Tailoring Agile in the Large: Experience and Reflections from a Large-Scale Agile Software Development Project

Knut H. Rolland[1,3](✉), Vidar Mikkelsen[2], and Alexander Næss[1]

[1] Westerdals Oslo School of Arts, Communication and Technology, Oslo, Norway
rolknu@westerdals.no, nesale14@student.westerdals.no

[2] Sopra Steria, Oslo, Norway
vim@soprasteria.com

[3] SINTEF, Trondheim, Norway

Abstract. It is not surprising that agile methods are tailored or customized in various contexts and projects. However, there is little advice for practitioners for how to go about tailoring agile methods in large-scale projects. Henceforth, the aim of this experience report is to highlight some of the challenges with large-scale agile software development and especially how to deal with these challenges involves continuous tailoring of the agile method in use. In so doing, we report from a large-scale agile software development effort involving more than 120 participants in a Governmental organization and running for 3,5 years. The project consisted of three deliverables, partly developed in parallel after a delivery model based on Scrum. After a much troubled start related to scaling challenges and architecture complexity during the first deliverable, the project was turnaround and the second and third deliverables were portrayed fairly successful by both supplier and customer. From a practitioner's perspective, we found that novel practices emerged through out the project that improved the way of working – especially across teams and stakeholders. Based on this, we describe some guidelines for tailoring agile in the large.

Keywords: Large-scale agile software development · Method tailoring · Software development practices

1 Introduction

In this experience report we draw from a recent large-scale agile software development project in a Norwegian Governmental organization. The project involved over 120 participants and was delivered through three distinct deliverables over 3,5 years. The project was highly prestigious and critical, as the Governmental organization had failed in two previous projects in replacing their core IT-systems. The specific context and complexity of the project with numerous external stakeholders, integration with existing portfolio of IT-systems, public contracting legislation, and replacing core legacy IT-systems made tailoring of a Scrum-based delivery model necessary. Existing literature on agile methods has for long underscored the need for tailoring to fit specific contexts and different types of projects [1–3]. However, the empirical literature on tailoring is not substantial, and there is little concrete advice for

DOI: 10.1007/978-3-319-33515-5_22

practitioners for how to go about doing tailoring and what to tailor in practice. Arguably, especially when agile methods and practices are scaled to larger projects in terms of involving multiple teams, heterogeneous users needs, complex software architectures, and numerous integration efforts with existing IT-systems, there is a pressing need to tailor and blend different agile methods [4]. Henceforth, *the aim of this report is to contribute to a richer understanding of tailoring agile methods in the context of large-scale projects – and based on this, to carve out some guidelines that would be useful for others.* We believe our experience and reflections from this project would be of interests to both project managers and developers as experience and guidelines for tailoring agile methods are hard to come by.

The remainder of this experience report is structured in the following way. The next section explains the case context. Then, we describe and analyse some of the experiences through out the project. Next, based on our experiences and some literature we try to give advice for tailoring agile in the large.

2 The Case: The Brownfield Project

Context. A case study of a major software development project was conducted from September 2014 to December 2015. The project, referred to as the Brownfield project, was a large-scale agile development effort involving over 120 participants over 3 years from 2011 to late 2014. The project was organized as four development – or 'Scrum' teams and one team loosely related to the project developing a business intelligence solution. Experience from this project is especially interesting in many respects. Firstly, the supplier, the Consulting company had just recently before starting on the Brownfield project been part of a prestigious large-scale agile software development project that was especially known nationally for being a success – and often used as a template for other large-scale agile projects in Norway. Secondly, the customer had tried two times before earlier in the 2000s to implement the Brownfield project and failed considerably in both cases.

This report is written based on 20 in depth-interviews of project participants, 2 workshops, project documents as well as numerous meetings with different participants. Additionally, one of the authors was the project manager for the Consulting company on the project during the third deliverable.

Three authors have written this report: one practitioner, one student, and one academic. One of us was the project manager for the project during the last of three deliverables. He has more than 10 years of experience as a project manager on large software development projects and agile projects in particular. The other author is currently a researcher working on a scientific case study of the project. Previously, he has also been a practitioner for many years participating in large-scale agile software development projects. The third author is a student of information systems management and innovation, who also has a background in industry. Obviously, our differences in experiences and background made the writing process especially interesting, as we were able to challenge each other's biases.

