The Level of Agility in Testing Process in a Large Scale Financial Software Project

Marian Jureczko

1Capgemini Polska Sp. z o. o., Legnicka 51-53, 54-203, Wrocław, Poland
2Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland

Abstract. Software testing is a very important part of agile software development methodologies. Unfortunately, few, real, large-scale projects that implement agile quality processes have been reported and currently there is a lack of data about agile testing. This paper describes the testing process in a real, large-scale, financial project, compares the process with principles of agile testing and discusses the differences. Each of the discovered differences is carefully analyzed and the advantages and disadvantages, that come from the difference, are depicted.

Keywords: Agile software development, agile testing, large-scale software project, scaling agile.

1 Introduction

Agile software development methods are easy to implement in small projects, that are realized by small companies. There are many examples and empirical proofs for the facts, that agile methodologies work better than the traditional in small projects. But unfortunately, when the scale of projects grows, problems are appearing. There are many papers, that discuss problems with scaling agile [4,6,9,13,18,19,20 and many others]. Those papers refer many different areas of scaling. There are described problems with communication in a large team, coexistence with other, “traditional” projects in a shared environment, cooperation with internal customer and finally agile testing. What is the reason for this paper when everything is already so well depicted? In fact, there are not many empirical data that come from real project. Most of the mentioned earlier papers describe experimental projects, that were developed under special conditions and when it comes to figures, it turns out, that the investigated projects was not very large.

This paper is organized as follows: the next section describes the research setting, Section 3 discusses related literature. Section 4 briefly introduces the most important rules of agile testing. Then testing process and testing tools, that are used in the Futura development project, are presented. The fifth section discusses differences
between agile testing and the testing process in the Futura development project. Some concluding remarks are presented in the final section.

2. Research setting

This section describes the research background and flow. In Section 2.1 is described the investigated project. In Section 2.2 is depicted the research method.

2.1 Research background

The author was a participant observer in a very big team, that develops a large scale software project for a real, external customer. The project is being developed since 2002. At the beginning it was planned for 150 working years, but later the required functionality has been increased and the duration time was lengthened. More than 150 high skilled developers, testers and managers are currently involved in the project. The project is divided into five subprojects. This paper is concentrated on only one of the subprojects. The subproject is being developed by a group of about 40 people.

Project is being developed for one of the biggest German social insurance companies – BVK (Bayerische Versorgungskammer). The system, called Futura, must allow to manage data about 1.3 million insured people. The ultimate goal of Futura is to give to the financial institution competitive advantage in the domains of customer intimacy and cost efficiency. As a custom build solution it aligns tightly to the business processes and requirements of the industry. It supports more than 1500 case scenarios initially identified. As in many European economies, German regulations of social insurance industry are complex and are still evolving. The Futura system is based on well defined technical framework, called Quasar. Hence, writing source code is focused on solving business requirements, rather than technical obstacles. The Futura development team delivers new releases with new functionalities frequently. There are two major releases per year: in spring and autumn. They are used to deliver large changes and bulks of eventual bugs. The two major ones are supplemented by hotfix releases. They target the most important bugfixes only.

2.2 Research method

The research approach is a qualitative approach. The author sought to understand the testing process from the participant point of view. Additionally the team members were asked about principles, that they follow during the development and testing process. It was an insider perspective. The author played a role in the software development process.
3 Related literature

Scaling agile is a quite popular topic. There are many papers, where the scaling process is discussed. A summary of difficulties, that arises from scaling agile methods to fit large projects is depicted in [18]. The overview of four pilotage installations of agile practices in large organizations (ABB, DaimlerChrysler, Motorola and Nokia) is presented in [13]. One of those four experiments is detailed in [4]. Results of those experiments may encourage to install agile practices in large organizations but it depicts some difficulties too. In a large organization, a project can not be truly independent, but must coexist with other projects (some of them may be developed in the traditional way) and must obey the rules of the organization. Lindvall [13] enumerates also several other risks, that were discovered in the experimental installations of agile practices:

- large refactorings may create significant project defects (but it should not when the project is well covered with regression tests),
- creating user stories leads to double work when an interaction with a traditional approaches (another projects, or another departments in the organization) occurs, the traditional one may require traditional documents,
- pair programming is not always as good as formal reviews,
- becoming agile too fast may turn a mature organization into a chaotic one.

