DOI 10.7603/s40601-013-0029-6

GSTF Journal on Computing (JoC), Vol. 3 No. 3, Dec 2013

Agile Software Development and Service Science

How to develop IT-enabled Services in an Interdisciplinary Environment

Andreas Meier, Jenny C. Ivarsson

Abstract— **This paper shows the necessary steps, which should be taken in order to get the most out of agile software development in interdisciplinary settings involving scientific experts. If applied properly, Agile delivers increased productivity, higher quality and, last but not least, higher customer satisfaction. The task of developing high quality software is already difficult. Developing software for a new IT-enabled service in an interdisciplinary team however, is even more challenging.**

In interdisciplinary projects scientific experts from different fields need to work together with computer scientists, developers, testers, business analysts and domain experts. Software engineering is very time-consuming and scientific experts who have never been involved in a software project, often find it hard to understand why progress sometimes seems so slow. Therefore, it is important that they understand what it takes to write high-quality code, i.e. code that is clean, tested, documented and extendable at the right points. The best way to achieve this goal is to expand the software team, make the scientific experts an integral part of it and thus profit from their know-how.

Keywords-component; agile software development, service science, scrum, scientific expert

I. INTRODUCTION

In the last decade, there has been a shift away from traditional towards agile software methodologies. Many people felt that the overhead imposed by traditional methods like the waterfall model, Unified Process etc. slowed down the development process and did not deliver the needed quality. Nowadays, many companies and software developers have adopted agile methodologies like Scrum [1], eXtreme Programming [2] (XP), Lean [4] or Kanban [5]. There are several reasons that brought about this shift from traditional, Big Design Up Front[1] (BDUF) to Agile Software Development [7], [10].

The sixth annual "State of Agile Development" survey [11] sheds some light on the benefits of Agile Software Development and its worldwide adaption. The survey data includes information from more than 6,000 respondents from different countries and was collected at the end of 2011.

According to the survey, the following main benefits were obtained by implementing Agile:

- 84% responded that the *ability to manage changing priorities* got better.
- 75% reported that they experienced an *increased productivity*.
- 71% had a *faster time-to-market*.
- 68% said that the *quality of the software* got better.
- 68% responded that the *alignment between IT & Business Objectives* got better
- *65% found that Agile reduced the risk in the project.*

These are impressive figures which show that agile methodologies improve productivity and increase both quality and customer satisfaction.

In the *Swiss Agile Study* [12], a survey conducted by Meier and Kropp, these findings have been confirmed for companies and IT-professionals. More than half of the participating companies are using an agile methodology like Scrum or XP. – Agile has become mainstream! Agile methodologies deliver results successfully and are superior for some categories of projects.

In this paper, we will first show which category of projects benefit from using Agile and in what category the IT-based service projects belong. To determine the category, the concept of project noise level has to be introduced. Next, we will give a short overview of Scrum, its terminology and how it works. Scrum alone or used in combination with eXtreme Programming (Scrum / XP hybrid) is the most widely used agile methodologies. They make up about two-thirds of the agile methodologies being used. Therefore, the focus will be on Scrum and XP in this paper. Following this, we will have a look at the different levels of agile planning and the importance of feedback cycles. To give a practical example of how scientific experts can be made an integral part of a functional Scrum team, we will then tell you about a project conducted by our Institute in the form of a case study. Finally, it will be discussed how software quality can be improved by following agile practices.

II. DEFINED AND EMPIRICAL PROCESS CONTROL MODELS

Agile methodologies, as opposed to traditional methodologies, are an implementation of the empirical process control model. When should I use agile methods and when traditional? To answer this question we first have to look at the noise and complexity in a development project.

[1] "Big Design Up Front is a software development approach in which the program's design is to be completed and perfected before that program's implementation is started. It is often associated with the waterfall model of software development." http://en.wikipedia.org/wiki/Big_Design_Up_Front

DOI: 10.5176/2251-3043_3.3.278

Published online: 28 February 2014

A. Noise Level and Project Complexity

Noise in a system development project is a function of these three vectors:

1. Requirements
2. Technology
3. People

If the requirements of the new service or product are well known, the noise of the first vector is probably low. If they are only partly understood, the noise level is higher, i.e. the uncertainty increases. Often the requirements are not well understood, poorly communicated or incomplete.

In software development projects we usually work with five or more different technologies like programming languages, databases, scripting languages, query languages, web services, markup languages, servers, networks etc. If all the technologies are familiar, the noise level is probably low. If any of the technologies are complicated, emergent and have not fully been tested, then the noise level is high.

When people are involved, nothing is simple. In IT-related service projects we have interdisciplinary teams and this increases the noise level even further.

