An Empirical Comparison of Two Agile Projects in the Same Organization

Noura Abbas, Andrew M Gravell, and Gary B Wills

School of Electronics and Computer Science, University of Southampton
Southampton, SO17 1BJ, United Kingdom
{n.abbas, amg, gbw}@ecs.soton.ac.uk

Abstract

The appearance of Agile methods has been the most noticeable change to software process thinking in the last fifteen years[6]. Although many papers, articles and books have been published about these methods, few empirical studies focus on their impact on software quality. Therefore, the main goal of this research is to investigate the quality of Agile projects empirically, in order to help software development organizations increase their understanding of Agile methods, principles and practices. This paper presents a multi-case study that was conducted using semi-structured interviews with two project teams that are using Agile methods within one organization. Our data was analyzed using the constant comparison method. The results are presented to illustrate how the teams adopted Agile methods and a comparison between the two projects is provided. From this it can be concluded that both projects were successful with multiple releases, the quality is generally seem to be as good as other projects in the same organization, the time release is reduced, and the differences between the two projects in terms of communication, the iteration length and the approach to quality, may result from the different team sizes.

1. Introduction

Agile methods are gaining interest from both academia and industry. Researchers expect to see increasing use of Agile methods for projects such as financial services, E-commerce, and air traffic control [4]. Although many papers, article and books have been published about Agile, empirical studies about how do organizations adopt Agile methods still needed. In addition, providing the evidence for what does and does not work is always needed when a new methodology is introduced. Most importantly, we need more studies about the impact of Agile methods on software quality. Therefore, we decided to empirically investigate how do organizations adopt Agile methods. The empirical study descried here is part of bigger research that aims to investigate the impact of Agile methods on software quality, in order to give a clear understanding of the topic, furthermore, to help people and organizations who work with Agile methods to produce high quality software. In his paper, we present a multi-case study where we describe the adoption of Agile methods is two projects within the same organization using qualitative research methods. The presented results are based on data collected from the 15 interviews with two teams. The paper is organized as the following: we will start with the background of the research and why such empirical studies are needed, and we will review the related work and studies. Then we will describe the nature of the empirical research and the methodology we used to collect and analyze the data. Finally we will present our results for each of the projects and will provide a comparison. We will conclude with the future work.

2. Background and Motivation

The study of software engineering has always been complex and difficult. This is mainly because of the intersection of machine and human capabilities [17]. Therefore, and because software development is a human-based activity [3] we need to apply empirical studies in order to understand important problems in the domain. Organizations need to know what are the right processes for their business, and what is the right combination of methods, and they need answers which are supported with empirical evidence. So now Agile methods have been around for a while, and they became very popular in industry, yet we still need to provide the evidence of how effective these methods are and what are the best ways to adopt them for software development.

The term “Agile” refers to a philosophy of software development [6]. This term was agreed in a big gathering when seventeen of the proponents of the “lightweight” approaches came together in a workshop

in early 2001 [9]. Under the umbrella of "Agile" term sit more specific approaches such as Extreme Programming (XP), Scrum, Crystal Methods, Adaptive Software Development (ASD), Dynamic Systems Development Method (DSDM), Feature-Driven Development (FDD), and Lean Development. Although these methods vary in practice, they all share the same principles and values. Barry Boehm defined Agile methods as "very lightweight processes that employ short iteration cycles; actively involve users to establish, prioritize, and verify requirements; and rely on tacit knowledge within a team as opposed to documentation" [4].

Survey results showed that Agile methods are gaining more interest in industry. A survey conducted in March 2007 [2] shows that Agile methods are wide spread within organizations. This found that 69% of 781 respondents worked for organizations currently using Agile methods. More interestingly many organizations that have adopted Agile methods have gone beyond pilot projects [2].

However, in a review and analysis of Agile software development methods that was conducted in 2002, the reviewers stated that there are not many experience reports available. In addition, scientific studies are hard to find [1]. Furthermore, many people claim that Agile methods are a better way to do software [5, 8, 12, 14]. However, as far as we know, there is no strong empirical evidence to support these claims. Therefore, we decided to investigate Agile methods, how organizations are using them and their impact on software quality empirically, in order to provide the evidence for what does and does not work as well as when it works in the Agile methods world.

