2007

An Empirical Study Of Software Quality Improvement Practices From Multiple Perspectives – An Australian Case Study

Lesley Pek Wee Land
The University of New South Wales, l.land@unsw.edu.au

Jeremy Higgs
The University of New South Wales

40. An Empirical Study Of Software Quality Improvement Practices From Multiple Perspectives – An Australian Case Study

Lesley Pek Wee Land
The University of New South WalesFaculty of Business
School of Information Systems, Technology and Management
Sydney, NSW 2052, Australia
l.land@unsw.edu.au

Jeremy Higgs
The University of New South Wales
School of Information Systems, Technology and ManagementFaculty of Business
Sydney, NSW 2052, Australia.

Abstract
The literature on software quality research to date has shown a lack of empirical insight into the use of methods for achieving quality in a real context. Further investigation is required if we want to increase our understanding on how to improve software quality practices. In particular, we wanted to explore the key factors which impact software improvement practices in a case study, by studying the perspectives of three key stakeholders - developers, managers and customers. The selected exemplary case is a small Australian based software company with exceptional record in terms of organizational growth and performance. A qualitative approach was adopted to conduct this exploratory study. Results were analyzed using the Grounded Theory approach to develop an initial framework from the empirical data. We hope that this framework will provide a deeper understanding of the reasons and perceptions for the use of quality methods in practice, and thus have an impact on both researchers and practitioners.

Keywords: software quality, software improvement, software quality practices

Introduction

The failure of many large software projects in terms of not meeting user/business requirements, prone to errors etc has led to software quality becoming one of the key issues from all stakeholders' perspective. As society comes to rely more on the interaction with software in all aspects of life, questions of 'quality' are increasingly important. In the literature, there is a lack of depth of study into what quality initiatives are successful. Software quality techniques have been studied using mostly experimental methods manipulating variations within each technique (e.g. testing and inspection). However, little is known about the practice of quality techniques in the real context. For example, how are various techniques used in combinations and under what circumstances are they more effective? Moreover, the use of quality techniques is only a component of software improvement practices/initiatives. Other components such as the use of various development standards, methodologies, tools, as well as non-technical aspects such as communication etc all come into play. In a real organizational setting, what are the key factors impacting software improvement initiatives?

Surprisingly, little work has been done on what quality initiatives are actually used in practice. Even though each organization is different in terms of context (e.g. setup, structure, size, maturity, experience, development practices and culture), there is much value in learning about exemplary practices. An in-depth understanding of how improvement practices evolve to achieve an exemplary level, and lessons learnt on why certain techniques/practices (in isolation or in combinations) work well or not at all, will assist stakeholders evaluate and evolve their own practices. These reasons motivate the study

reported in this paper. Moreover, we also believe that different stakeholders have diverse interests and objectives through the different roles that they play in the software development project, and therefore understanding diverse perspectives represented by the key stakeholders will present a more holistic picture of the software process improvement undertaken.
This study is exploratory in nature. Our main objective is to develop an initial framework from a case study in order to create an understanding of the key drivers of a successful software improvement initiative.
This paper is structured as follows. In section 2, we provide a literature review. In section 3, we describe the research methodology, followed by the results in Section 4. Lastly in Section 5, we conclude with study with a summary of findings, research limitations and suggestions for future work.

Literature Review

Researchers have debated whether quality is defined by the number of defects, quality-carrying properties or quality factors. Garvin suggests integrating different perspectives into quality, so a more holistic view can be formed (Garvin 1984). At the smallest scale, the literature covers tasks and activities that can be performed to improve software quality. Practices change the approach to development in such a way as to construct quality software from the outset. Finally, development methodologies have a wider impact, providing a framework for the transformation of the entire software development process. Table 1 provides a sample of the quality methods drawn from a review of the literature. Though this is far from an exhaustive list, it gives an indication of the types of quality initiatives that have been covered in the literature.

Table 25. Examples of software improvement activities, practices and methodology

Name	Description	Type
Unit Testing	Automated source code testing	Activity
Functional Testing	Determines if system fulfils customer acceptance criteria	Activity
Code Coverage	Determine coverage of the tests	Activity
Code Review	Manual defect identification	Activity
Automated Defect Detection	Automatic defect identification	Activity
Pair Programming	2 developers working on the same task	Practice
Test-Driven Development	Writing tests before development	Practice
XP	Increase quality through quick release cycles, testing & feedback	Methodology

We summarize in Table 2 a sample of case studies investigating practices and methodologies that have been reported in the literature. The absence of user feedback poses a problem for assessing the results of quality initiatives from different stakeholder perspectives. The literature review shows that there is little or no consensus on what methods are successful in creating quality software. Without a clear indication of ideal methods, implementing and achieving quality in software may be difficult.

