SPLMT-TE: A Software Product Lines System Test Case Tool

Crescencio Rodrigues Lima Neto[1,3], Eduardo S. Almeida[2,3], Silvio R. L. Meira[1,3]

[1]Center for Informatics – Federal University of Pernambuco (CIn/UFPE)

[2]Computer Science Department – Federal University of Bahia (DCC/UFBA)

[2]Reuse in Software Engineering (RiSE)

{crln,srlm}@cin.ufpe.br, esa@dcc.ufba.br

Abstract. *The product lines approach requires specific testing tools that should help to manage reusable testing assets and automate the test execution. Despite of the increasing interest by the research community regarding software testing tools, Software Products Lines (SPL) still need tools to support the testing process. This work presents briefly the results of a mapping study on software testing tool and defines the requirements, design and implementation of a soft- ware product lines system test case tool, aiming at the creation and management of test assets. A controlled experiment was also conducted to evaluate the tool effectiveness.*

1. Introduction

Software testing tools are available for testing in every stage of the software development life cycle [Fewster and Graham 1999]. Some organizations have used testing tools to manage, store and handle the tests. According to [Nakagawa et al. 2007], the tools availability make testing a more systematic activity and minimizes the costs, the time consumed, as well as the errors caused by human intervention.

Integrating test environments and the test asset repository is a cumbersome work, that is even worse due to the lack of automated support tool, which leads to be a manually performed task. There is a need for specific testing tools that should help to manage the reusable testing assets and automate the execution of tests and the analysis of their results as well [Tevanlinna et al. 2004].

In the Software Product Lines (SPL) context, the amount of available tools decrease drastically, and the need of tools to reduce the effort during the SPL Testing Process is a gap that need to be filled. We need testing tools to manage the variability and avoid the explosion of test cases [Neto et al. 2012].

The Reuse in Software Engineering Labs (RiSE[1]) dedicates its earlier research to study SPL, based in the experience gained during industrial SPL project development, the Software Product Lines Management Tool (SPLMT) was implemented [Cavalcanti et al. 2011] to coordinate SPL activities, by managing different SPL phases and their responsible, and to maintain the traceability and variability among different artifacts.

[1] http://labs.rise.com.br

For this reason, we preferred to extend the SPLMT instead of start the creation of a new tool. Based on the SPL Testing gaps identified by the RiSE Labs [Neto et al. 2011b] we developed the test module of the SPLMT named SPLMT-TE.

In this context, this paper presents a tool developed in order to optimize the system test case creation and decrease the time necessary to manage test assets from an SPL testing process. The remainder of this paper is organized as follows: Section 2 discusses the related work. Section 3 presents the mapping study. Section 4 details the SPLMT-TE. Section 5 describes the experimental study; and finally, Section 6 concludes this work and addresses some future work.

2. Related Work

There are few studies describing and detailing tools for testing SPL projects. If we decide to narrow the scope, encompassing the search for only SPL system testing level tools, the findings are worse. As a result from the mapping study [Neto et al. 2012], we identified three Software Product Lines testing tools.

According to [Reuys et al. 2005], the ScenTED-DTCD (Domain Test Case Scenario Derivation) is a prototype tool focused on generating test cases scenarios that describe the test engineer's activities and the system response if modeled within the activity diagram.

Another prototype tool can be found in [Nebut et al. 2007]. It generates system test scenarios from use cases. The approach is based on the automation of the generation of application system tests, for any chosen product from the system requirements of a product line.

Finally, [Oster et al. 2011] presented a tool chain based on commercial tools since it was developed within industrial cooperation. It contained a pairwise configuration selection component on the basis of a feature model.

3. Mapping Study

A mapping study was undertaken to analyze important aspects that should be considered when adopting testing tools [Neto et al. 2012]. A set of four research questions were defined in which 33 studies, dated from 1999 to 2011, were evaluated. From the total of 33 studies considered, 24 of them described single system testing tools and the other 9 described SPL testing tools.

The objective of the mapping study is to investigate how do the available tools support the Software Product Lines Testing process? The study aims to map out existing testing tools, to synthesize evidence to suggest important implications for practice, as well as to identify research trends, open issues, and areas for improvement. We derived four research questions:

- **RQ1 -** *In which context the proposed tools were proposed and evaluated?*
- **RQ2 -** *Is it possible to use the single system testing tools to test software product lines?*
- **RQ3 -** *Which testing levels are supported by existing tools?*
- **RQ4 -** *How are testing tools evolving?*

Based on the analyzed tools, it was possible to identify that the tools are usually developed to support a specific testing level, under the justification that there are no tools supporting all functionalities of a testing process. None of the selected tools supports the overall SPL life cycle.