This multi-case study is part of a research that aims to answer the following research questions:

1. How do organizations adopt Agile methods?
2. What is quality within Agile context?
3. What are the impacts of Agile practices on software quality?
4. Can Agile methods assure the quality under time pressure and with unstable requirements?
5. What are the best ways to assure the quality of Agile projects?

3. Review of Related Work

In this section we will review the empirical studies that have been conducted about Agile methods, their practices and principles. We could recognize two kinds of empirical studies about Agile methods. The first one discussed the use of an Agile method within an organization and conclude with success factors, pretty much similar to our approach in this paper.

The second one is investigating the impact of different Agile practices on software quality and how effective they are.

Following the first category we found two empirical studies about the use of Agile methods were published in the journal of Empirical software engineering in 2006. The first one discussed the advantages and difficulties 15 Greek software companies experience applying extreme programming. The study was conducted using sample survey techniques with questionnaires and interviews. The paper concluded that pair programming and test-driven development were found to be the most significant success factors in addition to interactions, communication between skilled people [18].

The second paper presented a qualitative case study of two large independent software system projects that have used extreme programming for software development within context of stage-gate project management models. The study was conducted using open ended interviews. The paper concluded that it is possible to integrate XP in a gate model context, and the success factors are the interfaces towards the agile subproject and the management attitudes towards the Agile approach [11].

Another paper with more focus on the human factor was published in the Agile Development Conference in 2005. It explored the nature of interaction between organizational culture and XP practices via three-based case studies. The paper findings suggest that XP can thrive in a range of organizational cultures and that the interaction between organizational culture and XP can be complex and subtle with consequences for practice [15].

On the other hand, a number of experiments investigated the impact of different Agile practices on software quality, such as a study about test-driven development where the results from a comparative case study of three software development projects were presented. The results showed that the effect of TDD on program design was not as evident as expected, but the test coverage was significantly superior to iterative test-last development [19]. Another study was based on a post hoc analysis of the results of an IBM team who has sustained the use of TDD for five years. The study reported that TDD practice can aid in production of high quality products[16].

Finally a replicate empirical comparison between pair development and software development with inspection using two classroom experiments and one industry experiment reported that in the classroom experiments, the pair development group had less average development effort than the inspection group with the same or higher level of quality. In the

industrial experiment the pair development had a bit more effort but fewer major defects [13].

4. The Study

Understanding a discipline demands observation, model building and experimentation. When studying a human-based activity such as software development, our research must deal with the study of human activities [3], preferably, within a real world settings. Qualitative methods are designed to study the complexities of human behaviors [17]. Qualitative data are represented as words and pictures, not numbers. Qualitative research is mainly useful when no well known theories or hypothesis have previously put forth in an area of study. As this is the case for the adoption of Agile methods and their impact on quality, and because the big goal of this multi-case study is to generate hypothesis that can be tested in future stages of the research, we conducted our case studies using qualitative methods, mainly semi-structured interviews [20]. Interviewing people provides insights into their work, their opinions and thoughts [10]. The reported results in this paper are based on interviews with 2 teams used Agile methods within the same organization. We conducted 10 interviews with 8 subjects, 5 interviews with each team. Each interview lasted, on average, 1 hour with two researchers interviewing one subject. At this stage of the research, the main purpose of these interviews is to understand how organizations adopt Agile methods and what is their approach to quality.

4.1. Data Collection

The interviews were conducted from January to November 2007. We accomplished 10 interviews in total. The interviews were conducted with members from a large organization working on two different projects. For each project we interviewed a project manager, an architect, a developer, and a tester. In addition, we interviewed the project manager twice; the first time was in early stage of the project and the second was in later stage. The main purpose of the multi-case study is have a deep understanding of Agile adoption within one organization that have different projects and approaches to software development, and to measure the quality and compare it with projects that used more traditional methods within the same company.