Primarily, research so far has been characterized by a lack of insight into how and why quality methods are used outside of an experimental or controlled context, and the influences that give rise to their uses. This study seeks to address such problems. Furthermore, we believe that greater insight can be achieved through a careful selection of an exemplary case of a successful process improvement initiative. Our research objective explores the following: *What are the main factors driving software quality improvement in an organizational context?*

Research Methodology

In order to answer the research question posed in Section 2, a case study was conducted at a Sydney based software company. Although this study adopted a 'strong pluralist'/multi-method approach (survey and semi-structured interviews) to *"deal effectively with the full richness of the real world."* (Mingers 2001), due to lack of space, this paper will only focus on the qualitative results (generated from the semi-structured interviews); without significant loss of meaning and interpretation to original data set.

Grounded Theory methodology was employed. There are three main reasons for this choice. Firstly, no previous research on software quality has attempted to understand the use of quality practices from different perspectives. Grounded Theory allows the researcher to induce and discover a theory from the empirical observations and data. Thus, it is useful when trying to build new theories. Secondly, Grounded Theory allows the researcher to incorporate the complex nature of organisations and social interaction into the study. This is important in researching the empirical context. Finally, the purpose of this research is to understand the complex reasons behind the use of quality practices. This is likely to involve processes, organisations and social interaction, which Grounded Theory facilitates (Orlikowski 1993a) Grounded Theory thus allows the researcher to develop a theory on the use of quality practices in an empirical context. Finally, the purpose of this research is to understand the processes, organization and social interaction behind the quality improvement practice, which Grounded Theory facilitates (Orlikowski 1993b).

3.1. The Case

The study was performed at an Australian software development company. This was made possible through the first author's involvement with the company in a University work placement program. The researcher as a 'participant observer' is afforded credibility by reducing *"barriers to disclosure"* (Nandhakumar and Jones 1997) during the data collection process. Thus, 'convenience sampling' is adopted in this study. The sample company employs around 50 staff, including 20 to 30 developers, with a young and open work culture. There are two major product teams (Product A and B) with different project leaders and markets. The company fosters open relationships with its customers by making its issue-tracker publicly available. This allows customers to report defects and suggest new features. In order to facilitate honest responses, all data collected from the sample company will remain anonymous.

Data Collection

Semi-structured, qualitative interviews were performed at the sample company to investigate subjects' perceptions and understanding of software quality. This method provided the flexibility to investigate topics of interest to the interviewer and interviewee as they arose (Yin 2003). Interviews have been used extensively in the IS realm for case study research (as evidenced by the summary of case study research on software quality). Previous research employing the Grounded Theory research methodology has also made use of interviews (Orlikowski 1993b).

Table 26. Summary of case studies on software improvement

Study	Quality Method(s) Studied	Research Methods Used	Description	Context	Company	Developer Feedback	User Feedback	Outcomes
(Johansen and Gilb 2005)	Evolutionary Development	Observation	Analysis of a 3-month trial of the Evolutionary Development Methodology	Prior use of Waterfall model and CMM	Future Information Research Management	Yes: • Increased developer motivation • Difficult to define good requirements	Yes	• Methodology has a focus on measurable product qualities • Able to focus on delivering value to the user • Company intends to use in the future
(Williams et al. 2003)	Test-Driven Development	Observation, Document Analysis	Aims to determine how effective TDD is in defect reduction	Used TDD to develop a hardware driver, with comparison to the legacy development process	IBM	Yes: • Developers positive about TDD • Practice has been continued	No	• 40% lower defect density • Similar productivity • Creates regression test library
(Wilson and Hall 1998)	Quality	Interview	Investigates the perceptions of software quality among developers, managers and software quality practitioners.	Case studies at four Australian companies	N/A	Yes	No	• Different groups have different perceptions of software quality • Culture affects the quality approach • Developers often reject formal or procedural approaches to quality • Successful quality initiatives require attitude alignment
(Biyani and Santhanam 1998)	Defect Analysis	Document Analysis	Analysis of defect data and reliability over a multiple software releases	Defect analysis performed on the defects discovered during development and 4 releases of an application for 'high-end systems'	N/A	No	No	• Modules that have more defects during the development process are likely to have more defects found by customers • Ratio of development to field defects can be used to assess the relative quality of a software release
(Horgan et al. 1994)	Code Coverage	Document Analysis	Investigates use of a code coverage tool in assessing test completeness and quality for two real-world software projects	Code coverage performed on two autopilot software systems	University of Iowa, Rockwell/Collins Avionics Division	No	No	• Relationship between high unit test coverage and low fault levels • Coverage data is a good feedback mechanism for programmers

