An Empirical Study and Impact of Software Quality in Software Development

Zuhab Gafoor Dand
School of Science & Technology/Department of Computer Science, Shri Venkateshwara University, U.P, India

Abstract— In this paper, we empirically investigate the impact of responsive methods on different aspects of quality as well as the different factors that affect these aspects. Quantitative and qualitative research methods were used for this research, including semi-structured interviews and surveys. Quality was studied in one of the project that used responsive software development. The empirical study showed that the project was successful with multiple releases. The data analysis produced a list of 13 refined grounded hypotheses out of which 5 were supported throughout the research. The project was studied in-depth by collecting quantitative data about the process A variety of statistical analysis techniques were applied and these suggested that when responsive methods have a good impact on quality.

***Index Terms*—Software Quality Assurance, Extreme Programming, Statistical Analysis, Software Development**

I. INTRODUCTION

Software development is gaining interest from both academia and industry. Researchers expect to see increasing use of responsive methods for projects such as financial services, E-commerce, and air traffic control (Boehm 2002). It was used to form the initial research questions: what is the impact of responsive software development on software quality? And what factors affect quality, and project success when using responsive software development approaches? In order to answer the research questions we had to fully understand what is responsive software development, and how did the idea of responsive software development evolve? In addition, we had to understand what is quality for responsive software development? The literature review showed that the available quality models were based and designed for traditional approaches to software development, mainly the sequential or the waterfall model. Therefore, we argue that there is no systematic way to integrate quality assurance within a responsive method. In a systematic review about the empirical studies of responsive software development (Dyba et al. 2008), the authors concluded that that there "is a need for more and better empirical studies of responsive development within a common research agenda", still these are mainly focused on one responsive method, namely Extreme Programming (XP). As software development is a human-based activity, the best way to study this activity and to get valid applicable results is to apply empirical approaches within real world settings. Therefore, this approach was chosen for our investigation. It was therefore decided to find a case study. The project manager of one of the responsive project in Merge Software Solution agreed to be interview for the research. Moreover, the analysis of the interviews produced 26 hypotheses about the impact of software development on the different aspects of software quality. This list was refined and reduced to 13 hypotheses to be selected as the focus for paper. In addition, the empirical study enabled us develop and trial a new technique for quality: The iteration monitor. This monitor was designed to first test the produced hypothesis, and to collect data about the iteration to understand how things are changing over the iterations. In addition, analysing the data yielded in statistically tested relationships between the different aspects of the iteration. The three responsive methods which are commonly used are

Extreme Programming (XP)

XP is the most widely recognized responsive method (Boehm et al. 2003). Kent Beck developed this method during his experience with the C3 project (Comprehensive Compensation System). XP practices were originally intended for small, collocated teams. Although some practitioners like Kent Beck and Ron Jeffries may envision that XP can be extended to larger teams, they do not try to convince people that it can work for teams of 200 (Highsmith 2002). XP is based on four values, communication, simplicity, feedback, and courage. XP practices include pair programming, continuous integration, refactoring, test-first programming, and user stories (Beck et al. 2004).

Scrum

Ken Schwaber and Jeff Sutherland developed Scrum when they realized that software development is an unpredictable activity. Scrum, along with XP is one of the most widely used responsive methods (Ambler 2006). This method defines a project management framework, managed by the Scrum master. Scrum is one of the few responsive methods that have been scaled up to medium projects (Boehm et al. 2003). In Scrum, the iteration length is 30 days and it is called a "sprint". The sprint will be preceded by pre-sprint planning and will be followed by a post-sprint meeting. Scrum practices include the daily scrum meeting and product backlog (Schwaber et al. 2001).

Crystal Methods

Crystal is a family of methodologies that was developed by Alistair Cockburn. According to him, there is no one Crystal methodology but different Crystal methodologies for different types of projects. The factors that influence the methodology selection are staff size, system criticality, and project priorities. Crystal Clear is an optimization of Crystal family and it is targeted at projects where the team consists of two to eight people sitting in the same room or in adjacent offices. Cockburn stated that Crystal Clear shares some characteristics with XP but it is less demanding (Cockburn 2005).

Software Quality

The IEEE Glossary (IEEE 1990) defines software quality as:

1) The degree of which a system, component, or process meets specified requirements
2) The degree to which a system, component or process meets customer or user needs or expectations.

