Lecture Notes in Computer Science: Scaling Scrum for Large Projects

Naum Puroski [1],

[1] Seavus, 11 Oktomvri 33A, Skopje, Macedonia
{naum.puroski}@gmail.com

Abstract. We all know what the *best is*, but not what *enough is*. We design and implement things aiming at perfection and thus spending more money and effort than real need and real justifications are. This problem is also present in the world of software engineering. In traditional software development processes a significant number of bureaucratic procedures exist that are spending a lot of money and time, leading to insignificant or unnecessary quality. Agilists are trying to solve this problem by creating lightweight development processes with minimal set of actions needed for successful project completion. In the past ten years, agile methodologies are successfully applied for small and medium projects, but there is still mistrust when it comes to large and critical projects. This paper summarizes the most common problems of Scrum agile process for large projects and gives guidance how Scrum process should be scaled in order to be more applicable for them.

Keywords: Scrum, Agile, Large Projects, Software Development Processes.

1 Introduction

Traditional or heavy software development processes are used by software teams more than 60 years. In their life they clashed with various problems; problems were analyzed, addressed and processes were upgraded in order same problems to not recur anymore. In this way over time, mature processes were created which are very accurate and resistant to bugs, but on the other hand they are bureaucratized, with high cost, slow start and long duration for development, fixed scope, heavy documentation and poor communication between people.

Nowadays software companies run into frequent changes in business demands and pressure to come to market as quickly as possible. In such dynamic environments traditional methodologies are not suitable because as previously mentioned in these methodologies requirements are fixed, start slow, and changes very easily lead to failure of the project [1]. Agile methodologies as an alternative to traditional methodologies, eliminate many of these weaknesses because they are adaptive rather than predictive; they rely more on people rather than process; they are open for changes and current needs of the client; they rely on code and communication rather than documentation; they implement the simplest possible solution instead solution which may never be used.

S. Markovski, M. Gusev (Editors): ICT Innovations 2012, Web Proceedings, ISSN 1857-7288

Agile methodologies in the past ten years are proved as successful approach for small and medium projects [10], but despite on these successful stories it is still difficult for software companies to decide to use them for large or critical projects. The biggest reasons why companies stay away from agile methodologies for large projects are difficulties in managing of large teams with this approach [2, 7, 4] and the lack of initial architecture, design and documentation [4, 5]. However, most experts agree that agile and traditional approaches are philosophically compatible [6, 3], even some of the methodologies (like Scrum [8]) give directions how to scale process in order to be more applicable for large projects. On one conference Ken Schwaber, founder of the Scrum methodology, was asked "Are there projects that are not meant to be Scrum projects?" on this he replied "Don't use Scrum if you want to ignore your impediments and if you don't want to know that your project may fail in an early stage" [9]. This kind of analyzes and activities by the experts and the agilists gave courage to people from industry to try agile methodologies for large projects.

Minimal use of agile methodologies for large projects and insufficient literature on this topic motivated me to analyze and examine whether the mistrust of companies for using agile methodologies, i.e. Scrum as their representative, for large projects is justified. I performed this analysis as part of my master thesis [11] and conclusions from my master thesis are briefly presented in this paper.

2 Top 10 Problems in Scrum for Large Projects

1. Although there is a general mistrust of companies for using agile methodologies for large project, there are companies who applied Scrum for large projects and shared their experience in papers or web articles. Analyzing this shared experience, taking in consideration the existing critics about agile methodologies and my previous professional experience with Scrum, I derived the most crucial and critic problems with which companies were faced when they tried Scrum for large projects and I categorized them based of their root problem:*Lack of initial solution design*. Scrum does not invest in creating of the initial architecture and design because in the early start of the project all requirements for the system are not well known and on the other hand as drafted they are expected to change. But anyway, initial design is essential for large projects. Lack of initial design leads to production of non-scalable, rigid, fragile and immobile code which is hard for maintenance i.e. each modification requires major changes in late stages of implementation. Also the lack of initial design complicates the process of creating independent teams that would work for implementation of isolated modules.

2. *Scaling of development teams*. The lack of a complete scope and design of the system disables proper sizing of the project teams at the beginning of the project. This is why agile methodologies have ad-hoc strategy for creation of new teams. Such a strategy leads to creation of teams that strongly depend on each other and which are not effective because most of the time they spend communicating with each other and correlating with other teams. On the other hand agile teams rely on people who are able to complete all types of tasks, from collecting requirements to testing. If we consider the required type and number (over 100) of such a people needed for large

projects, finding them is a real challenge. Impossibility of finding a sufficient number of this type of people leads to the need to introduce people with specific skills (technical writers, testers, GUI designers, etc.) in agile teams and their proper integration into them.