Interviews were held with three groups (developers, project leaders, customers) related to the sample company. The three perspectives provide a rounded understanding of software quality to be developed from the practical context, and triangulate the perceptions of software quality. Developers from Products A and B (Developer A1-A4, B1-B4) were interviewed to establish the reasons behind the use of quality methods and determine the approach and perceptions to software quality. Interviews were also held with two company directors (Director A and B) and two project leaders (Product A and B Lead), to explore how software quality has been implemented through the company culture. One customer of each product (Customer A and B) was interviewed to establish how they perceive the quality of the software. They were chosen on the basis of their knowledge of the products. This ensured that the customer had time to form an accurate judgement of the software quality.

Each interview was held in person or over the phone. Interviews were recorded (with permission), and lasted between 1/2 and 1 1/2 hours. Hand-written notes were also taken by the researcher. Interviewees were required to sign consent forms accepting participation in the study, in accordance with the UNSW Ethics Committee guidelines. The questions were developed in an iterative fashion to uncover the perspective of each stakeholder. The sample of developers involved in qualitative interviews was derived using stratified sampling. 2 junior developers and 2 senior developers were selected from each team, making a total of 8 developer interviews.

Data Analysis

Grounded Theory was adopted in the qualitative interviews to guide the collection of data and reveal themes in the interviews. Open coding was first performed to *"generate an emergent set of categories and their properties."* (Glaser 1978) (p.56) Following this, axial and selective coding were used to construct a theory from the empirical data. Thus, the coding process created a new theory on the use of quality methods to be derived from an empirical grounding.

Analysis followed the steps of open, axial and selective coding. A 'selective' approach to transcription was adopted in this study (Strauss, 1987). The large number of interviews and limited time and resources constrained the ability to perform full, verbatim transcriptions of the interviews. Instead, notes were made on the recordings of all interviews. Important passages were selected and transcribed verbatim, based on the researcher's understanding of the data, as suggested by (Strauss 1987). 'Constant comparison' enabled further refinement of the selections as the researcher developed a better understanding of the data. There were also opportunities to perform additional transcription when theoretical gaps were identified. In addition, written field notes from the interviews were integrated. Using these 'rules', each interview typically took around 3-4 hours to complete.

The first stage of analysis involved open coding to *"generate an emergent set of categories and their properties."* (Glaser 1978) (p. 56) As each interview was analysed, comparisons were made to previous interviews to incrementally refine the coding process (Ezzy 2003). Interviews were read to discover important codes or themes, which were highlighted on paper. As this was the researcher's first experience of Grounded Theory, a couple of attempts were made at open coding, as the initial attempts were not comprehensive enough. This process occurred while interviews were being performed, in line with the concept of 'constant comparison'. This continued for a number of weeks, until the codes were saturated.

Once it was felt that the codes were saturated, axial coding commenced. The aim of this stage was to let a theory emerge from the empirical data by identifying common themes. All identified codes (and the relevant quotes) were laid on a table and grouped into related areas. A number of attempts were made, as the previous emergent categories did not fit the data. At

the conclusion of this stage of analysis, a set of categories had emerged, explaining the underlying codes that had been identified.

Finally, selective coding was performed to determine a central category for the data. This involved exploring the relationships between the categories established in the previous step. Various combinations of categories were attempted, until one emerged that linked together the categories in a suitable manner. At this point, the theory had been saturated (Ezzy 2003). The results will be presented in Section 4.

Results

The phases of data collection have revealed that the company is undergoing a period of change. In this section, we present an initial framework for the improvement of software quality.

Developer responses varied greatly. Many focused on attributes of the source code, including documentation, modularity and maintainability, as well as the presence of unit tests. "User Experience" was a central theme. Responses clustered around the expected functionality, performance and usability. The final area of focus was the level and speed of customer support. The responses from the management team were quite similar to the developers. The focus remained on the user experience and code attributes. It is important to question whether this focus has influenced the perceptions of the developers through the company culture. The customer responses reveal an unsurprising focus around the "user experience" of the software. The number of bugs present and reliability were key factors. One customer mentioned the "intent" of the software as one of the key quality factors:

"If your product is marketed for 24 by 7, a lot of things that [Product B] has to do is provide cluster-based support, right, which is not there today." (Customer B)

These perspectives provide a wide range of factors that make up quality for the company. Clearly there is an overlap between the perspectives in the focus on user experience. Both management and developers also focused on code attributes. This definition of quality, as an amalgamation of the views of the stakeholders, provides the basis from which the company's approach to software company can be understood. According to (Wilson and Hall 1998), this is the key to successfully achieving software quality.
Grounded Theory analysis of the interview data has led to an empirically grounded framework, shown in Figure 1.