We used semi-structured interviews for all our subjects. We had two sets of questions, the first was about general Agile projects experience: number of projects, size of projects, working with Agile vs. traditional approaches if any exist, and how they rate the quality of an Agile project in terms of code quality and user satisfaction. The second set was about their experience in the current project: communication within the team, with customers, iteration and incremental development, and how satisfied they are with the whole process.

In each interview two researchers were present, and both took notes. In the same day of the interview the notes were reviewed and written up. Having two researchers talking notes was used in a study of COTS integration within NASA [17]. We tried audio taping couple of times; however transcription the recordings was time consuming and very expensive and the level of details we have got was more than what needed. Therefore we decided to use note-taking by both researchers which was successful in getting the required level of detail with an acceptable level of accuracy.

4.2. Data Analysis

As mentioned in the previous section the field notes were written up and reviewed. Each interview produced, on average, 8 pages (A4 size). In order to analyze our interviews we used the constant comparison method described by Glaser and Strauss [7]. In addition, we were influenced by the guidance from Carolyn Seaman to use this method for software engineering empirical research [17]. In this method we start with coding the field notes which means attaching labels to pieces of text which is relevant to a particular theme or a topic. We generated our list of code while we were reading through the data, with a big influence of the research questions. As a result we got a list of categories and sub codes (see Appendix A).

The next step was to group the passages of text into patterns and themes according to these codes. We did not cut and paste paragraphs or sentences as we did not want to lose the context of the data, instead we used MS Word find feature to trace each code. After that field memos were written to record our observations from the coded data. These field memos are the base for the results presented in the next section, and it will articulate a preliminary hypothesis to be considered in the next stage of the research.

Our results will describe each project in detail, and will discuss the main emerged themes from the coding. These themes are: the team, Agile adoption (iterative and incremental development, Agile practices, communication, and customer), quality issues and traditional software engineering (requirements, documentation, and testing).

5. Project A

Project A started in January 2007, with 2 week iteration and a high level of agility. We conducted our interviews between March and June 2007. The first release was due to be released after iteration 13 (6 month after the project started).

5.1. The Team

Project A has a team of 16 people (In both projects the team size varied over time, and the number reported here is the size at the time we interviewed the project manager for the first time), of which 12 are on-site and 4 off-site. The 12 people are mostly developers, 1 architect and 1 development manager with two sub-teams each with a lead who rotates. The testing team is off-site with a test lead on site.

The team was put together before deciding on using Agile methods. The criteria of choosing the team members was mainly the ability to deliver and work in a team, self-directed people, with high level of communication and language skills which according to the project manager are the essential skills for any project.

The team is seated in an open plan area consists of 3 bays of 4 people each. The project manager sits in one of these bays. The layout seems to work quite well, however they can't have a white board because of security reasons as they are sharing the area with other teams. This affects the communication as well as they have to respect other teams who use different kind of methods that does not involve high level of communication/interaction between the team members.

In general the team is happy, motivated and hard working. One comment was: "Shared view and ownership - yes this is good"

5.2. Agile Adoption

5.2.1. Iterative and Incremental Development. In project A the iterative and incremental approach which is the heart of Agile methods was in use. The team used 2 weeks iterations. At the beginning of each iteration they decide what to do. Each Iteration begins with a list of priorities (tasks). The first iteration was planned in details, next iteration in some detail, the others in less details (3 bullet points). They used Agile modeling on whiteboard, discussions, refine and tune the plan for the next iteration. Although the small lead team is doing the design, the whole team should understand the architecture, therefore it is reviewed by the team and continuously improved over the iterations. Decisions to drop line items are not very strict or formal; they may roll over to next iteration or reword it to close it off, although in the future they are planning to be stricter.

An interesting practice was to have an iteration for stabilization and consolidation and to improve code quality. At the time of the interviews the project was in iteration 13, and 3 iterations were devoted to this purpose. These tidy-up-iteration help to pace the work and they allow some breathing space for the team.

A team member emphasized that Agile and iterative development gives less illusion of control but we get more control in reality.

5.2.2. Agile Practices. Test-driven development was in use and it worked well for simple tasks. However, it has been helpful to have specialist testers as well as developers in the team, who have the skills to oversee all testing. Also, they used Pair programming for new team members to help integrate them in the team.

