Testing in an Agile Product Development Environment: An Industry Experience Report

Andreia M. dos Santos*†, Börje F. Karlsson†, André M. Cavalcante†, Igor B. Correia† and Emanuel Silva†
*Federal University of Amazonas (UFAM)
Av. General Rodrigo Octávio Jordão Ramos, 3000, 69077-000, Manaus-AM, Brazil
†Nokia Institute of Technology (INdT)
Av. Torquato Tapajós, 7200, 69093-415, Manaus-AM, Brazil
{andreia.santos, borje.karlsson, andre.cavalcante, igor.correia}@indt.org.br,
emanuel.silva@indt.org

***Abstract*—Product development nowadays requires great focus on time to market, as well as in quality, in order to meet customer expectations. Several agile methods and methodologies have been proposed to tackle the early release of software products and meet these stringent deadlines. However, in an agile team, the quality guarantee usually performed by tester groups is directly affected by these huge changes in the way and time when tasks are performed during a project; the traditional tester role usually does not adjust well to these new scenarios. This paper presents empirical observations on test practices in agile projects. These projects were developed at Nokia Institute of Technology (INdT) in the Network Technologies group, where protocol compliance, performance, low level details, and other requirements have to be guaranteed. We provide an experience report on agile testing and identify some important issues in dealing with it and on adapting the tester role to this kind of environment.**

I. Introduction

The level of consumer demand for software products is increasingly high, accordingly, factors such as shorter delivery times and quality in the product or service became more and more crucial. Problems such as high cost, high complexity, difficulty of maintenance, and a disparity between the needs of users and the product being developed, become increasingly evident in the processes of software development [1].

In order to alleviate this situation, different development methodologies have been proposed and implemented in software development companies. In contrast to traditional methodologies, the so called agile methodologies (which have seen wide adoption) focus on different principles and are known to aim at reducing the bureaucracy associated with product development activities [2].

In this context emerged The Agile Manifesto [3], from a gathering of experts in software development who were concerned about the difficulties faced in the traditional methods of development. The Manifesto presents a set of principles that define ideal criteria for development processes. The values to be followed by agile development teams, and which were described in the Agile Manifesto, are:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

Testing is a central component of agile software development, as several core practices used by agile teams relate to it. Techniques such as TDD[4], unit testing[5], refactoring[6], and even best practices to work with legacy code[7] are some examples. Also, the Agile Manifesto itself relates to testing, with its emphasis on individuals and interactions, working software, customer collaboration, feedback and conversations.

In an agile process, the whole team works together at the same time. As such, traditional test teams need to undergo a major transformation to adapt to this new reality, since they must stop being reactive and now play a vital role in the interaction with the developers, business analysts, and customers. This study seeks to provide accounts of experiences in software development at a research and development institute, which will report strategies for adapting the testing process within the reality of an agile development process following Scrum [8]. To accomplish this goal, an analysis was made on the principles of the most used agile methodologies and we seek to confront the position of software testing in the traditional processes and on agile ones. Also, the evolution of a agile test "process" is outlined. Although we use Scrum and our proposal is built on top of its framework, our suggestions can be applied to different agile methods.

This paper is organized as follows: Section II presents a brief description about agile methods and testing in the software development context. Section III describes the case study used to provide this experience report on agile testing. Additionally, this section is intended to identify some important issues in dealing with agile environments and how to adapt the tester role for this kind of environment. Finally, conclusions are given in Section IV.

II. Agile Methods and Software Testing

A. Agile development methodologies

According to Ambler [9], the waterfall process of software development limits developers. Sometimes being a cumbersome and expensive development process, many organizations - especially small ones - end up choosing not to use any type of consolidated process. This fact highlights the need to use agile methods that are not extremely focused on documentation [9]. Projects using agile methodologies assume that change is common in software projects (and software-heavy projects) and thus value ongoing planning, emphasizing human aspects and adaptability to rapid changes in features and scope.

