks, namely, problem investigation, treatment design, and treatment validation. These tasks are multiple times iteratively implemented to develop a complete and useful design. The results of this stage are transferred to the real world in the second stage "The engineering cycle" which decomposed into two tasks, namely, treatment implementation, and implementation evaluation.

4.1 The implementation of the design and engineering cycles

First, we will carry out observational case studies at agile client organizations and software development organizations which will help to carry out a proper problem analysis. We assume the problems from a vendor and client perspectives would be different. Also, large government organizations that embark on agile might experience a different set of problems compared to private mid-size companies using agile. Furthermore, case studies will be used to understand which treatments companies are currently using to cope with the identified problems. Studying whether the treatments are useful and whether they meet the expectations of companies will be also part of the case studies.

The first four sub-questions of our research are corresponding with the problem investigation tasks of the design cycle (Fig.1). Answering those questions will allow us to drive a proper treatment design (good practices) to deal with the identified problems. With the design of the good practices the fifth sub-question of our research will be answered.

Once the good practices are developed, the plan is to validate and evaluate their utility and usefulness in follow-up case studies with practitioners working on agile projects [19]. We will use two methods to validate the framework, namely, Expert opinion and Technical Action Research (TAR). The developed good practices are useful if the identified problems can be solved using them.

4.2 Work Process and Collaboration with Companies

The PhD research project will follow an iterative work process based on the following work principles:

- The case studies will be part of each iteration to ensure better analytical results.
- Feedback loops will be part of the interactions between industry partners and researchers to ensure the relevancy of the researched subjects.
- Refinements and improvements of our good practices and solution method will be done incrementally on step-by-step basis. In this approach, case studies are essential to ensure the practical applicability of the research for agile software businesses in the Netherlands.

5 Expected Outcome and Contribution

The main expected outcome of this research is the good practices that can be used to identify, implement and test the QRs in large-scale distributed agile projects. The contribution of this research is in four specific aspects: (1) Giving insight in the current challenges agile teams meet regarding QRs in large scale distributed projects. (2) The current way of working to cope with those challenges. (3) The evaluation of the current way of working and 4) The good practices and the improvement that could be applied to real-life projects that use those practices. Although there have been several

proposals in scientific literature on how to treat QRs in agile projects, we found no proposal to be empirically evaluated and used in real-life settings.

6 Reflection and Progress of the Research

The PhD research is started on May 2015 and expected to be finished in 2020. However as a software engineer with a solid experience of developing software using different software development processes I faced the challenge of neglecting the QRs in almost all the agile software projects I was part of. This motivated me to dive into the literatures searching for an appropriate solution to deal with QRs in agile context.

The search process resulted in nothing sufficient. Thereafter I started to visit seminars and workshops provided by different software companies in the Netherland to find an answer to my question. I found out that each company I spoke to, had its own not validated practice to deal with the QRs. One of those companies introduced a non-official scrum term "Sprint zero". In this sprint they define the overall software architecture. This approach explicitly opposes the agile spirit which emphasizes the emergence of the software architecture during the development cycle. Another company defined the QRs as constrains within user stories. Constrains are simply another type of requirements such as QRs but they are not the same. Constrains are not just a matter of opinion. Any solution that does not meet the specified constraints is simply unacceptable [20]. In contrast to QRs constrains are not negotiable.

The next step in this doctoral research is to conduct a Systematic Literature Review (SLR) [21] to answer the following questions:

- What are the existing agile practices used for engineering QRs?
- What are the challenges created by the use or absence of those practices?
- What are the existing solutions according to published empirical studies?

Answering these questions will give us insight in State of Art of QRs within an agile context.

7 References

1. The Standish Group.: CHAOS MANIFESTO 2013: Think Big, Act Small. Standish Gr. Int. 1–52 (2013).
2. De Lucia, A., Qusef, A.: Requirements engineering in agile software development. J. Emerg. Technol. Web Intell. 2, 212–220 (2010).
3. Helmy, W., Kamel, A., Hegazy, O.: Requirements engineering methodology in agile environment. Int. J. Comput. Sci. Issues. 9, 293–300 (2012).
4. Boehm, B.: Requirements that handle IKIWISI, COTS, and rapid change. Computer (Long. Beach. Calif). 33, 99–102 (2000).
5. Cao, L., Ramesh, B.: Agile requirements engineering practices: An empirical study. IEEE Softw. 25, 60–67 (2008).
6. Agile Alliance.: Manifesto for Agile software development. (2001).

7. Shore, J., Shane, W.: The Art of Agile Development. O'Reilly Media, Inc (2007).
8. Ramesh, B., Cao, L., Baskerville, R.: Agile requirements engineering practices and challenges: an empirical study. Inf. Syst. J. 20, 449–480 (2010).
9. Lauesen, S.: Software Requirements: Style and Techniques. Addison-Wesley Professional (2002).
10. Blaine, J.D., Cleland-Huang, J.: Software quality requirements: How to balance competing priorities. IEEE Softw. 25, 22–24 (2008).
11. Boehm, B., In, H.: Identifying Quality-Requirement Conflicts. 30602 (1996).
12. Kazman, R., Bass, L.: Toward Deriving Software Architectures From Quality Attributes. Engineering. 1–44 (1994).
13. Haigh, M.: Software quality, non-functional software requirements and IT-business alignment. Softw. Qual. J. 18, 361–385 (2010).
14. Inayat, I., Salwah, S., Marczak, S., Daneva, M., Shamshirband, S.: A systematic literature review on agile requirements engineering practices and challenges. Comput. Human Behav. (2014).
15. Farid, W.M., Mitropoulos, F.J.: NORMATIC: A visual tool for modeling Non-Functional Requirements in agile processes. 2012 Proc. IEEE Southeastcon. 1–8 (2012).
16. Domah, D., Mitropoulos, F.J.: The NERV Methodology : A Lightweight Process for Addressing Non-functional Requirements in Agile Software Development. (2015).
17. Daneva, M., Van Der Veen, E., Amrit, C., Ghaisas, S., Sikkel, K., Kumar, R., Ajmeri, N., Ramteerthkar, U., Wieringa, R.: Agile requirements prioritization in large-scale outsourced system projects: An empirical study. J. Syst. Softw. 86, 1333–1353 (2013).
18. Wieringa, R.: Empirical research methods for technology validation: Scaling up to practice. J. Syst. Softw. 95, 19–31 (2014).
19. Wieringa, R., Daneva, M.: Six strategies for generalizing software engineering theories. Sci. Comput. Program. 101, 136–152 (2015).
20. Robertson, B.S., Robertson, J.: Mastering the Requirements Process. Addison Wesley Professional (2006).
21. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering. Engineering. 2, 1051 (2007).