Journal of Universal Computer Science, vol. 20, no. 5 (2014), 615-639
submitted: 28/7/13, accepted: 15/2/14, appeared: 1/5/14 © *J.UCS*

Controlled Experiments Comparing Black-box Testing Strategies for Software Product Lines

Paola Accioly
(Federal University of Pernambuco, Recife, Brazil
prga@cin.ufpe.br)

Paulo Borba
(Federal University of Pernambuco, Recife, Brazil
phmb@cin.ufpe.br)

Rodrigo Bonifácio
(University of Brasília, Brasília, Brazil
rbonifacio@cic.unb.br)

Abstract: SPL testing has been considered a challenging task, mainly due to the diversity of products that might be generated from an SPL. To deal with this problem, techniques for specifying and deriving product specific functional test cases have been proposed. However, there is little empirical evidence of the benefits and drawbacks of these techniques. To provide this kind of evidence, in a previous work we conducted an empirical study that compared two design techniques for black-box manual testing, a generic technique that we have observed in an industrial test execution environment, and a product specific technique whose functional test cases could be derived using any SPL approach that considers variations in functional tests. Besides revisiting the first study, here we present a second study that reinforce our findings and brings new insights to our investigation. Both studies indicate that specific test cases improve test execution productivity and quality.

Key Words: Black-box Testing, Software Product Lines, Empirical Software Engineering
Category: D.2.7, D.2.5, G.3

1 Introduction

Efficient testing techniques are important for achieving software quality and reliability. Since an SPL can generate a range of different products, it is challenging to write black-box test cases based on product specifications. The main difficulties are the large number of products that can be instantiated from an SPL and the variation points scattered through different scenarios.

To address those challenges, SPL tests specification techniques like Pluto [Bertolino and Gnesi, 2003] and ScenTED [Pohl and Metzger, 2006] have been proposed. They introduce constructs that represent product variability and provide means to derive product specific test cases. Nevertheless, the research community still lacks empirical studies that evaluate these proposals in order to give a solid foundation for SPL testing in industry [Tevanlinna et al., 2004,Engström and Runeson, 2011].

Perhaps the absence of evidence about the benefits of such techniques discourages their industrial adoption. As a result, from what we have observed in an industrial test execution environment, companies use test documents with use case scenarios that usually describe family behaviour as a whole, describing most commonalities, and abstracting the fact that some steps are optional or alternative —in some cases, non mandatory steps are even omitted. For example, a test case that specifies the scenario of a report generator feature would contain all possible variants for report formats such as PDF, HTML and XLS, and testers would use this generic test case to test all SPL products, even those that are not configured with all these options. Such test suites may hamper manual execution because the lack of details can mislead testers that have to strictly follow the test steps. This is particularly true when different teams are responsible for the construction and testing activities. This leads to unwanted consequences, such as escaped defects –when testers do not find an error prior to product release. In addition, testers may take longer to execute test cases and report defects that do not exist, decreasing test execution productivity.

Alternatively, with the adoption of an SPL technique, it would be possible to derive different versions of the same test suite customised for the different product configurations in the SPL. This way, testers would not get confused during test execution and the problems mentioned above could be avoided. However, organisations cannot decide to introduce new techniques or change their usual methods based only in assumptions, they need evidence.

In a previous work [Accioly et al., 2012] we presented an empirical study that brought evidence to help decision making in such contexts. Here we describe such a study and extend it by presenting a second experiment, adding new results and insights. The motivation to extend this work is twofold. Replicate the experiment with different subjects, in order to gather more evidence to the previous results; and investigate the impact on productivity of the different activities done during test execution. Both experiments' results support the hypothesis that an SPL approach for software testing brings benefits to both productivity and quality. In the next section we present the differences between the Generic Technique (GT) which specifies most variants specifications together without variability representation, and the Specific Technique (ST) which specifies product customised test suites (Section 2). After that, we present our empirical studies (Section 3) comparing both experiments and their results. Finally, we present related work in Section 4 and our final considerations in Section 5.

2 Generic and Specific Test Cases

To better understand the two techniques, we describe examples of how test cases may turn up to be generic (describing inaccurate family overall behaviour) or product specific (showing the specific steps and data values suitable to each product). Besides presenting this difference, we also explain the consequences of using the GT. We have observed the use of the GT in test teams that focus only on the test execution process of a company that outsources test execution activities to test centers located in different countries. To illustrate the techniques, we use an example of a mobile SPL that manages the interaction of mobile multimedia content (pictures, videos and music), Multimedia Messaging Service (MMS) and some requirements made by a specific mobile carrier called here as Blue Carrier (BC) feature. These examples represent some of our observations in the mentioned industrial context.