While other processes try to address the problems in the waterfall model with interactions and an incremental approach (e.g. Rational Unified Process - RUP) they are still viewed as too rigid and bureaucratic. Currently there are numerous approaches to agile software development; some of the most

978-1-4577-1490-0/11/$26.00 ©2011 IEEE

popular models are Extreme Programming (XP), Scrum, Lean, and Kanban. No less important and also used by a variety of organizations are Crystal and Agile Modeling (AM), but all have the following in common:

- Continuous planning;
- Involving the client in all phases of the project;
- Interactive and incremental process;
- Clear definition of roles;
- Iterations costs and scope well defined;
- Discipline in the workflow.

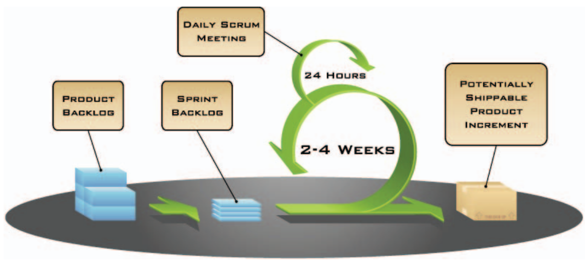


Fig. 1. Scrum Process Flow [10]

B. *Agile Development Environments with SCRUM*

Scrum reinforced the idea of empirical process control. Projects are divided into sprints, which typically last from one to three weeks. After each sprint, actors and staff members meet to assess project progress and plan their next steps. Scrum assumes the premise that software development is a complex effort and too unpredictable to be fully planned beforehand. Instead, one must use a defined empirical control process to ensure visibility, inspection, and adjustment of plans [11]. Fig. 1 illustrates the life cycle in developing projects using Scrum.

Also according to Schwaber [11], using the Scrum agile process is vital to have the vision of the product to be created. It is this vision that defines the list of features expected by the client/sponsor. It stands out from other agile methods for its greater emphasis on project management (on a strategic level). Scrum brings together monitoring and feedback in general with fast and daily meetings involving the whole team, aiming at identifying and correcting any deficiencies and/or impediments in the development process as soon as possible [8]. In this case, the process of product development is really iterative and incremental, i.e. planning is always ongoing and from time to time assessments are made about the product already created and the list of features not yet implemented. This allows adaptation to changes, since the process itself has already decided to account for possibilities for change in the scope of product development.

Scrum has three key roles [8]:

- Product Owner: Responsible for communicating the product vision to the team and must also represent the customer interests through requirements and priorities;
- Scrum Master: Liaison between the team and Product Owner, but does not manage the team. Instead, he/she works to remove the obstacles that are blocking the team from achieving the sprint goal;
- Team Member: Responsible for completing the work. Cross-functional and empowered to make effort estimates and take technical decisions.

C. *Agile Test and Quality Assurance*

As discussed by Rocha et al. [12], quality assurance must be conducted in all phases of the development process so that defects are eliminated as close as possible to the phase in which they are introduced, to avoid increased costs if left for later steps. When working with agile methods/frameworks it is no different. According to the SWEBOK [13], software testing means to verify the dynamic behaviour of a program via a finite set of test cases, suitably selected from a field of usually infinite execution paths, through checking the expected behavior. However, as Crispin and Gregory claim [14], in agile projects testers do more than just perform tasks of testing and therefore should also be considered as developers (as the rest of the team). This is due to the fact that the whole team is focused on delivering a high quality product and business value. Testers, or the quality assurance (QA) department, are no longer viewed as the quality gatekeepers, the team is.

III. Case Study

A series of projects developed at INdT during the last two and a half years, all using Scrum as the overall management framework, formed the basis for this study and are briefly described below. Two of them are more closely presented here (projects M and N) as they were the main testbed for the evolution of our current approach to agile tests, but lessons learned and adjustments performed on the other projects also provided insight into different issues and served as input for our better understanding of the problem space.

Project M aimed at a marketing system to reward mobile phone usage. Users would accumulate credits that could be converted into minutes when making calls or used to buy products and services from partner companies. The system works like a loyalty program similar to the mileage programs offered by airlines around the world. The project was developed in two phases and was composed of backend system, mobile clients and web client. It was developed for over one year and trialed with final users and a partner company in Brazil.