3. *Not effective face to face communication.* Face to face communication is effective in cases where the information should be shared with one or two people, but in cases when information should be shared with 10+ people this type of communication is extremely ineffective. In such situations there is a need for more documentation, not less.

4. *Unavailability or inconsistency of the client*. Client is directly involved in the Scrum development process and has a role of product owner. At the beginning of project and during sprint planning meetings he presents his needs in general, and after that during iteration, on daily bases, he defines details about them and gives feedback about completed tasks. This approach is good for small/medium projects, but it is almost impossible for large projects where project owner has to serve more than ten people concurrently. Because of this problem the development teams may be often stalled due to lack of information. If as solution we decide to add more people with role of product owner then we have problem with inconsistent requirements from different sources because there is no central point for verification and validation of customer needs.

5. *Managing and organizing product backlog*. Product backlogs for large projects can reach above 10000 customer stories. This size of product backlog is difficult for analyzing, monitoring and maintaining. On the other hand if we decide to create a one product backlog per module then we may lose the global picture of the system being built.

6. *Losing the big picture*. Creation of user stories in Scrum is a good idea because it allows frequent deliveries and focusing on small problems, but on the other hand it is a huge problem because by decomposing of the general functionalities into user stories we are losing the big picture of the system. Once when the general functionalities are decomposed and development teams start with implementation of the created user stories it is hard to reconstruct the big picture and to see exactly where the progress of the project is. Also user stories force agile teams to be focused on implementation of small functionalities rather on general functionalities. For large projects, top-down rather than bottom-up software development approach is more applicable in order the big picture to be kept and progress to be tracked better.

7. *Losing information about the system*. No investment in documentation leads to loss of information and decision points about developed functionalities. After a while without documentation no one, including developer and client, cannot remember why certain functionality is implemented as it is implemented. This problem grows exponentially with the size and length of the project, and is even greater when people in development teams are changed frequently and the code is not regularly commented or tested with unit tests.

8. *Planning and resolving dependencies***.** Unlike small projects where each requirement is assigned to one team in large projects new requirements depend on many teams. Client requirements need to be decomposed into multiple user stories

and they should be assigned to different teams. Synchronization and resolving dependencies between teams is needed and that presents problem because in Scrum there is no central point that define and assign tasks. Instead of this in Scrum tasks are defined by self-organized teams during sprint planning meeting according to some local priority.

9. *Working with distributed teams*. An inability to find sufficient number of people for large project on one geographical location induces companies to develop projects with distributed teams. Distributed teams have limited ability for communication because of their remoteness and different time zones. Thus violates one of the primary characteristic of agile methodologies and makes agile approach not very suitable for this type of teams. In order to apply agile methodology on project with distributed teams, teams have to be independent as much as possible and with that need for communication between team to be as low as possible.

10. *Change of people's habits and behavior*. Agile is completely new approach for managing of software projects and with that in contrast of traditional methodologies requires different behaving and acting of people. Changing of the mindset, habits and behavior of the people is not easy task, especially for large projects where number of people is large and management should encourage all of them to do something different than they usually do. It takes time and perseverance to train people to use new practices, to show and prove that this new approach gives results. Involvement of experienced Scrum Master into a project is crucial for success of this task.

3 Ten Rules for Successful Scaling Scrum for Large Projects

Based on findings presented in previous chapter and based on my previous professional experience with different software development processes, including Scrum, I am suggesting the following list of rules/practices which should be applied in order Scrum to be more applicable for large projects:

1. *Involvement of experienced Scrum Master*. Experienced Scrum Master's primary role is to encourage people to grow out of existing habits inherited from traditional methodologies and to try to transform hierarchical organized teams into self-organized teams. His experience will also contribute to the education of other team members and their motivation not to be distracted from the real road. From Scrum Master is also expected to assist in properly resolving the problems without adding bureaucratic activities into the process. For large projects, without an experienced Scrum Master the chaos could easily get out of control.

2. *Involvement of business analysts in agile teams*. At least one business analysts should be involved in each team who will act as a proxy to the client. In this way the role of Product Owner is scaled with people who are specialists in specifying software requirements, i.e. people who know best how to extract information from customers and who will best organize and pass on the desired functionalities to the team. Business analysts except as members of the development teams are acting as a special team of business analysts. The purpose of this team is to translate global requirements into user stories and to clarify dependencies and inconsistencies between the teams.