The scrum-based delivery model. The project followed a Scrum-based model that interestingly had been used by a recent large-scale project where the Consulting company was involved. This previous project was perceived as highly successful, and is generally regarded as 'best practice' for doing large scale agile in the Norwegian IT industry.

The Scrum-based delivery model is characterized by splitting up a large project in different deliverables as shown in Fig. 1 below. For each deliverable then, a semi-agile process is followed by first defining user stories, then architectural design, overall UX design, and refinement of user stories – but with a minimum of effort not to plan things in too much detail.

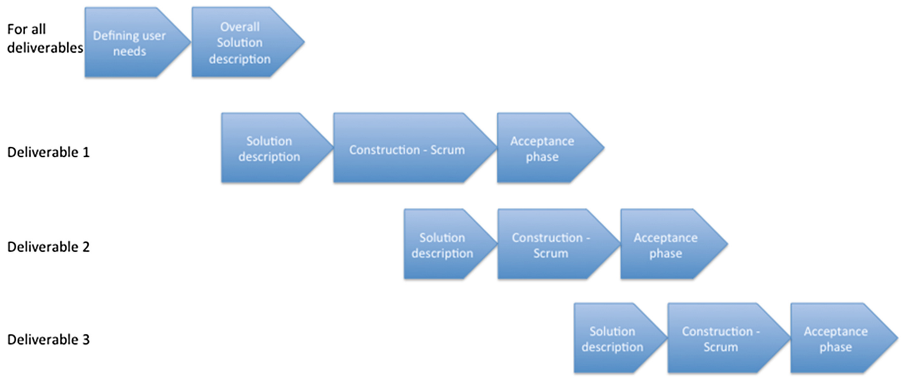

Fig. 1. Scrum-based delivery model of the project.

Project description and goals. The Brownfield project was established in order to replace The Client's outdated legacy IT systems with a new integrated system for case management. The new system was to be based on a Service Oriented Architecture, with support for integration with a large number of external and internal systems. In addition, the new system would include a web-based self-service solution aimed at the general public, as well as a rule-based application processing engine reducing the need for manual processing.

The project thus set out with four main goals:

- To replace fragmented case management systems with one integrated system.
- To replace manual processing with automated, rule-based processing.
- To establish a self-service web interface for the general public.
- To decommission legacy systems.

These goals were further elaborated in the form of a "dual goal matrix", specifying main business goals and main IT goals for each deliverable. The business goals were divided by functional areas, reflecting the existing organizational and system structure. The IT goals were more focused on architectural requirements, cutting across the functional areas in order to establish what was seen as a desirable "future state" of architecture in the organization. The background for the two subsets of goals was somewhat divided: The business requirements were related to limitations of the existing system portfolio in supporting new government regulations, interfacing with external parties, and efficiency in case management and processing. The technical requirements were driven by the strong internal technical organization's vision of a future-proof, platform independent architecture which would allow the organization to "pick and choose" technical components in a vendor independent manner. These technical requirements were communicated in the form of architectural standards and policies. In addition, the Customer had already purchased a number of technical components as part of existing vendor purchasing agreements for related database systems. The Consulting company was asked to consider the use of these components in developing the new architectural platform. At the outset of the project, it quickly became clear that the Customer was overwhelmed by the amount of work required in order to determine and specify requirements. The technical and architectural requirements seemed especially unclear, and resulted in a lot of time being spent by both parties in order to better understand what was actually required to be developed by the Consulting company. As a result, the first deliverable was delayed, and ultimately merged with the second planned deliverable in an effort to save time by skipping one of the planned production migrations.

3 Agile Method Tailoring in the Project

After a much troubled start related to scaling challenges and architecture complexity during the first deliverable, the project was turnaround and the second and third deliverables were portrayed fairly successful by both supplier and customer – including their end users. Noteworthy, we came across the following new practices as the project had been 'turned around':

(1) 'Task forces' were established across teams to deal with common challenges such as performance issues;
(2) 'Champion roles' were implemented working across teams on specific technology issues for example databases or java scripting;
(3) 'Specifying up front' in terms of close collaboration between customer and supplier in preparing user stories, uncovering dependencies and prototyping prior to sprints;
(4) 'Re-distributing development tasks' within the current sprint in order to utilize competence across teams and scale the project.
(5) 'Mini demos' were improvised in the middle of sprints to get users' feedback as soon as possible, and to do smaller adjustments to features and/or interaction design;

We will briefly describe these practices in more detail in the following sub-sections below.