Each of those risks must be considered before the successful installation of the agile software development process.

This paper is concentrated on agile testing in large projects. That is a quite wide topic and there are many papers that covers some areas of agile testing. Johansen [7] described the process of establishing an agile testing team. Some problems with interaction with other involved in project teams, that were used to traditional testing, are solved in [7]. However, the agile testing team become a quality assurance division in [7]. The area of responsibility of the quality assurance division differs from area of responsibility of an agile testing team. Probably the traditional culture dominated over agility and achieved result is only a successful compromise – it works, but it is not pure agile. It shows how difficult may be the transformation from traditional testing to agile one. Puleio [17] depicted that the transformation is especially difficult when developers and testers had experience in traditional methodologies and do not know the agile approach. Despite of many difficulties Puleio [17] presented a successful transformation of the testing process – from traditional to agile, but it is transformation of small team (only five members).

Talby [20] described installation of the agile software testing practices in a large-scale project in a traditional environment. The way of working of the quality processes after adjusting them to the principles of agile software development is discussed in the paper. More details about software project from [20] can be found in [6]. There are some indirect information about the scale of the project. In fact, the project is a subproject of larger one. The larger one was developed by 60 skilled developers and testers. We can only guess how many people were involved in the subproject, but for sure it was not a very big team.
4 Principles of Agile Testing

The Agile Manifesto does not deliver values, that are directly connected with testing, although testing is a very important part of most of the agile methodologies. XP [1,3] gives a very important principle: Test-first programming (TDD). The way of using TDD is detailed in [2]. Formal definition of a TDD based process is presented in [8]. Beck [3] wrote a whole chapter about testing: Testing: Early, Often, and Automated. Name of this chapter contains the most important features of XP testing. We should start test as early as it is only possible – before we start writing source code. We should run the test very frequent – before every source code integration (source code integrations occur very often because of another XP principle: Continuous Integration) and before every release, internal too (releases are frequent because of principle: Weekly Cycle).

![Fig. 1. Coding and testing in XP](image)

Automation is essential in agile testing. The summary of chapter Manual Tests in [5] is only one sentence: No manual tests. Why automated tests are so much better than manual tests? Manual tests use a lot of resources. A group of high skilled testers must test for several hours or days and despite of this, the tests results are less reliable than results of automated tests. According to Crispin [5], people used to cut corners, omit tests and miss problems. According to Beck [3], manual tests are stressful. Finding new errors in the software project increases the level of stress in team. The more under stress developers are the more errors they make. It is a vicious circle, that leads to decreasing product quality. Manual tests cannot be performed very often because of the duration time and high level of resources consumption.

There are XP principles, that are indirect connected with testing. Pair programming should assure early detection of defects and high quality of source code and automated tests without formal reviews. Weekly cycle and Continuous Integration force frequent testing and give early feedback. Root-Cause Analysis decreases the probability of repeating similar errors and decreases the level of defects in future. Real Customer Involvement assure that we develop the right system and increases the quality of acceptance tests.

According to Johansen [7], there are important values, which should guide us during establishing an agile testing team: individual and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, responding to change over following a plan. Authors of [7] established an agile testing team. They were basing on those values and they achieved interesting results. They decided not to test everything – a
try to test as widely as possible could only deliver worse performance. They did not
decide to resist on obeying all quality standards. This approach has a big advantage.
When something was absolutely necessary in their opinion, developers gave them
more support. In fact they were extremely agile, but in a totally different way than it
is in XP.

The described set of rules should be enlarged with communication – weak
communication leads to problems with cooperation between testers and developers
[17]. Solution is given in [15]: testers and developers should sit in the same room.
Full integration of testers and developers was a productive choice in project
described in [20] too. Another point to communication: tests results should be
reported via public and easy to read charts. According to Marick [15], the evolution
of tests plan is very important in agile testing. Test plan can not be fixed. It should
be modified adequately to requirements changes and difficulties encountered in
development.