From the selected tools, most of them were developed in the academic environment (14 for single systems and 7 for SPL), while 7 were developed exclusively in industry (6 for single system and 1 for SPL). The remaining 5 tools were developed in both environments, academic and industrial (4 for single systems and 1 for SPL)

However, there is insufficient information about publications describing tools used in the industry. Hence, we performed a search on sites to find out some of the most common tools. More details about it can be seen in: http://wp.me/p157qN-q

The testing tools support is also important to increase the quality and at the same time reduce the effort to perform them.Although, even with several existing testing tools, they are mostly unable for directly support an SPL Testing process [Tevanlinna et al. 2004], since there is no tool suitable to all testing levels of a SPL. Thus, researchers need to consider the feasibility of adapting existing tools or constructing new tools.

4. A Tool for System Testing Case Creation

This section presents the SPLMT-TE, developed with the aim to assist the elaboration of system test case based on the use cases from an SPL. With the tool, the users can manage test assets from an SPL (such as: test cases, test plans, and test suites). The information extracted by SPLMT-TE is derived from the text content of the use cases through regular expressions. Furthermore, the tool also focus on usability, thus the users can have better experience during the creation of test cases.

4.1. Requirements

Based on the mapping study performed [Neto et al. 2012], each one of the tools from system testing was analyzed in order to map its functionalities that were considered as requirements for the SPLMT-TE development.

[Nakagawa et al. 2007] identified functional requirements through investigation of software testing processes, testing tools and testing ontology. It is noticed that these requirements refer to the core activities of testing. The functional requirements that we implemented are presented next.

4.1.1. Functional Requirements

The identified functional requirements and their rationale are described as follows:

- **Include Test Cases Manually -** The tool provides manually test cases creation.
- **Store Test Cases -** Stores the test cases into a repository.
- **Generate Test Cases Automatically -** It creates test cases automatically.
- **Enable Test Cases -** Activates test cases inside of the test suite.
- **Disable Test Cases -** Deactivate test cases from the test suite.
- **Export Test Cases -** It allows to export test cases into PDF files.
- **Test Cases Report -** Generates reports about the selected test cases.

- **Remove Test Cases -** It allows the exclusion of duplicated and obsolete test cases.
- **Test Cases Visualization -** It provides the visualization of test cases.
- **Test Suite Management -** It groups test cases.
- **Test Plan Management -** It allows the management of test plans.
- **Variability Management -** It manages the variation points and the variants of the software product lines assets.

4.1.2. Non-Functional Requirements

The non-functional requirements are following described:

- **Reusability -** the proposed tool must allow the reusability of test assets such as test plans, test suites, and test cases. They can be reused to create new test assets.
- **Extensibility -** The architecture must presume the addition of new functionalities and the level of effort required to implement them without impacting to existing system functions. For this reason, the proposed tool must be well-structured, with low coupling modules to accommodate maintenance and extension demanded.
- **Usability -** The graphic interface of the tool must be built-in with intuitive components to perform the functionalities.
- **Performance -** Performance is measured in terms of the response time. In distributed scenarios, variables such as network traffic, geographical distance, and number of components must be considered.
- **Platform Independence -** An integrated reuse environment must be used in order to maximize its user base and provide more effective results. The implementation of the environment functionalities must be based on technologies portable across platforms.

4.2. SPLMT-TE Architecture

The SPLMT architecture was designed to be extensible providing the capacity to add new features by including new components, which allows the creation of the SPLMT-TE. Figure 1 shows the architecture of the proposed tool combined with the SPL Metamodel [Cavalcanti et al. 2011]. Details of the layers are described next:

- **Browser:** Users access the application through web browsers.
- **Template:** A Django template separates the application from the data. A template defines the presentation logic that regulates how the information should be displayed.
- **URL Dispatcher:** It defines which view is called for a given URL pattern. Basically, it is a mapping between URL patterns and the view functions that should be called for those URL patterns.
- **View:** It contains the business logic. Python view methods call a restricted template language, which can itself be extended with custom tags. Exceptions can be raised anywhere within a Python program. • Model: It describes the data structure/database schema. It contains a description of the database table, represented by a Python class. This class is called a model. Using it, is possible to create, retrieve, update and delete records in the database using simple Python code.

SPLMT is composed by the Product Management (PM), Risk Management (RM), Scope (SC), Requirements (RQ), and Test Module proposed in this work.

- **Persistence Layer:** All the data information is recorded at the repository.

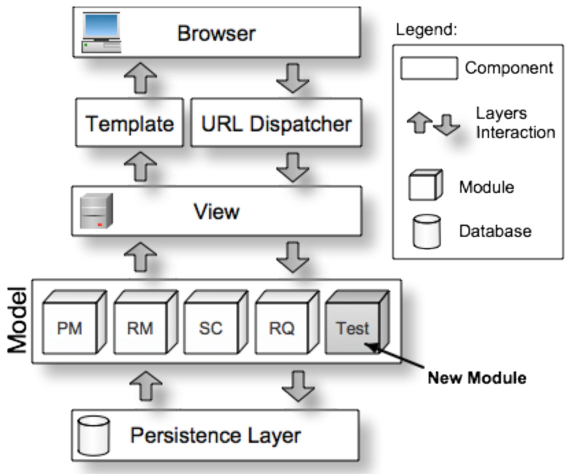


Figura 1. Architecture improved from [Neto et al. 2011a]