Task forces. In traditional agile development, participants in projects are supposed to work within teams. In this project, however, an informal role of temporarily 'task forces' was formed. Task forces were formed on developers' own initiative for tackling specific pressing problems relevant across the four development teams. These were typically problems related to non-functional requirements. For example, security issues, performance problems and ways of integrating with external systems and standardized components.

Task forces were initially not initiated by management, but grew out of a need recognized by some developers at one of the development teams. The developers recognized that they had common problems across teams and started informally to sit together with fellow developers belonging to a different team. This practice was later sanctioned and even facilitated by team leaders and project management for better solving problems across teams.

Our analysis is that task forces not only solved common problems, but also greatly helped coordinating work across teams and helped building a more common understanding across teams both regarding software architecture and business domain. In this respect, task forces became a necessary addition to scrum-of-scrums in that they had a much more detailed focus on solving specific problems.

Champion roles. While the task forces explained above were of a more temporary nature, the champion roles were more permanent. Champion roles also started bottom-up from a perceived need in the teams to coordinate and standardize certain ways of doing things in the project. For example, it was established champion roles for java scripting and databases ensuring a common way of working with and implementing these technologies across teams.

Champion roles rotated among competent individuals, and over time this also became more sanctioned and facilitated by management.

Similar to task forces, but more stable – champion roles implied better inter-team coordination and standardization of working. Additionally, it also increased learning among teams and members from different teams.

Specifying up front. In collaboration with the customer, the project started to have a more formal process before a new sprint was initiated and sprint planning started. This process where referred to as the 'ready-to-sprint' processes, and engaged all the relevant actors for coordinating and planning of the work to be conducted in the upcoming sprint in more detail. Depending on the specific challenges and type of work to be conducted the process ensured that all involved actors had contributed and were coordinated. This process could include further specification of user stores, flow diagrams, description of technical as well as functional dependencies, and more overall architectural issues.

A crucial skill in agile development is to conduct the Product owner role and the ability to create Epics and user stories upfront the sprints. The project organization addressed these issues by including two persons from the customer in each scrum team with the role "functional responsible". The role was part of the customers Product owner team, and participated both in specification work and to cope with functional clarifications throughout the sprints.

Already from the first sprints conducted, it proved major challenges to establish effective ways of handling the product backlog, agile collaboration that supported both common understanding of specifications, ensure consistent architecture implementation across teams and handling clarifications of upcoming issues. Corrective actions were issued by training the Product owner in necessary skills and adding trained functional architects to the scrum teams. The actions, which were taken, did help to some extent, but it was necessary to make some fundamental adaptations to ensure a more robust process.

Our analysis is that this made the initiation of the sprints more effective and ensured that key participants were coordinated irrespective of team and role in the project. Although this practice inevitably implies more planning up-front seemingly in conflict with the agile principles and practices, we will argue that this practice is more aligned to the characteristics of large-scale agile where there is an increased need for more standardization and coordination across teams and roles.

Re-distribution of work tasks. Partly as a consequence of the previous practice, the project got increased flexibility to re-distribute work tasks across teams within a sprint. This practice was a part of striking a balance between the need for competence and efficiency at the one hand, and the evenly distribution of work among teams on the other. The practice was especially useful in the last sprints of a deliverable when user stories belonging to different domains did not imply equal distribution of work effort between teams.

Again, this practice may seem odd, and even unproductive, from the perspective of 'textbook agile'. However, this gave the project as a whole better utilization of the teams and also helped spread competence across teams. On the other hand, we also see that this practice should be used with care and only for smaller tasks when necessary typically late in the project.

Mini demos. The project had some especially competent project members who had long experience from other large-scale agile projects. Some of the practices they adopted from a previous project were the practice of 'mini demos'. The crucial point in doing mini demos in the middle of sprints was to demo features as soon as they were developed irrespective of when. Typically, this was practiced as a way of negotiating and getting feedback on details regarding functionality and interaction design. In that way, developers and designers could easily do the last finishing touches right away, without going through a more formal demo and going back to those details in the following sprint.