How do you know when to use which process control model? Well, the higher the noise level, the more complex the project. So, if you have a simple project with a low noise level, a defined process like waterfall or Unified Process can be used. If the noise level is high, i.e. a complex project, an empirical process is better suited. Scrum is built on an empirical process model and therefore works well for complex IT-projects like the ones found in service science.

III. Scrum Basics

A. Scrum Terminology

Scrum is made up of three roles, four ceremonies, and three artifacts. These are the three roles that Scrum prescribes [13]:

- Product Owner: responsible for the business value of the project.
- Scrum Master: ensures that the team is functional and productive.
- Cross-functional Team: self-organizes to get the work done.

The following are the four ceremonies:

- Sprint Planning: the team meets with the Product Owner to choose a set of tasks to deliver during a sprint.
- Daily Scrum (aka Daily Standup): the team meets each day to share struggles and progress.
- Sprint Reviews: the team demonstrates to the Product Owner what it has completed during the sprint.
- Sprint Retrospectives: the team looks for ways to improve the product and the process.

And the following are the three artifacts:

1. Product Backlog: prioritized list of desired project outcomes/features.
2. Sprint Backlog: set of features from the Product Backlog that the team agrees to complete in a sprint, broken into tasks.
3. Burn Down Chart: at-a-glance look at the work remaining.

B. The Scrum Framework

The Scrum Framework according to the description of the *Scrum Alliance* [14]: A Product Owner creates a prioritized wish list called a Product Backlog. During Sprint Planning (aka Iteration Planning), the team pulls a small chunk from the top of that wish list, a Sprint Backlog, and decides how to implement those pieces. The team has a certain amount of time, a sprint, to complete its work - usually two to four weeks - but meets each day to assess its progress (Daily Scrum). Along the way, the Scrum Master keeps the team focused on its goal. At the end of the sprint, the work should be potentially shippable, i.e. ready to hand to a customer, to put on a store shelf, or to show to a stakeholder. The sprint ends with a Sprint Review and a Retrospective. As the next sprint begins, the team chooses another chunk of the Product Backlog and begins working again.

C. Agile Planning

Agile Planning [6] is an important part of the empirical process model and is done on different levels:

Daily Standup: 78% of the respondents of the "State of Agile Development" survey [11] report that they actually have Daily Standups.

Iteration Planning: 74% of the respondents use this agile technique, too. In this kind of meeting, which is based on the newly gained knowledge, the goals of the iteration are planned.

Release Planning: In intervals of typically three to six months, there is a Release Planning meeting. In this meeting, the content of the next release is planned on a rather coarse-grained level. 65% of the respondents report using this agile practice.

At the end of every sprint there is a demo of the new piece of functionality to the customer. The customer tests the new functionality and gives feedback. Does it meet the customer's expectations? Does it not? This feedback is used for planning the next iteration and helps ensure that the best possible system is built.

IV. Airline Catering Case Study

A. How to Integrate Scientific Experts in Scrum Teams

Where do scientific experts belong in Scrum? Are they customers, business analysts or do they belong to the team? We will try to clarify these questions by looking at a case study.

A while ago, we had an R&D project at our institute. This was a joint project between different institutes of our university, an airline catering company and an IT-company. The goal of the project was to deliver a new IT-enabled service for companies in the airline catering business. Andreas Meier was the academic project leader whose task it was to coordinate the scientific experts and the developers. There was another project manager in the role of the Scrum Master. He was responsible for the business stakeholders and their developers. We had a Product Owner (PO) from the senior management of the airline catering company. The PO was responsible for the items on the Product Backlog. He made sure that the items were ordered by business priority and developed correspondingly. Most importantly, there was the Scrum team, which consisted of four to eight people.

In the Scrum team, there were software developers, a user interface expert, testers and domain experts (which varied depending on the features that were to be developed during the sprint). The domain experts were from different fields in the airline business. For instance, there was a dispatcher, a chef, people from the front and back office, a lawyer and even the CEO. The domain experts all had their normal jobs to do and so it was quite a challenge to get them to work in the team. The CEO made sure, that all the domain experts understood the importance of this project and that they would sporadically be part of the Scrum team. The Scrum Master did a tremendous job guaranteeing that their time was used to the most. In the beginning there were some misunderstandings, but once the domain experts could see, at the Sprint demo, that their know-how and expertise was actually correctly transformed into working software, they became really committed.

The scientific experts had a very important part in the project. They came from the fields of Data Analysis and Process Design and were responsible for the innovation, which was new at that time for the airline catering industry. This innovation was one of the main reasons for the project. The scientific experts had already done a lot of research and had developed a prototype, which was based on artificial but not live data. In other words, they were ready and eager to get their prototype implemented and tested in a productive environment.

