Improving Quality in Agile Development Processes

Pryscilla Marcilli Dóra[1,2], Ana Cristina Oliveira[1,3] and J. Antão B. Moura[1]
[1]*Systems and Computing Department, Federal University of Campina Grande (UFCG), Campina Grande, Brazil*
[2]*University Center of João Pessoa (UNIPÊ), João Pessoa, Paraíba, Brazil*
[3]*Federal Institute of Paraíba (IFPB), Campina Grande, Paraíba, Brazil*

Keywords: Software Quality, Independent Testers, Agile Process Improvement.

Abstract: Software quality control in agile software development is based on two main principles: pair programming and test-driven development. More recently, "post-agile" techniques seem to favor releasing early over quality. Pressure for low cost, rapid development and to code for new features leads to the allocation of resources to software development tasks preferably rather than to quality control. Such practices may put the responsibilities for development and test on the same team and even facilitate sloppy testing. Albeit in prototyping this may be acceptable and even make business sense that is not the case of scenarios that include system software (e.g., a general purpose mobile operating system) or critical applications for airspace, military, banking or healthcare purposes. In this article, we present our experience in organizing an agile team which is divided into two cells with different responsibilities: software development *per se* and testing exclusively. Preliminary results for the case of a grid computing backup system indicate higher test efficiency and surprisingly, possible shorter time-to-market of the two-cell organization *given complimentary practices are also adopted*. These results may contribute for the on-going discussion on the role and impact of testing in agile development.

1 INTRODUCTION

As stated in (Guerra and 2002): "the quality of software is closely linked to the process used to develop it, and finding a process that fits exactly the specificities of the development environment is almost impossible". Hence, it may be better to adapt and adopt the process that most resembles the characteristics of the environment (Dinakar, 2009). Some environmental features increase the complexity of that task, such as when you have a small team (Crispin and Gregory, 2009).

Agile methodologies, such as eXtreme Programming (XP) and Scrum, treat quality as a responsibility of the entire development, small team. However, in many situations, teams spend more time in production (coding) activities rather than activities related to quality, so the results still show unsatisfactory levels of quality and software discard remains high (Chaos Report, 2011).

Mechanisms for quality control reduce the agility of a development team. In fact, if viewed in isolation, software testing activities require time, more physical resources, and properly trained personnel (Lycett et al., 2003). There is a growing debate in the industry about the need to stress delivery speed over testing in "post-agile" processes–see for instance (Savoia, 2011). For economy of resources, there is a trend to (continue to) embed testers in product teams with the consequence of "the role of test and Quality Assurance (QA) management becoming unclear" (Heuser, 2012). Another trend indicates that testing activities are concentrating more on checking business alignment (uprooting idea bugs) rather than on code bug fixes (Lent, 2013)–i.e., post-agile practices seem to suggest end user testing after product launch. Trends or practices that favor speed over testing may lead to defective software being released more often. Albeit in some scenarios–such as in idea testing by startups or in prototyping–this may be acceptable and even make business sense, that is not the case of scenarios that include system software (e.g. a general purpose mobile operating system) or critical applications–for the banking or healthcare industries, say–which have stringent quality requirements.

Our own experience in developing system

Marcilli Dóra P., Cristina Oliveira A. and Antão B. Moura J..
Improving Quality in Agile Development Processes.
DOI: 10.5220/0004559704110416
In *Proceedings of the 8th International Joint Conference on Software Technologies* (ICSOFT-EA-2013), pages 411-416
ISBN: 978-989-8565-68-6

software however, indicates that agile techniques can be improved with additional or adjusted practices that improve quality and speed simultaneously. This is surprising since additional practices would tend to make the process slower. This paper provides some details a case in such experience in the hope of contributing to the discussion about agile speed *v.* testing controversy.

The case study we consider here is a backup utility (OurBackup (Oliveira, 2007)) from the Our Grid project, an open source free-to-join peer-to-peer (P2P) grid that aggregates computational resources (grid machines) to support the execution of bag-of-tasks (Boot) parallel applications on demand. The project was developed at the Distributed Systems Laboratory at the Federal University of Campina Grande (DSL/UFCG) in Brazil. Several strategies to mitigate the risks of low quality were adopted during the project, including the definition of a software development process originally named ***OurProcess*** (OP), an adaptation of the XP methodology for the development of distributed systems. Further (practice) additions to OP–including the adoption of an independent team for Quality Assurance–led to an agile, mainly quality-centered process named ***OurQualityProcess*** (OQP).