These two definitions reflect Crosby's (conformance to requirement) and Juran's (fitness for purpose) definitions.

Software quality will mean different things to different people. Each stakeholder has his/her views about quality. The user will see quality as "what I want", "fast response" or "cheap to run" where from the designer point of view it can mean "good specification" "technical correct" or "well documented" (Gillies 1992).

The de facto definition of software quality consists of two levels; the first is product quality, as limited to product defect rate and reliability. (Kan 2002) refers to this narrow definition as the "small q". The broader definition includes product quality, process quality and customer satisfaction, which is referred to as the "big Q". This definition has been used in a number of industries, such as automobile, software, hardware, and consumer electronics. Now although the final product confirms to requirements, it might not be what the customer wanted, therefore, the customer should define the quality as conformance to the customer's requirements.

Another view is from (Gillies 1992) who stated that "quality is people". He explained that this is because quality is determined by people who are facing the problem to be solved by the software, people will define the problem, people will find the solution and people will use the system and make judgments about the quality.

Gillies stated that software quality does appear to be particularly problematical when compared to other arenas. According to a group of professionals, this is because 1) software has no physical existence 2) the clients do not know exactly what they need 3) clients need change over time 4) the change in both hardware and software is very fast and 5) the customers have high level of expectations.

Software Quality Assurance (SQA)

The IEEE Glossary (IEEE 1990) definition of SQA is:

1) A planned systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established functional technical requirements

2) A set of activities designed to evaluate the process by which products are developed or manufactured

II. RELATED WORK

The study of software engineering has always been complex and difficult. This is mainly because of the intersection of machine and human capabilities (Seaman 1999). Therefore, and because software development is a human-based activity (Basili et al. 2007), we need to apply empirical studies in order to understand important problems in the domain. Organisations need to know what are the right processes for their businesses, what is the right combination of methods, and they need answers that are supported with empirical evidence. However, there is a "need for more and better empirical studies of responsive development within a common research agenda" (Dyba et al. 2008). Therefore, we decided to use empirical methods to investigate the impact of software development on the different aspects of quality. The empirical studies reviewed previously gave us an idea about the different research methods used by different researchers when conducting empirical research. In order to broaden our knowledge regarding empirical software engineering, we took a look at the available references such as (Basili et al. 1986; Basili 1996; Seaman et al. 1998; Seaman 1999; Perry et al. 2000; Wohlin et al. 2000; Kitchenham et al. 2002; Sharp et al. 2004b; Zannier et al. 2006; Basili et al. 2007). We focused on studies that investigated responsive methods The systematic review of empirical studies of software development focused on thirty three primary studies. The studies were categorised in four groups: introduction and adoption, human and social factors, customer and developer perceptions, and comparative studies. Different empirical approaches were used to conduct these studies as can be seen in table 1.

Research method	**Number**	**Percent**
Single-Case	13	39
Multiple-case	11	13
Survey	4	12
Experiment	3	9
Mixed	2	6
Total	33	100

Table 1 Empirical studies by research methods (Dyba et al. 2008)

Of the 13 single-case studies, nine were done on projects in industrial settings; the other four were conducted on students' projects. Three of these studies took their data from the same project. Regarding maturity, only one single-case study in industry was done on a mature development team. All the 11 multiple-case studies were conducted in industry; only three of these studies were on mature teams. Three of the four surveys were done on employees in software companies, while one survey was done on students. The three experiments were all done on students, with team sizes ranging from three to 16. For the two mixed-methods studies, one reported on a survey amongst students in addition to interviews and notes from discussions. The other study reported on 10 case studies in companies, as well as the findings from discussion groups. The empirical study conducted in this research is done on mature teams; therefore we took a closer look on the four studies cited by the systematic review that fit this category. The studies had different focus, the first one focused the human factor and it explored the nature of interaction between organisational culture and XP practices (Robinson et al. 2005a). The second paper focused on the social side of XP practices (Robinson et al. 2005b) and the third focused on team characteristics and discussed how these characteristics are embedded in XP practices used in two particular settings (Robinson et al. 2004). The final paper was a study of XP practices and it identified five characterising themes within XP practices (Sharp et al. 2004a). The approach used in these papers was an ethnographic approach (Fielding 2001) which is a non-subjective approach that forces researchers to attend to the taken-for-granted, accepted, and un-remarked aspects of practice, considering all activities as ''strange'' so as to prevent the researchers' own backgrounds affecting their observations. The researches applied this approach by using close observation of the day today business of XP development, documenting practices using field notes, photograph of the physical layout, copies of documents, records of meetings, and discussions and informal interviews with practitioners. More recently, two empirical studies about the use of responsive methods were published in the Journal of Empirical Software Engineering in 2006. The first one discussed the advantages and difficulties which 15 Greek software companies experience applying XP. 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 (Sfetsos et al. 2006). The second paper presented a qualitative case study of two