Continuous improvement of software quality is the driving force behind the company's approach. The other constructs contribute to the improvement of the development process, which in turn contributes to software quality. A culture of improvement creates teams in which there is a drive to constantly improve the team and the product. This can be seen in two areas: developer self-improvement and a culture of continuous process improvement.

According to Director A, developers in the company are expected to have an intrinsic motivation to increase their knowledge and improve the way they work. This is based on the belief that intrinsically motivated developers will seek to make high-quality software and continuously strive to improve it. The directors have tried to achieve this from the outset, by adopting a policy of hiring only the best developers. Developers and management subject potential employees to rigorous interviews and programming tests. This allows technical people to evaluate the suitability of a potential employee in terms of their personality and abilities. A company with staff that are talented and intrinsically motivated to improve themselves is therefore likely to function well in a culture of continuous improvement.
The management team expressed that a culture of **continuous process improvement** had been cultivated in the company. The culture supports continuous feedback, which encourages developers to address problems, try new development practices and see which work best.
Primarily, this is achieved by establishing processes to allow developers to do good work:

"The way you get better quality software is that you have the processes in place that ... give people the room to do thorough work and encourage people to do thorough work, and at the same time give people clear direction and make sure that there's time to complete tasks and make sure there's a culture of things like ... automated testing and continuous testing, and the important part is that ***whatever successes and failures we have in that, we can discuss them openly and feed that whole thing back into the development process.****" (Product B Lead)*

Developers shared this approach to continuous process improvement through reflection and problem solving:

"... the assumption that after we do something, we should sit down and have a chat about it and work out ways to improve what we did,… that's kind of the fundamental good thing that we do to achieve quality." (Developer B3)

This reinforces (Wilson and Hall 1998) the view that culture affects the approach to quality in a company. This culture clearly addresses the SPI success factors of incremental approaches and 'tailoring' of the development process (Stelzer and Mellis 1998; Rainer et al. 2003). The ***gradual*** **introduction and refinement** of new development processes also embodies the continuous improvement approach. Previous experience has shown that implementing XP at once does not always work:

"Introducing them little by little also allows us to see which ones work. …" (Product B Lead)

Continuous process improvement is normally achieved in the company through small, incremental improvements to the development process. This allows developers and teams to learn from previous mistakes and implement practices or processes to overcome them, improving the development process and ultimately software quality. An example was provided by Developer A3. The daily meeting was shortened and provided greater focus after developer feedback was implemented:
"Our standups [meeting] changed this morning... Previously I think it was about half an hour; this morning's was 8 minutes. So, very quick and precise and we sort of just covered what needed to be done and talked about commitment for the day …" (Developer A3)

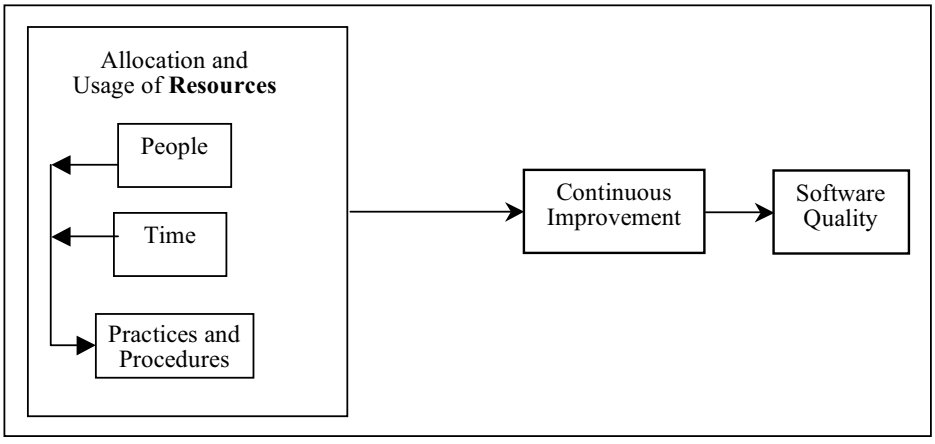

Figure 1: Key factors driving software process improvement

Continuous improvement is essentially driven by how the organization utilizes its resources to achieve the software quality objective. Resources include:

- People
- Time

- Practices and procedures enacted by people.

Each of these components needs to be utilized and managed in the best possible way because they are limited and deliver value propositions. Resource availability has already been identified as a success factor in SPI initiatives (Baddoo and Hall 2002a; Paulish and Carleton 1994). Consequently, resource availability is a key factor in process improvement and quality. These will be addressed below.