5 Testing the large scale financial software project in the Capgemini Company

Testing tools play important role in the testing process. Hence this chapter is divided
into two sections. First section describes the process, the second is concentrated on
tools.

5.1 Testing process

There are some general principles about quality management, that are used in the
company. All project must obey those principles. Those principles force a project
structure and define quality related roles. The structure is depicted on fig. 2.
Following roles are strongly connected with quality:

- Quality Manager is responsible for installing the company quality policy. He
  should verify if the quality policy is understood and followed at all levels. Quality manager is also the person, who improves the quality
  processes.
- Quality Consultant advises employees in quality policies application. He
  should carefully monitor the software development process and detect all
  quality problem as soon as it is possible.
- Quality Supervisor is responsible for quality assurance. He creates the
  quality management plan. He monitors the implementation of the quality
  assurance according to previously created plan and he regularly informs
  project manager about the status of quality assurance.

The rest of this section concerns on testing in a subproject. Project organization,
that is presented above, is just a background for testing process in the subproject.
Testing process in subproject utilizes many agile practices. The tests are performed
as soon as possible. When a feature request arrives, the specification is discussed
with the customer to create a formal document with definition of the new
functionality. When the specification is ready, testers start with test concept. The test
concept is consulted with customer too. With well written specification and test concept, the implementation can be started. Developers write source code and perform manual tests when testers write new automated tests. The new automated tests are based on the previously written test concept. Each of the tests is immediately added to the regression tests set. Regression tests are executed daily. Information about results of execution regression tests are available for developers. The results contains information about bugs and not yet implemented features. It is not possible to execute regression tests before every change in source code repository. Execution of the regression tests lasts several hours.

Creating new automated test cases is a complicated activity. Often developers are ready with implementation, but not all tests from test concept are automated. Sometimes the test automation is postponed and the not automated yet tests are performed manually because of the deadlines. Before every release (internal too) there is a code freeze. It is a short time period, when changes in the central source code repository are forbidden. The time period is necessary to integrate source code from branches. Yes, there are branches. The branches are necessary because of project size. Each of these subprojects is developed in different branch. Sometimes it is also profitable to get separate branch for each team. One subproject is developed by more than one team. The Futura system has been already installed in customer environment. Hence, there is a productive branch too, where is the version of system, which is currently used by customer.

Subsystem tests are performed after the integration. Subsystem tests are usually manual, and they are concentrated on new features. If a subproject has a release an extra step must be perform in all others subprojects, that depend on the subproject – integration tests. According to Myers [16], one of the efficient test methods is error

Fig. 2. Project organization, the quality point of view

Information flow
Organization hierarchy
guessing (based on tester's experience). The integration tests are performed in this way. The integration tests and regression tests are able to indicate the quality level.

The sequence of activities in a iteration is depicted on fig. 3. Those activities are performed for each feature request separately. Hence the sequence may be performed several times in one iteration.

Communication
Communication between developers and testers is very important in the tests process. Developers and testers work in the same open space area. They have got lots of opportunities to conversation.

New features usually require a workshop, that introduces the team into the problem domain. Testers and developers are trained together. A shared environment of domain knowledge is being tried to be build. Using workshops to ensure good knowledge flow is recommended in [9] as a ideal solution for agile methodologies.

Good communication between developers and testers leads to other advantages too. Good communication has a positive influence on the software quality. It works almost like pair programming. When two people (tester and developer) have to solve a problem, it is unlikely that they both will solve it in the wrong way. Hence, the software quality is better. When tester and developer work together, it is possible to start testing earlier and the bugs are found and removed faster.

**Fig. 3.** Sequence of activities in iteration
Automation

The testing process in Futura project is not as extreme with automation as it is suggested in [5]. There are some manual tests too, but the advantage of using automated tests and of not using manual tests is fully understand. Unfortunately creating new and maintenance of existing automated tests cases requires a lot of time and there is not always the possibility to postpone deadlines. Some things are very difficult to automate and it is better to test them manually – the cost of automation could be too high.

