Software Testing Process in a Test Factory - From Ad hoc Activities to an Organizational Standard

Rossana M.C. Andrade, Ismayle S. Santos, Valéria Lelli, Káthia Marçal de Oliveira, Ana Regina Rocha

Software Testing Process in a Test Factory
From Ad hoc Activities to an Organizational Standard

Rossana Maria de Castro Andrade[1,*], Ismayle de Sousa Santos[1,†], Valéria Lelli[1],
Káthia Marçal de Oliveira[2] and Ana Regina Rocha[3]
[1]*Federal University of Ceará, GREat, Fortaleza, CE, Brazil*
[2]*University of Valenciennes, LAMIH, CNRS UMR 8201, Valenciennes, France*
[3]*Federal University of Rio de Janeiro, COPPE, Rio de Janeiro, RJ, Brazil*

Keywords: Software Testing, Test Factory, Test Process.

Abstract: Software testing is undoubtedly essential for any software development. However, testing is an expensive activity, usually costing more than 50% of the development budget. Thus, to save resources while performing tests with high quality, many software development companies are hiring test factories, which are specialized enterprises for the delivery of outsourced testing services for other companies. Although this kind of organization is common in the industry, we have found few empirical studies concerning test factories. In this paper, we report our experience in the definition, use, and improvement of a software testing process within a test factory. To support the implantation of the test factory, we applied the PDCA (Plan-Do-Check-Act) cycle using the lessons learned in the PDCA check phase to improve the testing process. As a result, we have decreased the number of failures found after the software delivery and thus a higher value for DRE (Defect Removal Efficiency) measure. We also present 12 lessons learned that may be applicable by other test factories.

1 INTRODUCTION

Software testing is one of the most expensive and time-consuming activities during software development (Shamsoddin-motlagh, 2012). The costs associated with this activity can reach more than 50% of the total costs of producing software (Myers et al., 2011). This high cost is because test requires time, knowledge, planning, infrastructure, and skilled personnel (Myers et al., 2011). Also, as the software often has to be delivered as soon as possible, the time available for the testing activity is usually compromised. Besides, the testing of particular kinds of applications (*e.g.,* mobile applications) can be challenging (Dantas et al., 2009; Bezerra et al., 2014).

Currently, hiring testing services from independent organizations, named test factories (Sanz et al., 2009), has become common in the industry. Test factories can be seen as software factories specialized in software testing. By leveraging test factories services, a software development project can benefit from tests with high quality and low cost since a software project does not need to invest in its own test team. Therefore, we decide to implement a test factory to provide testing services not only for the Research and Development and Innovation (R&D&I) projects executed in our research group[1] but also to external companies. With this belief in mind, we have been working in the software testing process definition in a test factory[2] since 2013.

We highlight that test factories should work in a close relation with their customers (other software organizations that develop the software system to be tested) to be efficient. Thus, the definition of roles, responsibilities and competencies should be clearly defined in both parties: the test factory and the customer (software organization). Moreover, the control and tracking of the correction of bugs detected should be made to ensure traceability. Unquestionably, a software process that defines all activities, roles, and artifacts (used and produced in the activities) is a good

*Researcher Scholarship - DT Level 2, sponsored by CNPq

†PhD Scholarship (MDCC/DC/UFC) sponsored by CAPES

[1]Group of Computer Networks, Software Engineering and Systems (GREat) - http://www.great.ufc.br/index.php/en/

[2]http://fabricadetestes.great.ufc.br/

Andrade, R., Santos, I., Lelli, V., Oliveira, K. and Rocha, A.
Software Testing Process in a Test Factory - From Ad hoc Activities to an Organizational Standard.
DOI: 10.5220/0006333301320143
In *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017) - Volume 2*, pages 132-143
ISBN: 978-989-758-248-6

practice to address all these requirements. We argue that the definition and institutionalization of a testing process should be performed on a continuous improvement cycle mainly in the case where two or more organizations should interact to assure the execution of the activities and their internalization in all parties involved.

This paper presents how software testing activities were continually integrated using a process continuous improvement cycle based on the lessons learned collected. So, the main contribution of this work is to report our experience on a definition, use, and improvement of a standard test process within a test factory. We also present 12 lessons learned that can be applicable by other test factories or in-house test teams to improve their software testing process.

The paper is organized as follows. Section 2 reports our experience in software testing process improvement from the *ad hoc* testing to the *standard* test process. Section 3 discusses the results achieved. Section 4 present the related work and, finally, Section 5 presents our conclusion and future work.

2 SOFTWARE TESTING PROCESS DEFINITION, USE AND IMPROVEMENT IN PRACTICE

To define software testing activities, we followed the well-known cycle for process continuous improvement proposed by Johnson (Johnson, 2002): the PDCA (Plan-Do-Check-Act). The PDCA has four steps. It begins with the planning and the definition of improvement objectives. Next, the planning is executed (*Do*) and then evaluated (*Check*) to see whether the initial objectives are achieved. Actions are taken based on what was learned in the check step. If the changes are successful, they are included in the process. If not, it is necessary to go through the cycle again with modifications in the original plans. Figure 1 shows the PDCA cycle used during the testing process improvement described in this paper. We highlight that the several *Do* in that figure represent the application of the planning in more than one testing project. We followed this strategy to acquire more data to support the *Check* phase.

It is worth noting that we have applied our testing process on several software projects (*e.g.,* mobile and web projects) from the same company, but they have different product owners and/or project managers. Thus, this scenario enabled us to collect feedback from several customer's teams while we ran the

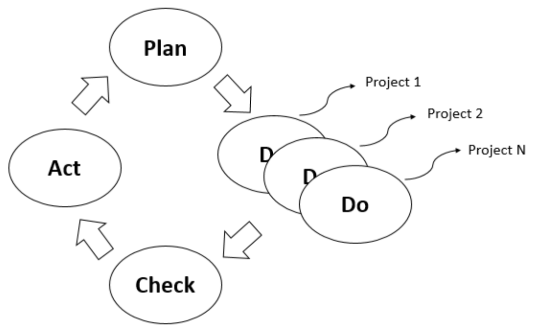

Figure 1: PDCA Cycle used in the testing process improvement.