2.1 Test Case: User Sends MMS with Attached Picture

Our first example considers the scenario of a user sending an MMS with an attached picture, as detailed in Table 1. This scenario applies for most products of the SPL in discussion. However, it does not apply for products containing the BC feature, which corresponds to a group of requirements associated to this carrier that requires that, before sending an MMS, a message pops up asking if the user really wants to send that message —since data transfer will be further charged. Table 2 specifies the test case for the products that follow the BC feature requirements. On steps 7 and 8 we can see the differences from Table 1.

Table 1: Generic test case: user sends MMS with picture attached.

Step ID	User Action	System Response
1	Go to Main Menu	Main Menu appears
2	Go to Messages Menu	Message Menu appears
3	Select “Create new Message”	Message Editor screen opens
4	Add Recipient	Recipient is added
5	Select “Insert Picture”	Insert Picture Menu opens
6	Select Picture	Picture is Selected
7	Select “Send Message”	Message is correctly sent

In the GT, the test case detailed in Table 1 would serve to test all SPL products and the tester would be confronted with an unexpected output while testing products configured with the BC feature. On the other hand, when using

Table 2: Specific test case for products configured with the BC feature.

Step ID	User Action	System Response
1	Go to Main Menu	Main Menu appears
2	Go to Messages Menu	Message Menu appears
3	Select "Create new Message"	Message Editor screen opens
4	Add Recipient	Recipient is added
5	Select "Insert Picture"	Insert Picture Menu opens
6	Select Picture	Picture is Selected
7	Select "Send Message"	Dialog appears: "Are you sure you want to send this message? Data transfer shall be charged"
8	Hit "Yes"	Message is correctly sent.

the ST, there would be two different test cases. The first, detailed in Table 1, would serve to test the products not configured with the BC feature, whereas the second, detailed in Table 2, would serve to test products configured with the BC feature.

In manual black-box testing, when the test specification fails to agree with the product behaviour, it often means that the test case has revealed a defect. However, when testing a product with the BC feature, that is not what happens with the generic test case just described. Here, the product works fine according to its configuration. The problem is that the generic test case does not consider all steps required to perform the task correctly. The implementation is correct, but the specification is vague, so that it roughly fits different SPL members. In this context, when the tester is not familiar with the products specificities prior to test execution, he/she might interpret the test inaccuracy as a product defect. This misunderstanding can be solved if the tester investigates and finds evidence that the test specification does not apply for that product. The tester can do that by contacting a requirements analyst or reading the products specification. However, if he/she cannot find this evidence, he/she will report an invalid product defect, wasting her and other people's time to analyse the inaccuracy.

Besides reporting false defects, a different type of issue might happen, when a product is configured with the BC feature and does not present the alert message before sending the MMS. In this case, the product was not correctly implemented, and the generic specification is vague. There is a product defect that should be reported so that the development team could fix it. However, the tester will not be able to notice this defect because the generic test case does not consider the step that shows the alert message. If the defect is not reported, the product might be released to the market without considering the BC requirements. This will likely lead to an *escaped defect*, that is, defects that are found after the product release. This scenario is worse than reporting invalid defects because it directly affects products quality, whereas reporting invalid defects only affects testing productivity.

In this example, the generic test case presents fewer steps than necessary to specify the behaviour for products configured with the BC feature. However, this situation can be inverted if the test analyst evolves the generic test suite with the expected behaviour for the BC feature. In this context, the test case described in Table 2 would serve to test all SPL products including those that were not configured with the BC feature. While testing such products, testers would be confronted with an unexpected behaviour when pressing the "Send Message" button. Instead of showing the warning message, the product would send the MMS right away. In this case, the generic test case presents more steps than the specific product requires, bringing the same problems already discussed.

The last two examples discussed generic tests as presenting fewer or more steps than necessary. Nevertheless, a third situation may happen since test cases sometimes specify icons and labels that the tester should check during the tests. So if, for example, the BC feature requires that some icons need to be changed so that they display their brand logo, the GT may fail to specify what icons and labels the device should display. Again, testers confronted with this kind of inaccuracy might waste time, report invalid defects or even let a defect escape.

In summary, generic test cases may differ from specific test cases in three different ways. They might present less or more steps than an specific product requires or they might present different data values such as icons and labels. Note that the generic test suites described here have no means of representing variability to inform which steps apply or not to each product. They simply describe most commonalities, abstracting possible variations, and are partly not correct depending on the product configuration.