4.2.1. Technologies

The architecture of the tool was structured based on a set of technologies, as follows:

- **Django[2] -** is a Python Web framework that follows the Model-View-Controller (MVC) pattern. Through Django, the metamodel mapped entities and their relationship into Python[3] classes, and then it is automatically created a relational database for these entities.
- **Python -** It is the programming language used to implement the logic business of the tool. We decided to work with python because there is plenty of material available. It is also free to use because of its open source license.
- **MySQL -** It is the database for the application. Such database was chosen because it features indexing of content, stability, good performance, good documentation, strong open source community, and it is a free project.
- ***re* - Regular expression operations[4] -** It is a Python module that provides regular expression matching operations. *re* was used to construct the test cases steps through the manipulation of strings.

4.3. Implementation

The proposed tool was developed during 4 months of development (20 hours/week). It contains about 3000 lines of code, with approximately 85% of Django/Python code, 10% of HTML code and 5% of SQL. All the modules presented in the tool can run on multiple operating systems and supports several browsers.

Figure 2 presents the low level architecture that translates the process explained

[2] http://www.djangoproject.com
[3] http://www.python.org
[4] http://docs.python.org/library/re.html

at the previous sections and shows how the tool work precisely. After selecting the use cases at the proposed tool and selecting the option to create the test cases, the commands will be converted into ORM - Object Relational Mapping and the information will be directly extracted from the database.

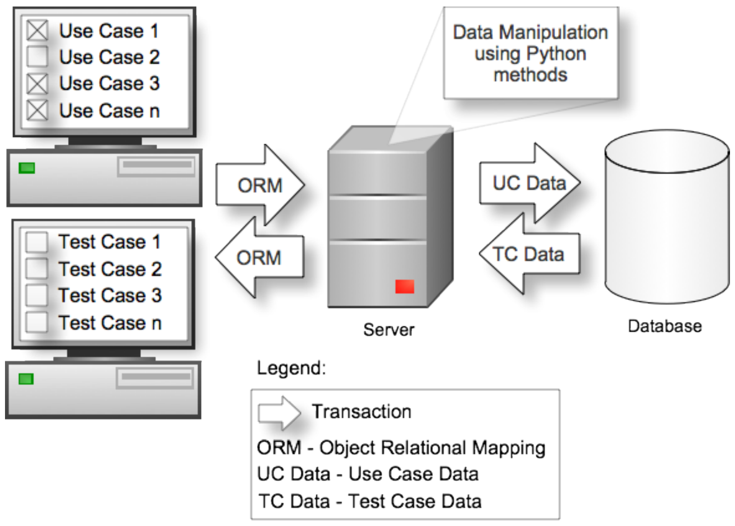


Figure 2. Low Level Architecture

The server will manipulate the data using Python methods implemented using regular expressions in order to create new test cases. Subsequently, these new test cases will be saved directly at the repository. Finally, the tool can visualize the test cases and the test assets can be organized into test suites and test plans, reducing the effort spent to manage the test assets of the product line.

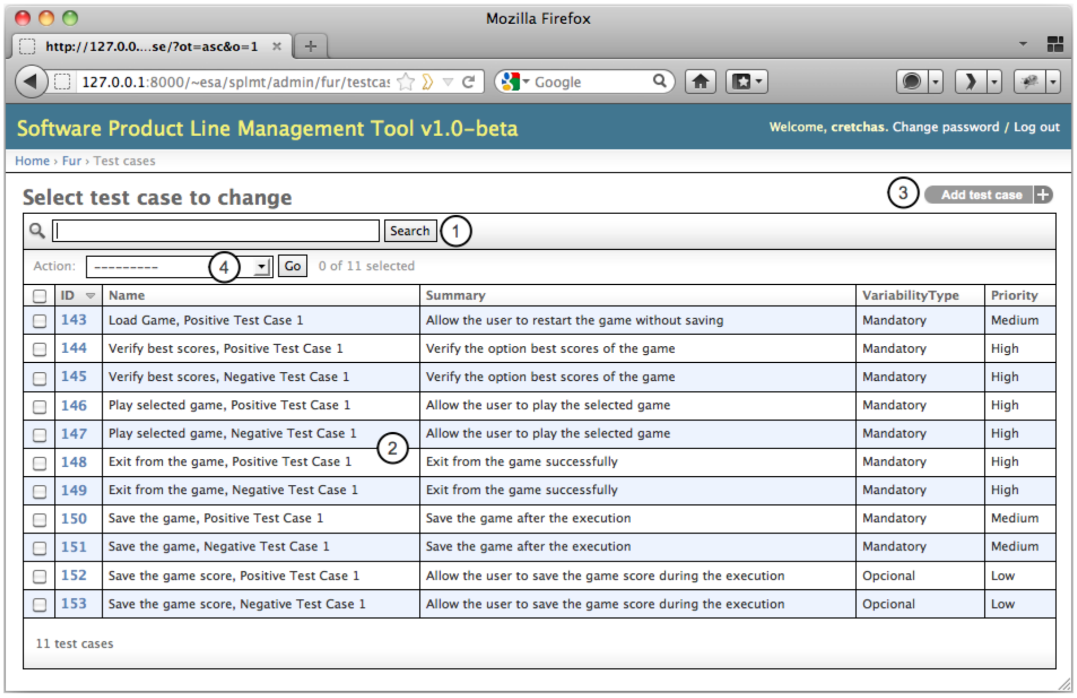
4.4. SPLMT-TE in Action

Figure 3 shows a screenshot of SPLMT-TE: on the top there is a field available to define the search keywords (1), at the middle of the application there is a list of test cases (2), if necessary, the testers can create more test cases (3) or invoke others functionalities (4). The functionalities are described as following:

1. **Search Bar -** This is the part of the tool, which the tester can search for specific test cases. It is also in that field where the search are specified by name of the test case, specific words and responsible for the tests.
2. **Test Case Visualization -** This field shows all the test cases of the repository, it also displays the results of the searches. Each line represents a test case, with information about: unique identification number of the test, Name of the test case, brief summary about the test case, variability type and priority.
3. **Adding Button -** The adding button can be selected when the creation of manually test cases is needed. The test case will be saved only if the mandatory fields were filled properly. Warning messages are displayed until the test case obeys the demands. Figure 4 presents one of these alert messages, requiring the filling of the Name field.

4. **Combo Box -** The combo box displays all the extra features of the tool such as the creation of reports, the deletion of one or more selected items, the test case creation, etc.

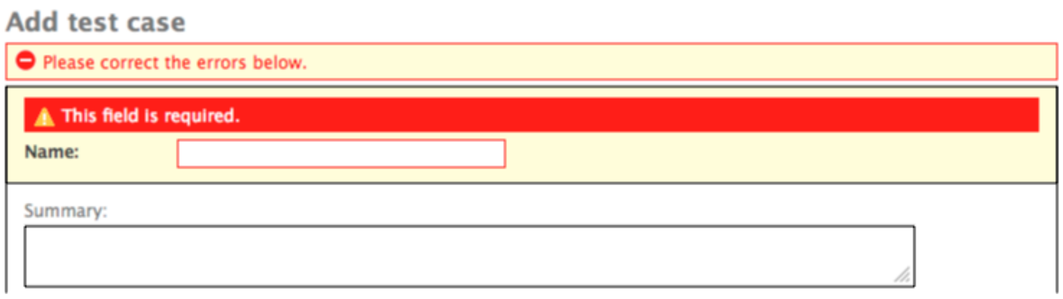
It is also possible to create tests suites. When the test suite creation option is selected, the test suite will be saved only if the mandatory fields were filled properly.


Mozilla Firefox
http://127.0.0....se/?ot=asc&o=1
127.0.0.1:8000/~esa/splmt/admin/fur/testcas
Google
Software Product Line Management Tool v1.0-beta
Welcome, cretchas. Change password / Log out
Home › Fur › Test cases
Select test case to change
3
Add test case
Search
1
Action: ---------
4
Go
0 of 11 selected
2

ID	Name	Summary	VariabilityType	Priority
143	Load Game, Positive Test Case 1	Allow the user to restart the game without saving	Mandatory	Medium
144	Verify best scores, Positive Test Case 1	Verify the option best scores of the game	Mandatory	High
145	Verify best scores, Negative Test Case 1	Verify the option best scores of the game	Mandatory	High
146	Play selected game, Positive Test Case 1	Allow the user to play the selected game	Mandatory	High
147	Play selected game, Negative Test Case 1	Allow the user to play the selected game	Mandatory	High
148	Exit from the game, Positive Test Case 1	Exit from the game successfully	Mandatory	High
149	Exit from the game, Negative Test Case 1	Exit from the game successfully	Mandatory	High
150	Save the game, Positive Test Case 1	Save the game after the execution	Mandatory	Medium
151	Save the game, Negative Test Case 1	Save the game after the execution	Mandatory	Medium
152	Save the game score, Positive Test Case 1	Allow the user to save the game score during the execution	Opcional	Low
153	Save the game score, Negative Test Case 1	Allow the user to save the game score during the execution	Opcional	Low

11 test cases

Figure 3. SPLMT-TE screenshot



Add test case

Please correct the errors below.

This field is required.

Name:

Summary:

Figure 4. Alert Message

Warning messages will be displayed until the test suite obeys the demands. The responsible for the test suite can be associated with the suite and a summary can explain and add information referred to the suite.

Moreover, at the test plan creation process, the test plan will be saved only if the mandatory fields were filled properly. In accordance with the other requirements, warning messages will be displayed until the test plan respects the demands. The responsible for the test plan can be associated and a summary explains and adds important information related to the test plan. The acceptance criteria field defines how many test cases should pass in order to validate the test plan. Finally, the release field defines the version of the application under test.