People

People include employees at all levels of the organization. Beginning with the management, **proper management direction** is important. In this case, the direction given to developers is provided through frequent releases of the software and visible evidence of progress. The development teams have adopted the XP mantra of 'release early, release often'. Releases are partitioned into one or more two-week 'iterations'. Planned work is broken down into manageable sub-tasks at the start of every iteration. Developers and management estimate the amount of effort for each task and schedule according to priorities. The result of this is a short release cycle that allows developers to remain focused on small tasks. Previously all development tasks, such as bugs, enhancements and new features, would be entered into the online issue-tracker. Instead, development tasks for each iteration, are now written on index cards and placed on a wall in the team office, called the 'story wall'. As work is completed, the relevant index card is marked and moved onto an adjacent wall. This provides a visible indicator of progress and guidance on the work remaining.

Non-management employees also play an important role in taking **ownership and responsibility** of various aspects of the continuous improvement process. For example, the introduction of "XP Process Champions" has led to greater commitment and motivation for improvement, and thus improvement of the development process and software quality. The role of "XP Process Champions" was introduced into the Product B team earlier in 2006. At the time of the interviews, it had not been introduced to the Product A team. Developers were encouraged to take ownership of different aspects of the development process. The Product B Lead summarised their role as the following:

"Each person has been given an overview of a particular part of the process, like testing or making sure the builds work or looking for refactoring or all that kind of stuff, and each person has been given a small area of focus to look at how we can improve the way we do things and also the most important part there is coming up with metrics and measuring what we're doing." (Product B Lead)

The key to the champion's role is the time dedicated to the improvement of the one practice, rather than shared ownership of all. The role of the champions is to promote and champion the use of a practice, and to improve the way the practice is done. This allows developers to be more involved in the improvement of the development process, while giving management greater flexibility and perspective. Currently, there are six different champions in the Product B team as depicted in Table 3. There were no champion roles in the Product A team, as XP process champions had not been introduced at the time of the study. A sense of ownership and responsibility has been cultivated by making individual developers in charge of aspects of the development process. Champions feel a responsibility and commitment to ensure that their practice or process is providing the greatest benefit to the team. Responsibility also provides motivation for improvement. There appears to be no extrinsic reward for the champions. Rather, self-improvement and contribution to the team are likely to be the intrinsic motivations.

Time

Due to time limitations in every project, the interviews revealed a **pragmatic** approach to quality. This approach underpins the decisions to use practices and procedures in the

development process. The 'pragmatic' approach can clearly be seen in the use of testing, TDD and PP by developers. For example, Developers made it clear that a 'practical' approach to testing was adopted. An informal 'cost/benefit analysis' was cited as the way to judge whether tests should be written when working on a development task (Developer A2, A3, B3). The main determinant in this judgement was the difficulty in writing tests. If it was "too technically difficult" (Developer B3) or the code was "untestable" (Developer A3, B1, B2, B3, B4; Project Lead A, B; Director B), then it was unlikely that tests would be written. The architecture of the source code often affected this.

Table 3: XP Process Champion roles

Role	Responsibilities
Code Coverage	3. Provide reports on the amount of code coverage by tests 4. Explore ways to increase the usage of code coverage tools
Continuous Integration	5. Provide reports on the build stability for unit and functional tests 6. Ensure developers value testing and fix build issues quickly
Daily standup meetings	7. Ensure daily meetings happen 8. Explore ways to increase the effectiveness of the meetings
Frequent Releases	9. Develop processes to facilitate frequent software releases
Refactoring	10. Identify areas that require rework 11. Ensure there is one refactoring task included in each iteration
Velocity Tracking	12. Report on the amount of work required for tasks completed in each iteration, compared to the estimated tasks

Practices and Procedures

Proper allocation of people and time resources enable practices and procedures to be enacted efficiently and effectively. Practices may be enabled by the appropriate mechanisms such as metrics (measurement) and employee communication.

Measurement (e.g. in areas like product quality, productivity or speed) is a key aspect of any SPI initiative (Paulish and Carleton 1994). Metrics were officially introduced to the Product B team recently, in order to quantify improvements to the development process. Measurements are collected by the "XP process champions" responsible for each practice. Conversely, measurement can also reveal problems. Primarily, measurement of key aspects of quality quantifies the efforts of the developers and gives them an incentive to make further improvements

Metrics provide visible evidence of progress. Developers are encouraged to present graphs on key areas of the development process (e.g. build stability, code coverage and iteration velocity) in a bi-weekly 'process meeting'. These metrics provide evidence of success and a motivation to improve. Frequent releases provide the SPI factor of 'visible success' (Hardgrave and Armstrong 2005; Baddoo and Hall 2002b) and overcome a 'lack of guidance' (Hardgrave and Armstrong 2005). Therefore, developers and management both contribute to the direction of the development process at the company.