OQP's main characteristics and practices are briefly reviewed in section 2. Section 3 compares results of OP's and OQP's application to the case study. Analysis of the results and recommendations are made in section 4. Results from related work are compared to ours in section 5. Conclusions, caveats and further work are presented in section 6

2 OQP: SOFTWARE QUALITY CONTROL

XP was chosen as a starting point and base for QOP because our team at DSL/UFCG had familiarity with its concepts and usage.

The main goal of OQP is to maintain agility. But to also focus on producing clear requirements and automatic (Buglione and Hauck, 2012), reproducible tests, while being still minimally intrusive, additional practices were added to its XP base (or *Our Process* – OP, as we called it internally). OQP's additional practices and techniques focus on the number of defects identified before a new version is released. The main addition is the insertion of an external quality assurance (QA) team to focus exclusively on the quality of final products. (This does not eliminate the responsibility for quality of the development team which should cooperate with the inserted QA team).

Another adaptation of the base XP process entails validation of requirements, by analyzing and criticizing each specification sentence. While the development team writes software requirements and acceptance tests for the obvious cases, the QA team checks non-functional aspects, such as completeness, correctness and unambiguity. This practice minimizes problems of requirement writing and interpretation, leading to an executable documentation in the form of automatic, cohesive and correct tests that last the software "lifetime".

Yet another adjustment to OP to yield OQP is to halve the duration of XP's typical one-day long tasks. (This is because "software developers" at DSL/UFCG are usually students who need to take care of other daily duties–e.g., attending classes.)

During the implementation of the system functionalities, the practice of *Test-Driven-Development* (TDD) (Beck et al., 2001; Crispin and House, 2002) is also widely used by the development team, while the QA team identifies new test scenarios, sometimes by performing manual testing prior to automation. The practice of refactoring is also made to encompass both teams' codes, developers' as well as the QA team's.

Other additional practices include contract-driven development (Mitchell et al., 2002), execution of different test batteries (*builds*), constant revisions and synchronization between teams. With this incremented and adjusted set of practices OQP's usage is guided by three basic principles as seen next.

2.1 OQP Principles

i) Gradual QA - After the elaboration of basic acceptance tests (by the developers), a process called "explosion of test cases**"** begins with the purpose of stressing the code (when available). Each produced acceptance test leads to one or more tests, which are developed by the QA team.

Once the defects are fixed (by developers), the QA team runs the battery of (possibly manual) tests to validate the correction of defects and to identify new test cases. Testing stops when a set percentage of code coverage is reached. According to a survey of development practioners and managers that should be higher than 90% (Dóra et al., 2013).

ii) Maintainability of Code Health - Every new piece of code must go through a battery of automated tests to be integrated into the repository. At integration one can check the "health" of the

code. Different batteries of tests are defined with different objectives. At first coding, a battery is still simple with only unit tests and *mock* tests (Mackinnon et al., 2000) related to the module under development. The battery of tests grows according to the evolution of the software being developed. A battery of integration tests is performed where the *mock* tests are replaced by integration tests, and every night the full battery of tests is performed creating a daily status of the "health" of the code.

Furthermore, the integration of a new developed test should occur as soon as possible so that all team members have access to the new test and thus increase the verification of newly developed code.

Note that *Development by Contract* (DBC) also contributes to code health by mapping the responsibilities of classes and objects, making the implementation more robust. Business rules are checked by logical assertions that verify whether the input and output data are correctly processed.

iii) Code Review – is enacted during pair programming or by a person who is not involved in the actual coding, preferably by the team leader, either of development or QA. The adoption of this principle reduces errors, misinterpretations, increases code legibility, reduces breaches of contracts, and improves *design*. Its combination to the other two principles causes all developed code to be examined by at least two people in its life cycle.

2.2 Life Cycle

As in XP, integral development of the software occurs through a succession of coded and tested releases. The activities performed for a release are identified by the lines across Figure 1 and are detailed in Table 1. Each release is divided into four phases: Requirements elicitation, Development, Alpha and Beta Testing. During these phases, the activities of the development and QA teams are performed in parallel.