PDCA cycle.

Next section describes the study object, *i.e.,* the testing process used before the improvement. Then, the following sections present the two cycles of the improvement performed. Each cycle was started by the definition of goal (planning), and finalized with the definition of actions to be carried out in the next cycle. These actions were defined by the analysis of lessons learned collected after the execution of the planning in different projects.

2.1 Before the Test Factory: *Ad hoc* Testing

The creation of our test factory was motivated by the need of one mobile manufacturer company, our main client, who does not have expertise in software testing. Once this client worked with Scrum methodology (Schwaber and Sutherland, 2016), we have started by identifying how to integrate testing activities as a service in its agile process. Scrum works with the concept of sprint, a time boxed effort during which a potentially releasable product increment is created, and with three roles (Schwaber and Sutherland, 2016): (i) Product Owner, responsible for maximizing the value of the product; (ii) Development team, consisting of professionals who do the work of delivering a potentially releasable increment of "Done" product at the end of each sprint; and (iii) Scrum Master, responsible for ensuring Scrum is understood and enacted.

In the Scrum, the testing (now "agile") is integrated with development efforts, being required tests for each product backlog item (*e.g.,* a software functionality) (Myers et al., 2011). Thus, our initial goal was to encourage our customer to incorporate more test procedures in the software development. To do so, a test analyst was allocated to work closely with the software development team performing test activities as required. Following the Scrum, the test analyst performed functional testing at the end of each sprint. Furthermore, the developers created both unit and system tests without planning or documentation. One complete software project with eight developers, named here P1, was developed and tested following this structure. The duration of this project was about ten months (August 2013 until May 2014). With regard to the system tests, most of them failed (47%) and the failures were classified as critical, *i.e.,* bugs that block the software. This high percentage of failed tests and the delay to find critical bugs motivated us to investigate how to improve our test process to identify the bugs as early as possible.

2.2 First Cycle: Establishing Test Procedures

As mentioned in Section 2.1, the analysis of the results obtained from the *ad hoc* testing approach motivated us to adopt some practices and to formalize our testing process.

The first practice is to allocate the test analyst to support the implementation of both unit and integration tests performed by the developers. First, the test analyst pointed out the basic tests scenarios, which the developers should implement by using JUnit tool[3]. In some cases, the test analyst also performed "Pair Testing"[4] with each developer until they can create good tests cases at both unit and integration level.

The second practice is regarding the validation of the backlog items. For each activity from the sprint backlog, it was added a validation with the test analyst to get the status "done". We also included in this backlog specific activities for creating unit tests, emphasizing their importance within the sprint. When the development activity did not generate executable code, a review of the application code was made. On the other hand, if it is possible to execute the code, then automated tests or manual tests were performed to validate the functionality implemented in the activity. For each development activity, 1 or 0.5 points were added in the activity estimation for the validation of the product increment developed. Additionally, it was also added in the sprint Backlog one activity for the system testing of all products developed in the sprint.

The last practice concerns the test specification. The test scenarios were documented in spreadsheets that were frequently updated by the test team and shared among the client. For the sake of simplicity, the test case specification in such spreadsheets contains only three columns: test scenario, step-by-step and results.

Regarding the testing process, it was defined according to the sprint phases of the Scrum. Figure 2 shows the test process defined in the first improvement cycle. In the *Sprint Planning* phase, the test team estimates the testing effort and defines the test strategy according to the product requirements. Next, in the *Sprint Execution*, the test analyst creates the test scenarios, which are used: (i) by the developers to create both unit and integration tests; and (ii) by the testers to create automated tests scripts or perform manual tests. After the execution of the tests, their results are documented. Once a bug is found and fixed, the regression tests should be run again. Finally, in the *Sprint Revision*, the test team discusses the lesson learned to improve the testing process.

We applied (Do phase from PDCA) the test process defined and the practices aforementioned in two development projects, named here P2 and P3. The project P2 was conducted between July 2014 and September 2014 with 12 developers, while the project P3 had 10 developers and occurred from September 2014 to December 2014.

In the project P2, the low failures rate (15% of the total number of system tests) at the end of sprint shown us the greater participation of all developers in the product quality. It is worth mentioning that were not implemented automated tests in P2. In project P3, despite the failed test rate of 28%, none of them were of the critical type, representing an improvement comparing with the results obtained in project P1. We highlight that in this project the test automatization with tools (*e.g.,* Robotium tool [5]) made easier the regression testing.

Regarding the tests documentation, the test team report that it is easier to document, plan and prioritize the tests according to changes in the sprints and the customer requests. Furthermore, they also reported that a bug tracker is crucial to to follow the fixed bugs and thus perform the regression testing efficiently.

The main benefits that we obtained in the first improvement cycle are: (i) greater participation of all developers in the product quality; (ii) greater integration between developers and testers; and (iii) greater integration between the code and its test, once the test has become a part of the development of a feature.

The main lessons learned from the execution of *Do* phase of PDCA in two projects (P2 and P3) were:

- LL01: Test Documentation is essential. Our experience in the project P1 corroborates with our

[3]http://junit.org/junit4/

[4]It is a kind of "Pair Programming activity (Zieris and Prechelt, 2016) applied on the software testing.