The stand up Scrum meeting was used and the team was happy about it, it helped having the shared technical understanding.

The shared understanding was present during the interviews. All team members were able to describe the process and they mostly agreed. The whole team understands what every one is going to do; also they should be able to present the overall picture themselves. It have been presented twice so far, and they would like to do this more often. The management goal is that everybody should be able to deliver the presentation.

5.2.3. Communication. Communication within Agile teams plays an important role. In project A the team used different ways of communication such as meetings, whiteboards, wikis, presentations, and chalk and talk sessions.

The team has two meetings for the iteration preparation. One for the team lead only in order to produce a straw man list of items. This list is discussed and refined in the first day of the iteration with the whole team, and the tasks are allocated to developers. In this meeting they go over the status of the previous iteration and say "well done", go through each goal and who is responsible, and schedule design sessions. During the iteration, the team has daily stand up Scrum meeting for 15 minutes. In this meeting everybody says a couple of sentences to describe what they are doing at the moment, this may lead to further communication. In addition, they have a weekly meeting for one hour for the whole team. This meeting is a good opportunity for feedback and discuss on technical issues. The senior team meets three times a week for half an hour to discuss planning issues, feedback from customer and bug lists. The off-site test

team meets once a week for half an hour through a formal phone call to agree responsibilities. In addition, a test meeting will take place on the day before the iteration planning meeting. Also, they have a weekly chalk and talk session; originally it was for learning purposes, now explaining key areas, such as how to construct trace point and exceptions. The white boards are used to record the task lists, progress of the current release and to tick the completed tasks.

5.2.4. Customer. Project A has internal customers. Developers expressed that response to customer requests is very good with Agile project; however it depends on a good customer as in some cases where you need an effort to obtain some feedback. The customer provides priorities weekly by phone calls on the day just before the iteration planning meeting. As expected, the customer's demands and requests increase throughout the project. In this project they had 2 weeks internal delivery at the end of the iteration, and it was always on time. In later stages the deliveries will be available on demands. The first external release will be after 6 month from the start of the project.

5.3. Quality Issues

Assigning one iteration to improve the quality of the code is an effective practice; in addition they are using code reviews. Small number of defects was reported so far, some are missing features, and the others are reported by internal customers. The focus was on the good-enough factor which is the right thing at the time based on current knowledge.

The team put a lot of effort on fixing bugs; sometimes it took priority over agreed goals.

5.4. Traditional Software Engineering

Although the development method was very agile in project A, we thought that it will be interesting to discuss how the traditional aspects of software engineering were integrated within the Agile project. We will discuss requirements, documentation and testing.

5.4.1. Requirements. Risk was used to priorities the requirements. Actually some simple ones were picked first to show progress, as well as the most risky ones (to reduce risk). As we mentioned in the communication section a meeting is held at the beginning of the iteration in order to select the line items, priorities them and assign them to people.

After the first two months the customers become more forceful and they start asking for more features. Team members are expecting to have firmer requirements in the future.

5.4.2. Documentation. The team keeps a history of the development (change logs, wikis, etc.) but no "static" documents. Though with traditional approaches, the documentation can easily get out of date too. The architecture is documented as power point slides, basically UML diagrams, and some text (bullet points). It is about 30 slides, 90% are diagrams. They are using class diagrams, package diagrams, and sequence diagram. In addition, they use Java doc, they have up to date list of features for users (what is available and how to use it), also they use coding standards and design patterns. The off-site test team wrote a formal document for test case writing guidelines. They experimented with taking photos of the whiteboards as well. At the end of each iteration the project manager will write a report to the senior management.

5.4.3. Testing. As motioned in Agile practices section test driven development was used by developers to test their own code. Probably all developers write test first and then try it, all should pass. Testers write functional tests and the project manager review them.

The tests team is trying to keep ahead of the developers so they can run the tests when writing the code. When the requirements are met, the test suite is enabled. Builds picks up test suits and produces the report to show status of each function, if any test fails the build fails. When the build is broken it should be fixed in around 30 minutes. The first attempt will be by the person who last checked in the code.