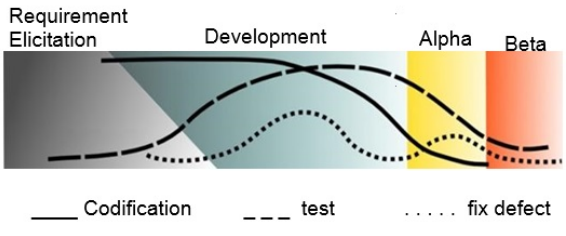

Figure 1: OQP release life cycle.

To validate OQP we applied it to a pilot project and compared the results to OP's at DSL/UFCG.

Table 1: OQP Activities and phases.

Teams	Requirement elicitation	Development	Alpha	Beta
Develop-ment	Write requirement	Code Implementation	Correction of defects	Correction of defects
	Define design	Implementation of unit and integration test		
	Acceptance tests	Correction of defects		
QA	V&V requirements	Implementation of new cases of automated acceptance tests	Manual and exploratory testing	Validation of defect correction
	Acceptance tests			

3 CASE STUDY

The proposed *Our Quality Process* (OQP) was applied to the *OurBackup* (OB) *Home* software (Oliveira 2007), a backup system based on social networks. Initially, a set of six macro-features were defined and implemented under the OP process. These features enable the user to install the software, log onto the system, build his/her social network (by addition and / or acceptance of friends), and lastly, to perform and restore backups. Upon conclusion of the first version (V1), eight new features were added now under OQP, producing a "quality" version 2.

For the comparative study, three *releases* developed with OP (*OurBackup Release*)–OBR_i, i=1,2 and 3; and three releases developed with OQP (*OurBackup Quality Release) OBQR*$_i$, i=1,2 and 3 were considered.

Although every effort was directed to the production of automatic tests, some manual testing was needed. However, if a critical defect was discovered during manual testing, the manual procedure would be interrupted and a new test would be developed to detect the defect or to validate the correctness of the corresponding code.

3.1 Data Collection and Analysis

Initially collected measures were (Table 2): a) % of test classes; b) % testing methods; c) Number of cases of manual testing; and, d) automatic testing coverage. These measures were collected with the aid of the following tools: Jira (bug tracking); FindBugs code static analysis); JUnit (for test development); easyMock (for mock object testing); Bamboo (for continuing code integration); and, SVN (version control).

The number of tests in a project is not the most

appropriate metric to attest to its quality, but it may suggest amount of effort towards quality control. In column *a* in Table 2, we note a gradual increase in the percentage of test classes as OQP is adopted – reaching an increase of 50% over OP's percentage (22% over 14,8%).

The increase in the amount of classes of tests by itself is not an indication that there has been an increase in the effort to produce automatic tests. So, we collected other data that indicate such an increase: column *b* shows the proportion of testing methods relative to the total of developed methods.

Column *c* shows an increase in manual testing as one switches from OP to OQP to produce $OBQR_1$ and $OPQR_2$. But a consistent decrease from $OBQR_1$ to $OBQR_3$ and a lower amount of manual testing with $OBQR_3$ relative to OBR_3 This seems to indicate that OQP's sharper focus on testing tends to reduce manual testing which is tedious and error prone.

The relative larger number of manual tests for OQP can be attributed to this process' permanent availability of testers coupled with the functional code-breaking idiosyncrasies of OB's target distributed environment: different operating systems (OS) or different features across instances of a same OS (such as different versions, Network Address Translation, firewalls, antivirus software, and so forth). Environments such as OB's tend to reduce the realistically possible amount of automatic testing (as a percentage of the entire code) to the range of 20-40% (Harrison, 2013).

Table 2: Initial data comparison.

Version	a)Classes of tests (%)	b)Testing Methods (%)	c) # of Manual Tests	d)Automatic tests' code coverage (%)
OB R1	15,8	8,0	0	0
OB R2	14,4	11,0	74	21
OB R3	14,8	10,8	160	18
OBQ R1	13,5	13,4	275	34
OBQ R2	21,0	16,3	229	62
OBQ R3	22,2	17,7	143	91

Code coverage was measured in terms of lines, methods and classes covered by tests and it was collected using the Clover tool (Clover, 2012). Column *d* brings these data and it shows a consistent increase in code coverage as OQP is continually employed to reach 91% with $OBQR_3$ (meeting the quality baseline of over 90% as indicated by 60% of the respondents in the international survey in (Dóra et al., 2013)). In contrast OP shows a somewhat haphazard behavior.