5. Experiment

We performed a controlled experimental study (in vitro) in order to evaluate the proposed tool. The experimental study consisted of two phases. Firstly, a pilot study was performed in order to analyze the experiment feasibility, as well as to identify possible deficiencies in the design of the proposed experiment. Secondly, the experiment was performed.

5.1. Experiment Design

This study applied the GQM method [Basili 1992] in order to collect and analyze metrics that are intended to define goals and was structured as follows:

Goal. The objective of this experimental study is to analyze the SPLMT-TE tool for the purpose of evaluation with respect to its efficiency and effectiveness from the point of view of the potential users (testers) in the context of a SPL testing project in an academic environment.

Questions. To achieve this goal, the following questions were defined:

- **Q1 -** *Is the time required to design system test cases reduced when the tool is used?*
- **Q2 -** *Is the amount of test cases increased when the tool is used?*
- **Q3 -** *Is the time required to execute the designed test cases reduced when the tool is used?*
- **Q4 -** *Is the amount of errors detected increased when the tool is used?*
- **Q5 -** *Is the effectiveness of test cases improved when the tool is used?*

Metrics. The questions were mapped to a measurement value, in order to characterize and manipulate the attributes in a formal way. Hence, the metrics used in this analysis are described next:

- **M1 -** *Designed Test Cases (DTC)*[Juzgado et al. 2004]: It refers to the number of designed test cases. This metric refers to Q2.
- **M2 -** *Efficiency in Test Case Design (ETCD)*[Juzgado et al. 2004]: This metric is related to the number of designed test cases *(DTC)* over the amount of time spent to create test cases *(TSC)* [Juzgado et al. 2004]. It is related to Q1.

$$ETCD = \frac{DTC}{TSC} \times 100 \qquad (1)$$

- **M3 -** *Efficiency of Test Cases Execution (ETCE)*[Juzgado et al. 2004]: Comprehends to the amount of executed test cases *(ETC)* over the amount of time necessary to execute the designed test cases *(TSE)* [Juzgado et al. 2004], including errors report. This metric is connected to Q3.

$$ETCE = \frac{ETC}{TSE} \times 100 \qquad (2)$$

- **M4 -** *Number of Errors Found (NEF)*[Juzgado et al. 2004]: It represents the number of errors found by the subjects during the execution. This metric is linked to Q4.
- **M5 -** *Test Cases Effectiveness (TCE)*: [Chernak 2001] proposed the metric to mea- sure test case effectiveness. He defines this metric as the ratio of defects found by test cases to the total number of defects reported during a test cycle. Test cases that find more defects are considered more effective. We adapted this metric to our context, so *TCE* consists of the amount of errors found (M4) reported to the total number of test cases created (M1). This metric is associated with the Q5.

$$TCE = \frac{NEF}{DTC} \times 100 \qquad (3)$$

5.2. Planning

In order to avoid the risks of conducting experiments in real projects, the experiment was conducted in an academic environment (not industrial software development). We also performed a pilot study to analyze the experiment feasibility, as well as to identify possible deficiencies in the design of the proposed experiment.

The subjects of the pilot study were 14 undergraduate *students* from the Software Engineering course at Federal University of Bahia, Brazil. The pilot was executed from June to July in 2011 and addressed a problem that was analyzed based on two *specific* SPL projects scenarios. All activities of the experiment were performed at the pilot study.

After performing the pilot study, the actual experiment was implemented. The subjects of the experiment were composed by graduate students (7 M. Sc. Students and 5 Ph. D. Students) from Federal University of Bahia and Federal University of Pernambuco, Brazil. The experiment was performed in July 2011, and used the same projects as pilot study.

5.2.1. Hypothesis Formulation

In this experimental study, we focused on ten hypotheses. They are formally stated with the measures needed to evaluate them.

Null Hypothesis (H_{0n}) - There is no benefit of using the proposed tool (described below as *Tool*) to support the design and execution of system test cases, if compared to the manual process (described below as *manual*), in terms of effectiveness. The Null hypotheses are:

$$H_{01} : \mu_{DTC_{manual}} = \mu_{DTC_{Tool}}$$
$$H_{02} : \mu_{ETCD_{manual}} = \mu_{ETCD_{Tool}}$$
$$H_{03} : \mu_{ETCE_{manual}} = \mu_{ETCE_{Tool}}$$
$$H_{04} : \mu_{NEF_{manual}} = \mu_{NEF_{Tool}}$$
$$H_{05} : \mu_{TCE_{manual}} = \mu_{TCE_{Tool}}$$

Alternative Hypothesis (H_{1n}) - Conversely to what was defined as Null Hypotheses, the alternative hypothesis determines that the proposed tool produces benefits that justify its use. We herein define the set of alternative hypotheses, as follows:

$$H_{11} : \mu_{DTC_{manual}} \neq \mu_{DTC_{Tool}}$$
$$H_{12} : \mu_{ETCD_{manual}} \neq \mu_{ETCD_{tool}}$$
$$H_{13} : \mu_{ETCE_{manual}} \neq \mu_{ETCE_{Tool}}$$
$$H_{14} : \mu_{NEF_{manual}} \neq \mu_{NEF\_without\_Tool}$$
$$H_{15} : \mu_{TCE_{manual}} \neq \mu_{TCE_{Tool}}$$

5.3. Operation

The experiment was hold at the Software Engineering Laboratory (LES[5]) from the Federal University of Bahia. The subjects used the Eclipse IDE, the software of the Arcade Game Maker[6], the software of the NotepadSPL extracted from the FeatureVisu[7], and the proposed tool. The requirements documents, all the forms and reports were provided in digital copies.