6. Project B

Project B started in October 2005. The interviews were held between January and November 2007. The First release was out after 10 months (Sep 2006), quicker that other products within the organization (the average is 18 months). The second release was out in May 2007. When the last interview was conducted the team was preparing for the third release.

6.1. The Team

Project B has a bigger team of 55 in total, of which 17 developers increased to 24, 20 testers, 2 architect, 2 project managers and 7 off-site. The team is divided into three smaller teams, each working in one area.

Regarding people experience, developers stated that iterative development requires experienced people, who are open to change, and with communication

skills. One interviewee commented "It will not work for people who need to be told what to do".

All team members are located in the same area, though it is not an open plan area but small offices where testers and developers often share the same office.

The team expressed personal satisfaction with the new way of working; however the high pressure might cause some conflicts. Developers don't get bored with Agile because of the constant changing which encourages them to be creative. In each iteration they had new code to write and some to maintain. Mostly the team was satisfied with the new approach as they had more input to the design and more influence to the architecture, besides they are having more fun. Another important point is that they can see the customer using their product quicker than before. In other words they can see the value of their work. This satisfaction was expressed in comments like "The team is like a democracy", "current project has more interaction", "in the waterfall days we didn't talk to anybody", "In agile 5 minutes discussion can solve the problem". On the other hand some interviewee had concerns about things going very fast, and the time pressure.

6.2. Agile Adoption

6.2.1. Iterative and Incremental Development. In Project B the team used four weeks iteration. In each iteration the team has some code to write and some to maintain, the process has developed over the iterations. Every iteration has to deliver something new

As mentioned before each iteration will last four weeks. However, in reality up to two weeks are added for testing and correcting the code. At the same time the next iteration will start so the two iterations will overlap. At the end of week four the next iteration will start and a code cut off will occur in the current iteration which will enter the fifth week where the testers will start testing the code. This means that the developers will be under pressure in the first week or two of each iteration, because they may have to start the next iteration while correcting the code from the previous one. In the first week of the iteration they will determine functionality (agree the design and the scope). Test cases and code were written in parallel (the developers wrote some unit testing) so that tests were written first. The second and third weeks are for developing. The last week is for testing – usually this week overlapped with week one (and sometimes week two) of the next iteration. During this time testers test stable code and developers stabilize code and plan for next iteration.

The first release was after 9 iterations. The team had a tidy up iteration after the first release, mainly for re-factoring. They had another tidy up iteration just before the second release. An interesting idea that the team is trying to have an iteration and release focus, which means working in themes, for example in the first release the focus was on functionality, the second on robustness. The same with iterations, in the tidy up iteration the focus is on refactoring or improving one aspect of software quality such as maintainability, extensibility or scalability.

6.2.2. Agile Practices. Refactoring was the main theme for the tidy up iterations. The team thought that it worked quite well. They didn't do a lot of pair programming, some at the beginning of each iteration. Similar to project A the shared understanding was clear during the interviews as expressed the ability to describe the process of working; also they had the same overview picture. An interviewee pointed out that this was very important "If everyone understands what is going on, this is what really matters".

When asking about code ownership, the answer was that it was ok to change other people code, even testers and developers can change (people are comfortable about it). The same applies to line items were anyone in the team can open one at any time and they can make changes.

6.2.3. Communication. The iteration starts with 2 hours meeting for the whole team. After this meeting each development team leader will have a meeting with his or her team on the same day to make sure that they understand everything and to see if they have any questions. On the second day of the iteration the project manager and the development team lead and the architects will meet for 2 hours. In the third day the project manager will meet with the architecture to agree the feature list and who's doing what. During the iteration there will be a daily walk in the area and a meeting with the architects. The project manager and the architects will meet weekly to discuss architecture reports. The architects and project manager will meet every day for an hour to focus on the external view and to decide on high level priorities. The triage meetings (with architects, testers, developers and service representative) will be held to decide what should be fixed and which to be deferred to the next iteration or the next release, also the architects have a daily meeting for an hour.

