Improving Quality, One Process Change at a Time

Caryna Pinheiro
MCIT Solutions
Caryna.pinheiro@gmail.com

Frank Maurer
University of Calgary
frank.maurer@ucalgary.ca

Jonathan Sillito
University of Calgary
sillito@ucalgary.ca

Abstract

We report on one organization's experience making process changes in a suite of projects. The changes were motivated by clients' requests for better time estimates, better quality, better stability and more reliable test scheduling resulting from the high number of bug reports and constant delivery delays. The teams embarked on a series of top-down process changes inspired by the IBM Rational Unified Process. Changes included adopting the Rational Tools, introducing iterative development, and later the hiring of a formal manual testing team and support for refactoring activities. To assess the impact of these changes we have collected fault data from 23 releases of the systems including releases from before and after these changes were introduced. In this report we discuss the challenges and impact of these process changes, and how the development teams leveraged these successes to gradually introduce other process improvements in a bottom-up fashion.

1. Introduction

Agile Methods such as XP and Scrum, have iterative development in the core of their engineering practices [8], intended not only for the development phases of a project, but also for maintenance of a code base [3,7]. As Agile methods become more mainstream, many project teams are customizing their process by choosing which agile practices to adopt and which to ignore. However, companies that wish to follow a complete Agile process might introduce practices such as continuous integration, pair programming, test driven development and automated tests. This makes it hard to measure which techniques provide the most visible benefits during the trying transitional stages. This paper presents a longitudinal experience report of a large (900+ employees) corporation's journey towards the goal of adopting an iterative and incremental process. We gathered and analyzed available project metrics and found that the iterative aspects of the Rational Unified Process (RUP) [5] together with investment into test teams and refactoring succeeded in improving software quality and software stability.

Recent literature provides many reports of the move from waterfall to Scrum that were supported and promoted in a top-down fashion by managers to initially skeptical engineering groups [6,12,14,16]. Many industries are faced with constant and strict conformance to rules and regulations that result in formal decision-making processes [9]. As many of us have experienced, in IT projects this often translates into long sign-off processes, pre-defined scope documents, and tight change control. In reality, this simply moves the accountability around the table [15]. In this context, changing to an Agile process represents a paradigm shift based on trust. Jochems and Rodgers first lesson when migrating to Agile was: "*if the team is resistant to change, gradually introduce new concepts*" [4]. Although the authors directed this comment to resistant developers, this advice can also apply to management. This experience report shows how the improvements of a top-down adoption of the RUP process provided gentle ways to introduce Agile-related concepts and practices in a bottom-up fashion into an environment that initially lacked management support for agility. It also contributes to an understanding of the advantages and challenges in adopting semi-agile iterative development practices in an industrial setting. Such an understanding is particularly important to managers that prefer to adopt processes that have been successfully implemented by others to reduce the risk of failure [15].

2. Related Experiences

Before we move into the specifics of our experience, we would like to share some related literature that we found extremely helpful and relevant

to those of us transitioning to iterative methodologies in large organizations.

Lewis and Neher [18] give an experience report of the challenges encountered by a pilot development team while migrating to Scrum at Microsoft IT. To diminish the impact on the overall IT management, they provided a map between the existing software development life cycle to the Scrum practices, added iterative periodic reviews, and excluded the complex User Acceptance Testing (UAT) process. The pilot team considered the adoption beneficial and attributes the success to staying on course with the changes, even when the team resisted them. The authors suggest that one of the key success dynamics is to compose the team of "agile minded" individuals. They forecast that there will be issues with complex dependencies and with the "co-habitation" of teams in the future.

Megan [2] relates the challenges faced by a QA team used to complex UI testing cycles while moving to Scrum. The author advises getting help from Agile experts early in the process, being ready for roles and responsibilities changes, and establishing shared ownership of software quality. Beavers [6] reports on his experience as a manager of a team migrating from a waterfall process to Scrum. He advises managers to trust the methodology and intuition, to empower the engineering team, and to become more engaged with the team than only in a process control role. When adopting Scrum, Seffernick [16] suggests establishing an agile project management framework, engaging product owners from the start, and adding developer best practices. Fry and Greene [12] describe the challenges and opportunities related to a "big-bang" rollout of Agile initiated by the company founder of an R&D company. Their suggestions included to get professional coaching early in the process, engage as many individuals as possible, and give the key executives concrete deliverables around the rollout. Jochems and Rodgers [4] describe their experiences with the adoption of an Agile development process mandated by management. They offer six lessons learned: gradually introduce new concepts if the team is resistant to change, get more buy-in from the entire project team, provide the right training for agile testing methods and tools, allow the project team to evolve and own the process, select a consulting team that meets the needs of the project team, keep the lines of communication open, and quickly address issues.