5.3.1. Execution

The activities involved in the experiment were executed by the subjects. The characterization was made when the subjects filled in the background form, allowing us to organize the subjects into two groups.

After the characterization, the subjects were trained at the concepts of software testing. Both groups learned how to create a good test case, how to execute test cases, and how to report errors.

Before using the proposed tool, the subjects learned how to use the tool. Both groups performed the experiment using the tool in one phase and without using the tool in the other phase. At the end, they had to fill out the feedback questionnaire.

5.4. Analysis and Interpretation

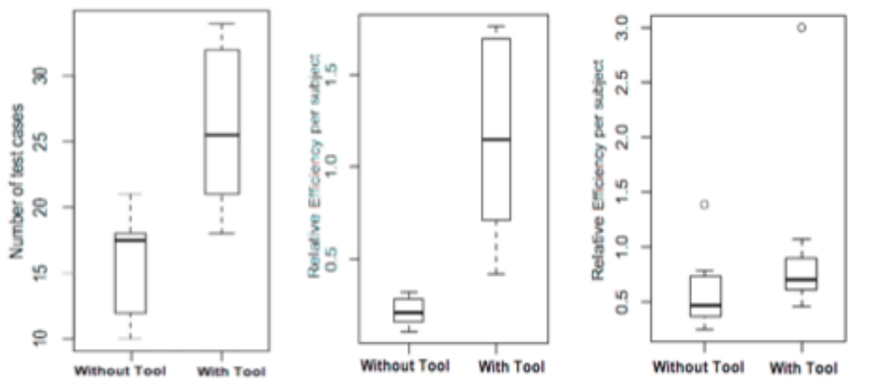
We analyzed all the artifacts produced by the subjects, including the error report forms and the feedback questionnaires.

[5] http://wiki.dcc.ufba.br/LES/WebHome
[6] http://www.sei.cmu.edu/productlines/ppl/index.html
[7] http://fosd.de/FeatureVisu

5.4.1. Designed Test Cases

Boxplot shown in Figure 5(a) contains data from the distribution of test cases created by tool usage. In groups (1 and 2) using the tool, the *mean* was 26.10 with *Standard Deviation (Std.Dev.)* of 6.06; and 15.70 of *mean* with *Std.Dev.* of 3.83 in groups (1 and 2) without using the tool. Median of groups using the tool is higher than groups that did not use tool support.

According to Figure 5(a) the amount of designed test cases created with the tool is higher than without tool support. The proposed tool enabled the creation of more test cases, 261 with tool support and 157 without it.



(a) Designed Test Cases (b) Efficiency in Test Case Design (c) Efficiency in Test Case Execution

Figure 5. Bloxplots

5.4.2. Efficiency in Test Case Design

Figure 5(b) presents the distribution of time to design the test cases per subject. In groups (1 and 2) using the tool, the *mean value* was 25.70 with *Std.Dev.* of 10.48, while in groups (1 and 2) without using tool support, the *mean value* was 82.00, with *Std.Dev.* of 43.69. The *median value* of groups using the tool is greater than groups without using tool support.

The efficiency in test case design was higher with groups that used the tool, which allowed the creation of more test cases faster than without tool support as presented in Figure 5(b). The time needed to create system test cases decreases using the tool, reducing from 820 minutes to 257 minutes, saving 563 minutes.

5.4.3. Efficiency in Test Cases Execution

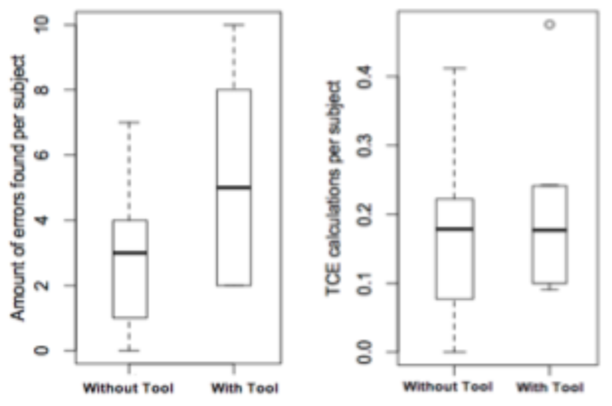
The *mean* in groups (1 and 2) using the tool was 0.94 with *Std.Dev.* of 0.74. In groups without the tool, the *mean* was 0.59 with *Std.Dev.* of 0.33. *Median* of groups using the tool is higher than in groups without using the tool (see Figure 5(c)).