Here we ran into a timing problem, which is very common: The scientific experts were ready before the developers had even started to write the first line of code. In the Airline Catering project it took about half a year, before the developers could start implementing the prototype.

In Scrum, the aim of a sprint is to develop working software, i.e. something that adds value for the customer. Implementing a standalone prototype, in our case software, without the underlying components and data, is not useful to the customer. Therefore it should be avoided. In our project, the developers first had to develop working components before they could think about starting on the innovation. How can you avoid that the scientific experts get frustrated and lose focus while waiting? Our approach was to make them part of the Scrum team.

B. Tasks Performed by the Scientific Experts in the Scrum Team

If the scientific experts are part of the Scrum team, it is no longer just cross functional but interdisciplinary. Ideally, the whole team is co-located and the scientific experts take part in the daily Standup Meeting. Often that is not possible, since the scientific experts only work full-time sporadically for the project. If they work part-time, it is important that the scientific experts attend at least the Scrum Demo as well as the Iteration and Release Planning Meeting at the end of each sprint.

Which tasks do the scientific experts perform?

- They write the user stories for the features, which are part of their field of expertise. That includes the acceptance criterions.
- They write the acceptance tests for their features, preferably automated tests. Acceptance tests are useful in two ways: 1. They serve as documentation for the developer who is implementing the feature. 2. They can be run every time after new source code has been added to the source code repository[2].
- They write or support developers writing automatic unit tests. Occasionally, it is advisable that the scientific experts write low-level unit tests. This makes sense for calculations, where the output is a direct function of the input and is not dependent on some internal state of the software system.
- They do testing.
- They perform tests to further improve the usability of their features.
- They adapt the newly gained knowledge. This is probably the most important task. During the course of a software development project two important things happen: working software is written and new knowledge is gained. Both are equally important. Based on the new knowledge, better decisions can be made, which in turn might change the requirements and improve the service.

It is important to note that the knowledge of the scientific experts will spread throughout the Scrum team, which in turn results in better solutions, higher quality and customer satisfaction.

V. Improving Software Quality

A. Common techniques

There are a number of techniques known to improve software quality [8]. Both traditional and agile development

[2] This is also known as Continuous Integration (CI), one of the agile practices. Usually, the build server notices when software has been added to the repository and automatically starts building and testing the system. If there is a problem, the team gets notified. There are many CI-Systems available.

approaches have merit. Best is to use a combination of the two. The following are some of these practices:

- *Code Inspections* improve the code quality. Code inspections can take the form of peer reviews, code walkthroughs or pair programming (see below). They are best done daily and in small batches.
- *Pair Programming* is an agile software development practice in which two developers work together at one computer. The Driver writes code while the Observer reviews each line of code as it is typed in. The two programmers switch roles frequently.
- *Clean Code*. What is clean code? There are many definitions [9]; probably every programmer has one. Most seem to agree, that clean code is a simple and elegant one[3].
- *Professional testers and Quality Assurance* (QA) make sure that the system is without defects and possible problems. They should be part of the Scrum team from the beginning of the project.
- *Design Patterns* are solutions to well-known problems within a given context in software design. The use of design patterns helps to reduce defects.
- *Automatic Unit Testing*. There are a number of frameworks for automatic unit testing. The most widely used is the family of xUnit[4].
- *Test Driven Development* (TDD) is a practice in which a developer first writes automated unit tests that define code requirements and then immediately writes the code [3]. TDD ensures that the *code is right,* i.e. correct.
- *Acceptance Test Driven Development (ATDD)* is a practice in which the customer, the scientific experts and other stakeholders discuss acceptance criteria and write the acceptance tests for the system[5]. Writing the acceptance tests before development begins ensures that the *right code* is built.
- *Continuous Integration* aims to improve the software quality. For every increment to the software all tests are executed and the system is built. A build server[6] that also reports any problems back to the team usually does this.
- *Limited Work-In-Progress.* "Limit work in progress and deliver often". This is one of the core practices of the Kanban method [5]. Limiting work in progress helps to avoid that too much work is pulled into the developing process and therefore has a positive effect on software quality.

B. Agile Techniques in Practice

The "State of Agile Development" survey [11] lists the following figures about the techniques employed to improve software quality:

Unit testing is used by 70%, Continuous Integration by 54%, Automated Builds by 53%, Coding standards by 51%, Refactoring by 48%, Test-Driven Development (TDD) by 38%, Pair programming by 30% and Automated Acceptance Testing by 25% of the respondents.