[5]http://robotium.com/

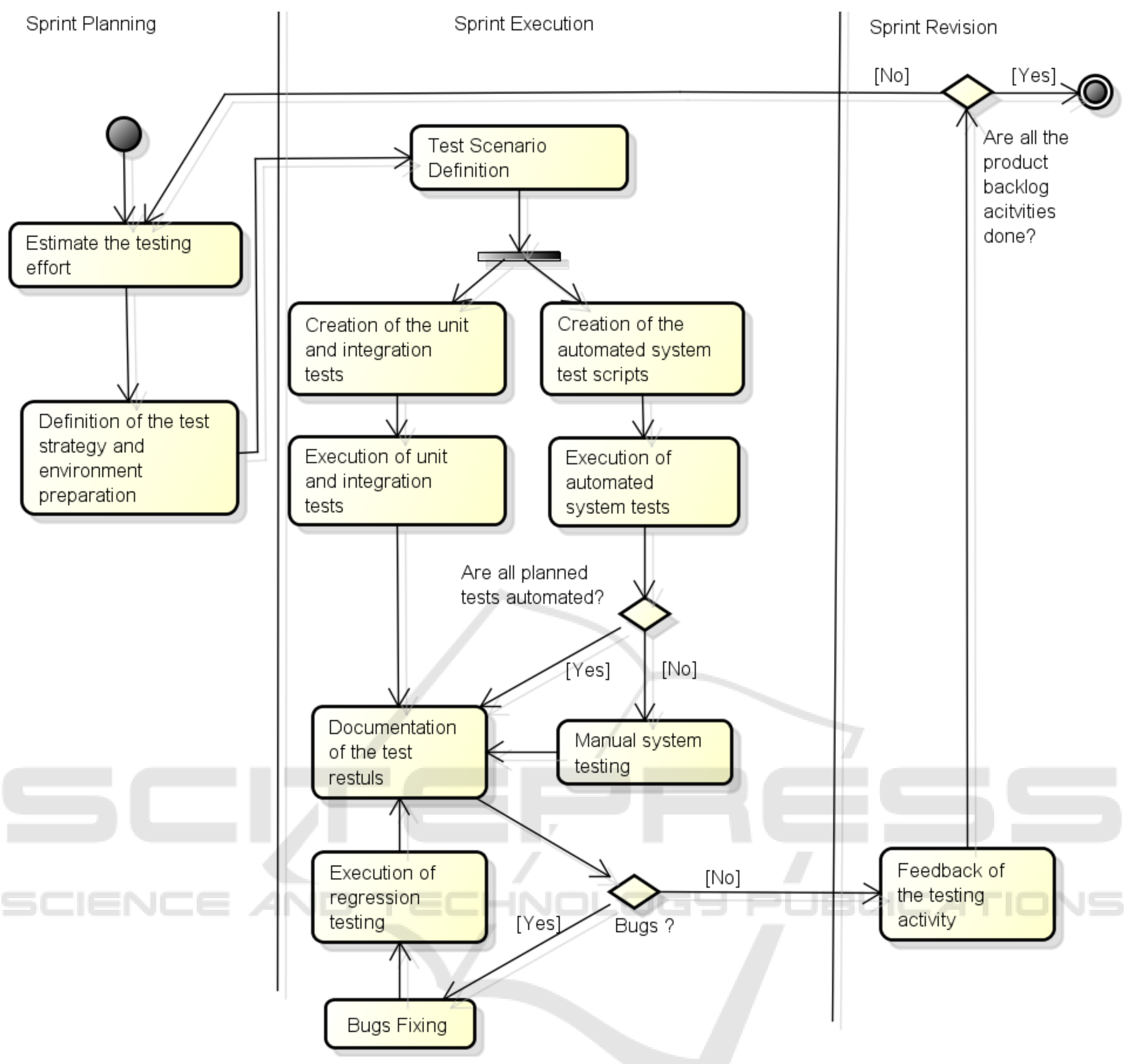
Sprint Planning
Sprint Execution
Sprint Revision
Estimate the testing effort
Definition of the test strategy and environment preparation
Test Scenario Definition
Creation of the unit and integration tests
Creation of the automated system test scripts
Execution of unit and integration tests
Execution of automated system tests
Are all planned tests automated?
[Yes]
[No]
Documentation of the test restuls
Manual system testing
Execution of regression testing
Bugs Fixing
Bugs ?
[Yes]
[No]
Feedback of the testing activity
Are all the product backlog acitvities done?
[No]
[Yes]

Figure 2: Test process defined according to sprint phases.

feeling that the lack of test documentation makes it difficult to monitor, replicate and evaluate them;

- LL02: There is no need for specialized roles, but it is essential both to follow a test process and to have a team of experts. During the testing of P1, where developers and testers do testing without planning, many of the defects found at the end of the sprint were critical. In projects P2 and P3, the participation of an expert test analyst and the use of a testing process enabled critical defects to be identified in the early stages of development;
- LL03: It is not always possible to automate the tests, but it is advisable. Automating tests is interesting because it facilitates continuous feedback. The planned use of a systems test automation tool in P3 supported the fast defect identification;
- LL04: Pair Testing is a good practice to exchange experience between novices and expert testers. The application of Pair Programming to implement tests during P2 had a very positive impact, resulting in qualified developers involved with test activities and better communication among testers and developers;
- LL05: The client should be aware of the testing activities. When the customer cannot participate in a decision at some point, we realized as a good practice to document the decisions taken concerning the testing activity to confirm them with the customer afterward (*e.g.,* regarding test prioritization). We employed this practice in the three projects P1, P2, and P3 and we obtain a good feedback from the clients.

Based on the aforementioned lessons learned, the next step (*i.e.,* the *Action* phase in PDCA) is to define a standard software testing process.

2.3 Second Cycle: Defining a Standard Software Testing Process

Based on the previous experience, we decided to evolve the test procedures for a formal software testing process that establishes activities, responsibilities, roles, and artifacts to be used and produced. To that end, several specific policies were created based on the Reference Model for Brazilian Software Process Improvement (MPS.BR) (SOFTEX, 2006)

For the test project management, we defined the following policies:

- The project planning of the test factory results in the test plan and should be carried out by the test factory manager;
- The test plan should have the approval by the stakeholders and client aiming to establish commitments to the project;
- The test factory manager should monitor the project continuously, based on the test plan and schedule;
- The final phase constitutes project milestones;
- Any changes in the test plan and schedule can only be implemented after obtaining new commitments with the client and all involved in the test factory.

For the requirements management, the policies establish that:

- All the testing requirements should be evaluated by the client and the stakeholders of the test factory;
- Any change in the testing requirements should have impact assessment before being approved;
- The plans, artifacts, and activities of the testing requirements management process should be maintained to assure their consistency;
- The specification of the testing requirements should have the approval and commitment of all stakeholders;
- The change requests in testing requirements must be recorded in e-mail or meetings and should be stored.

Finally, for the quality assurance, all the test factory projects should be audited for adherence to the process and product quality, and the non-compliance that are not resolved in deadlines (up to three days) should be scaled to higher hierarchical levels.