In project N, the goal was to provide a flexible billing platform for pre and post paid mobile users, based on specific business rules for listed protocols and URLs; with differentiated billing by data connection. A real-time monitoring tool displaying information on data usage per user was also created. A prototype version of the system was created and interacted with different pieces of equipment inside a mobile operator network. A new project is now building on the created system.

Additionally, three other projects also served as learning sources and their experience was used as input on tailoring our current approach. Projects T1 and T2 involved the reception and decoding of digital TV transmissions on mobile phones. T1 includes adaptations to a hardware device for reception of the digital TV signal, decoding of the data stream, and the development of an embedded TV player. T2 involved the design of a middleware to support interactivity on top of the previous solution (according to the Brazilian digital TV standard) and also involved distributed teams.

Finally, project P dealt with the development of a geo-referenced messaging system with social characteristics. This project involved both backend and mobile application development, had international stakeholders, and was trialed in Brazil and the US with groups of end users.

While not central to the process reported on this paper, experiences from projects T1, T2 and P helped inform the evolution of the test process currently in use and also allowed us to analyze the same issues from different points of view.

A. *Project Characteristics Analyzed*

In order to have this study better serve as guidelines for new teams facing agile tests we decided to focus on the point of view of a new team, quite green into agile development. The same core team participated in project M during the development of the first prototype and then during the second phase of the project. And the same test focused team members moved to project N. So this is the timeline we follow to comment on test process evolution.

There were many challenges faced by the team during phase 1 of project M, and specially by the testers on the team, since they were new to agile tests and the number of people focused on this function was very small (one test analyst and one intern). Also, in line with allowing the team to have autonomy, there was no top-down imposed standard process.

The first approach at project M was to define test cases soon after the Sprint Planning meeting, immediately after the definition of which stories to develop in a given sprint, and in parallel to code development. Even though the idea of developing both code and tests in parallel was right, there was an informal separation of the team into two sub-teams: a development team and a testing team. This led to the test run only starting after all development was complete. As the "handover" to testing happened late and the number of test-focused team members was small (2), the testers got swamped with too many tasks and too little time, and were not able to test everything into the sprint. Some stories were not completed because tests were not performed. Also, the test team relied heavily on the programmers to set up the test environment for execution.

Similarly to many new teams into agile, this quickly leads to a micro waterfall cycle were testing comes after coding is completed, but before the next iteration. Unfortunately, if development takes longer than anticipated or there was an estimation error, there is only a small timeframe for the tests. Not only testing has little time, but when bugs were identified, there was not enough time to fix them and re-test before the sprint was over.

In trying to fix the issues, the team's second approach was to run the tests after completion of the current sprint, i.e. features would be broken into two stories; the first (development) happening on a sprint, and the second (testing) only happening on the following sprint. This approach led to stories (features) being considered done before actual testing was performed; giving the team a sense of progress. However, there was a major hidden problem. Later tests could reveal serious defects in the system structure and since the test execution occurred out of phase with feature development, programmers had already added new features to the system, making the problem worse and solutions much more complex. Often it was necessary to remedy a defect immediately, delaying the features and failing the sprint.

As the project matured the team refined the approach to testing and started to get convinced that a process where the whole team did testing was the way to go. Being pro-active and automating test cases were the main axes of this process.

According to Crispin and Gregory [14], the key factors for a successful agile testing approach are:

- Look at the Big Picture;
- Collaborate with Customer;
- Build the Foundation for Agile Core Practices;
- Provide and Obtain Feedback;
- Automate Regression Testing;
- Adopt an Agile Testing Mindset; and
- Use the Whole Team Approach.

Having this in mind, and the team real-life experience with the issues described, the underlying problems with the previous approaches were discussed and a new model was gradually adopted. Under this paradigm the team sought to adapt to the new reality of agile development and at the same time, guarantee the quality of each sprint deliverable.