All the above experience reports had the support, if not the mandate, of the top-level management in order to adopt an Agile methodology. Our report contributes to the IT community by providing complementary empirical evidence of the migration to the RUP iterative methodology in a bureaucratic environment. It explains how such adoption provided the grounds for bottom-up introduction of other Agile practices. We use quantitative and qualitative project metrics as evidence that iterative development paired with testing and refactoring improved software quality and software stability.

Blotner describes a hybrid Agile process introduced at Sabrix [10]. This hybrid process treated each iteration as a mini waterfall project similar to the process adopted by our company during the initial transition stages. The author provides the reasons for the adoption of a hybrid process and why each Agile technique such as iteration, unit tests, and business involvement were introduced. Although Blotner describes the influences of the new process adoption on the bug fixing process, he does not provide empirical evidence of software quality improvement. Kobayashi *et al.* conducted two interview case studies with companies adopting XP and found that solo programming introduced more bugs and misunderstandings than pair programming [17]. Rational followed the Booch iterative-development methodology for the third major iteration of the Rose product and found that high code quality was achievable, thus supporting the 80-20 defect clustering and bug criticality (80% of all bugs are found in 20% of the code). The author focused largely on bug distribution and established that more abstract subsystems had higher rates of defects, so showing that subsystem hierarchy plays an important part on the bug distribution [13]. Berger discusses her longitudinal study findings based on field observations and interviews dealing with the issues of trust, the lack of collaboration, and the censure between business managers when an Agile process is introduced into a bureaucratic government agency [9]. Our experience report provides an example of how companies can use available project metrics to measure how a new practice, in our case iterative development, benefited the company in the long run and how you can leverage this success to support further process improvements.

3. Background

Our experience comes from a case study conducted in a large oil & gas government agency. This agency has a workforce of over 900 employees, with a large IT department comprised of more than 10 different IT programs (portfolios of many projects bundled under a common business area domain). The information provided in this report is based on the experiences of

one of the authors as a consultant for the company, on data collected from the issue tracking system, and on seven semi-formal interviews with the agency's business clients, developers and managers.

We collected data from three complex projects in the largest IT program, one native client-server application and two on-line web applications. These multi-million dollar systems support business critical functions including the digital submission of information, the internal processing of the information, and the publishing of the results to the public. The client representatives for these projects were the internal business area leaders (business clients). The business clients held regular focus groups with industry users.

Development for the first release of the client-server application and the external publishing site started in early 2001 with a group of 4-6 developers. The original vision was to develop a simple application to provide a central data management point for the business clients. The development team followed a waterfall development approach: gather all the system requirements, develop the entire application, and finally hand off to the business clients for testing and approval.

After the first year the scope of the project increased significantly. The new plan was to create a workflow system to provide quicker turn-around processing times for digital submission of data. Late in 2002 most of the original team had left the company and a new team started the development of the second on-line application to support the submission and validation of the electronic data. By early 2004 the small team had evolved from four developers to a larger team of over 15 developers. There were a total of 40+ team members. Development was fast paced and many releases were rushed or delayed due to fixed delivery dates that forced unreliable acceptance testing schedules on the business clients. The result was poor software quality and low team morale. To address these problems, business clients communicated to management the need for better time estimates, better quality, better stability and more reliable testing schedules. At the same time the development team voiced the need to implement a process with more automated practices, better business involvement and a faster response to changes.

Management, motivated by the IBM Rational Unified Process (RUP), made a small number of process changes over a period of approximately 13 months. A part-time RUP process engineer from IBM was contracted to engage expert help as early as possible, as suggested by Megan [2], and Fry and Greene [12]. The process engineer's role was to support the customization of the RUP framework. The specific process changes made and the results of those changes are discussed in section 5.