large independent software system projects that have XP 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 (Karlstr et al. 2006).

III. QUALITY IN RESPONSIVE PROJECTS: AN EMPIRICAL STUDY

The previous review convinced us that the best approach to answer the identified research questions is to apply empirical methods including qualitative and quantitative ones.

The Empirical Study

When studying a human-based activity such as software development, the research must deal with the study of human activities, preferably, within real world settings (Basili et al. 2007). Qualitative methods are designed to study the complexities of human behaviours (Seaman 1999). Qualitative data can be represented as words and pictures, not numbers. Qualitative research is mainly useful in the early stages of research when no well known theories or hypotheses have previously been put forth in an area of study. As this is the case for the adoption of responsive methods and their impact on the different aspect of quality, we started the research with qualitative methods, namely case studies (Yin 2003).

The next step was finding the suitable case studies. The best option was a mature team that is using responsive software development who will agree to participate in the study. We had the opportunity to study the responsive project within Merge Software's which they did not apply a specific responsive method but they adopted responsive practice and principles so in effect they had their own responsive method.

Data Collection

Semi-structured interviews (Wohlin et al. 2000) were used to collect the data. Interviewing people provides insights into their work, their opinions and thoughts (Hove et al. 2005). The projects were studied within Merge Software. As total, 5 interviews were conducted. The interviews were informal and conversational. The notes were reviewed and written up. In the early interviews, we asked initial questions about general projects experience: number of projects, size of projects, working with responsive vs. traditional approaches if any existed, and how they rate the quality of an responsive project in terms of code quality and customer satisfaction. Also, we asked about the interviewee 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 later interviews, and as research evolved, we added more questions with more focus on product and process quality, as well as stakeholders' satisfaction.

Data Analysis

In order to analyse the interviews the constant comparison method, described by Glaser and Strauss (Glaser et al. 1967), was used. This method is one of the theory generation methods. We were influenced by the guidance from Carolyn Seaman to use this method for software engineering empirical research (Seaman 1999). When using this method, we start with coding the field notes, which means attaching labels to pieces of text that is relevant to a particular theme or topic. Then a list of codes will be generated while reading the data, with a big influence of the research questions. 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 they will articulate the preliminary grounded hypotheses.

The Results

The results will describe the agile adoption in each project as it was described by the interviewees with minimal subjective views from the researchers. The results will be organised in three categories: people, process, and quality. Each category includes subcategories, which represent the codes.

Process

When the project started it did not use any specific responsive method. The team followed a 1 week iteration that begins with a list of priorities (tasks). The team used responsive modelling on whiteboard and discussions to refine and tune the plan for the next iteration. The whole team should understand the architecture; therefore, it is reviewed by the whole team and continuously improved over the iterations. Decisions to drop line items are not very strict or formal; the team may roll them over to the next iteration or reword them to close off the iteration. During the interview with the team lead, he stated that the process has changed and it became very influenced by Scrum. The team is using iteration planning, review meetings including demos to stakeholders, and web conferences with business partners (external customers). One of the crucial points in the project is to have all the teams are working in an responsive way.

Estimation

Estimation is in days but the team is moving to story points so velocity can easily be measured and improved.

Tidy up Iteration

An interesting practice was to have iteration for stabilization, consolidation and to improve code quality. Out of 13 iterations, three were devoted to this purpose. These tidy up- iterations help to pace the work, fix problems, and allow some breathing space for the team. The consolidation iteration happens every six iterations or as needed. In addition to resolving defects, these iterations can be used to slow the pace and allow some time for reflection.