B. *Testing in an Agile World*

There were many doubts about the role of test analysts and testers in general within the agile philosophy, and also specifically in Scrum. Where do programmers and testers fit in a new agile project? How can there be code and tests in such short interactions? Who is in charge of avoiding defects getting to production? In the recent literature [14] it is argued that the role of software testing in an agile methodology can be represented by four quadrants (shown in Fig.2).

Programmers write unit, component-level, and integration tests to make sure the smaller units of code work together as intended. These are all critical to a successful project. Marick [15] describes these types of tests as "supporting the team" as they help programmers know they have a solid base and decide what to code next. Also they facilitate changes to the system by serving as a guarantee that unintended changes do not get into the system.

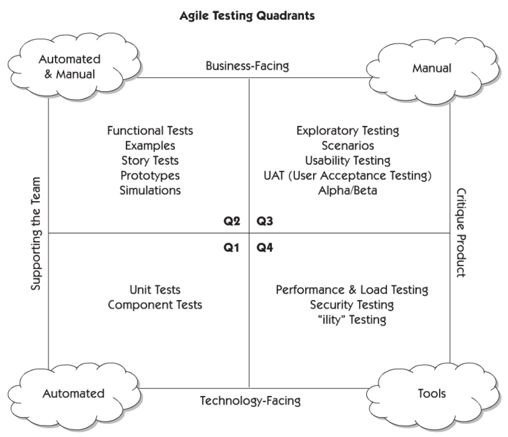


Fig. 2. Agile Test Quadrants

This concept of testing as support for developers is new to many team members (both programmers and testers) and is one of the biggest changes from traditional testing. Tests in quadrants 1 and 2 can be seen more as testable requirements specification, and design aids, than validation/verification tests. Tests in Q2, particularly, describe the business details of each story at a functional level, each one verifying a business condition. Often these tests duplicate some of the effort at the unit/integration level; however, these tests are oriented toward confirming the desired system behavior at a higher level.

Ideally, all of these tests should be run as part of an automated continuous integration (CI), build, and test process. During project M the team reached this same conclusion and decided to setup a CI server. We are now moving towards a common CI environment for most development projects in the Networking Technology group.

Quadrant 3 contains the business-facing tests that prod the created software to check if it does not meet expectations or lacks functionality. These business tests try to emulate how real users would use the system and identify its shortcomings. Automated processes can be used to help generate data, but this is usually only manual testing. Often users perform these types of tests as some kind of user acceptance testing (UAT). Exploratory testing, as in Q3, is not ad hoc testing, which is impromptu and improvised. In exploratory testing, a tester designs and performs test scenarios, analyzing the results.

The types of tests that fall into quadrant 4 are again technology-focused and intended to critique non-functional requirements such as performance, scalability, robustness and security. An approach to them is to quantify measurements and the acceptance criteria for each story, or to deal with them as specific stories.

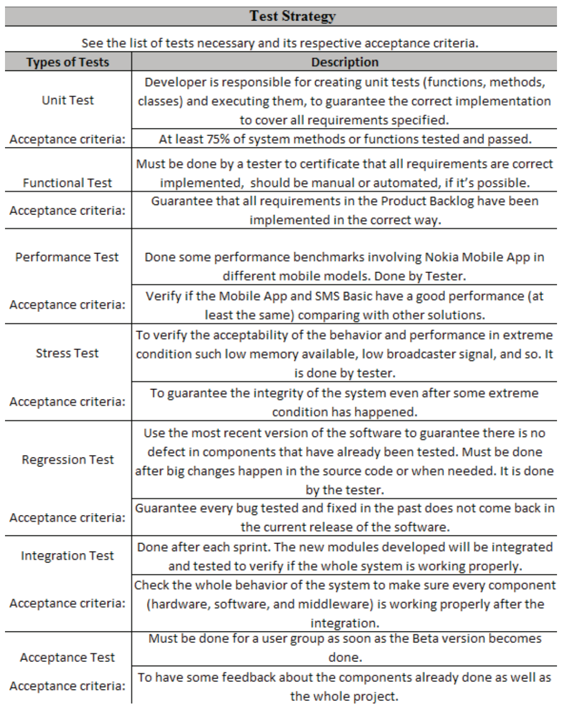
C. *Evolution of the Agile Test Process*