3. *Adding teams with special purpose*. In order the big picture to be kept and to cover tasks that are not in direct responsibility of any single development team, practice is adding of teams with special roles into a project. This kind of teams will cover segments of the software development lifecycle like verification and validation of customer needs; creation of user stories, their prioritization and resolving of dependences; creation of architecture and design; integration of solution and maintenance; system testing; etc. These teams may be composed of members of the existing development teams or by entirely new members.

4. *Involvement of specialists in the agile teams*. Except experienced people with wide range of software engineering knowledge, agile teams should also involve people with some specific field of knowledge. This practice will reduce need for such a large number of people with wide range of knowledge (specialists will perform work for which they are specialized; other will cover all other tasks) and also will improve effectiveness of the team (It is expected that the specialist will perform the specific task faster and with greater quality then others). For example, each agile team may involve GUI designer, QA engineer, technical writer, database developer, etc.

5. *Putting bigger accent on software architecture and design*. At the beginning of a large project we have to invest more in software architecture and design in order developed solution to be more robust, more scalable and more maintainable. Creation of the initial architecture and design will allow easier tracking of big picture, easier tracking of the impact from performed modifications, creating independent teams, including of less experienced people in teams, easier resolving of dependencies and inconsistencies in the system etc.

6. *Minimization of dependencies between teams*. Agile teams must be as independent as possible between each other in order need for communication and synchronization between them to be as low as possible. Ideal situation would be when the need for communication between agile teams would be at most once a week. This can be achieved by initially creating the architecture and design solution, and then assigning implementation of independent components on each team separately.

7. *Modification of the planning process*. In contrast to the small projects where one team is responsible for whole implementation and all tasks are defined and assigned during one sprint planning meeting, for large projects any required modification may have impact on more modules and with that on more teams. Taking into account that each team plans its tasks on separate sprint planning meeting, practices for synchronization of activities between teams is necessary in order modification to be completed at the end of a sprint. As a solution, each modification should be analyzed by team of business analysts and architects (equal on scrum of scrums team), during Scrum-of-Scrums meetings. During these meetings they have to perform impact analysis on required modification, to create user stories for all of the impacted modules and to define priorities for each of impacted teams. Expectations about these stories will be passed to the team by their representative on Scrum-of-Scrums meeting.

8. *Top-down approach for managing of the Product Backlog*. In Scrum general requirements are defined at the beginning of the project and then these requirements are decomposed into smaller user stories (top-down approach). For large projects,

systems used for managing of the product backlog has to allow managing of all level items (general requirements, user stories, tasks,…) and to allow establishment of links between them. Linking between backlog items will allow keeping of the big picture for the project, better tracking of the project scope, project status and dependences between user stories. The system for backlog managing should also allow filtering and grouping of items by components. This functionality allows keeping of all items in one product backlog and setting of different views by defining custom filters.

9. *Creation of minimum required documentation*. In large projects face to face communication is ineffective and therefore there is a need for creation of documentation that will contribute for easier communication between and within teams. In most cases it is practical to create software design document, technical documentation for specific algorithms and protocols used in the system, regularly commenting the code, test plan for system testing or SRS specification (not both because there is a 1:1 relationship between them). Type of documentation which will be created should be defined at beginning of the project and it must be created or updated in same iteration concurrently with the implementation of the required functionality. At the each iteration end we have to have potentially shippable product with completed documentation and code.

10. *Minimal introduction of new practices*. If new practices are introduced for all of the detected problems then agile process will soon become as bureaucratized as traditional ones. Instead of this, each problem should be analyzed during a sprint retrospective meeting, in order the root of the problem to be found and to be disused how, in an agile way, it can be avoided in future. New practices should be added only if the same problem occurs several iterations in a row. Experienced Scrum Master has the crucial role for control of this point.

4 Scaling of Scrum

The suggested rules and practices in previous chapter may be applied in Scrum very easily because Scrum is defined as a framework, not as a classical process where all activities and steps are defined in details. The only prerequisites in Scrum for adding of new practice/activity in the process are: people who suggest change have to understand the problem really; proposed modification must not violate any of the agile principles and at the end Scrum Master have to confirm that proposed modification will give positive results.

For the addressed problems about Scrum for large projects, explained in chapter 2 and following the rules for Scrum scaling, presented in chapter 3, I am suggesting scaling of the Scrum by adding of the these activities (see figure 1):

1. *Collection of requirements and creation of product backlog*. Initially a team of business analysts is created who should conduct the business analysis communicating with the client and the other stakeholders who are going to use the new system. Based on this analysis they create general requirements, after they decompose them into smaller user stories, detect dependencies and inconsistencies between the stories and finally all the stories are entered into the system for managing product backlog. This

stage can last for weeks; but it is practical to be limited on one iteration or maximum one month.