A decline in communication is one of the main reasons for the most recent quality initiatives in the company. Consequently, communication has received significant focus from the developer, management and customer perspectives. Communication is effective when it promotes visibility of the development process and when developers' reflect on their practices.

Table 4 shows that customers provide feedback to the development teams through various methods which facilitate communication between development teams and customers. The customer perspective of quality is thus integrated into the development process. Until recently, teams often met (daily) to discuss product status, problems developers encounter and any other pertinent issues. However, it was recognized that these meetings did not provide a way to reflect and improve on the development process itself. Product planning meetings were also held infrequently. As a result, this meeting was differentiated into three separate meetings with distinct foci. A "daily standup meeting" was established, where each team member outlines their 'commitments' for the day:

"So, very quick and precise and we sort of just covered what needed to be done and talked about commitment for the day and not all of the shit we did the previous day, so it went a lot quicker." (Developer A3)

Developers gain a better understanding of the team status through focus on daily tasks. It also provides a medium to express development problems. A fortnightly "process meeting" was also introduced to Product B. Developers and managers in each team discuss problems with the development process and possible improvements:

"We don't talk about ... features, bug fixes, what bugs are being fixed, we just talk about what we're doing well, what we could be doing better and how we could be doing it better, and that's ... an important thing for building good quality software... because it keeps the conversation going." (Product B Lead)

Table 4: Communication methods between developers, managers and customers

Name	Purpose	Focus	Frequency
Blogs	Share knowledge and customer feedback	Internal/External	Ad-hoc
Support/issue-tracking System	Request support, report bugs, provide suggestions	Internal/External	Ad-hoc
Forums	Request support, report bugs, provide suggestions	External	Ad-hoc
Email	Request support, report bugs, provide suggestions, share information	Internal/External	Ad-hoc
Instant Messaging	Share information, chat	Internal	Ad-hoc
Planning Meeting	Plan major software release, discuss problems with planning	Internal	End of each release
Iteration Meeting	Plan 2-week development iteration, discuss problems in planning/estimation with the previous iteration	Internal	Every 2 weeks (Product B only)
Process Meeting	Discuss how the development process is working, suggest improvements to the development process, present reports and metrics on practices/procedures	Internal	Every 2 weeks (Product B only)
Standup Meeting	Discuss what tasks will be performed by each person that day	Internal	Daily
Wiki	Share knowledge and development plans	Internal	Ad-hoc

According to Director B, it was also one of the key changes made to this implementation of the XP methodology over previous attempts:

"The process meeting is our attempt to implement XP better this time, … We don't talk about the application, we don't talk about how well we're doing in terms of developing features. We don't talk about what the features are. We talk just about the process that the team is undergoing. How do we think we're doing at XP? …" (Director B)

This meeting has allowed the team to be intimately involved in the appraisal and evolution of the development process, in line with the 'involvement' SPI success factor [14]. Such a meeting had not been introduced to the Product A team at the time the interviews took place. Developers from that team mentioned that they could only raise development issues in the daily meeting. Often, they felt their suggestions would be skipped or not discussed at length. Conversely, developers from Product B seemed quite happy with the level of discussion of

the development process. The third meeting, the "iteration planning meeting", is held every other week to the "process meeting". In the iteration planning meeting, team members plan the tasks for the next two weeks.
Discussion of the preceding iteration also reinforces the concept of reflection:

"When we finish an iteration... or when we finish a release cycle, we regularly get together and say 'ok, what was good about this, what was bad about this, what can we do better?' Quite a bit of the feedback will come from [Director B] and [Director A] and myself, but a significant amount comes up from the crew." (Product B Lead)

The net effect of these differentiated meetings is greater communication about product development and the software development process. These meetings provide feedback forums in which developers and management can contribute to the improvement of the development process, and indirectly the quality of their products. These are in line with SPI success factors 'communication' and 'feedback' (Stelzer and Mellis 1998; Conradi and Fuggetta 2002). There is also evidence of more informal methods of communication. In particular, the interviews revealed that there is a degree of 'diffusion' of ideas between the teams. An example is the formal introduction of Pair Programming to the Product A team. The Product A Lead had been discontent with the number of bugs being reported. A combination of positive feedback from the Product B team and the suggestions of the Product A developers led to the subsequent introduction of Pair Programming. Thus, there is likely to be informal inter-team communication, which aids in the improvement of the development process. Another key aspect of communication is the developer interaction with customers. The company recently established dedicated support staff for both products. However, 2 developers rotate onto the support team every few weeks. This gives them exposure to customer problems and expectations, increasing the communication between the groups. Thus, developers can then understand customer problems and develop the software in a way as to overcome them. The communication methods allow the stakeholders to discuss and develop the products further. In particular, the differentiated meetings have encouraged further feedback and reflection on the development process by developers and management. These improvements feed back into the improvement of the development process and overall quality.