There was no significant difference during the execution of the created test tools. Both groups, with and without tool support, executed a similar number of test cases during the same period of time. The effort of test case execution were almost the same too, 328 minutes with tool support and 300 minutes without, we need to consider that the number of test cases executed were higher with the tool (256 test cases) than without tool support (149 test cases).

5.4.4. Number of Errors Found

Figure 6(a) shows the Boxplot with distribution of the number of errors found per subject. *Mean* of the groups (1 and 2) using the tools was 5.20 with *Std.Dev.* of 2.97. The *mean* of the groups (1 and 2) without using the tool was 2.90 with *Std.Dev.* of 2.23. *Median* of the groups that used the tool is higher than in the groups that did not use it.

According to Figure 6(a), the amount of errors found with the test cases created by the tool was slightly higher than with the test cases created manually. Moreover, the subjects were able to find more errors during the use of the tool, 38 errors found with tool support and 26 without.



(a) Errors Found (b) Test Case Effectiveness

Figure 6. Bloxplots

5.4.5. Test Cases Effectiveness

Concerning the use and applicability of the TCE metric, to determine whether if a test case created by the SPLMT-TE was effective, we compared the TCE from both groups, using and without using tool support. We measured the test case effectiveness as a ratio of the total amount of designed test cases by the total of errors found.

By applying the TCE formula, we obtained the distribution presented in Figure 6(b). It shows the data similarity. *Mean* of the groups (1 and 2) using the tool was 0.19 with *Std.Dev.* of 0.11. The *mean* of the groups (1 and 2) without using the tool was the same, 0.16 with *Std.Dev.* of 0.12. *Median* of groups using and not using the tool is the same, 0.17.

There was no significant difference between the test case effectiveness. The number of errors found per number of test cases was almost the same using and without using the tool. Test case effectiveness with and without the tool support was almost the same, 14.55 using the tool and 16.56 without using it.

5.4.6. Hypotheses Testing

Since the experiment has one factor with two treatments, completely randomized design, the data collected during the experiment were submitted to parametric tests, **t-test** to analyze the hypothesis. The tests are primarily presented for a significance level of 5%.

Regarding the Q2, the amount of test cases created using the tool is higher than without using it (see Figure 5(a)), *t-value* = -4.59, degrees of freedom (*df*) = 15.2, and p-value = 0.0001729. The number of the *p-value* is lower than the significance level, rejecting the null hypothesis.

Time spent creating test cases without the tool is higher than using it (see Figure 5(b)). For this reason, in order to answer the Q1, the Null Hypothesis H_{02} is rejected, since, *t-value* = 5.9,*df* = 9.4,and*p-value* = 0.0001999. This *p-value* allowed the rejection with high significance level.

Figure 5(c) shows the efficiency in test case execution, which supports the Null Hypothesis H_{03}. Thus, the question Q3 is answered, *t-value* = 1.4, *df* = 12 and *p-value* = 0.09917. Since, the *p-value* is lower than 0.1, we raise the significance level to 10% in order to reject this hypothesis.

The Null Hypothesis H_{04} is rejected. The number of errors found were higher using the tool support, *t-value* = -1.96, *df* = 16.7, and *p-value* = 0.0373, answering the Q4. The *p-value* is lower than 5% rejecting the null hypothesis.
As a result, the Null Hypothesis H_{05} cannot be rejected. *t-value* = -0.498, *p-value* = 0.3122, and *df* = 17.9. Since the *p-value* is higher than the significance level the hypothesis cannot be rejected and no conclusion can be drawn. Regarding Q5, there is no significant differences between the effectiveness test cases values during the tool usage and without use it.

5.4.7. Qualitative analysis

Data from subjects who used the tool were qualitatively analyzed. Only 20% of the subjects needed additional information other than the available artifacts. The majority of subjects approved the use of the tool, except the factor on how effective the tool was. In summary, the overall comments regarding difficulties referred to lack of expertise in software testing, which impacted on their activities.

Ninety percent of the subjects agreed that the SPLMT-TE helped them at the creation of test cases and find more errors. Sixty percent of the subjects believed that the tool created sufficient test cases. Moreover, 33% changed the test cases created by the tool and 35% created more test cases.

5.5. Lessons Learned

After concluding the experimental studies, we gathered information that was used as a guide to the replication. The structure presented in this section can be reused in other general experiments in the SPL Testing practice.

Some important aspects should be considered, specially the ones seen as limitations in the experiment. We tried to mitigate some of these problems that we raised at the pilot study. The general impressions gathered from the experiment are described below:

Training. To eliminate the problem with lack of experience, we selected subjects with experience in the SPL and software testing area.

Motivation. To mitigate the complaints with boredom, we removed the 2 hours limitation for creation and execution. A possible solution for future replications could be to define only one activity by subject. The subjects with experience will be responsible for test creation activities and the inexperienced subjects will only execute and report errors.

Questionnaires. To suppress some problems related to the wrong data collection, we changed some questions of the questionnaires. The questionnaires should be constantly updated and calibrated before new replications.