6.2.4. Customer. Because of legal issues the first delivery was after iteration 6, after that they deliver after each iteration. With each delivery, the customer is expecting something they can use. So it is important to

understand how the customer is going to use the capabilities they provide. The customer can install, use the product and send feedback or queries or even suggestions. Requirements can be changed always on customer requests. In case of requests conflicts they will follow the majority. There is an external news group to add comments and questions, this group can be shown by all customers.

6.3. Quality Issues

In project B the project manager stated that testing in the main factor to assure the quality of the product. One developer stated that he/she thinks that the quality is slightly less, another two stated that it is no worse than in other products. In this project they didn't use code reviews, but developers expressed that they would like to do code reviews as they have used them before and they were effective. An interesting comment from a tester was that although the number of reported bugs is bigger in Agile projects however they are minor and easier to fix than what they used to have in more traditional projects.

As in project A, the idea is to provide what is needed from the customer point of view. Team members stated that the project is a great success as all releases are on time so far, the defect rate is very low comparing to other products within the organization and the customers are satisfied.

The team gave a lot of time and effort on reviewing new defects and setting priorities. Although defect rate is one important aspect of quality, measuring customer satisfaction is another important aspect. Therefore, they measures user satisfaction through talking to customers and collecting feedback from them, as well as having measures for the number of reported problems.

6.4. Traditional Software Engineering

As we did in project A, we thought that it will be interesting to discuss how the traditional aspects of software engineering were integrated within the Agile project. We will discuss requirements, documentation and testing.

6.4.1. Requirements. The project manager stated that initially they were prepared to be flexible with requirements. They commit to some requirements, might do other stuff, can always change as a result of customer requirements or for sales people. He pointed out that requirements management in Agile is very critical, in order to decide what is important at the time. Requirements' prioritizing happens during the management daily meeting where they focus on external view and select customer requests.

6.4.2. Documentation. Team B did not have much design documents; however the architects provide weekly reports and power points to document the architecture. They produce good customer documentation, range of approaches are in use including Java doc.

6.4.3. Testing. The project manager indicated that the success factor in the project is the automated tests. All automated tests are executed overnight. As mentioned before, at the end of week four the next iteration will start and a code cut off will occur in the current iteration which will enter the fifth week where the testers will start testing the code. So, the testers are writing code to test the code written by the developers and most developers are writing unit tests. Test cases and code were written in parallel.

Test team structure mirrored the development team division. Testers attended design and brainstorming sessions to understand the design and to suggest testability improvements. For critical problems testers will go to talk with development team.

7. Comparison

A multi-case study with two projects of different sizes and domains within the same organization is quite interesting. In this section we will compare the two projects. We will discuss how the different variables affected each other for each project.

Table 1 summarizes the main themes for each project. The most interesting fact is the team size. We argue that the team size affected the level of communication in the team. For example in team A we can see more channels of communication within the team. In addition the whole team is involved in most of the meeting and this is understandable for a team of 12 (on-site).

On the other hand, with 55 people, team B has more meetings that involve high level of leadership (project managers, architect and teams lead); however, this doesn't affect the shared understanding and the ownership within the team.

In both projects we can see a good amount of documentation, however similar to communication, in project A we can see more documents than in project B. Probably it will be expected to be the opposite, as more documentation is needed in a larger project where communication between team members will be more difficult.