One may also note that, differently from OP, OQP meets baseline values for other metrics in this international survey: percentage of erroneous deadline and programmer-month effort estimations (within 5 to 15% as indicated by 48% of respondents) and percentage of defects discovered after release delivery (1 to 5%).

Regarding the lifetime of defects, or how fast the team is in resolving defects, a significant improvement with OQP was observed (please refer to Table 3).

Again, Table 3 illustrates a gradual improvement in quality as OQP usage continues (by contrast, OP degrades on the average, while OQP's min, max, average and median times to fix defects improve).

Table 3: Lifetime of defects.

Software Version	Minimum (days)	Maximum (days)	Average (days)	Median(days)
OB R1	-	-	-	-
OB R2	3	801	65	19
OB R3	1	328	168	221
OBQ R1	1	102	24	60
OBQ R2	0	29	9	13
OBQ R3	0	7	6	5

4 EVALUATION AND LESSONS

We believe OQP's superiority over OP in the pilot project of *OurBackup* is due to:

Independence of testers in the QA team: there is a QA boss who is not the development leader. Testers are well regarded by the leader when they find critical defects and vulnerabilities in the software. Testers do not feel guilty when they reveal defects that they have not inserted themselves in the code.

Promoting testing competencies: testers should be trained to improve their skills in detecting failures and writing tests.

Sharper focus on quality: a tester is more productive than a developer that only tests his code in the remaining time of development. Moreover, an external tester, in general, is less likely to ignore errors caused by programming vices.

Also and contrary to agile processes that typically allow little emphasis on testing tasks initially (Reichert, 2012), OQP recommends concentration on tests right from the onset of the project. This may not need to increase budgets by much. Test outsourcing may reduce the need for costly, in-house testing environments, thus easing

the internal competition for resources between developers and testers. In turn, this should make it easier and cheaper to have a two-cell organization as advocated here.

5 RELATED WORK

The debate on agile speed *v.* testing seems to have been kindled by the inability of agile practices of unit and acceptance tests to always meet the need for quality of delivered products (Hislop et al., 2002). This paper indicates that testing and speed need not be traded off if practices that lead to development and independent testing activities are added to agile processes. The results presented here may have shed light on this debate. and may help practitioners' make informed decisions regarding quality management of software development projects.

One may contend at this point that the benefits of continuous, independent software testing activities having been established long ago, are undisputable and for that, need not be revisited. The on-going debate in the marketplace indicates otherwise: practices of yesteryears are criticized for being in want of reform to meet new challenges. Also and despite recent progress, most companies still present very low levels of testing maturity (Experimentus, 2011). As stated in this last reference: "It is perhaps a damning indictment of the industry that after all these years we can consistently design and plan testing, but have no thought or regard for effectively measuring the success and efficiency of this activity (which, combined with the costs of rework, forms a significant proportion of project costs)". This paper offered some insight into measurements of test results.

State-of-the-practice requirements needed to measure (expected) software quality were elicited in an international survey of expert software development managers (Dóra et al., 2013). This survey yielded a software quality metrics baseline for the accuracy of project estimates, the detection of defects before product release, and the test coverage. This baseline was used for comparing results of the test-driven, adapted agile OQP process proposed here against those of its foundation XP process.

The authors of the work in (Artho et al., 2006) have proposed and studied a framework to scale up unit tests, and, as a result, they achieved test coverage of over 99 % with 36 % of the code dedicated to testing. In the case study worked out here, OQP achieved a test coverage of 91 %, with a total test code of 18 %. Although results of both works exceed the test coverage baseline of (Dóra et al., 2013), OQP ended up having half of the test code percentage of total coding effort. One cannot vouch for OQP's superiority (or the framework in (Artho et al., 2006) for that matter), however, given environmental differences underlying both works. A more detailed scrutiny and comparison of both works could reveal interesting, complementary aspects that could be explored to support decisions concerning code coverage against test code amount trade-offs, which was not intentionally made here.

6 CONCLUSIONS AND OUTLOOK

This paper proposed complementing the basic aspects of Agile development processes with a few but significant techniques and practices that, taken together, have been shown effective in improving quality and defect-fixing-delays for the case of a backup utility in a large scale, open source free-to-join, peer-to-peer (P2P) grid computing environment.

The case studied compared results for two different versions of the backup utility. Although this may hinder the significance of conclusions and recommendations, it offered some evidence that investing in independent testing may indeed pay off not only in software quality but in development time as well.