The standard process is organized as a work breakdown structure where phases are decomposed in activities and those in their turn in tasks. Figure 3 gives an overview of the activities of our test factory process. This process is based on the iterative and incremental lifecycle. Three main phases (planning, elaboration and execution) were defined as follows.

1. *Planning* phase aims at starting a project and performing the planning activities. The first activity "Initiate project" involves kick-off/initial meetings to understand and review (if applicable) the product requirements to be tested. In the "Plan Project", the test factory manager elaborates the test plan with test requirements as agreed with the client. Also, the SQA (Software Quality Assurance) analyst must manage the adjustments, for instance, if nonconformities related to the testing process exist, such as a non-written confirmation that both parties agreed with the test plan, they should be solved in an established period.
   - The main artifacts produced in this phase are minutes of the kickoff meeting, test plan with schedule, process and product checklists for phase 1, and project status report.
2. *Elaboration* phase aims at preparing the test execution. This phase starts with an initial meeting involving the test team to align the test plan. Then, test scenarios and the test cases are specified by the test analysts in the "Design Tests" activity. This specification is based on product and test requirements provided by the client. To be approved, the test specification must be examined by another test analyst as a peer review task. Other tasks carried out in this phase are the establishment of traceability matrix, the environment configuration, and the creation of automated test scripts (if applicable).
   - The main artifacts produced in this phase are traceability matrix, test specification with test scenarios/cases, test environment with load test mass (if applicable), checklists for phase 2 (change control, audit) and test artifacts evaluation reports, such as test cases evaluation report.
3. *Execution* phase aims at carrying out the execution (manual or automated) of the tests artifacts specified previously (*e.g.,* test scenarios/cases). Similarly, to the elaboration phase, the test team attends an initial meeting. In the next activity, the tests are executed, and their results are reported to be evaluated. It is worth noting that, the tests can

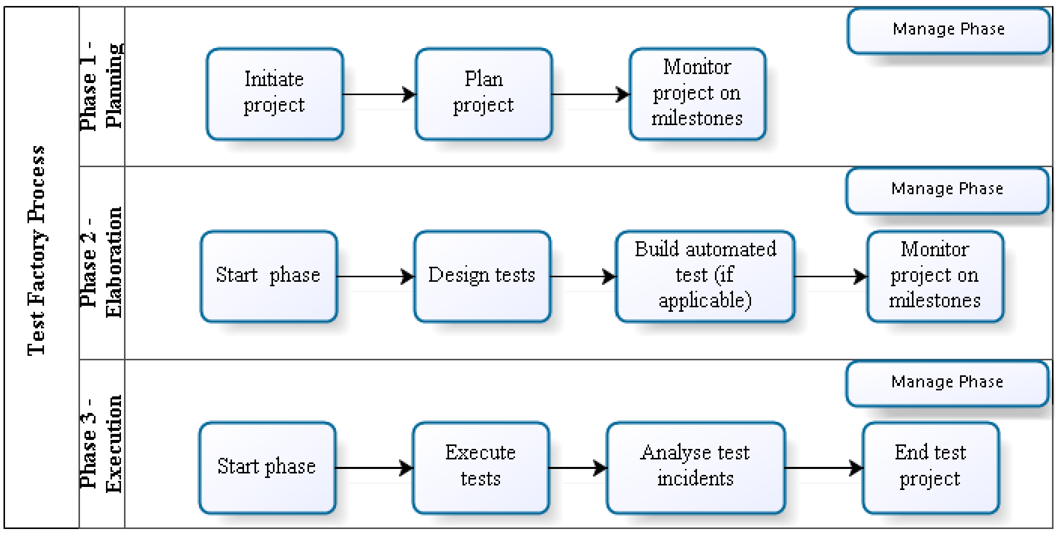

Figure 3: Overview of our test factory process.

be executed by a tester (*e.g.,* a testing intern) but the test leader must determine in the “Analyze test incidents” activity whether the reported incidents are actual bugs. Once the incident is confirmed as a bug, it is registered in a bug/issue tracking tool. The *Execution* phase ends with a meeting with those involved in the project to record lessons learned.

- The main artifacts produced in this phase are test incident reports, bugs/failures report, checklists for phase 3 (audit, adherence to process), lessons learned, and project closing statement.

Our test factory process is adherent to the MR-MPS-SW (Rocha et al., 2005) and includes the quality assurance process and activities related to configuration management and verification. As depicted in Figure 3, two management activities are presented in all phases. The first one is “Manage Phase” that is executed throughout the phase. The second one, “Monitor project on milestones”, is the activity for monitoring the project in the final milestone of the phase. For example, this activity registers the project status, stores the phase artifacts (*e.g.,* test plan, project status report) and audits the adherence of the process at a phase following a specific checklist.

The organizational chart of our test factory is shown in Figure 4. The factory test team is composed of a high quality team with academic and professional experience on software such as PhDs with expertise in Software Testing, certified professional testers (International Software Testing Qualifications Board - ISTQB certification), usability experts, software quality experts, postgraduate and graduate students of Computer Science/Software Engineering courses. The roles and the main activities within the factory test process are described in Table 1. Also, the artifacts that should be produced by each role are presented in this table.

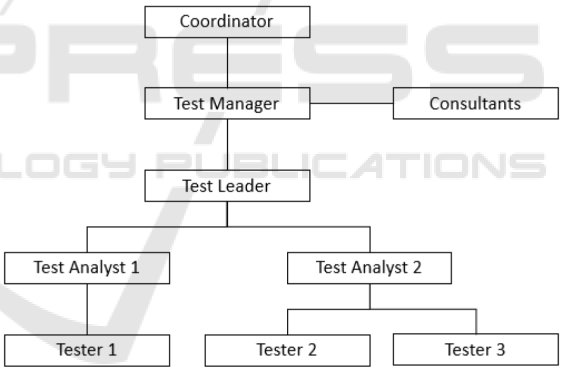

Figure 4: Organogram of our test factory.

We have applied the second improvement cycle on two projects (P4 and P5) that performed several sprints. Both were developed under the Scrum methodology and, therefore, we have considered a release with one or more sprints as a whole testing project. Indeed, the development process adopted by the client is transparent and thus independent of our test factory process. The main difference is the kinds of artifacts that we have to produce. In a Scrum project, we start by attending the planning meetings to understand the functional requirements and to have an overview of the test scenarios must be covered. Once the test scope is defined, the project manager elaborates the test plan with the test requirements, schedule, human resources, and risks.