Testing, Automated Testing, and Relation between Testing and Development Team

Testing started just after writing the project's high-level statement. Developers typically write unit tests. The test team try to keep ahead of developers in writing functional tests so developers can use them while writing their code. The team lead reviews, selects the tests, and adds them to the build. When the code is checked in, it needs 45 minutes to build and then they run all the tests. The project had 50 new functional tests written by the test team; these tests are longer and more complicated than the tests written by the developers. The team found it important to be able to automate tests as part of continues integration. The team has three levels of automation: instant (unit tests), longer tests which can be combined in a suite that completes in less than half an hour and the long

running tests that includes the overnight re-build. In the interview with team lead, he mentioned that the system verification test has not changed.

Documentation
The architecture is documented as power point slides written by the architect and reviewed by the team. These slides include high-level decisions and some detailed description. The team is using class diagrams, package diagrams, and sequence diagram. An up to date list of features is available for users including how to use them.

The People
Size of the Team: The team has 10 members of which 7 are on-site and 3 off-site. Out of the 10 people, 9 are writing code including the senior team, (7 coders and 3 testers) and one performance person. The team had one architect, one team lead and two sub teams each had a lead who rotates, one sub team is responsible for the core functionality and the other is responsible for everything else including the add-ons. The team size did not change over the period we studied the project.

Roles: The team lead is also a technical lead, so he watches the process day by day and has a full understanding of the process. In addition, he monitors the defects and provides direction and guidance to the whole team. The architect facilitates high-level thinking for requirements design and review. Developers write code and unit test it, and regard testing as part of their role. Testers write functional tests, and then the team lead selects the tests and adds them to the build.

Delivery to the Customer: The project started with 2 weeks internal delivery at the end of the iteration, all were on time. Then they moved to weekly (mid iteration) delivery. In the future, the deliveries will be available on demand.

Quality and Quality Assurance
Assigning an iteration to improve the quality of the code is an effective practice; in addition, the team is using code reviews. The team expressed that these stabilization and consolidation iterations are important to increase the quality of the code.

Defects
Since the project has started only 20 defects were reported, some are missing features others are internal customer's defects. Defect prediction is not easy with responsive methods, and current Merger Software specific techniques that predict cost of defects and time needed for system test are in use although they are not designed for software development. Sometimes fixing a defect may take priority over agreed goals. At later stage of the project, according to the team lead during the interview, many problems were fixed without a bug report. The team managed to stay within the bug prediction, which is estimated by company. They found that responsive software development helps finding problems early, and then it is a business decision to fix them or continue adding functions.

Quality Assurance
The team lead stated that responsive quality assurance is different from traditional development quality assurance as in responsive assurance it should be more flexible and adaptable.

Customer Satisfaction
The team are doing what the customers want, deliver something to the customers to use and keep business partners happy. However, there are some areas that are not. The team lead expressed that responsive methods give a good way to assess customer satisfaction, he stated, "in waterfall you may find lots of bugs that do not matter to customers". The project achieved better customer satisfaction compared to other projects.

Quality measures
The team lead does not prefer formal management metrics as they need to be firmly accurate and does not work well with 2 hours build cycle, instead informal mechanisms work much better. The metrics in use for project are LOC (Line of Code), number of bugs, and code coverage (79% currently).

Generated Hypotheses
The generated hypotheses are organised in four categories: customer satisfaction, product quality, people quality, and process quality. These hypotheses are mainly based on the interviews conducted

Customer Satisfaction Hypotheses
1. Using agile methods improves customer satisfaction
2. Customer involvement/demands/requests increase throughout the project
3. The customer satisfaction has the same level throughout the project
4. Response to customer requests is good when using agile methods

Product Quality Hypotheses
1. Using agile methods can achieve high levels of reliability, availability and serviceability
2. When using agile methods, testers receive more minor bugs comparing to traditional approaches where the bugs are fewer but more critical
3. Automated tests can assure high quality code
4. The code developed using responsive methods has the same defect rate than traditional methods
5. The code developed using agile methods is less maintainable
6. The quality of the code increases as the number of iterations increases
7. Assigning an iteration to tidy up the code improves the quality of the software in terms of defects and maintainability
8. Code reviews can help improving product quality in responsive software development

People Quality Hypotheses
1. Agile software development requires people with high level of communication skills
2. Iterative development requires developers with high level of experience
3. Integrating new team members is harder with responsive methods
4. The smaller the team the higher the communication level between the team members

Process Quality Hypotheses
1. When using responsive methods testing is the responsibility of all team members