Different practices and strategies are used for test activities by different agile methods. In Scrum, tests are usually represented as tasks to be executed during a development sprint (that deliver a increment of potentially shippable software), or story may have the sole purpose of performing test-related tasks, code improvements and defect corrections. Nonetheless, a feature is only ready to be delivered to the customer after undergoing integration, functional and system tests. It is extremely important to include testing in estimates of story size as, in some cases, testing a piece of functionality might take longer than coding.

Bellow we outline the employed test strategies as they evolved during the presented projects and then we describe our current validation and verification model.

During the first stage of project M, the project team decided on a test strategy composed of: white box tests, to be performed during functionality development; black box tests, to test functionality implemented during the iteration; Definition of test cases, test cases should be documented in a test plan, be them automatic or manual; Integration tests, to be run as in each iteration test plan and to generate error reports; Bug fixing, to address issues discovered during testing, bugs found are prioritized and distributed among team member for correction; and QA Audit, for monitoring the iteration, to check the status of activities and make changes in the plan if necessary. Later, stress tests and regression tests were included. Fig.3 shows some more detail on the test strategy during this phase, along with some metrics.

This was a first team attempt at properly using tests in an agile project and reflected some of the issues mentioned regarding the internal separation of the team between programmers and testers, as well as the reliance on the old policy of QA as quality gatekeepers. After problems with this approach were identified, the team started refining the strategy and moving to a new model, involving more members of the team on the test-related tasks.



Test Strategy

See the list of tests necessary and its respective acceptance criteria.

Types of Tests	Description
Unit Test	Developer is responsible for creating unit tests (functions, methods, classes) and executing them, to guarantee the correct implementation to cover all requirements specified.
Acceptance criteria:	At least 75% of system methods or functions tested and passed.
Functional Test	Must be done by a tester to certificate that all requirements are correct implemented, should be manual or automated, if it's possible.
Acceptance criteria:	Guarantee that all requirements in the Product Backlog have been implemented in the correct way.
Performance Test	Done some performance benchmarks involving Nokia Mobile App in different mobile models. Done by Tester.
Acceptance criteria:	Verify if the Mobile App and SMS Basic have a good performance (at least the same) comparing with other solutions.
Stress Test	To verify the acceptability of the behavior and performance in extreme condition such low memory available, low broadcaster signal, and so. It is done by tester.
Acceptance criteria:	To guarantee the integrity of the system even after some extreme condition has happened.
Regression Test	Use the most recent version of the software to guarantee there is no defect in components that have already been tested. Must be done after big changes happen in the source code or when needed. It is done by the tester.
Acceptance criteria:	Guarantee every bug tested and fixed in the past does not come back in the current release of the software.
Integration Test	Done after each sprint. The new modules developed will be integrated and tested to verify if the whole system is working properly.
Acceptance criteria:	Check the whole behavior of the system to make sure every component (hardware, software, and middleware) is working properly after the integration.
Acceptance Test	Must be done for a user group as soon as the Beta version becomes done.
Acceptance criteria:	To have some feedback about the components already done as well as the whole project.

Fig. 3. Initial Test Strategy

On project M's second phase, after the proof-of-concept system was approved, and taking into account the lessons learned in the previous phase, the team felt more comfortable in structuring a more efficient process for quality assurance. At this point a new strategy was defined were the team decided that functional tests should be automated (as much as possible) and continuous integration (CI) was introduced into the development process to better take advantage of frequent automated testing (unit/integration/functional). Fig.4 illustrates the new testing process. However, one can note that there was still some functional separation of testers and programmers in this flow. In this stage the team also defined a set of validation and verification criteria to be applied to development and to the closeout of each sprint and release candidate.