Table 1: Roles, main activities and artifacts.

Role	Main Activities	Artifacts
Coordinator	Coordinate the test factory project (decision making, fundraising, conflict resolution); develop and/or review artifacts; organize, participate and conduct meetings	Test project work plans, budgets
Test Manager	Apply the test factory process and define measures appropriate to collect the test project status; elaborate process artifacts; plan and control the team activities	Test plans, meeting minutes, project status reports
Consultants	Conduct the research activities in test project; organize and conduct training of advanced testing techniques; audit the adherence to the process and the product quality; provide new testing techniques (quality measures or fault models) to be applied to advanced application domains (ubiquitous/IoT domain)	Usability test report, checklists, technical/researcher reports
Test Leader	Ensure compliance with planning execution; disseminate the technical information to the test team; evaluate test artifacts/deliverables	Test artifacts evaluation reports
Test Analyst	Elaborate test artifacts; support the test leader to verify technical problems raised by the client; report the impediments to the test leader; validate the incidents reported by testers	Test specification, test scripts, failures report
Tester	Execute the tests; report the test results (incidents) and submit to the test analyst to be validated	Incidents report

We have faced several challenges in the implementation of the second cycle. Firstly, we have worked on several maintenance projects, whose the maintenance requirements are organized as a backlog of users stories. In these projects, most of the functional requirements are not specified or there is no previous test specification. To minimize the risks, we have to define the test scenarios according to some test criteria provided by the client and thus ensure minimal test coverage.

Secondly, the phases defined in the second cycle require several test artifacts. Then, we have implemented this cycle incrementally. We have started by defining the test templates that must be followed in the elaboration/execution phases. Then, we have provided training sessions to explain the process and the artifacts that must be produced. This task was crucial since the test team of the previous cycles has a limited experience in test case specification.

As previously mentioned, we applied (Do phase) the test process defined and the practices aforementioned in the testing of two development projects, named here P4 and P5. We conducted their test projects in four months (March 2016 to June 2016). The project P4 involved the development of three software systems (A, B and C) and was organized in ten sprints, while the project P5 concerns three other systems (D, E, and F) developed in the total of six sprints.

The application domains of projects P4 and P5 are mostly mobile and web, respectively. Thus, the number of tests depends on systems requirements and also their domains. For example, we have tested for Software A and Software C, both the client (mobile) and the server (web). However, they are simpler than Software D, which is a game mobile that requires the testers' skills to play it, and also the tests must cover the game rules. In some sprints, the tests only focus on bug fixed and their impact.

We highlight that these results do not cover the unit and integration tests performed by the developers team and the acceptance testing executed by the client test team (if applicable). We have experienced in the second cycle a better control of our test activities since we have more tests documented and, therefore, we could leverage them for the next sprints, for instance, to perform regression tests.

With the execution (Do phase of PDCA) in 16 software releases in projects P4 and P5, the main lesson learned were:

- LL06: Process templates adjustment could be needed to some specific client. Although we standardized the test documents used in our testing process, in some cases new fields (*e.g.,* indicating if the test is for basic, alternative or exceptional flow of use cases) are added to meet the needs of specific clients;
- LL07: The application of the test process should be supported by the test team, which needs to be trained firstly. We have adopted an implementation of our test process in a bottom-up way starting by the test documents used daily (*e.g.,* test scenario/case specification) to minimize the impacts. Also, we have provided intensive training to balance the test team that had never done documentation and regular evaluations of both test documentation and test scripts quality;

- LL08: The hierarchical structure and roles/activities were essentials to centralize and manage the demands for the test factory. Besides, the inclusion of roles helps the team focuses on its objectives (*e.g.,* test manager monitors all the activities, testers run tests specified by test analysts);
- LL09: At the first moment, the test process increased the time spent and costs in the testing activity. For instance, the test analyst took a longer time to produce the test artifacts (*e.g.,* test scenarios and their test cases) when we implemented the Elaboration/Execution phases. We have observed that the evaluation report took a meaningful time (between 1 hour and 3 hours). Moreover, this activity works as a round trip process: check-correct-check until the test specification is satisfactory. One improvement here is to provide checklists to support this activity;
- LL10: In spite of the testing process that we follow rigorously, we observed that the experience-based testing (IEEE, 2015) is interesting for a first interaction with the application by using, for instance, the exploratory testing. In our process, we have started with this kind of testing when the requirement specifications or user stories (in Scrum projects) are not provided by the client. In such a case, the client must provide at least the test acceptance criteria.
- LL11: Sharing the same database between tester team and development team could lead to rework, especially in the test documentation because the test data defined before could not be more available (*e.g.,* if a developer deletes a database while implementing some new feature). Such problem also impacts on several test scripts that have to be updated. Furthermore, the previous versions of the application could be used as test oracle in the regression testing and also to verify intermittent bugs; and
- LL12: An emergency release called "Hot fix" requires a fast test feedback. We call "Hot fix" the time box for fixing and testing critical bugs that often have to be deployed to a client in less than 24h. In such releases, the test team usually does not have time to run all the process as defined, so the team just creates the main test artifacts, for instance, the test scenarios/cases specification.

Currently, we have worked on the specific checklists defined by the activities "Manage the phase", and "Monitor the project in the final milestone of the phase".

2.4 Results

To evaluate the effectiveness of our test factory process, we used a well-known metric called Defect Removal Efficiency (DRE) (Jones, 1996), which is given as follows:

$$\text{DRE } (\%) = \frac{TestFailures}{TestFailures + ProductionFailures} \times 100$$

TestFailures are the number of total failures found by the test team before the software delivery. *ProductionFailures* are the number of failures found by clients and/or users after the software delivery.

Figure 5 shows the DRE results collected for three software projects: software A, B, and C. We have measured the DRE in the first cycle (without the standard test process) and the second cycle (with the standard test process), with a total of 28 software releases: 13 without and 15 with the process. We did not distinguish the different software phases (*e.g.,* requirements, design) to collect the DRE.