2. In responsive software development governance increases when the team is larger
3. In responsive development the process matures throughout the project
5. Each release should have a clear focus on one aspect of quality
6. Prioritizing defects is important in responsive development
7. In responsive development, longer iterations are needed when the team is larger

IV. RESULT

The data was collected and recoded using Excel and SPSS. SPSS was used as a tool for applying the analysis. First, because the software is provided by the University with introductory training, many books are available for self training, and most importantly it is a well respected tool among statisticians. In order to apply statistical methods on the current data we had to recode it into numbers using SPSS. This was done using a simple syntax that has to be applied on all columns to be recoded. The result is a new set of columns with coded data. The frequencies of the emerging data were compared against the original ones to make sure that the recoding was done correctly. Unfortunately, the survey questions are not suitable for statistical testing; therefore, the descriptive statistics will be used to present the results.

Descriptive Statistics

There are many ways of presenting univariate information about variables including frequencies, graphs, and statistical measures (Nardi 2002). For our data, we will present a frequency table that shows how each response was given by the respondent to each item; frequencies are useful when the variable has a limited number of values such as nominal or ordinal measures. It is less useful when an interval/ratio variable has many values.
In addition to the frequencies, the measure of central tendency provides a quick summary of where the responses are clustered. For our data, we will use the mean, the sum of the values divided by the number of values. Although the mean is more suitable for interval/ratio, we will use it for our ordinal variables as our scale (Likert -scale) looks like equal appearing interval scales. In order to see how well the mean represents the data we will use the standard deviation. The standard deviation (s) is the square root of the variance which is the average error between the mean and the data points (Field 2005).

$S = \sqrt{\frac{\sum(xi-\bar{x})^2}{N-1}}$ where xi is the data point for the ith position, $\bar{x}$ is the mean values and N is the total number of responses.
Different ways were used to present the results depending on the collected data and the suitability of these ways for our results clear presentation. For the iteration questions, we used frequency tables and the measure of central tendency. For the remaining sections, we used the frequency tables and bar graphs as they gave a good quick look at the results. Also in some occasions, we used the filtering feature in Excel to find and work with a subset of the data.

Iteration 1

For the first iteration we collected 24 responses from the team. The questionnaire was distributed to the whole team (55 people at the time of running the experiment). Table 1 shows the SPSS frequency table of the role variable as it was the first question in the monitor.

	Frequency	Percent	Valid percent	Cumulative percent
Developer	8	33.3	33.3	33.3
Information developer	2	8.3	8.3	41.7
Manager	1	4.2	4.2	45.8
Team Lead	3	12.5	12.5	58.3
Tester	10	41.7	41.7	100
Total	24	100	100	

Table 1 The frequency table for the role variable

Effectiveness of Responsive Practices

By looking at figure 1, we can see at a glance that most of the practices were not used during the iteration, which was interesting as the project manager reviewed before collecting the data, yet he did not remove this section. This can be seen in two ways: either the project manager is flexible about what techniques the team members are using and they are free to choose whatever they see appropriate, or the team are using these practices without naming them, because during the interviews some of these practices or at least their description was mentioned by the team members. Only two practices, user stories, and unit testing were reported by more than 75% of the team and they had an average level of effectiveness.

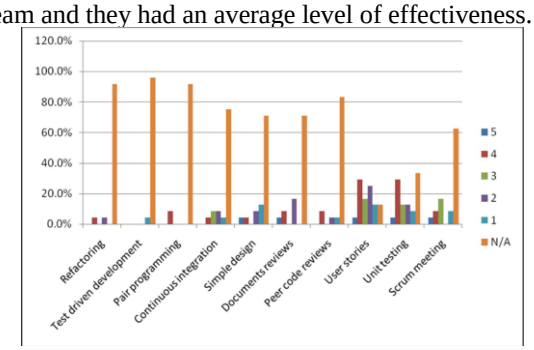

Figure 1 The practices' effectiveness

Communication within the Team

In this section, we asked how often the team members used different ways of communication. We can see that instant messaging is the most used method with as 75% of the team reported (often) and 12.5% (always). Email, meetings and informal chat are always popular, however the phone was not on the top of the list as 33.3% of the team never used the phone. The communication results are also presented in figure 2.