Performance tests create another automation problem. Execution of one performance test lasts several days and requires enormous amount of data. The test conditions depend on system architecture (data model). Hence, all data model changes must be applied to the test conditions. When the size of the test conditions is equal to several GB, applying changes may be very difficult. Automation of performance tests is not acceptable because of those problems. Such a long execution time and such a big effort with handling the test condition data makes tests useless. In fact, even Puleio [17] had not automated everything (Pueilo considered test automation as essential practice in agile software development methodologies).

The automated tests are used as regression tests. The structure of the automated tests is similar to acceptance tests. The tests simulate user actions and check the results. Positive results of tests ensure, that all system functionalities still work correct. Hence, the regression tests play almost the same role as acceptance tests in XP methodology.

RUP influence

Is RUP an agile methodology? People, that are involved in developing RUP, consider this methodology as an agile one [11], but others are not so sure about it.

Some RUP practices have been installed in the testing process. According to [10 and 12], there are two parallel flows in the RUP testing discipline. The traditional testing is in the first flow. Only one activity is in the second one - verify test approach. The verify test approach activity is important in the Futura project too. The test approach is being constantly analyzed and is modified, when a possibility of improvement is noticed. Recently a discovery has been made, applying the feature driven approach to create new test cases (new test cases is created when a feature request comes from customer) leads to irregular tests coverage. Currently some tests concepts are being prepared to fulfill the testing gaps.

Verification test approach is not far away from agile testing. Marick [15] recommends evolution of test plan as one of the agile testing principles. In the Futura testing process, the test plan is not fixed too. Usually only new tests are added, but every modification is allowed as well.

Soft roles

Talby [20] wrote: In a traditional project, everyone is responsible for quality, but in agile projects, everyone actually writes tests. This rule is very difficult in a large scale project – every person must be very good in writing source code and in writing tests. It means, that every person knowledge must be very big. In large scale project developer must have got knowledge about the whole (or at least a big part) project to
write good source code and tester must know a lot about test framework and system specification to write good tests. It is much easier to be skilled in only one area.

Connecting testing and developing without pair programming leads to another bad situation. In Myers [16] opinion, it is strictly forbidden to allow the some person to write source code and to test.

Testers and developers are in the same team but they are not the same people. However the developers are encouraged to write tests and testers get sometimes a developing task. Developers utilize 80% of their time to develop the system and testers utilize 80% of their time to test, but they get also tasks, that show what other people are doing. The result is impressive. The team members understand what their colleagues are doing. When you understand other people, you can better cooperate with them. Sometimes an implementation task force team is set up inside the team to solve especially difficult problems. Differences in setting up the implementation teams depend on scope, bulk, level of complexity and level of cross boundary impact of changes.

5.2 Testing tools

A test framework has been created for the Futura project. The test framework is developed simultaneously with the project. There is a possibility to add new features to test framework, to adjust it better to the project. The project specification and the system architecture is changed sometimes. New, previously not planned features, are added, but the test framework still works good. If it was not working good enough, it could be improved.

The test framework consist of two application. The first one, called TMDB, is dedicated to constructing test plans and to managing with existing tests. The second one, called TDB is dedicated to constructing automated acceptance tests. TDB works with standalone tests and tests suits as well. A standalone test is dedicated to test one scenario of a use case. Test suit is a sequence of standalone tests. Each of tests has conditions, that defines the state of system before test execution. System size induces number of entities in tests conditions. There are usually about thousand entities in test conditions of one tests. However preparing the test conditions is not connected with a lot of work, because it is a semiautomated process. Each of the standalone tests simulates some users activities and checks results of those activities. TDB has got an open architecture, hence there is possibility to write plug-ins, which check non-typical tests results. Currently the data that is presented in gui (results of counting) is checked. Structure of documents that are printed after successful realization of a use case and changes in data base are tested by plug-ins. Test cases definitions and tests suits definitions are written in XML files, and are stored in the version control system. The test framework is integrated with an automated, continuous build process. Summary of regression tests execution results is presented daily on a web page. The web page is available for all developer and tester. Test framework is integrated with the development environment. Launching a test case is a one click task.
6 Differences between Agile Testing and testing in the large scale, financial, software project

The most significant principles of agile testing are listed below. Each of those principles is compared with testing process from the Futura project and differences are depicted. The reason why certain agile practices fit well and why others do not are briefly discussed. A level of adoption is presented for each of agile testing principles. A principle can be not adopted – 0; can be adopted partly – 0,5; and can be adopted fully – 1.