4. Measuring Quality

To assess the effects of the process changes made we have gathered quantitative and qualitative information about the quality and stability of releases of the application. Although there are a number of metrics that can be considered as measures of software quality [20], the most convenient source of quantitative data for the projects under study comes from bug report data. Bug report data was the only metric available from before and after the process changes were made that could be quantitatively analyzed. In this paper, we use the word "bug" and "defect" interchangeably. Both refer collectively to faults and failures [20] as the teams made no distinction when bugs were being logged. Enhancements requests were logged in a separate part of the logging system and are not included in our calculations. Business clients defined bugs as "*something that should not be happening in the system but it is, or that should be happening but it isn't.*" More formally, IEEE Standard Glossary of Software Engineering Terminology defines a fault as a potential "flaw in a software system that causes a failure, and errors as a passive fault execution leading to incorrect system behavior or state" [1]. Mohagheghi *et al.* conducted a series of six industrial case studies using problem reports to assess software quality [19,20]. Also the authors provide a practical discussion of the impact of the diversity in the terminology used to define defects compared to terms such as errors, faults, and failure, and the political implications of the categorization of bug's criticality and severity.

Specifically, the data we present below is *bug-density* data for each release of the software. A release happens at the end of an iteration cycle after two weeks of code-freeze in an integrated production-like environment. Production releases happen after hours on the last weekend of the month to minimize disruptions to the business clients (approximately 6300 external users). We define bug-density as the number of bugs found per one thousand lines of code changed, added or deleted in the source control repository. This is in-line with the definition of bug-density provided by Mohagheghi *et al.* [19]. The normalization by coding activity provides a better representation of varied levels of effort per release.

Bug reports were extracted from IBM Rational ClearQuest, which also contained reports migrated

from older logging systems. Code activity data was extracted from two different source control systems: Microsoft Visual Source Safe and IBM Rational ClearCase. We developed data scripts to extract activity data from the code repositories. Both the bug and repository data were parsed and imported into a relational database to facilitate analysis. We grouped the data into three time periods based on release dates, Pre-RUP adoption, transition, and partial RUP adoption to measure the variances in bug-density per release per time period. The bug-density for a particular release was calculated based on the number of bugs found during development, testing and in production (after the release was deployed) divided by the thousand lines of code changed, added or deleted in the source control repository for the development period of the release (bug/KLOC activity). Thus bug-density equals the number of bugs reported (during development, testing and after the code was deployed to production) divided by the code activity for the development period. The development period was defined as the first day after a production release up to day of code-freeze for the release.

The data used to calculate bug-density has several important limitations. Reliable and complete bug data was unavailable prior to July 2004. During this period, only critical bugs that could not be resolved immediately were logged in excel spreadsheets. These spreadsheets are no longer available which means that only a few reports were migrated into the new bug logging system. For this reason, the data sample used for the pre-RUP period starts with releases six (R6 – see Figure 1) of the systems dated from July 2004. It is also important to note that, although data was available from July 2004 on, no formal logging process was in place and developers indicated that it is very likely that bugs were still not being consistently logged. Another limitation includes the loss of code activity for five months of the first Transitional RUP release. The development team started using the IBM Rational ClearCase tool that presented numerous challenges and overhead to the team, and a decision was made to migrate the code back to Visual Source Safe. The first five months of development were comprised mostly of proof of concept code. To minimize the effect of this limitation, the numbers of lines of code in the new files were used as the activity data for that period. The most likely effect of these limitations is that our numbers potentially underestimate the bug-density of the releases made before the process changes we are describing in this report.

We also gathered data using interviews. These interviews were designed in part to compensate for