Theme		Project A	Project B
The Team	**Team size**	16	55
	Team Distribution	12 on-site, 4 off-site	48 on-site, 7 off-site
	Seating Plan	Open plan	Conventional office space
	Team Satisfaction	Satisfied	Satisfied
Agile Adoption	**IID**	Used	Used
	Iteration Length	2 weeks	4 weeks
	Tidy-up-Iteration	3 times over 13 iterations	Once every release (every 6 months average)
	Agile Practices	Scrum meeting Test-driven development Shared understanding and ownership	Refactoring Shared understanding and ownership
	Communication	Meetings Whiteboard Wikis Presentations Chalk and talk	Meetings Walking though offices
Customer	**Customer Delivery**	Internal customers Delivery after each iteration	External customer Delivery after each iteration
	Feedback	Prioritize requirements through phone calls	Feedback through emails forums
	Satisfaction	Satisfied	Satisfied
Quality	**Quality of People**	High communication skills Self-oriented	High communication skills Self-oriented
	Quality of Code	Low defect rates	Minor defects Low defects rate
Traditional Software Engineering.	**Requirements**	Start with simple ones Becoming firmer over time	Initial item list Can always change to response to customer requests
	Documentation	Change log, wikis Presentation for architecture (UML diagrams) Java doc, lists of features, design patterns Test cases guidelines Reports to senior managements at the end of the iteration	Architecture reports Customer documents Java doc
	Testing	Developers: TDD Testers: test suites	Developers: unit testing Testers: test cases

Table 1. Comparison between Project A and Project B

Interestingly the team size didn't affect the quality of the code or customer satisfaction. It only affected the communication within the team. The same apply for the seating plan; the first team had an open plan area where the other team is seated in offices.

A question arises here, is the level of communication between the team members independent of the quality including the process and product quality and customer satisfaction.

The other variable is the iteration length, for the first team it is 2 weeks where it is 4 weeks for the second team with up to 2 weeks of overlap. The question here is do we need longer iterations for bigger teams?

The final point is that project B has more developed approach to quality. Quality measures were in place for release 3, this includes defects rates, test coverage and user satisfaction measures. We do not know if this is because of the size or because the project age is longer than project A.

8. Validity

The presented study was conducted within one organization only. So it could be generalized to cover other projects within the same organization or to similar organizations. However in order to generalize the results on other organizations we need to expand our study to include projects from different companies. On the other hand the study was done with real software development on two projects of a significant size and duration.

Regarding the validly of the collected data, we did 5 interviews with each team and the participants mostly agreed with each other. In addition we had two researchers taking notes which gave our data higher level of quality and accuracy. However we had only one coder during the analyzing phase of the study.

9. Conclusion and Future Work

In this paper we presented the results of an empirical study that was conducted using semi-structured interviews with two project teams that are using Agile methods within one organization. Our data was analyzed using the constant comparison method. The results were presented to illustrate how the teams adopted Agile methods, the team organization, the approach to quality, the communication within the team and the relation with the customer. In addition we provided a comparison between the two projects.

Although the two projects were of different sizes (16 vs 55), the level of quality was not different. However we argue that the size may affect the level of communication and the iteration length and the approach to quality.

From this it can be concluded that both projects were successful with multiple releases, the quality is generally seem to be as good as other projects in the same organization, the time release is reduced.

The future work will be to conduct more interviews with different organizations in order to generalize our results and to focus more on the quality. In addition the collected data will be used to generate hypothesis that can be tested in next stage of the research using the quality measures provided by the organization. If the data are available, we will compare these measures with ones from more traditional projects within the same organization.

10. Acknowledgement

We would like to thank all participants for their time and valuable input. At the time of writing this paper we are waiting for the organization permission to mention the name and more details about the organization. We hope that at the time of the camera ready version we will get this permission.

11. References

[1] Abrahamsson, P., O. Solo, J. Ronkainen, and J. Warsta, Agile Software Development Methods. 2002, VTT technical Research Centre of Finland.

[2] Ambler, S., Agile Adoption Rate Survey. 2007, www.ambysoft.com.

[3] Basili, V.R. and M.V. Zelkowitz, Empirical studies to build a science of computer science. 2007. 50(11): p. 33-37.

[4] Boehm, B. and R. Turner, Balancing Agility and Discipline: A Guide for the Perplexed. 2003: Addison-Wesley Longman Publishing Co., Inc. 304.

[5] Cockburn, A. and J. Highsmith, Agile Software Development: The Business of Innovation. Computer, 2001. 34(9): p. 120-127.

[6] Fowler, M., The New Methodology. 2005, www.martinfowler.com.

[7] Glaser, B.G. and A. Strauss, The Discovery of Grounded Theory: Strategies for Qualitative Research. 1967: Aldine Transaction

[8] Highsmith, J., Agile Software Development Ecosystems. 2002: Addison-Wesley Longman Publishing Co., Inc. 404.