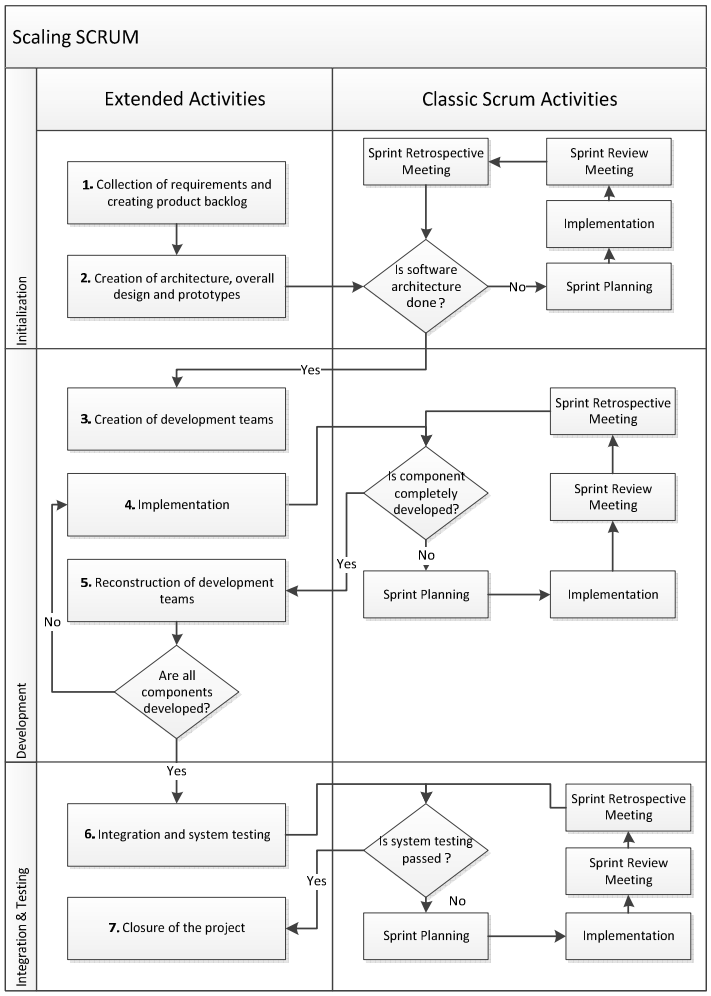

Fig. 1. Each phase of the scaled Scrum process uses classical Scrum process for execution of the team tasks. In the first phase classical Scrum process is used for creation of software architecture and design, in the second phase for implementation and in the third phase for integration and system testing.

2. *Creating the architecture, design and prototype of the solution*. Once the product backlog is created, a team for creation of initial architecture and solution design is established. In this team experienced technical people and business analysts who previously worked on collecting of requirements are included. The goals of business analysts in this stage are transfer of information to technical team and decomposition

of more general user stories into a smaller. Because this phase may last several months technical team, concurrently with software architecture, creates prototypes for proposed solution. Creation of prototypes allows showing something valuable to the client as also the end of iterations. In this way it is easier for the client to check whether the offered solution meets his needs. It is practical the team who works on this task to be not distributed in order communication between team members to be without limitations.

3. *Creating teams*. Teams should be created based on created software architecture teams. Ideally is for each separate component one separate team to be created. The teams created in this way very little depend on each other because they work on different components i.e. modules that can be easily isolated by mock/stub objects. The new teams, beside certain number of people with wide range of skills and certain number of specialists (tester, technical writer ...), have at least one of the team members from the architects and business analysts teams. They have a role to convey the current state, to present to the team into Scrum-of-Scrums meetings and do impact analysis on requested changes. In order to reduce the time needed for Scrum-of-Scrums meetings and increase the efficiency, Scrum-of-Scrums teams can also be defined by the architecture of the solution. One Scrum-of-Scrums team should have only members of teams that directly depend on each other (see figure 2).