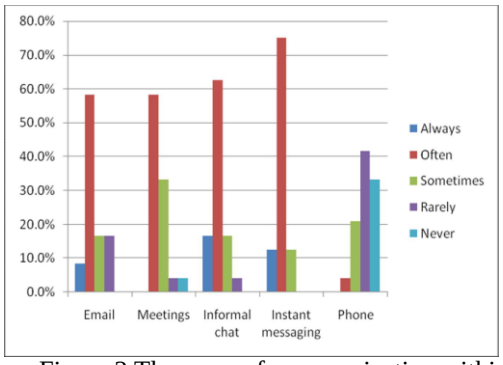

Figure 2 The ways of communication within the team

Iteration Focus

The percentages are presented in figure 3. From the table and the graph, we can see that 29% of the team stated that too much time was spent on functionality, at the same time 29% stated

that the amount of time spent on functionality was just right. Also, the team stated that too little time was spent on documentation (58%) and defect fixing (62%). The responses for refactoring were not very different from the practices section where 91% reported it was not applicable. In this section, 54% of the team did not know how much time they spent on refactoring as an activity. However, 25% said they spent too little time on refactoring as a responsive practice.

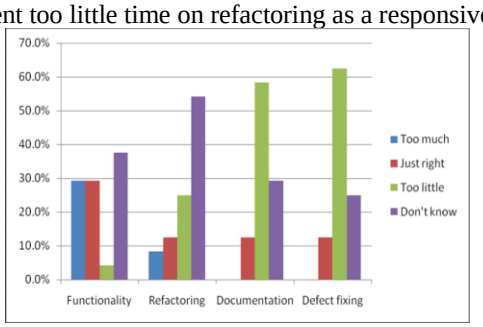

Figure 3 The iteration focus

Iteration 2

Iteration 2 received 13 responses. The motivation and happiness variables improved slightly in iteration 2 with mean=4.53, 3.46 and SD=.66, .66 respectively. The time spent on functionality looked about right with 38% of the team voted for "too much" and 30% for "just right". However, most of the team (53% for documentation and 46% for defect fixing) thought that too little time was spent on these two activities. Improvements focused on asking to limit the planned content for the iteration/release, which is consistent with the iteration focus results (too much functionality). The team asked to focus on important issues during meetings without going into too much detail.

Iteration 3

Iteration 3 received 10 responses. Similar to iteration 2, high level of good relationship (mean=4.50, with SD=.70), trust (mean=4.40, with SD=.70) existed within the team. In this iteration the team needed more time to resolve defects as the results showed (mean=1.80, with SD=.91) for the statement "there was sufficient time to resolve defects". Slightly different answers to the iteration focus question were found as 50% of the responses stated that the iteration had too little focus on refactoring and documentation. The architect influence on prioritizing requirements remained high with 70% voting that it was too much.

V. CONCLUSION

The first case study gave us a good understanding of the adoption of responsive software development and how it is related to the different aspect of quality within a organisation such as Merge Software. The project did not follow any specific responsive method at the beginning but as the time passes, the used process was influenced more and more with Scrum. The project started with 10 members and the team size was stable throughout the period of the interviews. The team followed 1 week iterations, and used iteration planning, Test Driven Development (TDD), refactoring and continuous integration. The team was happy and motivated which played a big role in the success of the project. Off-site members did not work very well so they moved to join the rest of the team on-site. The company culture affected the development in a number of ways including delays in early deliveries because of the legal issues, the team was unable of keeping whiteboards overnight, and the quality plan was not flexible enough to fit the responsive way of working. However, we argue that the company culture had a more positive impact on the project than negative. This is because the project followed the existing good practices in Merge Software such as the emphasis on measurements. In addition, although quality plans were inflexible, they worked well and they are on the way of producing an responsive quality plan. The project delivered on time, defects count was as predicted and customer satisfaction was improved.

REFERENCES

[1] Ambler, S. (2006). "Results from Scott Ambler's 2006 Agile Adoption Rate Survey" Retrieved 28/07/2008, from www.ambysoft.com.

[2] Basili, V. R. (1996). The Role of Experimentation in Software Engineering: Past, Current, and Future. Proceedings of the 18th International Conference on Software Engineering. Berlin, Germany, IEEE Computer Society.

[3] Basili, V. R., R. W. Selby and D. H. Hutchens (1986). "Experimentation in software engineering." 12(7): 733-743.