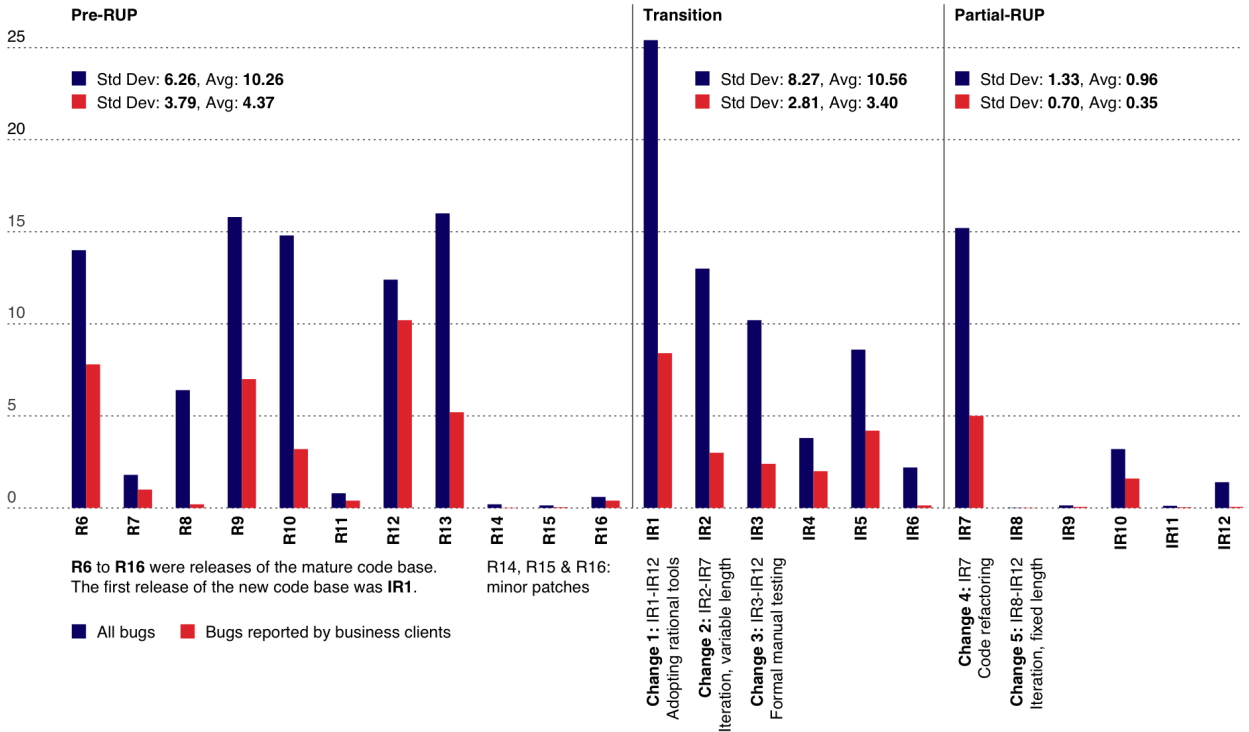

Figure 1. Bug-density data by release.

limitations in our quantitative data [19]. The interviews also aimed to gather information about various stakeholders' views on the process changes. We conducted and recorded seven interviews using the responsive interviewing technique [21]. The interviews lasted from 35 to 50 minutes each. We later transcribed them and analyzed the data.

5. Changes

Figure 1 presents the bug-density distribution and severity breakdown for all releases grouped by stage. The prefixes on the x-axis of the graphs indicate R{number} for a standard waterfall release and IR{number} for releases of the new code base undergoing the process changes.

Change 1: Adopting Rational Tools and New Development/Architectural Approach (release IR1)
At this stage, the adoption path for existing projects was limited to the adoption of the Rational Tool set to make project deliverables more visible to all stakeholders. Release dates were still booked according to business needs as they had been before, following a fixed time and fixed scope approach. The Rational tools included software for source code repository management, for requirement gathering, and for bug and task logging. As well as the organizational adoption of the IBM Rational tools, an Enterprise initiative required that the high profile systems move to .NET, to follow a new Object Oriented Analysis and Design (OOAD) approach. This approach was supported by an in-house development framework primarly responsible for consolidating the serialization and remoting of business objects and database connectitivity. This new enterprise approach was also one of the drives to move to iterative development since it would require many systems to be completely re-written. The motivation behind this enterprise initiative was to assist teams in building better, more reliable and maintainable systems and to consolidate the numerous different architectures that used to cause implementation and configuration issues during deployments.

Change 2: Iteration Length Based on Tasks (releases: IR2 through IR6)
After release IR1, the team moved into what one of the developers called a *"fire-fighting mode"* due to the high number of bugs and software instability. This meant long hours of overtime because of continuous system issues that had lead to a high volume of support calls. The number of high-priority/high-criticality tasks that needed to be delivered at this stage, and how quickly business clients needed them defined the length of the iteration, but the iteration could not exceed 12 weeks until production deployment.

