ISSN 2278 – 0211 (Online)

Scaled Agile Framework: A Blight

Aditya Pancholi
Assistant Professor, University of Delhi, India
Sapna Grover
Assistant Professor, University of Delhi, India

Abstract:
The traditional notion of software lies in plan-driven model of development. However, agile development methods universally rely on updating and improving the plan along with the passage of time. They are best suited for application development where the system requirements usually change rapidly during development. Suited for small sized teams and small projects, agile development cannot be easily applied in large companies with large projects. Much work has been done in past to scale the agile development techniques. In this paper, we discuss scaled agile development framework, highlighting the problems associated with this scaling.

Key words*: Agile method, Scrum, Sprint, Agile Scaling*

1. Introduction

The notion of 'software engineering' was first proposed in 1968 at a conference held to discuss what was then called, 'software crisis'. The conference was attended by international experts on software, who agreed on defining best practices for software grounded in the application of engineering. The discipline of software engineering was coined to address poor quality of software, bring projects exceeding time and budget under control, and ensure that software is built systematically, rigorously, measurably, on time, on budget, and within specification.

A software process model is a simplified representation of a software process. Each process model is an abstraction of process that can be used to explain different approaches to software development. The earliest and the most primitive process model, waterfall model, define the fundamental activities as specification, development, validation and evolution and represent them as separate process phases. One moves to next stage only if the current phase is completed, documented and tested. It believes in the notion that time spent early in the software production cycle can lead to greater economy at later stages. A bug found in the early stages is cheaper in money, effort and time to fix than the same bug found in the later stages of process. Critics of waterfall model argue that it is impossible for any non-trivial project to finish a phase of software product's life-cycle perfectly before moving to the next phases and learning from them. Also, the idea falls apart when the problem constantly changes due to requirement modifications and new realizations about the problem itself by the customer [5].

Another class of process models are based on the idea of developing an initial implementation, exposing this to the user for feedback and evolving it on the basis of received response through several versions until an adequate system has been developed. Also, known as incremental development, this approach adapts to changing needs and provides rapid delivery and deployment of useful software to the customer as early as possible. This approach has major drawbacks from management perspective. Firstly, the process is not visible to the managers and secondly, the system structure tends to degrade as new increments are added. These problems become particularly acute for large, complex, long-lifetime systems, where different teams develop different parts of the large systems.

In majority of projects, software reuse concept is applied. Reuse-based software engineering is another software engineering strategy where the development process is geared to reusing existing software components. This is in response to demands for low production and maintenance costs, faster delivery of systems and increased software quality. The open source movement has made a huge reusable code base available at low cost [5]. The potential problems with reuse-based software engineering includes lack of control over functionality and performance, lack of control over system evolution, the need for support from external vendors, and difficulties in ensuring that systems can inter-operate.

This heavyweight plan-driven development believes that the best way to achieve better software is through careful project planning, formalized quality assurance, use of analysis and design methods supported by CASE tools and controlled and rigorous software development processes. However, when this heavyweight, plan-driven development approach is applied to small and medium-sized

business systems, the overhead involved is so large that it dominates the software development process. More time is spent on how the system should be developed than on the program development and testing. This led to the proposal of 'agile methods' in late 1990s. It allows the development team to focus on the software itself rather than on its design and documentation. But these agile methods were suitable for small projects. Lately, a lot of research has been done on scaling of agile methods to suit the requirements of large scale project [1]. In this paper, we try to explore the weaknesses of scaling agile methods for large size projects.
The rest of the paper is organized as follows. We describe agile methodology in section 2 along with its strength and weaknesses. Section 3 talks about techniques to scale agile methods. We identify the problems with scaling of agile methods in section 4. Section 5 concludes.

2. Agile Software Development

Agile methods, described in late 90s, are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads and producing high-quality code. The Agile movement proposes alternatives to traditional project management. Agile approaches are typically used in software development to help businesses respond to unpredictability [3].
In February 2001, the manifesto for Agile Software Development was published. The Agile Manifesto reads, in its entirety, as follows [7]:
"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value
Individuals and interactions over Processes and tools
Working software over Comprehensive documentation
Customer collaboration over Contract negotiation
Responding to change over following a plan
That is, while there is value in the items on the right, we value the items on the left more."
The agile manifesto is based on 12 principles, described as follows [7]:

- Customer satisfaction by rapid delivery of useful software.
- Welcome changing requirements, even late in development.
- Working software is delivered frequently (weeks rather than months).
- Close daily cooperation between business people and developers.
- Projects are built around motivated individuals, who should be trusted.
- Face-to-face conversation is the best form of communication (co-location).
- Working software is the principal measure of progress.
- Sustainable development, able to maintain a constant pace.
- Continuous attention to technical excellence and good design.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- Self-organizing teams.
- Regular adaptation to changing circumstances.

One of the most Extreme Programming is perhaps the best known and most widely used of the agile methods. The name was coined by Beck (2000) because the approach was developed by pushing recognized good practices, such as iterative development, customers' involvement etc, to 'extreme' levels. It integrates a range of good programming practices such as frequent releases of the software, continuous software improvement, and customer participation in the development team. The strength of extreme programming is the development of automated tests before a program feature is created. All the tests must successfully execute when an increment is integrated into a system.
Another agile method, Scrum method provides a project management framework [4]. It is cantered around a set of sprints, which are fixed time periods during which a system increment is developed. Planning is based on prioritizing a backlog of work and selecting the highest-priority tasks for a sprint. Hybridization of scrum is common as it does not cover the whole product development life-cycle; therefore, organizations find the need, to add in additional processes, to create a more comprehensive implementation.
Scaling agile methods for large systems is difficult. Large systems need up-front design and some documentation. Continuous integration is practically impossible when there are several separate development teams working on a project. There have been efforts on scaling agile methods, especially Scrum method [1], techniques for which are discussed in the next section.