The regularity of deliveries (builds or deploys) during a sprint, is defined by the team during sprint planning, as it depends on the features to be implemented in the iteration. A "Delivery Package" (DP) was created to represent the final deliverable of a sprint and some metrics and procedures to be followed were specified. This package would include a "Release Notes" describing the implemented requirements (according to the backlog), a report of all executed tests (manual and automated), and a list of known defects (also documented in a DTS system). At the end of a sprint, team and PO would review the information, re-prioritize stories, and plan what to do with the remaining open bugs (if applicable). It was also documented in the development process that, for every sprint: a certain percentage (defined by the team) of code shall be covered by unit testing; the number of major defects shall be less than or equal to 3 (no blocker defects); and all the evidences of attaining the Sprint Goal (as defined by the PO) would be summarized in a presentation. It was also decided that for every sprint that should result in a release candidate

(RC) for the client, a story of verification and validation must be executed, to make sure that:

- All test cases are reviewed and executed by the team;
- No blocker, critical or major bugs are open;
- Only a reduced number of minor defects are still open;
- The presentation of the work package is ready for PO review and presentation to clients.

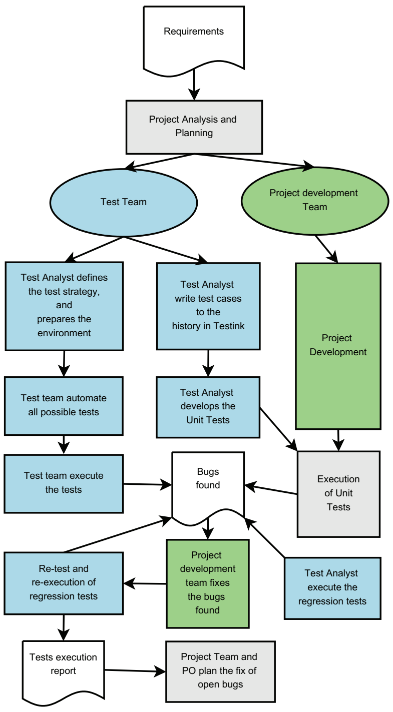

Requirements
Project Analysis and Planning
Test Team
Project development Team
Test Analyst defines the test strategy, and prepares the environment
Test Analyst write test cases to the history in Testink
Project Development
Test team automate all possible tests
Test Analyst develops the Unit Tests
Test team execute the tests
Bugs found
Execution of Unit Tests
Re-test and re-execution of regression tests
Project development team fixes the bugs found
Test Analyst execute the regression tests
Tests execution report
Project Team and PO plan the fix of open bugs

Fig. 4. More concrete Test Process

The results showed significant improvement not only in bugs fixing and overall quality of deliverables at each sprint, but also on development speed, communication with the PO and, consequently, on the development flow as a whole.

As previously mentioned, after project M, the team members most involved in testing started working in project N with another group of developers. Some changes were then introduced to the test process (4) to further improve it. More effort was put into the whole-team approach to testing as some artificial boundaries still existed and were considered as hindering the team. Automated tests for bugs found (and not previously covered by any test case) were added to the set of regression tests.

Also, some automated functional tests were added to the regression suite. However, one important issue surfaced during project development. Due to specifics of the project, data traffic needed to be generated by actual mobile phones and this data would then go through some networking hardware before reaching some modules of the system. As the hardware was also part of the solution being developed, many tests (and regression tests) needed to be performed manually, what slowed development and diminished somewhat the benefits of using CI. Later a hardware simulator was created to generate the network traffic and event dispatched by the equipment and the regression test are now being automated. Nonetheless, many functional tests cases still need to be performed by hand, but the benefits of the additional frequent automated tests far surpass the effort of the automation.

D. Common Problems and the Whole-team Approach

Even though some of the people involved on these projects had previous experience with agile projects (including some of the authors), in line with the idea of agile teams having autonomy for technical decisions, there was no top-down defined process at first. The premise was that the team needed to identify shortcomings in the test process and propose solutions to those. As expected, some common problems surfaced at different times during the case study timeframe. Some of the most important were: programmers left responsibility for quality in the hands of the testers (who were powerless to make changes to the process by themselves); testers got frustrated that testing was getting squeezed at the end (iterations were over before stories could be tested); testers acting as quality police; team afraid of making bugs visible during development.