Change 3: Formal Manual Testing (release IR3)
In the context of this paper, "formal manual testing" means testing being manually executed by a team of trained testers hired to perform this role. Such a team was not available during the Pre-RUP stages when the business clients performed all the acceptance testing. As suggested by the RUP key principles to business driven development [5], during IR3 management finally agreed to hire a formal manual testing team.

Change 4: Code Refactoring (release IR7)
Prior to the implementation of RUP, management considered re-factoring and the development of automated tests peripheral activities. Some developers felt that management saw these techniques as "developers' fads", but the team still believes that management support for refactoring came as a result of the improved communication channels supported by the iterations. The refactoring was started to deal with design flaws in the system especially in the security transactions of the in-house development framework. The developers saw this effort as an outlier to the standard releases as business functionality was not being directly delivered.

Change 5: Fixed 6 Week Iterations (release IR8)
Management reviewed the adoption strategy for existing projects and moved all project teams into time-boxed 6-week iterations with tasks prioritized to fit the timeframe. Many Agile practitioners may consider 6 weeks too long of a feedback cycle to be considered iterative, however for an organization like ours, that supports concurrent development projects with dependencies (especially old legacy systems such as mainframes), 6 weeks seemed to better handle the cross team dependencies and risks. This is particularly true when at the end of each iteration the code is indeed put into production and is available to internal and external business clients. It is important to note that according to our interviewees, during the Waterfall stages project releases took anywhere between one to three years to be moved into production with many smaller "fixer-up" releases to handle the missed or wrong requirements.

6. What Actually Changed

6.1. Iterative development, aided by code re-factoring and formal testing, improved software quality and software stability

As mentioned in the previous sections, we calculated the bug-density for all available releases and grouped the releases into three adoption stages: Pre-RUP (waterfall), Transition, and Partial RUP. These calculations are show in figure 1 and are discussed below.

Pre-RUP. Once we grouped the data by release, releases R14, R15, and R16 stood out. Using the interviews, we found that these three last releases of the Pre-RUP period only included minor patches to the client application. As they were not representatives of a standard release, they were not used in our averages and standard deviation calculations. The standard deviation in the bug-density was 6.26 between the releases in the Pre-RUP period, a good indication of the instability of the software being delivered. This was later confirmed by the interviews with developers and business clients. The instability generated issues of trust and censure between IT and business clients as detailed in a previous publication [22]. The highest bug-density encountered in this stage was 15.96 bugs/KLOC; this was also the last "standard" release of the legacy and mature code-base.

RUP Transition. The standard deviation of 8.27 in bug-density, as well as the highest data spike (25.33 bugs/KLOC), indicate that things actually got worse during the transition period. This corresponds to many of the experiences of large companies migrating to iterative methodologies [6,12,14,16]. Based on our interviews and observations, we found that in addition to challenges involved in moving to a new process, the following factors also contributed to these results:

Adoption of new technologies and implementation of new complex functionality. As mentioned in Change 1, the projects migrated to an OOAD development approach using .NET and an in-house development framework. The systems we measured here were the "guinea pigs" of the new in-house framework. They suffered from the lack of design direction with missed documentation about how to properly use this framework and from the bugs encountered during development. Table 1 summarizes the major system differences between the pre and post stages. Another substantial functional module was implemented that added yet another set of stakeholders to the picture. The new module's requirements were complex and poorly documented, in some cases spread over more than four documents. Since the projects under study were existing systems, they suffered from the initial unknowns of adopting a new methodology as well as the "direct migration syndrome" where code was simply adapted to the .NET framework instead of revisited and refactored. On many occasions, the team felt lost with limited support from the RUP Process Engineer who was part-time and a shared resource.

Increased development team. The development team more than tripled in size, from 4 to over 15 developers, during the first RUP transitional release.

Table 1. System differences.