Table 2 gives an overview of the software size, total of failures, DREs, and standard deviations results per software. The average DREs for the software A is 30,50% (first cycle) and 30,72% (second cycle). The average DREs for the software B is 15,58% (first cycle) and 35,08% (second cycle). The average DREs for the software C is 11,67% (first cycle) and 37,78% (second cycle).

We can observe that the highest value of DRE was 37,78% (Software C) against the great value 95% suggested by Jones (Jones, 1996). However, the DRE values had increased for the three software when our test factory process was implemented in the second cycle. Also, the *standard deviation* value is low for the Software A (8.62%) in the second cycle. By contrast, the deviation values have increased for the Software B - from 18.31% (first cycle) to 38.14% (second cycle) and Software C - from 14.53% (first cycle) to 43.32% (second cycle). We believe that this result was affected by the small sample size (*e.g.,* the number of failures and releases) and also by the team experience in applying the process as we discussed in the next section.

Although the standard deviation was a little high for Software B and C, we conclude that our testing process had improved their quality since their DREs had increased more than 100% in the second cycle.

3 DISCUSSION

The implantation of the test factory process has several lessons learned as we presented in the previous

Table 2: Total of Failures, DREs, and Standard Deviations per Software.

Software ID	Size (KLoC)	Total of Failures (#)	DRE (%) without proc.	Std. Deviation(%) without proc.	DRE (%) with proc.	Std. Deviation(%) with proc.
A	384	152	30.50	21.59	30.72	8.62
B	101	94	15.58	18.31	35.08	38.14
C	117	58	11.67	14.53	37.78	43.32

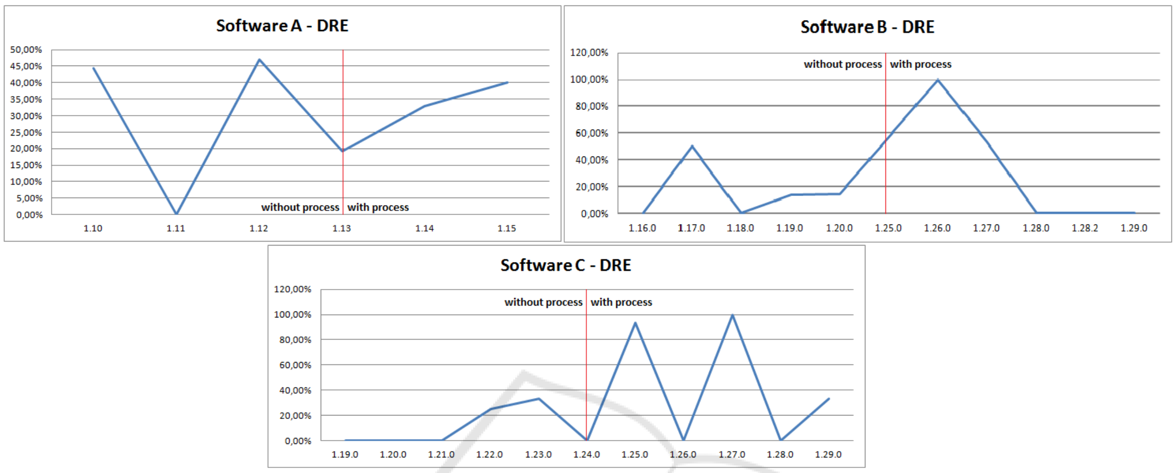

Figure 5: DRE results in the first cycle (without process) and the second cycle (with process).

sections. Such lessons help us to evolve from *ad hoc* testing activities to a standard testing process. We discuss some lessons below.

As we identified early (**LL01**), the test documentation is crucial. When we introduced the standard testing process and their artifacts, we have identified several problems related to the test specification such as uncovered test scenarios, duplicated test cases, etc. Furthermore, we have observed that the elaboration of the test artifacts has impacted on the team productivity (see **LL09**), mainly when the new test artifacts are introduced. To overcome this problem, we analysed the artifacts elaborated by the test team and also monitored the time spent to produce them. So, we provided some adjusts in the documentation (see **LL06**) to meet the needs of both the clients and the test team. Also, we provided several training sessions (see **LL07**) to ensure that some misunderstanding does not impact on the team productivity.

Currently, we have used a simple spreadsheet to specify the test scenarios/cases. Such specification contains: (i) test scenario (what is being tested); (ii) test steps; (iii) expected output; and (iv) actual result. However, we have observed that a spreadsheet is not a good choice when we have several test cases (*e.g.,* more than 50 longer test cases). Also, we cannot use version control efficiently with spreadsheets. As a future improvement, we intend to adopt a tool that allows the test analysts specify the tests directly on it and stores them on a server (*e.g.,* TestRail[6]). With regard to the failures, they are reported in both JIRA[7] software, which is shared with the development team, and spreadsheets. Note that, only the bug repository is shared with both teams. Our experience shows that we have to clearly separate the development and test environments (see **LL11**). In such a case, the configuration manager is responsible to handle the changes in both environments. Moreover, the repository's permissions are strictly managed, *i.e.,* none of the team have permission to delete bugs. For example, once a bug is registered, it is analysed by the test analyst, if the bug is invalid, only its state changes in that repository.

Another important point to be discussed is the team qualification (**LL02**). In the first cycle, the test team works together with the development team as a "Pair Testing" approach (**LL04**). This kind of work helped developers to create good unit and integration tests. In the second cycle, we applied the "Pair Testing" between the experienced test analysts and the novices, and the results were also positive.

Furthermore, the hierarchical structure of the team (**LL08**) by including the consultants is important to improve the test factory process. For example, the researchers are responsible for investigating new testing techniques (*e.g.,* test automation - **LL03**) or mea-

[6]http://www.gurock.com/testrail/
[7]https://www.atlassian.com/software/jira

sures while the test analysts are responsible for applying them. The other important role is the SQA analyst who must ensure that the process is followed by the test team. In some cases, the nonconformities pointed out by the SQA analyst in early stages of the testing process (*e.g.,* a non-written confirmation that the client agreed with the test plan - see **LL05**) help us to identify several problems that may compromise the test quality.