Thus, these mini demos both made the ongoing communication and collaboration with the customer smooth and at the same time reduced the administrative cost for both parties.

4 Implications for Tailoring Agile in the Large

In this section we propose some practical guidelines for tailoring agile in the large. We do not want to be too bold and generalize too much, as guidelines as these could easily be misinterpreted and used in contexts that are not comparable to our project. However, we argue that there is something more general worth mentioning based on our experience. The suggested guidelines are:

(1) **Experiment with new practices.** For tailoring agile in the large, projects should experiment with practices that highlight functional and technical interdependencies in the software being developed. This would help improve coordinating and communicating across teams and roles.
(2) **Facilitate novel practices to emerge.** It should be underscored that project managers should be wary of trying to enforce predefined tailored practices. However, although agile methods and principles tend to emphasize bottom-up initiatives, successful tailoring can be both bottom-up or top-down initiated.
(3) **"Record, and move on".** Do not wait for sorting out contractual details. Try to establish trust to that pragmatic decisions can be made and temporary solutions can be sought.
(4) **Improve inter-team coordination.** Establish both long term 'communities of practice' and short term 'task forces' across teams.
(5) **Scale the project in an evolutionary manner.** Plan for a ramp-up phase allowing customers to get accustomed to the working process. Conduct training activities to ensure customers are aware of what is required of them.
(6) **Adjust content in sprints.** Allow time for customers to absorb and process new information, and coordinate requirement elicitation with stakeholders in their organization. This can be done by inserting technical sprints where programmers focus on technical tasks, in order to allow customers a "programmer's holiday" [5].

5 Concluding Remarks

In this experience report we have emphasized the ways in which a large-scale agile software development effort has been tailored during the process. Here, tailoring was not done up-front, but rather emergent during the development over 3,5 years. Especially in this report we have highlighted and described five different practices and roles: (1) 'Task forces', (2) 'Champion roles', (3) 'Specifying up front', (4) 'Re-distributing development tasks', and (5) 'Mini demos'.

We argue that these novel practices are good examples of agile method tailoring reflecting the complexity and large-scale characteristics of the project. We do not argue that these actual practices denote any 'ultimate way' of tailoring agile projects, but more on an analytic level – that in succeeding with large-scale projects continuous tailoring throughout the process is necessary.

In reflecting upon the establishment of these practices we discuss how some are *bottom-up initiatives* (1, 2 & 4) largely initiated, planned and coordinated among team members themselves with no or little management involvement. Whereas some practices can be described as a *blend of bottom-up and top-down* (3 & 5) where management are much more involved.

Furthermore, we recognize that all of the practices turn out more *emergent*. They were not deliberately planned and adjusted ahead of starting the project – but emerged over time based on the involved actors' experiences. Collectively, then, the project seems to preserve a sense of agility in terms of 'learning from change'. Additionally, interestingly, some of these practices are seemingly also in conflict with the agile principles – notably (3) focusing on planning.

Acknowledgments. This article was written with support from the project Agile 2.0, which is supported by the Research council of Norway through grant 236759/O30, and by the companies Kantega, Kongsberg Defence & Aerospace, Sopra Steria and Sticos.

References

1. Williams, L., Cockburn, A.: Agile software development: it's about feedback and change. IEEE Comput. **36**, 39–43 (2003)
2. Dingsøyr, T., Moe, N.B.: Towards principles of large-scale Agile development. In: Dingsøyr, T., Moe, N.B., Tonelli, R., Counsell, S., Gencel, C., Petersen, K. (eds.) XP 2014. LNBIP, vol. 199, pp. 1–8. Springer, Heidelberg (2014)
3. Haugset, B., Hanssen, G.K.: Automated acceptance testing: a literature review and an industrial case study. Presented at the Agile 2008, Proceedings, Toronto (2008)
4. Jones, C.: Software quality in 2012: a survey of state of the art. Presentation by Namcook Analytics LLC. www.namcook.com
5. Martin, A., Biddle, R., Noble, J.: XP customer practices: a grounded theory. In: Proceedings - 2009 Agile Conference, AGILE 2009, pp. 33–40