Pre-RUP	Transition and Partial adoption
3 years of code base maturity	New code Base re-written in .NET
Simple 2-tier design	Distributed n-tier application
Releases booked at pre-established project-delivery dates	Releases booked on a need basis during transition, than moved to a 6 weeks iteration schedule.
Established technology	New .Net technology, New in-house development framework, and new document management tool
Code repository - Visual Source Safe	Code repository - ClearCase that caused many merging issues
Volatile requirements, using Word documents that were not kept up-to-date	Requirements using Requisite Pro, and Excel spreadsheets due to tool limitations.
No automated testing	Minimal automated testing
No formal manual testing team	Formal manual testing team

Adding extra resources at first reduces productivity and may lengthen the delivery timelines (Brookes' Law) [11]. Because of the lack of the business domain knowledge, new developers would fix an issue but also create new bugs particularly in the more abstract areas of the subsystems. This is in-line with Blotner's findings [10]. Accordingly, the first release of the new code base (IR1) that was initially estimated as a minor effort, did not follow an iterative approach and had the scheduled production date delayed four times. The total development time was over 47 weeks. The team morale was at its lowest, developers were burned-out due to long overtime hours, and there were over 680 bugs logged against the release.

As the transition was still at the very beginning, management and business partners attributed this

failure to the limited RUP adoption; *"honestly, it felt like nothing had changed, other than we had to do more documentation."*

Partial RUP. The standard deviation of 1.33 in bug-density between the releases after the Partial RUP adoption indicates an increase in software stability. The lowest average of 0.96 also indicates an increase in the overall software quality. We would like to discuss some interesting points in the data: the developers attribute the high bug-density of the first Partial adoption release (IR7) to a major system re-factoring (Change 4) done without comprehensive automated unit tests coverage. Since developers identified this release as an outlier to the standard releases, we did not include it in our averages and standard deviation analysis. The team attributes the spike seen in IR10 to another re-write of the system to support a new in-house framework version, again without any significant automated testing coverage.

6.2. Iterative development, aided by code re-factoring and formal testing, increased customer satisfaction

In Figure 1, we also isolated the bug-density of bugs logged by business clients alone. The quantitative analysis again points out that the transition stage presented challenges. However, after the first transition release, the bug-density logged by business clients dropped. The Partial RUP adoption stage has the lowest average of bug-density (0.35) compared to the Transition (3.40) and Pre-RUP (4.37) stages. As well, it has the lowest standard deviation (0.35) that indicates an increase in the quality and stability of the software delivered to user acceptance testing.

During the initial Pre-RUP development stages, the communication channels were open between clients and the small development team. After the team increased in size and the waterfall process broke down, management prohibited business clients from contacting developers: *"at one point it was so bad that, if we were seen talking to a developer, we would get a nasty e-mail from management."* Business clients missed the small team atmosphere and felt that they had *"lost the team work."*

When iterative development was adopted, the communication channels were slowly re-opened through prioritization meetings, iteration planning sessions, iteration assessments, and iteration user acceptance testing. Any new system developed at the company is now using a more complete RUP process that provides constant opportunities for business clients' feedback. They are welcome to "drop by" and view the progress at any time.

Our interviews with business clients and field observations revealed that iterative development provided better distribution of acceptance testing effort, less pushback on necessary changes, reestablishment of business involvement, introduction of a testing team, improved communication and management of expectations. We describe these business improvements in more detail in a previous publication [22].

7. Continuing Changes

The results above indicate that the transition to RUP was not smooth. Lack of direction, compounded by new technologies and growing development teams, resulted in a temporary decrease of software quality. Because of the complex system changes and methodology transition, the team members indicated that there was simply *"too much going on all at once"*. Supported by the new RUP process, two factors assisted the team to get back on track: the introduction of a testing team, and the major re-factoring of the code base. We asked team members what they think would have happened if a direct transition to an agile methodology had been attempted during the transition stage and several of them felt that the project would have been *"back to waterfall in no time."* However, management supported RUP and the process survived the transition stage. Due to its perceived success, management is now more open to developers' requests.

More Agile techniques are now being adopted at the company. In the past, problems fixed in prior releases would suddenly show up again in production. As the systems had no automated tests, it was very time consuming for the testing team to perform a full regression test. As mentioned in Change 4, management did not support test automation as it was considered to be peripheral and not visible enough to provide immediate business value. Recently top-level management has announced that projects must implement automated functional testing with a defined minimal coverage as the organization moves to test automation. The projects under study implemented a suite of automated functional tests that are scheduled to run daily at 6 AM with the results available on the intranet to all team members. This automation is greatly appreciated by the testing lead; *"that is awesome."*

The new system design included extra layers of dependencies with subsystems from other teams and

the in-house framework. This has caused many deployment coordination issues. A reader might think that automated builds and continuous integration could resolve integration and configuration issues, but during the waterfall days and even the initial RUP transition, there was simply no buy-in from management to invest in these perceived peripheral techniques. One of the teams has finally gained top-management support to hire a continuous integration expert. This expert is currently automating the build process to facilitate early code integration with other teams.