Visibility communicates progress and value to the development teams. The first affected area is metrics and communication of progress. Secondly, increased visibility can also communicate the importance of particular beliefs. Visibility of progress was introduced by the use and presentation of metrics. Metrics on the procedures and practices are presented in the meetings mentioned in the previous section. A developer provided an example of this:

"The Continuous Integration champion is tracking the build values every day, which is a good way of making sure that the tests are actually passing more often than not. So [Developer B4], basically every morning, he checks whether the tests are passing... and he tells everyone in the standup [meeting] or sends out an email if it's breaking. Everyone then looks at it and realises who broke it and then get finger-pointed." (Developer B3)

The reporting of metrics on build stability thus communicates progress of the improvement initiatives and relative increases in software quality. The communication of value was another area where visibility had an impact. A developer raised this concern, in relation to the collection of metrics:

"If something requires ongoing effort, then it has to have enough value that the person who's doing it feels like they should continue doing it." (Developer B3)

Under the previous build system, build results were ignored. A new Continuous Integration system was introduced in early 2006, which built the software and ran tests against it on every commit to the source code repository. Developers would then be notified of the build results by email and the internal instant messaging (IM) platform. These messages report the developer that made a change to the source code and the results of the build and tests. One of the benefits provided by its introduction was increased visibility and emphasis on quality:

"*There's a lot of value in just making it more visible that this is something you care about…*" (Product B Lead)

Thus, increased visibility communicates the importance of quality and attention to detail in development. The introduction of differentiated meetings and increased visibility have had a large impact on the development teams and process. Creating distinct meetings has allowed developers to openly discuss issues in daily tasks, the development process and in iterations. Higher visibility of values has been achieved through the Continuous Integration system, providing a common understanding to developers and management.

The enactment of practices and procedures enable their integration into the development process. This is important because increasing practices and procedure usages by developers help gain consistent benefits as a result. Developers were unlikely to use practices if they felt they were not essential. Developers are likely to skip practices under pressure. The 'institutionalization' of practices into the development process was suggested as a remedy for this by the management team. The clearest example of this is the integration of testing through Continuous Integration. The value of testing and quality is maintained by the automatic execution of tests by the CI system, independent of the amount of pressure on developers. An example of a lack of institutionalization also emerged through the interviews. Developers expressed the benefits of using code coverage analysis to determine how well written their tests were. Leading code coverage software that worked with the Integrated Developer Environment (IDE) had been purchased for the entire company and was available to all developers. Despite this, the Product A Lead commented that code coverage analysis is "not done very often, but it does happen". Developer A3 was aware of the available code coverage tool, but felt that inexperience with the tool and a lack of integration into the IDE workflow hindered its use. A lack of understanding and value on the practice is also likely to have contributed to the level of use. Management plays a key role in the integration of practices through institutionalization and repeatable processes. These aspects of integration contribute to the effectiveness of the development process, and consequently software quality.

The practices and procedures used in the development process serve as 'enablers' for the improvement of product quality. The use of suitable practices and procedures is therefore important to improve software quality. Testing, Test Driven Development (TDD), Pair Programming (PP) and CI serve as 'enablers' for the improvement of the development process and the SPI initiatives. The responses of the developers indicate largely positive attitudes to the development practices and procedures used, which contribute to the improvement of the development process and product quality. However, the code coverage and code review practices had less positive responses, which indicate issues in their use at the company.

People and time are therefore important ingredients to the use of practices and procedures. Equally, adequate and appropriate management of scarce resources are critical for ensuring quality in software development.

Conclusion

A case study was conducted to understand the key factors driving software quality improvement practices in a software company based in Sydney, from key stakeholder perspectives. At the centre of the framework is continuous improvement. We argue that continuous improvement is driven by efficient allocation and usage of scarce resources – people, time, software development practices and procedures.