Further work is needed to extricate and isolate cause-effect relationships (between added practices and the observed improvements), to establish the degree of significance of each cause to results, and to generalize conclusions. The early evidence presented here supports OQP's separation of testing from development. This separation may run against current industry trends but it may as well better support agile practioners, particularly those with responsibility for critical application development where a higher degree of compliance between requirements and implemented features is expected.

ACKNOWLEDGEMENTS

The authors thank anonymous reviewers whose comments clarified and enriched the contents of this paper.

REFERENCES

Artho, C., Biere, A., Honiden, S., Schuppan, V., Eugster, P., Baur, M., Zweimüller, B., Farkas, P. Advanced Unit Testing -- *How to Scale Up a Unit Test Framework*. AST 2006, Shanghai, China, May 2006.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D., (2001). "*Manifesto for agile software development*". http:// www.agilemanifesto.org. Accessed in: Dec 17th, 2008.

Buglione, L., Hauck, J. C., Gresse Von Wangenheim, C., Mccaffery, F., (2012). "Hybriding CMMI and Requirement Engineering Maturity & Capability Models". ICSOFT – *7th International Conference on Software Paradigm Trends*, Italy.

Chaos Report (2011). http://blog.standishgroup.com Accessed in: Jun 18th, 2012.

Crispin, L., Gregory. J., (2009). "*Agile Testing: Practical Guide for testers and Agile Teams*". Addison-Wesley Signature Series.

Crispin, L., House, T., (2002). "*Testing Extreme Programming*". XP Series.

Clover (2012). http://www.atlassian.com

Dinakar, K., (2009). "Agile Development: Overcoming a Short-Term Focus in Implementing Best Practices". In *Conference on Object Oriented Programming Systems Languages and Applications* (OOPSLA), Orlando, FL, pp. 579-588

Dora, P., Oliveira, A. C., and Moura, J. A.B., (2013). "A Baseline for Quality Management in Software Projects". In Proceedings of Informática 2013 – *15th International Convention and Fair*, March 18th to 22nd, Havana, Cuba, ISBN 978-959-7213-02-4.

Experimentus, (2011). "Test Maturity Model Integrated (TMMi) – Survey Results, How Mature are Companies' Software Quality Management Processes in Today's Market?" Update 2011, White paper, www.experimentus.com, 20 pp.

Guerra, A., Santana, M. (2002). "*Quality of Software Process or Quality of Software Product*?". In: International Conference on Software Quality, Canada.

Harrison, J. A., (2013). Cited in "*A debate on the merits of mobile software test automation*", James A. Denman, Published 23 May 2013, http://searchsoftware quality.techtarget.com/news

Heuser, M., (2012). "Exploring the shifting roles in test and QA management". In http://searchsoftwarequality. techtarget.com. Accessed in: Oct 12th.

Hislop, W., Lutz, J., Naveda F., McCracken, M., Mead, R., Williams, L. A. (2002). "Integrating Agile Practices into Software Engineering Courses". In 15th CSEET.

Lent, J., (2013). "Software Testing Trends 2012: Business Alignment, Not Bug Fixes". http://searchsoftware quality.techtarget.com. Accessed in: Jan 28th, 2013.

Lycett, M., Macredie, D., Patel, C., Paul, J., (2003). "Migrating Agile Methods to Standardized Development Practice". In: IEEE Computer Society, pp. 79-85.

Mackinnon, T., Freeman, S., Craig, P., (2000). *"Endo-Testing: Unit Testing with Mock Objects"*. XP eXamined by Addison-Wesley.Meyer, B., (1997). "Object-Oriented Software Construction". Second Edition, Prentice Hall.

Mitchell, R., McKim, J., Meyer, B., (2001). "*Design By Contract, by example*". Addison-Wesley Publishing Company.

Oliveira, M., (2007). "*OurBackup: Uma Solução P2P de Backup Baseada em Redes Sociais*". Dissertação de Mestrado, COPIN - UFCG, Campina Grande, PB, Brasil (In Portuguese).

Reichert, A., (2012). "*How to Focus an Agile Scrum Team on Quality and Testing*". http://searchsoftwarequality. techtarget.com, first published in August 2012.

Savoia, Al., (2011). "Test is Dead". *6th Annual Google Test Automation Conference* (GTAC). Uploaded on Oct 27, 2011.