3. Scaling Agile Development

Since 2005, efforts were made to apply scrum framework over products where development groups had a size of few hundred to even few thousands distributed over multiple sites. One of these methods, scaling agile development, given by Craig Larman, Bas Vodde [1], in which scrum framework is scaled for large projects, is described below.
In scrum framework for a large team size, say up-to 100 people, the entire team is divided into sub-teams. The number of sub-teams and size of each sub-team is not static. There exists one (overall) product leader (PL) who manages and coordinates with every sub-team.
Only one product definition/backlog is prepared for all the teams. A fixed sprint time cycle, usually two-four weeks, is considered for development of an increment after which it is reviewed and inspected. Once the product definition/backlog is available, a planning meeting, commonly one hour per week of sprint, is conducted between the PL and up-to two team representatives from each sub-team. High priority backlog parts are considered for development in the next increment. After this, another planning meeting is conducted

by every sub-team in which all the members discuss, in detail, about the development of the next backlog part. Besides this, during the entire sprint, every team independently conducts daily meetings (Daily Scrums) in order to track the development and increase information sharing.

At the middle of the sprint, in order to refine the product developed so far, a large group meeting is conducted in a big hall where each sub-team is given a separate area having white-boards, projectors etc. In its own arena, each sub-team displays its product development so far along with the techniques and tools utilized. Each team is requested to give relevant feedback for improvements to other teams. In this way, all members across all teams are eventually exposed to examine all items, which is critical for more team flexibility.

All code, across all teams, is integrated into one single item on a daily basis and verified with automated tests. At the end of the sprint, one running potentially shippable product increment is available. After the sprint is complete, a meeting (again of one hour per week of sprint) is conducted where the potential stake-holders and PL reviews the increment developed in the presence of two representative members from each sub-team. This review meeting is conducted in the same manner as the refinement meeting, with separate areas for each sub-team. A large room has multiple areas with computers, each staffed by team representatives, to record feedback, where the features developed by a team are shown and discussed. However, to increase overall feedback and alignment, in the beginning and end of the sprint review, everyone discusses the about the increment.

4. Scaled Agile Framework: A Blight

Scaled Agile Framework's fundamental assumption, “You’re large, therefore you need to scale” [1], turns out to be wrong. In actual situation, even in big companies, most of the projects are not really very large. Most of them can be handled by a single agile team, or a few agile feature teams. All of these can be handled in the standard scrum or agile fashion, with an empowered product leader to guide the team and a few joint meetings to keep coordinated. Most week to week and month to month planning doesn’t require a large-scale top-down coordination effort; in fact, the product leaders and team members are perfectly competent to work these things out without imposing a big process on them. Many multi-team projects can turn into single team projects once we get teams competent in doing “pure” agile software development [2].

Scaling Agile Framework tries to wrap good agile practices in a package that is designed to appeal to today's managers and executives, who do not understand agile. It may provide some benefits but endangers an organization's progress towards high functioning. In the name of picking the best of both worlds, it is merely an illusion that is being sold to large companies, who are too afraid to really change and just want to increase productivity, reduce defect counts, etc. The management still controls and manipulates every decision while giving an 'agile' fairy world illusion to the developers.

The whole concept of iterating over a product rather than simply incrementing features is fundamental to agile method but is completely bypassed with this framework. Continuous delivery, in order to tap into the market as early as possible, and adapt the product is ignored. Scrum serves as a very crisp mirror making transparent all of the fallacies, one hopes to preserve and lets you see immediately the impact of any change you make. But, when we synchronize the whole frigging organization's product development, it means any one team cannot adapt its process because it is locked into the organization's so called “Agile” framework. This murders the entire spirit of agile development.

5. Conclusion

It can be safely concluded that scaled agile framework will be a marketing success. But on the other hand, scaled agile framework is not really agile in its heart. It does have many good elements. It may be appealing to the managers and executives but the strength of agile gets lost somewhere. The fundamental attention is getting distracted from training and adapting team members to adjusting to larger organization. We believe, there is nothing we can rest on, except may be our values and principles. Any belief, practice, and by extension, framework, must be subject to rigorous and empirical review.

6. References

1. Larman C, Vodde B. Scaling Agile Development: Large and Multisite Product Development with Large-Scale Scrum: CrossTalk, May/June 2013.
2. Paetsch F., Eberlein A., Maurer F. Requirements Engineering and Agile Software Development. Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE’03).
3. Kar N.J. Adopting Agile Methodologies of Software Development. SETLabs Briefings, Vol. 4, No. 1, July - Sep 2006.
4. Schwaber K., Beedle M. Agile Software Development with Scrum. Prentice Hall, 2001.
5. Sommerville I. (2011). Software engineering. Addison-Wesley.
6. Pandey D, Suman U, Ramani A.K. A Framework for Modelling Software Requirements. International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 1, May 2011.
7. http://en.wikipedia.org/wiki/Agile_software_development