The proposed framework is supportive of the existing literature on software quality improvement practices. Since our case study was done in depth, we have provided a rich picture of how scant resources are utilised by organizational members to deliver quality improvement and ultimately software quality for the particular case. For example, we have shown that the organization has to be practical in implementing practices due to time constraints, thus pragmatism is an attribute in decision making of practices and procedures. Not surprisingly, human factors play a big role in software quality success in this case. This includes strong management and headship, as well as ownership and responsibility for

employees below. Equally appropriate practices and procedures must be integrated within the software processes, these were made possible by effective and efficient allocation and use of time and people resources. Our results were derived from multiple stakeholder perspectives, so is not bias towards developers, management or clients.

This study provides a deeper understanding of the perceptions behind software quality practices. A better understanding of these reasons can contribute to more effective software quality initiatives in practice. This framework proposes technical and non-technical resource factors that contribute to software quality improvement at the company. This study offers a unique insight; it has provided into a software development company run by developers. The project managers and company directors are all of a technical nature and often engage in the development of the software. This is in contrast to many companies in the industry. The level of influence and the team dynamics may therefore differ from the 'norm'. Only one case study was performed. Although this was a unique and exemplary case, there is a limited context. Future research should address process improvement practices of a different context. Further, future work could explore the resource view for enabling software process improvement along other resource dimensions, such as those that spans across organizational boundaries.

Acknowledgements

We would like to thank the management and development teams of the company involved in this study. Many hours were spent performing interviews and filling out the surveys, which are greatly appreciated. We would also like to thank the customers that donated their time to provide a user perspective on quality.

References

Baddoo, N., and T. Hall. 2002a. Motivators of Software Process Improvement: an analysis of practitioners' views. *Journal of Systems and Software* 62:85 - 96.

———. 2002b. Software Process Improvement Motivators: An Analysis using Multidimensional Scaling

Empirical Software Engineering 7:93 - 114.

Biyani, S., and P. Santhanam. 1998. Exploring defect data from development and customer usage on software modules over multiple releases

Paper read at Proceedings of the Ninth International Symposium on Software Reliability Engineering.

Conradi, H., and A. Fuggetta. 2002. Improving software process improvement *IEEE Software* 19:92 - 99.

Ezzy, D. 2003. *Qualitative Analysis*: Routledge.

Garvin, D. A. 1984. What Does "Product Quality" Really Mean? *Sloan Management Review* 26:25-44.

Glaser, B. G. 1978. *Theoretical Sensitivity: Advances in the methodology of grounded theory*. Mill Valley, California: Sociology Press.

Hardgrave, B. C., and D. J. Armstrong. 2005. Software process improvement: it's a journey, not a destination. *Communications of the ACM* 48:93-96.

Horgan, J. R., S. London, and M. R. Lyu. 1994. Achieving software quality with testing coverage measures. *Computer* 27 (9):60-69.

Johansen, T., and T. Gilb. 2005. From Waterfall to Evolutionary Development (Evo): How we rapidly created faster, more user-friendly, and more productive software products for a competitive multi-national market.

Kaplan, B., and D. Duchon. 1988. Combining Qualitative And Quantitative Methods In Information Systems Research: A Case Study *MIS Quarterly* 12:571-586.

Markus, M. L. 1994. Electronic Mail As the Medium of Managerial Choice *Organization Science* 5:502-527.

Mingers, J. 2001. Combining IS Research Methods: Towards a Pluralist Methodology *Information Systems Research* 12:240-259.

Nandhakumar, J., and M. Jones. 1997. Too close for comfort? Distance and engagement in interpretive information systems research
Information Systems Journal 7:109-131.

Orlikowski, W. J. 1993a. Case Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development. *MIS Quarterly* 17 (3):309-340.

———. 1993b. CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development *MIS Quarterly* 17:309 - 340.

Paulish, D. J., and A. D. Carleton. 1994. Case studies of software-process-improvement measurement. *Computer* 27:50 - 57.

Rainer, A., T. Hall, and N. Baddoo. 2003. Persuading developers to "buy into" software process improvement: a local opinion and empirical evidence Paper read at Proceedings of the 2003 International Symposium on Empirical Software Engineering.

Stelzer, D., and W. Mellis. 1998. Success factors of organizational change in software process improvement. *Software Process: Improvement and Practice* 4:227-250.

Strauss, A. 1987. *Qualitative Analysis for Social Scientists*. Cambridge: Cambridge University Press.

Williams, L., E. M. Maximilien, and M. Vouk. 2003. Test-Driven Development as a Defect-Reduction Practice Paper read at Proceedings of the 14th International Symposium on Software Reliability Engineering.

Wilson, D. N., and T. Hall. 1998. Perceptions of software quality: a pilot study. *Software Quality Journal* 7 (1):57-75.

Yin, R. K. 2003. *Case Study Research: Design and Methods 2003*. 3rd edition ed. Thousand Oaks, California: Sage Publications,.