The projects under study tried and failed to hold daily stand-up meetings. Instead, a weekly round-table team meeting is held and seems to be efficient. Not all business stakeholders were available, as this meeting was seen as less critical than other business engagements. This can perhaps be attributed to the adoption path for new projects as they follow iterative development since inception, thus getting business involvement first. These new teams, some of which are now following a more Agile version of RUP, hold daily stand-up meetings that also include the business clients, and, whenever possible, project sponsors. The adoption path for new projects included the iterative RUP lifecycle (inception, elaboration, construction, and transition), Rational Tools, role sets, and selected work products (primarily: Design and Use-Case Models, Software Architecture Document, Iteration Plan and Assessment, Risk List, Issues List, Test Cases, and Use-cases).

Project leads are reviewing the work products required by RUP and only pass Class and other system-related models to the development team. The leads assume the responsibility of producing iteration assessment documents, phase assessment documents, status updates and other required work products using the data gathered during daily/weekly stand-up meetings. Their goal is to minimize the list of required work products and this has been surprisingly well received by upper management. Some new projects have succeeded in replacing use-cases with user stories and in adopting TDD practices. However middle managers still encounter issues with some top-level managers that still require detailed early plans regardless of the iterative approach.

8. What We Learned

If you can, do less to achieve more. As we see in Change 1, there was simply too much going on together with the process changes. It made it hard for the people involved to distinguish if the initial failures were due to process changes or due to other technological/cultural factors.

Perseverance pays off. Iterative development was a difficult but very worthwhile process improvement. As we can see in Figure 1 it took some time before quality and stability actually started to settle down, but the final results were positive.

Don't underestimate the value of a good testing strategy. The developers believe that Change 4 (refactoring) was a very positive exercise that ultimately added to overall system reliability and quality. On the other hand, they were disappointed by the lack of management support to introduce automated tests to the development suite during the RUP Transition. A few developers used Nunit to test the creation of data structures but soon had to abandon the efforts due to aggressive timelines. The high bug-density of IR7 shows that, although automated tests may not seem to add tangible business value, their absence produces negative side effects such as many undiscovered bugs during refactoring activities. In IR7, the majority of bugs were found by the formal manual testing team (Change 3). The testers were able to prevent the bugs from making their way into the acceptance and production environments. As a result, refactoring was seen as a success by management, and teams are now allowed to periodically schedule refactoring iterations.

Think carefully about your process for making process changes. Give focus to the adoption strategy of existing projects. These are the people that are still suffering from existing problems so why not learn from them instead of keeping them as an after thought? In our case, the transition approach was reviewed many times for existing systems. Unfortunately these reviews were triggered due to failures that could have been minimized or even avoided by better processes.

Explain the process changes to your business clients. In our experience, the business clients knew things were changing but they often felt like *"puppy dogs being lead here and there."* We strongly believe that our business clients would have benefited from: training sessions that mapped the reasons for the adoption to the existing problems, clear definitions of what the new process was trying to accomplish, and how they could become involved to help!

Pick your biggest hurt area and measure it in real-time to access the success of your adoption strategy. In our case, software quality and software stability were major issues but we only measured it after the fact. As a result, we missed opportunities to show improvement and tune the process for improved and faster changes.

Listen to those in the "line of fire." In top-down adoptions, developers are sometimes the last ones to be consulted on process changes but the first ones to be made accountable for poor quality.

9. Limitations

In this paper we report on one organization's experience making process changes in several specific projects. Our aim has been to provide a description of our experience working through process changes and so care should be taken in drawing general conclusions based on our results. However, we believe that our findings can be valuable for organizations facing similar challenges, especially when aggregated with other findings [23].

Exploring the consequences of the process changes made using bug-density data is valuable as reducing bug's is a goal of many software development organizations. However, accurately capturing all of the consequences of the changes made in this organization would require a wider range of data (e.g., total development cost) than we are able to report on at this time. This limitation is mitigated by our qualitative data, which fills in some of the gaps.