Test driven development

TDD has been popularized by extreme programming. It is a very interesting approach with many benefits, but it is very risky too. Installing TDD in a large scale project, that was developed in a different way at the beginning, is almost impossible. Benefits from TDD are taken, but carefully, without increasing the risk. The result is a compromise. Test concept is written before source code, but tests automation sometimes takes place after source code implementation and the productive code may be ready earlier than the tests.

Some features of the system are difficult to test. Not using TDD allows skipping automation and performing some tests manually. Hence, software can be developed faster without decreasing quality level. It does not lead to decreasing regression tests value, because the automation is postponed only.

TDD is adopted partly (TDD=0,5).

Test automation

Test automation is essential in agile testing and the advantages of automation are fully understood in the Futura project. There is automated as much, as it is reasonable. There is an automated builds system based on the Cruise Control tool. All regression tests are executed during every build and the tests results are available to all developers, testers and managers. The principles of agile testing recommend to automate all tests, but according to Puleio [17], there is always something, that is impossible to automate and in a large scale system the number of not automatable things rise. The Futura system is very large and there are many not automated tests. But each of those tests is not automated because of an important reason: the automation may be too complicated, may require too much time and some thing just can not be automated. The level of automation is high because of very well adjusted test framework. The test framework has been developed especially for the Futura project. Hence, it is able to cover many Futura specific functionalities.

Benefits from automation are taken where it is possible. But the team is not strict with automation. When something is not worth it, it is skipped. This approach works great. The deadlines are not exceeded and high quality level is kept. Test automation is adopted fully (TA=1).

Continuous integration

The integration is continuous only on the level of subprojects. Continuous integration of the whole project may be risky. It may slow down work in the subprojects. Currently a customer model is being used. Subproject depends on
another subproject so it plays the customer role for the other subproject. The subprojects are integrated only when an release is made (internal or external). In continuous integration, members of a team should integrate their work frequently. Usually each person integrates at least daily. The team members have the possibility to integrate whenever they want, except short time periods called code freeze. The integrations are verified nightly by an automated build and execution of all regression tests.

In large scale projects continuous integration is more difficult than in small projects. Time period between 'check ins' is smaller. Keeping workspace up to date needs greater effort. Developers get more work with merging source code. Dividing the large scale project into several parts make the development work more effective. Creating branches anddropping continuous integration in the scale of whole project is one of the methods of dividing a large scale project.

Continuous integration is in opposition to branches. Branches create difficulties, but currently there are no other, easier solution. The project is so closely coupled with branches, that dropping them would be currently too risky. Dropping branches must be a continuous and slow process. Continuous integration is partly adopted (CI=0.5).

Communication
The significance of good communication is well understood. Good communication is assured on two levels: communication with customer and communication between team members. Communication between team members is problematic, because the teams are spread geographically. The gap is fulfill with periodical video meetings and phone conversations. With team members, that are not geographically spread, the principles of agility are followed – the team members are sitting in an open space area.

The customer is not 'on site' all the time. Meetings with customer representatives takes place only when they are necessary. But that does not mean, that there are communication problems. There are several customer representatives, that are constantly under the phone. There is always somebody available to discuss the problems. The communication with customer works good, but with 'on site' customer it may be better.

Scale of project has a significant influence on communication. But communication problems in large scale projects are very well covered in literature. Some of the solutions has been adopted in the Futura project (one team plays customer role for the other team, not each of the teams has contact with real customer).

The communication can be a bit more agile, but without doubts the communication principle is adopted (C=1).

Pair programming
Not each of the agile software development methodologies recommend pair programming. The positive impact of pair programming is also questionable. Bowers [4] concludes: The likelihood of pair programming eliminating all mistakes during coding is small. He suggests supporting pair programming with formal reviews. Without formal reviews source code may contain many defects. Madeyski [14]
presented an overview of several empirical studies, that investigate the impact of pair programming. Results are not astonishing. A pair speedup ratio is between 20% and 40% in comparison with a single programmer. According to Madeyski [14], there are no empirical evidences, that pair programming practice has an impact on software quality. According to Bowers [4], there are some organizational difficulties too. Sometimes it may be difficult to find a pair and according to XP principles, developers are not able to code without a pair. If there were empirical evidences, that pair programming is better than solo programming, the developers would be already programming in pairs. Formal code reviews are more valued in the Futura project than the pair programming. Pair programming is not adopted (PP=0).