Project. We would select projects with more specification available in advance. Subjects, mainly from the experiment, with a large experience in industry, complained about the lack of documentation to aid them to create the test cases. Moreover, as we are dealing with SPL projects, we need projects containing many variation points and variants in order to analyze the impact of the tool usage.

Design Type. We changed the design type at the experiment. At the first round, the subject from group 1 used the tool with the domain 1, and the group 2 also used the domain 1 but without tool support. At the second round, group 1 analyzed the domain 2 without tool support and group 2 analyzed the domain 2 with tool support.

Measurement. In order to eliminate the data loss that happened in the pilot study, we changed the instructions for the experiment, where each subject did the activities individually allowing a better interpretation of the data. We also enforced the importance of collect information during the creation of test cases with the tool.

6. Conclusion and Future Work

In this paper, we applied the mapping study [Neto et al. 2012], to collect evidence in the literature that allowed us to sketch and comprehend the state-of-the-art of single system and SPL testing tools research and practice field. The motivation was to identify the evidence available on the topic and point out gaps in the existing tools.

We presented the SPLMT-TE, a web-based tool developed to build system test cases from use case and manage test assets such as test cases, test suites and test plans [Neto et al. 2011a]. The tool was built based on the needs to facilitate these tasks reducing the effort and the costs of the Software Product Lines testing process.

Thus, the architecture was presented as well as its components, features for creating and managing the test assets, and implementation details. Moreover, we performed a controlled experimental study (in vitro) to evaluate the tool effectiveness in order to re- duce the effort spent during the system testing level of the SPL testing process. We also realized one replication of the experiment, but more investigation still needed including a replication in industry context.

The results for the quantitative analysis showed that the tool is adequate to perform the management of SPL test assets, in sense that it can reduce the time of analysis.

For qualitative analysis, all subjects preferred to use the SPLMT-TE. Finally, as future work, the tool will be evolved to include functionalities that can improve the support of the SPL Testing process.

As future work, we intend to incorporate new requirements in the tool based on the mapping study and feedback from the experiment. Moreover, we are planning the replication of the experimental study before using the tool in industry.

Acknowledgment

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08 and CNPq grants 305968/2010-6, 559997/2010-8, 474766/2010-1.

References

Basili, V. R. (1992). Software Modeling and Measurement: the Goal/Question/Metric Paradigm. Technical Report CS-TR-2956, College Park, MD, USA.

Cavalcanti, Y. C., Machado, I. C., Neto, P. A. M. S., Lobato, L. L., Almeida, E. S., and Meira, S. R. L. (2011). Towards Metamodel Support for Variability and Traceability in Software Product Lines. *5th International Workshop on Variability Modelling of Software-intensive Systems.*

Chernak, Y. (2001). Validating and Improving Test-Case Effectiveness. *IEEE Softw., 18:81–86.*

Fewster, M. and Graham, D. (1999). *Software Test Automation: Effective Use of Test Execution Tools*, volume 10. John Wiley Sons, Ltd.

Juzgado, N. J., Moreno, A. M., and Vegas, S. (2004). Reviewing 25 Years of Testing Technique Experiments. *Empirical Software Engineering*, 9(1-2):7–44.

Nakagawa, E. Y., Simão, A. S., Ferrari, F. C., and Maldonado, J. C.(2007). Towards a Reference Architecture for Software Testing Tools. In *International Conference on Software Engineering & Knowledge Engineering*, pages 157–162. Knowledge Systems Institute Graduate School.

Nebut, C., Traon, Y., and Jezequel, J. (2007). System Testing of Product Lines: From Requirements to Test Cases. *Software Product Lines*, pages 447–477.

Neto, C. R. L., Machado, I. C., Neto, P. A. M. S., Almeida, E. S., and Meira, S. R. L. (2011a). Software Product Lines System Test Case Tool: A Proposal. In *International Conference on Software Engineering & Knowledge Engineering*, pages 699–704. Knowledge Systems Institute Graduate School.

Neto, C. R. L., Neto, P. A. M. S., Almeida, E. S., and Meira, S. R. L. (2012). A Mapping Study on Software Product Lines Testing Tools. In *International Conference on Soft- ware Engineering & Knowledge Engineering*. Knowledge Systems Institute Graduate School.

Neto, P. A. M. S., Runeson, P., Machado, I. C., Almeida, E. S., Meira, S. R. L., and Engstrom, E. (2011b). Testing Software Product Lines. *IEEE Software*, 28:16–20.

Oster, S., Zorcic, I., Markert, F., and Lochau, M. (2011). MoSo-PoLiTe: Tool Support for Pairwise and Model-Based Software Product Line Testing. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, pages 79–82, New York, NY, USA. ACM.

Reuys, A., Kamsties, E., Pohl, K., and Reis, S. (2005). Model-Based System Testing of Software Product Families. *International Conference on Advanced Information Systems Engineering*, pages 519–534.

Tevanlinna, A., Taina, J., and Kauppinen, R. (2004). Product Family Testing: a Survey. *ACM SIGSOFT Software Engineering Notes*, 29:12–12.