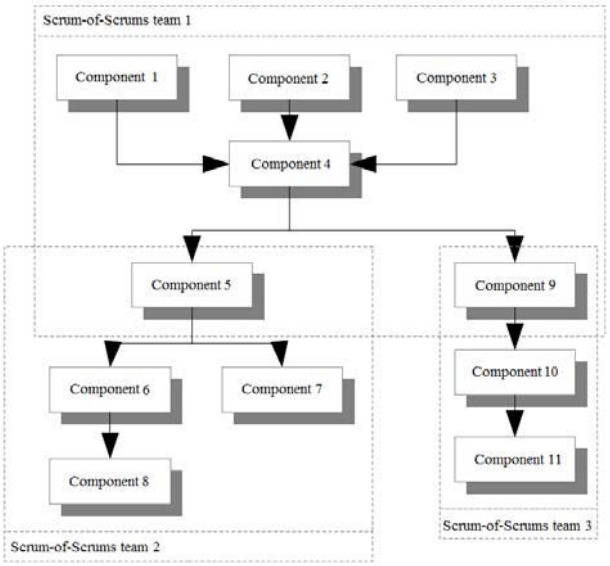

Fig. 2. One Scrum-of-Scrums team is composed of team members from teams which directly or indirectly depend. Team members from boundary teams (for example team 5 and 9) are part of two Scrum-of-Scrum teams and they have responsibility to pass information between two teams.

4. *Implementation*. Created teams implement assigned components using classic Scrum process.

5. *Reconstruction of teams*. Teams are reconstructed as soon as they finish with the implementation of some specific component. The team that has implemented some component may remain to function with smaller number of members if there are expectations for larger changes in the developed component. In opposite case the team is released and some of the members are transferred to the teams for system testing and maintenance. In both cases people who are experts on a certain part of the system are kept, these people will further be responsible for integration of all components in the solution and will be responsible for solving possible detected problems and system changes.

6. *Integration and system testing*. System can be integrated by using any integration approach: top-down, bottom-up, big-bang integration strategy, etc. The way how and when system will be integrated depends on selected integration strategy. In any case integration should be performed by the team that implemented the component which is about to be integrated or by the team members from the maintenance team. After integration is finished, system testing should be performed. This kind of testing should be performed by a special group of testers that can work within or without the maintenance team. This phase may last more than one iteration, so for this phase we also use classic Scrum approach.

7. *Closure of the project*. After integration and stabilization of the project whole team is released, except the maintenance team. This team is responsible for maintaining the system during his lifecycle. Optionally some of the teams may continue to work if they are responsible for some critical part of the system for which more frequent modifications are expected. After closing the product backlog product remains divided by components, but unlike before it will be run by one team.

Conclusion

Scrum methodology in the past 10 years had good results for small and medium projects, but it is still not too often used for large projects. Companies which tried Scrum for large project clashed with various problems as lack of initial design, loss of global picture, the impossibility of finding a sufficient number of “agile” people, spending too much time on communication and information loss. In this article I suggested several rules and practices that should be introduced into a Scrum in order these problems to be avoided or at least reduced. Some of suggested practices are investing more in software architecture, documentation and tools, introducing of the people with specific field of knowledge in agile teams, creation of agile team by some defined criteria, etc. These practices will make Scrum more robust and stabile and with this more applicable for large projects.

S. Markovski, M. Gusev (Editors): ICT Innovations 2012, Web Proceedings, ISSN 1857-7288

References

1. The Standish Group, Chaos Report, http://www.projectsmart.co.uk/docs/chaos-report.pdf, (1995)
2. Cockburn, A., Highsmit, J.: "Agile Software Development: The People Factor", Computer, vol. 34, no. 11, pp. 131--133 (2001)
3. Paulk, M.C.: "Extreme Programming from a CMM Perspective", IEEE Software, vol. 18, no. 6, pp. 19--26 (2001)
4. Boehm, B.: "Get Ready for Agile Methods, with Care", Computer, vol. 35, no. 1, pp. 64--69 (2002)
5. Rasmusson, J.,: "Introducing XP into Greenfield Projects: Lessons Learned", IEEE Software, vol. 20, no. 3, pp. 21--28 (2003)
6. Reifer, D.J.,: "XP and the CMM", IEEE Software, vol. 20, no. 3, pp. 14--15 (2003)
7. Fowler, B.,: "The New Methodology", http://www.martinfowler.com/articles/newMethodology.html (2005)
8. Cohn, M.,: "Advice on Conducting the Scrum of Scrums Meeting", http://www.scrumalliance.org/articles/46-advice-on-conducting-the-scrum-of-scrums-meeting (2007)
9. Smits, H.: "Implementing Scrum in a Distributed Software Development Organization". In Agile Conference (AGILE), pp. 371--375, IEEE Press (2007)
10. Results from the July 2010 State of the IT Union Survey, http://www.ambysoft.com/surveys/stateOfITUnion201007.html (2010)
11. Puroski, N.: "Scrum For Large Projects" ("Scrum за Големи Проекти"), master thesis in process (2012)