We also would like to highlight that we could not apply the standard testing process in *Emergency Releases* (**LL12**). The most difficulty is to elaborate all test artifacts and execute, for example, the functional and regression tests within one working day. In such a case, we only execute the tests already specified and perform the exploratory testing. Indeed, this kind of test plays a vital role in our testing process when the requirement specification is not provided (**LL10**). We rely on such test to explore the application and thus elaborate the test scenarios and their test cases.

The lesson learned presented above are not quantitatively measure. However, we leverage DRE measure to show the benefits of the standard testing process. We observed that the DRE results have increased in the second cycle, but they do not achieve the great value (*e.g.,* 95%). We identified several reasons as follows:

- The test team does not have any experience in applying testing process. Also, most of the team are novices in the projects;
- The software (*i.e.,* A, B, and C) that we applied the standard testing process are maintenance projects, and most of them had no documentation. In this case, we elaborated the test artifacts incrementally according to the release plan. We performed this cycle until the test documentation was complete; and
- The whole test process have been not implemented. We have adopted this process on March 2016, and incrementally the test artifacts are introduced in the projects. So, we believe that when all test artifacts can be produced, the DRE results will be better. However, we have obtained good results until the present moment.

4 RELATED WORK

We have found several test factories (CWI, 2017)(Cyber:con, 2017) but none of them describe their testing process in a literature. Thus, we also investigate papers related to testing processes. In this section, we discuss the related work into two categories: *Test Factory* and *Software Testing Process*.

4.1 Test Factory

We have found several test factories that offer test services for software development companies. For instance, there are the CWI's test factory (CWI, 2017), Cyber:con's test factory (Cyber:con, 2017) and FH's test factory (FH, 2017). The main characteristic of these enterprises is to have a dedicated test team able to provide specialized testing services (*e.g.,* test case specification, test case execution). Those test factories, however, do not present their testing process in scientific or white papers since they want to protect their business from the competitors. Therefore, we cannot do any relation between our testing process and their test factory processes.

By contrast, we have found two papers (Sanz et al., 2009; Cooper-Brown, 2015) that focus on only the implantation of test factories. For example, Sanz et al. (Sanz et al., 2009) define a process model to create a test factory, determining the roles, capacities, and responsibilities for each specified process. In their paper, Sanz et al. present succinctly 11 processes (*e.g.,* Testing Planning) that are classified into three categories: Management, Technical and Support. Thus, this model provides the key elements of the process of a test factory. In our work, we focus on describing the experience for defining a standard testing process in an test factory from *ad hoc* testing activities performed in-house.

Cooper-Brown et al. (Cooper-Brown, 2015) present a process to setup a test factory. This process is organized into three major phases: (i) Solution Definition, in which the existing organizational test processes are evaluated; (ii) Solution Design which involves designing processes (*e.g.,* related to test strategy, organizational structure, etc.); and (iii) Solution Implementation, which could be executed by steps or by using a big bang approach. In our work, we focus on the definition, use and improvement of a testing process to implement a test factory based on the lessons learned.

Additionally, Xia et al. (Xia et al., 2015) investigate the challenges in test outsourcing. To do so, the authors perform a empirical study through a survey with testers and interviews with SQA managers and team leaders. In our paper, we report the experience by applying the test factory process in real-world daily use involving several stakeholders (*e.g.,* tester, client, research, SQA analyst). Furthermore, the benefits of our test factory are evidenced by the positive results of DRE collected from industry software projects.

4.2 Software Testing Process

Regarding the software testing practices, we have identified several research studies in the literature, we discuss them below.

Engstrom and Runeson (Engström and Runeson, 2010), for instance, present the results of a qualitative survey conducted by using focus group meeting with 15 industry participants and an online questionnaire with 32 respondents. Based on this survey, the authors identified weaknesses and strengths in regression testing practices and also their good practices such as "run automated daily tests on module level". Collins e Lucena Jr. (Collins and de Lucena, 2012) describe the experience with the use of open source testing tools in automation testing into two development projects using Scrum. Santos et al. (Santos et al., 2011) also describe the use of testing practices in agile environments. In contrast to our work, they focus on specific issues within the software testing process (*e.g.,* the use of testing tools) whereas we present an overview of our testing process and how we evolve this process over two years.

Ramler and Felderer (Ramler and Felderer, 2015) propose a process for risk-based test strategy development that consists of seven steps: (1) definition of risk items, (2) probability estimation, (3) impact estimation, (4) computation of risk values, (5) determination of risk levels, (6) definition of test strategy, and (7) refinement of test strategy. Our testing process is not a risk-based test strategy. Instead, our process is adherent to the MR-MPS-SW(SOFTEX, 2006) and thus the risks are described in the test plan and managed during all the process.

Afzal et al. (Afzal et al., 2016) present the results of a systematic literature review (SLR) which identified 18 software test process improvement approaches. Furthermore, they perform an evaluation with two of that approaches using an industrial case study. In contrast to this work, our goal is to apply PDCA (Plan-Do-Check-Act)(Johnson, 2002), which is an process improvement approach, to establish a standard testing process into a test factory.

5 CONCLUSIONS

Several development enterprises have hired services of test factories to reduce the costs related to the software testing activities. These test factories are then an opportunity to reduce the costs and improve the quality of the tests. We presented in this paper our experience over two years by leveraging the PDCA cycle to define a standard testing process into a test factory. First, we identified the current *ad hoc* testing activities in Scrum projects and pointed out the main weakness of our testing approach. Next, the improvements were proposed based on the lessons learned to define a standard testing process. This process is adherent to the Brazilian Software Process Reference Model (MR-MPS-SW) and it is independent of the software development process. Last, we applied our standard testing process on 15 industry software releases and obtained good results for DRE.

We have also presented 12 lessons learned that can help practitioners to improve their test process. As future work, we aim to continuous improvement process to perform an official CMMI appraisal for the test factory organization unit. Also, we aim at expanding the services of our test factory to advanced domains (e.g., Ubiquitous and Internet of Things) focusing on the human-computer interaction quality of such domains (Carvalho et al., 2016)(Andrade et al., 2017)(Rocha et al., 2017) and their advanced graphical user interfaces (Lelli et al., 2015b)(Lelli et al., 2015a).