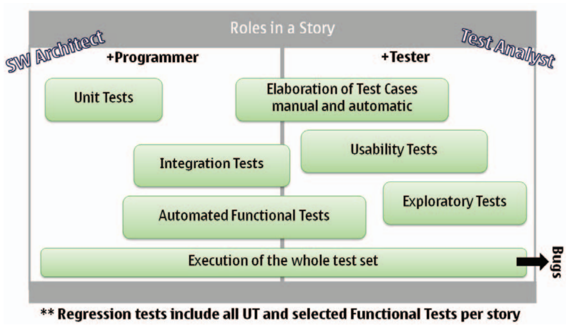

Roles in a Story
SW Architect
Test Analyst
+Programmer
+Tester
Unit Tests
Elaboration of Test Cases manual and automatic
Integration Tests
Usability Tests
Automated Functional Tests
Exploratory Tests
Execution of the whole test set
Bugs
** Regression tests include all UT and selected Functional Tests per story

Fig. 5. Roles in a Story

Focus on team communication and cooperation is paramount in an agile environment. The whole-team approach does not limit particular tasks to specific team members. Any task might be completed by anyone, or pairs of team members. In testing, this implies that the team takes responsibility for all kinds of testing tasks, not just the testers. Even when looking at the story-level, while programmers start to think about implementation, everyone should always keep a mind on how one could test the story. Getting people to change habits is a problem not only for testers, many programmers also present resistance to performing test tasks as if they are somehow inferior. Depending on what features are implemented in a given sprint, different types of tests need to be performed. Fig.5 shows the typical team member roles most related to in each test type (but not exclusively either).

It is important to highlight the importance of test-related tools when it comes to agile development. Projects M and N made use of: JUnit, for unit testing; Selenium, as automation tool for web application functional testing; TestLink, for managing test cases; and Bugzilla, for tracking bugs. Tools and automation help a long way in freeing programmers and testers to focus on higher level tasks and concerns.

E. Lessons Learned

The whole-team approach is a paradigm shift for many developers, but in agile environments (that are focused on dealing with change) it is extremely important. Having the team share a workspace helps communications, facilitates cooperation in performing tasks, keeps everyone aware of the context of each sprint (especially testers, that usually belong to other parts of the organization) and help everyone care for the developed product.

Have a test strategy defined, as well as having clarity about the whole testing activity, also facilitates team work. A test strategy helps the team visualize testing activities more clearly and enables them to better contribute. The definition of done [11][16] is very important to test planning for each story, so all doubts about it and the acceptance criteria of each story must be removed so that there is a common understanding of the effort by team members.

One of the main points of improvement, according to the team in Project M was the adoption of CI, as it forced the implementation of unit testing in every build and functional testing automation, which reduced execution time of tests and allowed for quick discovery and fixing of many issues. When programmers and testers cooperate on automating test, the whole team realizes that they help create and maintain a safety net, which enables everyone to accommodate changes with confidence, as the team will now as soon as possible about any unintended impact. Testers can then also help with different tasks in the project, from working with customers to help write test cases that better define acceptance for upcoming stories, to pairing with programmers to look for gaps in tests. Also, when the whole team is comfortable with the new focus on tests, testers feel more confident and contribute with their different mindset and points of view on even planning and design. Team members also realize that more detailed test specification needs to wait for iteration planning, but that the team still needs to think about testing at a high level and budget time for it on each story. Interestingly, if there are no "testers" in the team (or not enough), other team members start to wear a "tester hat" for each iteration, writing test cases, performing manual tests, tracking defects, etc.