[9] Highsmith, J., k. Beck, A. Cockburn, and R. Jeffries. Agile Manifesto. 2001 [cited; Available from: www.agilemanifesto.org.

[10] Hove, S.E. and B. Anda, Experiences from Conducting Semi-structured Interviews in Empirical Software Engineering Research, in Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS'05) - Volume 00. 2005, IEEE Computer Society.

[11] Karlstr, D. and P. Runeson, Integrating agile software development into stage-gate managed product development. Empirical Software Engineering, 2006. 11(2): p. 203-225.

[12] Larman, C., Agile and Iterative Development: A Manager's Guide, C. Alistair and H. Jim, Editors. 2004, Pearson Education, Inc.

[13] Phongpaibul, M. and B. Boehm, A Replicate Empirical Comparison between Pair Development and Software Development with Inspection, in First International Symposium on Empirical Software Engineering and Measurement. 2007: Madrid, Spain.

[14] Poppendieck, M. and T. Poppendieck, Lean Software Development: An Agile Toolkit. 2003: Addison-Wesley Longman Publishing Co., Inc. 240.

[15] Robinson, H. and H. Sharp, Organizational culture and XP: three case studies, in Agile Development Conference. 2005, IEEE Computer Society.

[16] Sanchez, J.C., L. Williams, and E.M. Maximilien, On the Sustained Use of a Test-Driven Development Practice at IBM, in Proceedings of the AGILE 2007 (AGILE 2007) - Volume 00. 2007, IEEE Computer Society.

[17] Seaman, C.B., Qualitative methods in empirical studies of software engineering. IEEE Transactions on Software Engineering, 1999. 25(4): p. 557-572.

[18] Sfetsos, P., L. Angelis, and I. Stamelos, Investigating the extreme programming system---An empirical study. Empirical Software Engineering, 2006. 11(2): p. 269-301.

[19] Siniaalto, M. and P. Abrahamsson, A Comparative Case Study on the Impact of Test-Driven Development on Program Design and Test Coverage, in International Symposium on Empirical Software Engineering and Measurement. 2007.

[20] Wohlin, C., P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, and A. Wessl, Experimentation in Software Engineering: an introduction. 2000: Kluwer Academic Publishers. 204.

Appendix A - List of Codes

Agile Adoption	
AA-CT	Communication within the team
AA-CC	Communication with the customer
AA-DC	Delivery to the customer
AA-OST	Off-site teams
AA-DTS	Developing team skills
AA-MET	Meetings
AA-PLN	Iteration planning
AA-GOOD	What is good about agile
AA-BAD	What is bad about agile
AA-CUL	Culture issues
AA-PRO	Process
AA-SU	Share understanding
AA-OWN	Ownership
AA-BV	Business value
TI-UP-IT	Tidy up iteration
Agile Practices	
AP-TDD	Test driven development
AP-PP	Pair programming
AP-IID	Iterative and incremental development
AP-XP	Extreme programming
AP-SCR	Scrum meeting
AP-CI	Refactoring
AP-CRC	CRC cards
Quality	
Q-CODE	Quality of the code
Q-PPL	Quality of the people
Q-T	Relation between quality and the time
Q-DEF	Defects
Q-CS	Customer satisfaction
Q-MEG	Quality measures
G-EN	The Good Enough
P-SUCS	Project Success
M-SUCS	Measure of success
Software Engineering	
CR	Code review
REQ	Requirements
DOC	Documentation
TEST	Testing
ARCH	Architecture
BUG-R	Bugs removal
PP	Project progress
P-REQ	Prioritising requirements
LI	Line items
AT	Automated testing

People Issues	
OI	Organizational team
DT	Development team
DT-SKILLS	Development team skills
DT-ORG	Development team organization
SP	Seating plan
ROLES	Roles
R-T-D	Relation between test team and development team
MT	Moral of the team
S-O-T	Size of the team
TT	Test team
TT-ORG	Test team organization
TT-SKILLS	Test team skills
TS	Team satisfaction