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

[5] Beck, K. and C. Andres (2004). Extreme Programming Explained: Embrace Change (2nd Edition), Addison-Wesley Professional.

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

[7] Boehm, B. (2002). "Get Ready for Agile Methods with Care." Computer 35(1): 64-69.
Cockburn, A. (2005). Crystal Clear A Human - Powered Methodology for Small Teams, Addison-Wesley.

[8] Dyba, T. and T. Dingsoyr (2008). "Empirical Studies of Agile Software Development: A Systematic Review." Information and Software Technology(50): 833-859.

[9] Field, A. (2005). Discovering Statistics Using SPSS, Sage.

[10] Fielding, N. (2001). Ethnography in N.Gilbert (ed.) Researching Social Life, Sage: 145-163.

[11] Gillies, A. C. (1992). Software Quality: Theory and Management, Chapman \& Hall, Ltd.

[12] Glaser, B. G. and A. Strauss (1967). The Discovery of Grounded Theory: Strategies for Qualitative Research, Aldine Transaction governance. (2009). "Compact Oxford English Dictionary Online." Retrieved 09/12/2009. Government Accounting Office (1998). Air Traffic Control: Evaluation and Status of FAA's Automation program, USA GAO.

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

[14] Highsmith, J. (2002). Agile Software Development Ecosystems, Addison-Wesley Longman Publishing Co., Inc.

[15] IEEE (1990). IEEE Standard 610.12-1990 Glossary of Software Engineerng Terminology. New York, The Institute of Electrical and Electronics Engineers. Kan, S. H. (2002). Metrics and Models in Software Quality Engineering, Addison-Wesley Longman Publishing Co., Inc.

[16] Kitchenham, B. A., S. L. Pfleeger, D. C. Hoaglin, K. E. Emam and J. Rosenberg (2002). "Preliminary Guidelines for Empirical Research in Software Engineering." IEEE Transactions on Software Engineering 28(8): 721-734.

[17] Karlstr, D. and P. Runeson (2006). "Integrating Agile Software Development into Stage-gate Managed Product Development." Empirical Software Engineering 11(2): 203-225.

[18] Nardi, P. M. (2002). Doing Survey Research: A Guide to Quantitative Research Methods Allyn & Bacon.

[19] Perry, D. E., A. A. Porter and L. G. Votta (2000). Empirical Studies of Software Engineering: A Roadmap. Proceedings of the Conference on The Future of Software Engineering. Limerick, Ireland, ACM Press.

[20] Robinson, H. and H. Sharp (2004). The Characteristics of XP Teams. Extreme Programming and Agile Processes in Software Engineering, Springer.

[21] Robinson, H. and H. Sharp (2005a). Organisational Cultur and XP: Three Case Studies. Agile Developemnt Conference, IEEE Computer Soceity.

[22] Schwaber, K. and M. Beedle (2001). Agile Software Development with Scrum, Prentice Hall PTR.

[23] Seaman, C. B. (1999). "Qualitative Methods in Empirical Studies of Software Engineering." IEEE Transactions on Software Engineering 25(4): 557-572.

[24] Seaman, C. B. and V. R. Basili (1998). "Communication and Organisation: An Empirical Study of Discussion in Inspection Meetings." IEEE Transactions on Software Engineering 24(7): 559-572.

[25] Sfetsos, P., L. Angelis and I. Stamelos (2006). "Investigating the Extreme Programming System: An Empirical Study." Empirical Software Engineering 11(2): 269-301.

[26] Sharp, H. and H. Robinson (2004a). An Ethnographic Study of XP Practice, Kluwer Academic Publishers. 9: 353-375.

[27] Sharp, H., M. Woodman and F. Hovenden (2004b). Tensions around the Adoption and Evolution of Software Quality Management Systems: A Discourse Analytic Approach, Academic Press, Inc. 61: 219-236.

[28] Wohlin, C., P. Runeson, M. Host, M. C. Ohlsson, B. Regnell and A. Wessl (2000). Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers.

[29] Yin, R. K. (2003). Case Study Research Design and Methods, Sage Publications.

[30] Zannier, C., G. Melnik and F. Maurer (2006). On the Success of Empirical Studies in the International Conference on Software Engineering. Proceeding of the 28th International Conference on Software Engineering. Shanghai, China, ACM Press.