Finally, one big issue of confusion between team members is how to deal with bugs and defects. Every new project faced this same problem. Our current approach is that intra-story issues should be solved before the end of the story. These generally are not fully tracked as bugs, but "bug-tasks" in the story itself. If some issues are prioritized as minor impact, they can be put in the DTS or in a bug backlog for dealing with them later. Bugs that are handy to keep in a DTS are the ones that are intermittent and hard to track down. When bugs are pulled to be fixed, they can be placed inside a bug-fixing story or in a story for a related feature; and the story is estimated including the effort to analyse and solve them. But when estimating new stories, team should not pad the effort estimates to account for possible bugs; the effort necessary for a feature assumes the best case scenario as it is not feasible to account for all kinds of problems.

IV. Conclusions

Our experiences in this set of projects (along the last two and a half years) highlight the importance of testing within agile methodologies and of implementing good practices following the whole-team approach. Although we built our proposal on top of Scrum, our findings and suggestions can be applied to different agile methods.

Testing practices support several other agile practices and values, such as continuous integration, refactoring, and mainly the adaptability to change during development. All increments of software must be validated by sets of test cases for unit, integration and acceptance testing to validate functionality. Exploratory tests and tests related to the operation of the system as a whole also should be used to criticize the system and to make sure that non-functional requirements are met.

The development of a cross-functional team is essential. Team members need to work closely with one another and communicate frequently. Basically, making testers think a little bit more like a programmer and programmers begin to think more like testers. By automating the tedious, low-level boundary condition test cases, both testers and developers are freed to focus on the bigger picture and on customer needs.

Another important issue to note is that it is necessary to give the team opportunity to make mistakes and discuss problems and possible solutions. This doesn't mean that the team shouldn't follow best practices or take time to analyze stories and think through the appropriate architecture and design. But if bugs initially make it to the next stage, nobody should be blamed (especially not only the testers). Instead, the whole team should analyze what happened and start taking steps to prevent a recurrence. Special attention should be payed to new teams in agile development to make sure enough time is budgeted for testing and making sure a feature works. Always! As Crispin puts it: "If there is no time to test a new feature, then there is no time to develop it in the first place" [14].

Despite the good results obtained with the proposed model, it can still be improved to enable the team to better perform, for example: making better use of continuous integration, improvements to how to systematically handle bugs, code coverage for higher level tests, more collaboration between testers and business team, and an analysis of metrics to try and identify other weaknesses in the process.

Acknowledgment

The authors would like to thank INdT for the opportunity to share these findings and all team members in each project for their support in collecting the data.

References

[1] I. Sommerville, *Software Engineering*. Pearson/Addison-Wesley, 2004.
[2] M. Fowler, "The New Methodology," 2001. [Online]. Available: http://www.martinfowler.com/articles/newMethodology.html
[3] K. Beck *et al.*, "Manifesto for agile software development," 2001. [Online]. Available: http://agilemanifesto.org
[4] K. Beck, *Test Driven Development: By Example*. Addison-Wesley, 2002.
[5] G. Meszaros, *xUnit Test Patterns: Refactoring test code*. Addison-Wesley, 2007.
[6] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.
[7] M. Feathers, *Working Effectively with Legacy Code*. Prentice Hall, 2004.
[8] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Prentice-Hall, 2002.
[9] S. W. Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. J. Wiley, 2002.
[10] "Mountain Goat Software - Scrum Overview," 2005. [Online]. Available: http://www.mountaingoatsoftware.com/scrum/overview
[11] K. Schwaber, *Agile Project Management with Scrum*. Microsoft Press, 2004.
[12] J. C. Rocha, A. R. C.; Maldonado and K. C. Weber, *Software Quality: Theory and Practice*. Prentice Hall, 2001.
[13] A. Abran and J. W. Moore, *Guide to the Software Engineering Body of Knowledge (SWEBOK), 2004 Edition*. IEEE Press, 2004.
[14] L. Crispin and J. Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley, 2009.
[15] B. Marick, "My agile testing project," 2003, http://www.exampler.com/old-blog/2003/08/21/.
[16] B. Gloger, *Scrum: Produkte zuverlässig und schnell entwickeln*. Hanser Fachbuch, 2009.