The Futura project does not answer if the pair programming can be applied in a large scale project. There was no attempt to install this practice.

Root-cause analysis
Root-cause analysis is interesting and may be very helpful, but even Beck [3] agreed, that it is dangerous without installing other, essential XP practices. This practice forces complex analysis for each of committed defects. We should discover not only how the defect can be resolved, but also how the software development process can be improve to avoid similar defects in future. Without other XP practices, there are too many defects to perform a complex analysis for each of them. According to Beck [3] root-cause analysis is useful when only one defect appears per week.

The Futura project do not utilize the full XP. Only some practices from XP are taken. The root-cause analysis may be very danger. Install of other XP practices is necessary before a successful root-cause analysis adoption, some of the practices are not connected with testing. Currently a complex analysis is perform for very few defects. The practice require complex analysis for each defect so root-cause analysis is not adopted (RCA=0).

Working software (over comprehensive documentation)
The developed system has been already installed in customer environment. Every feature request must be applied to working software. When it is not done, customer requirements are not fulfilled. Customer measures progress of development as the number of new functionalities in the system and the number of resolved bugs. Implementing new functionalities requires creating new source code. Resolving bugs requires modifying existing source code. The source code is compiled into working software in both cases. The working software must be valued over comprehensive documentation. But on the other hand it is very difficult to develop a large scale project without documentation. Many people are involved in such a project. Documentation may be used as another communication channel. Documentation is helpful in communication with customer too. Documentation does not have to be comprehensive in large scale projects, but cursory (or lack of) documentation may be even more risky than the comprehensive approach. Documentation plays important role in the Futura project but does not dominate over the working software. Working software is partly adopted (WS=0.5).
Responding to change (over following the plan)
Responding to a change is easy to implement in the domain of testing. Tests, even the automated, are much easier to change than the source code. Hence, changes are always applied. The test plan is an evolutionary artifact. Old test cases may be modified and new test cases may be added during and after the implementation. Responding to change is adopted (RS=1).

Level of agility in testing process
The level of agility in testing process can be counted from level of adoption of the principles of agile testing. Not all principles has the same value from the testing point of view, so weights were used: 3 – TDD, test automation; 2 - continuous integration, communication; 1 - pair programming, root-cause analysis, working software, responding to change. The weighted average of agile testing principles adoption is 0,64% (AT=0,64). The figure is informal and subjective. It creates only an overview of the level of agility in testing process.

\[
AT = \frac{3 \times TDD + 3 \times TA + 2 \times CI + 2 \times C + PP + RCA + WS + RC}{14}
\]  

7 Summary and Conclusions
The comparative analysis of agile testing and the testing process in a large scale financial project in Capgemini company did not show many differences. The most important practices have been already adopted, however not all practices are adopted fully and some, but not the essential, are skipped. Quality is very important in the software development process in Futura project and new, risky practices are always installed very carefully. Hence, only those agile testing principles are used, that increase the quality level without any doubts.

The testing process is not fixed. Improvements are constantly in demand. The last months show, that agile testing is a promising direction. Recently we have increased the significance of test automation. Currently the source code coverage is increased by writing new test cases and test suits. Recently the value of sharing domain knowledge has been rediscovered [9] and currently the domain knowledge creation process is being improved by performing workshops. The testing process in Futura project is going towards the agile testing. Probably, in the near future, the process will be more agile than it is now.

Some principles of agile testing were not installed. Those principles were not helpful in the Futura project, but it is hard to decide, if those principles can not be used in other large scale projects. They may work better in projects, that are developed in other environment. Some agile practices were not used in the Futura project for reasons, that are depicted in previous section. This explanation may be useful in configuring software development process. It gives information about potential difficulties and dangers, that appears during adoption agile practices.
References