VI. Conclusion

Scrum is the most widely used agile methodology in practice. It is an implementation of the empirical process model and therefore well suited for complex IT-based service development projects. Scrum is known to increase productivity, quality, and customer satisfaction:

- Increased productivity: the aim of every sprint is a new piece of working software. Progress is only measured in features that are tested and accepted by the customer. This assures that all project activities are focused on developing software that adds customer value.
- High quality: agile practices like for instance Test Driven Development, Refactoring and Continuous Integration deliver better software.
- High customer satisfaction: the ability to respond to changing requirements and managing oscillating priorities leads to satisfied customers.

In this paper, we have shown that special attention has to be paid to how scientific experts are integrated in Scrum teams and how they can actively help improve the software development effort and hence the new IT-enabled service.

Agile methodologies do not guarantee a successful project in an interdisciplinary environment – but applied properly they increase the possibility of a successful software development project considerably.

[3] In his noted book (Martin 2009) the author explains how to write good code and how to transform bad code into good code. This book has had a tremendous impact on the quality of our own development projects. We have even started teaching Clean Code in our software engineering lectures for undergraduate students with good results. The students have since been writing much better code and they seem to enjoy it.

[4] JUnit for Java, CppUnit for C++, NUnit for .Net etc.

5 Frameworks like FitNesse make it possible to run these tests automatically.

6 There are a number of continuous integration tools like Jenkins, CruiseControl etc. available.

Reference List

[1] Schwaber, Ken and Beedle, Mike (2002), *Agile Software Development with Scrum*, International Edition, Pearson Prentice-Hall, Upper Saddle River NJ, ISBN 0-13-207489-3

[2] Beck, Kent with Andres, Cynthia (2004), *Extreme Programming Explained, Second Edition: Embrace change*, Addison-Wesley, Boston, ISBN 0321-27865-8 (aka "Second White Book")

[3] Beck, Kent (2002), *Test-Driven Development: By Example*, 2002, Addison-Wesley, Boston, ISBN 0321146530

[4] Poppendieck, Mary and Poppendieck, Tom (2007), *Implementing Lean Software Development, From Concept to Cash*, ISBN 0-321-43738-1

[5] Anderson, David (2010), *Kanban: Successful Evolutionary Change for Your Technology Business,* ISBN: 978-0-9845214-0-1

[6] Cohn, Mike (2006), *Agile Estimating and Planning*, Prentice Hall, Upper Saddle River NJ, ISBN 0-13-147941-5

[7] Highsmith, Jim (2004), *Agile Project Management, Creating Innovative Products,*
Addison-Wesley, Boston, ISBN 0-321-21977-5

[8] Martin, Robert C. (2003), *Agile Software Development, Principles, Patterns, and Practices*,
Pearson Prentice-Hall, Upper Saddle River NJ, ISBN 0-13-597444-5

[9] Martin, Robert C. (2009), *Clean Code, A Handbook of Agile Software Craftsmanship*,
Prentice Hall, Upper Saddle River NJ, ISBN 0-13-235088-2

[10] Shore, James and Warden, Shane (2008), *The Art of Agile Development*, O'Reilly, Sebastopol CA, ISBN 978-0-596-52767-9

[11] VersionOne, *Sixth State of Agile Development Survey*, 2011, Online publication:
http://www.versionone.com/state_of_agile_development_survey/11/, 20.4.2013

[12] Martin Kropp, Andreas Meier, *Swiss Agile Study - Einsatz und Nutzen von Agilen Methoden in der Schweiz*. http://www.swissagilestudy.ch, 20.1.2013

[13] Scrum 101 http://www.scrumalliance.org/pages/scrum_101, 8.5.2013

[14] ScrumAlliance: http://www.scrumalliance.org/pages/what_is_scrum , 8.5.2013

Andreas Meier was born and raised in Zurich, Switzerland. Since the early 1980s, when he first got a *Sinclair ZX81* to play with, he has been interested in computers and programming. He studied computer science at the Swiss Federal Institute of Technology (ETH Zurich) and graduated 1992 with a master's degree. Afterwards he worked in the software industry for almost 10 years. In 1993 he was elected lecturer at Zurich University of Applied Sciences. He has been teaching programming and software engineering courses both on bachelor and master levels. His special interests are software engineering and agile methodologies.

Jenny Ivarsson was born in Sweden and grew up in Sweden, Denmark and Bermuda. She studied English language and literature as well as business administration at the University of Zurich where she graduated in 2002. She was working as a teacher before she became interested in agile software engineering. She is the founder and CEO of a software development company, which is successfully implementing its IT-enabled services with agile methodologies.