10. Conclusion

Our objective has been to describe the benefits and challenges involved when an organization transitions from a waterfall process to an iterative approach. To this end we gathered qualitative and quantitative data about a large organization's transition from a waterfall process to RUP. In reporting on the analysis of our results, this report makes three key contributions. First, through quantitative and qualitative analysis, it demonstrates that the iterative aspect of RUP together with investment into manual test teams and refactoring activities can improve software quality and software stability. Second, it confirms that the transitional stages of adopting a new process present challenges to the teams. Third, the perceived success of the iterative adoption can buy the development team enough traction to lighten the process, rebuild trust, and implement some Agile improvement techniques in a bottom-up fashion. While this may seem indirect, it helped to increase the acceptance of Agile ideas in the organization we studied.

11. References

[1] IEEE Std 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology–Description, 1990.

[2] U. Megan. From waterfall to agile - how does a QA team transition? In *Proceedings of the AGILE conference*, pp. 291–295, 2007.

[3] J. Favaro. Managing Requirements for Business Value. *IEEE Software*, 19, pp. 15–17, 2002.

[4] R. Jochems and S. Rodgers. The rollercoaster of required agile transition. In *Proceedings of the AGILE Conference*, pp. 229-233, 2007.

[5] J. Barnes. *Implementing the IBM rational unified process and solutions: a guide to improving your software development capability and maturity*. IBM Press, 2007.

[6] P. A. Beavers. Managing a large "agile" software engineering organization. In *Proceedings of the AGILE Conference*, pp. 296–303, 2007.

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

[8] M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for agile software development. http://agilemanifesto.org/, Last verified: April 2007.

[9] H. Berger. Agile development in a bureaucratic arena – a case study experience. *International Journal of Information Management*, 77(6), pp. 386–396, 2007.

[10] J. A. Blotner. Agile techniques to avoid firefighting at a startup. In *Proceedings of OOPSLA Practitioners Reports*, 2002.

[11] F. Brooks. *The Mythical Man-Month*. Addison-Wesley, 1975.

[12] C. Fry and S. Greene. Large scale agile transformation in an on-demand world. In *Proceedings of the AGILE Conference*, pp. 136–142, 2007.

[13] J. F. Walsh. Preliminary defect data from the iterative development of a large c++ program (experience report). In *Proceedings of the conference on Object-oriented Programming Systems, Languages, and Applications*, pp. 178–183, 1992.

[14] A. Ruhnow. Consciously evolving an agile team. In *Proceedings of the AGILE Conference*, pp. 130–135, 2007.

[15] F. Hartman. *Don't park your brain outside*. Project Management Institute, 2000.

[16] T. R. Seffernick. Enabling agile in a large organization our journey down the yellow brick road. In *Proceedings of the AGILE Conference*, pp. 200–206, 2007.

[17] O. Kobayashi, M. Kawabata, M. Sakai, and E. Parkinson. Analysis of the interaction between practices for introducing XP effectively. In *Proceeding of the international conference on Software engineering*, pp. 544–550, 2006.

[18] J. Lewis and K. Neher. Over the waterfall in a barrel – MSIT adventures in scrum. In *Proceedings of the AGILE Conference*, pp. 389–394, 2007.

[19] P. Mohagheghi, R. Conradi, O. M. Killi and H. Schwarz. An empirical study of software reuse vs. defect-density and stability. In *Proceedings of the International Conference on Software Engineering*, pp. 282–291, 2004.

[20] P. Mohagheghi, R. Conradi, and J. A. Borretzen. Revisiting the problem of using problem reports for quality assessment. In *Proceedings of the International Workshop on Software Quality*, pp. 45–50, 2006.

[21] H. J. Rubin and I. S. Rubin. *Qualitative Interviewing: The Art of Hearing Data* (2nd Edition). SAGE Publications, 2005.

[22] C. Pinheiro, F. Maurer, and J. Sillito. Adopting iterative development: the perceived business value. In *Proceeding of the conference on Agile Processes and Extreme Programming*, pp. 185–189, 2008.

[23] M. Myers. Qualitative Research and the Generalizability Question: Standing Firm with Proteus. *The Qualitative Report*, 4(3/4), 2000.