ACKNOWLEDGEMENTS

We would like to thank the GREat's test factory team for the technical support on this work. The authors thank the research groups involved in the CAcTUS - Context-Awareness Testing for Ubiquitous Systems project supported by CNPq (MCT/CNPq 14/2013 - Universal) under grant number 484380/2013-3.

REFERENCES

Afzal, W., Alone, S., Glocksien, K., and Torkar, R. (2016). Software test process improvement approaches: A systematic literature review and an industrial case study. *Journal of Systems and Software*, 111:1 – 33.

Andrade, R. M. C., Carvalho, R. M., Oliveira, K. M., Maia, M. E. F., and Arajo, I. L. (2017). What changes from ubiquitous computing to internet of things in interaction evaluation? In *5th International Conference on Distributed, Ambient and Pervasive Interactions, DAPI 2017. Held as Part of the 19th International Conference on Human-Computer Interaction 2017.*

Bezerra, C., Andrade, R. M. C., Santos, R. M., Abed, M., de Oliveira, K. M., Monteiro, J. M., Santos, I., and Ezzedine, H. (2014). Challenges for usability testing in ubiquitous systems. In *Proceedings of the 26th Conference on L'Interaction Homme-Machine*, IHM '14, pages 183–188, New York, NY, USA. ACM.

Carvalho, R. M., Andrade, R. M. C., Oliveira, K. M., Santos, I. S., and Bezerra, C. I. M. (2016). Quality characteristics and measures for human–computer interaction evaluation in ubiquitous systems. *Software Quality Journal*, pages 1–53.

Collins, E. F. and de Lucena, Jr., V. F. (2012). Software test automation practices in agile development environment: An industry experience report. In *Proceedings of the 7th International Workshop on Automation of Software Test*, AST '12, pages 57–63, Piscataway, NJ, USA. IEEE Press.

Cooper-Brown, B.; Ludhani, C. C. S. (2015). Test factory setup for sap applications. https://www.infosys.com/IT-services/independent-validation-testing-services/white-papers/Documents/test-factory-setup.pdf.

CWI (2017). CWI's test factory. Available: http://www.cwi.com.br/Services/TestFactory. [Online, Accessed: 19-Feb-2017].

Cyber:con (2017). Cyber:con's test factory. Available: http://www.cybercon.de/en_GB/testing. [Online, Accessed: 19-Feb-2017].

Dantas, V. L. L., Marinho, F. G., da Costa, A. L., and Andrade, R. M. C. (2009). Testing requirements for mobile applications. In *2009 24th International Symposium on Computer and Information Sciences*, pages 555–560.

Engström, E. and Runeson, P. (2010). A qualitative survey of regression testing practices. In *Proceedings of the 11th International Conference on Product-Focused Software Process Improvement*, PROFES'10, pages 3–16, Berlin, Heidelberg. Springer-Verlag.

FH (2017). Fh's test factory. Available: http://www.fh.com.br/en/servicos/technology/software-development/test-factory/. [Online, Accessed: 19-Feb-2017].

IEEE (2015). IEEE Draft International Standard for Software and Systems Engineering–Software Testing–Part 4: Test Techniques. *ISO/IEC/IEEE P29119-4-FDIS April 2015*, pages 1–147.

Johnson, C. N. (2002). The benefits of pdca. *Quality Progress*, 35:120–121.

Jones, C. (1996). Software defect-removal efficiency. *Computer*, 29(4):94–95.

Lelli, V., Blouin, A., and Baudry, B. (2015a). Classifying and qualifying gui defects. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–10.

Lelli, V., Blouin, A., Baudry, B., and Coulon, F. (2015b). On model-based testing advanced guis. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–10.

Myers, G. J., Sandler, C., and Badgett, T. (2011). *The Art of Software Testing*. Wiley Publishing, 3rd edition.

Ramler, R. and Felderer, M. (2015). A process for risk-based test strategy development and its industrial evaluation. In *Proceedings of the 16th International Conference on Product-Focused Software Process Improvement - Volume 9459*, PROFES 2015, pages 355–371, New York, NY, USA. Springer-Verlag New York, Inc.

Rocha, A. R., Montoni, M., Santos, G., Mafra, S., Figueiredo, S., Albuquerque, A., and Mian, P. (2005). Reference model for software process improvement: A brazilian experience. In *Proceedings of the 12th European Conference on Software Process Improvement*, EuroSPI'05, pages 130–141, Berlin, Heidelberg. Springer-Verlag.

Rocha, L. C., Andrade, R. M. C., Sampaio, A. L., and Lelli, V. (2017). Heuristics to evaluate the usability of ubiquitous systems. In *5th International Conference on Distributed, Ambient and Pervasive Interactions, DAPI 2017. Held as Part of the 19th International Conference on Human-Computer Interaction 2017*.

Santos, A. M., Karlsson, B. F., Cavalcante, A. M., Correia, I. B., and Silva, E. (2011). Testing in an agile product development environment: An industry experience report. In *Latin American Test Workshop*, pages 1–6.

Sanz, A., García, J., Saldaña, J., and Amescua, A. (2009). A proposal of a process model to create a test factory. In *Proceedings of the Seventh ICSE Conference on Software Quality*, WOSQ'09, pages 65–70, Washington, DC, USA. IEEE Computer Society.

Schwaber, K. and Sutherland, J. (2016). MPS.BR - Melhoria de Processo do Software Brasileiro, Guia Geral (v. 1.1). http://www.Scrumguides.org/docs/Scrumguide/v1/Scrum-guide-us.pdf.

Shamsoddin-motlagh, E. (2012). Article: A review of automatic test cases generation. *International Journal of Computer Applications*, 57(13):25–29. Full text available.

SOFTEX (2006). The scrum guide - the definitive guide to scrum: The rules of the game. http://www.softex.br/mpsbr/.

Xia, X., Lo, D., Kochhar, P. S., Xing, Z., Wang, X., and Li, S. (2015). Experience report: An industrial experience report on test outsourcing practices. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 370–380.

Zieris, F. and Prechelt, L. (2016). Observations on knowledge transfer of professional software developers during pair programming. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 242–250, New York, NY, USA. ACM.