2.2 Problems With Generic Test Suites

In a black-box manual test execution environment, testers are not required to have specific knowledge of the application code structure. Only by reading the test steps, they are aware of what the system under test is supposed to do. They follow the user action steps checking if the product behaves according to the described system responses. Whenever there is an inconsistency between the test case and the product behaviour, testers must investigate whether this inconsistency is a defect. However, the use of generic test cases for SPL products may hamper test execution because when they fail to specify a certain product behaviour, testers are not able to identify if there is an issue in the test case. Instead, they might interpret it as a product defect.

The activity diagram described in Figure 1 considers the scenarios that typically happen when a test case is not accurate. The activity flow starts when the test case does not correctly specify the behaviour of the system under test, presenting an issue similar to those described before. Then the fork on the diagram indicates two possible scenarios. In the left branch the product matches the test case description, in other words, there is a defect but the tester will not be able to notice because the test case is also wrong. So the tester passes the test and the consequence is an escaped defect. For instance, in the first example described in Section 2.1, Table 1, step 7, the mobile company would release the product with the BC feature without the message prior to the MMS dispatch.

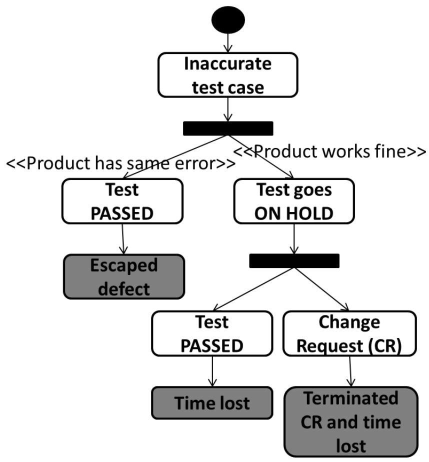

Figure 1: Possible consequences of generic test cases.

In the right branch, the product works fine according to its configuration, so the tester will find an unexpected output, pausing the test case execution to start investigating whether there is an issue in the test case or in the product. The tester might search throughout requirement documents or eventually speak to a requirements analyst, to the development team or to other testers who already executed that test case. Then, if the tester finds evidence that the product works as expected by costumers, he/she passes the test, writing an observation about the test inaccuracy, and the consequence is time lost with the investigation.

Based on our observations in a medium size test organisation, this kind of investigation can take a small amount of time if the tester talks to a technical leader or a requirement specialist available personally or via instant messaging. On the other hand, it can take a lot of time if, for instance, the tester needs to look throughout requirement documents to find out about the expected system behaviour. Finally, the tester might also contact the development team, having to wait for an answer. For instance, in the organisation that we mentioned before, the development team worked in a different time zone, so the questions took longer than one day to be analysed. Meanwhile the test case remained on hold and the tester moved on with the test suite.

Either way, if the tester cannot find evidence about the expected product behaviour, he/she will assume that there is a product defect. The tester will create a Change Request (CR)[1] and, when the development team gets to analyse the CR, they will get to the conclusion that the product works fine, then terminating the CR. In this case the consequences are time lost and an invalid CR which is a negative metric indicating that the tester reported a product failure that did not exist.

In summary, the GT might impact SPL development with respect to two aspects: quality and productivity. Quality because some defects might escape, and productivity because of the time lost during investigations and possible invalid CRs. The more often these inaccuracies appear on the test cases, the more significant is the impact on the test execution process. Particularly, SPLs that contain more variation points are more likely to present such problems.

As a final remark, we remember that poorly specified test cases may present issues like the ones described in this section despite of which technique the company uses for SPL products or even for single products specification. So, using the ST may improve the test cases quality but it is not a guarantee that they will not present any kind of issue.

2.3 Product Specific Test Cases for SPL

A number of techniques can be used to derive product specific test cases for a given SPL. A naive alternative is to copy the same test document for each product line configuration to be tested and manually adjust the differences between them. However, this solution is not quite appropriate because the more complex the SPL is, the harder it is to maintain each product's test documents. The alternative to obtain product specific tests is to reuse test cases for the different products in a given SPL. This reuse can be done in, at least, two different ways. First, we can use an SPL technique that manages test cases variability and derive product specific test cases. Some of the existent approaches are PLUTO [Bertolino and Gnesi, 2003] and ScenTED [Reuys et al., 2005]. The second alternative is to structure requirements specifications using modularisation mechanisms so that it is possible to generate requirements specifications for SPL products. An existing technique for this matter is MSVCM [Bonifácio and Borba, 2009]. After deriving the expected behavioural descriptions (using

[1] In this work we use the CR definition only in the context of products defect report.

scenario specifications, for instance), these specifications can be used as input to an automatic model-based test suites generation tool such as TaRGeT [Neves et al., 2011].

Either way, we believe that having product specific test cases might help to solve SPL test execution problems. Nevertheless, the benefits regarding the test execution productivity are not so obvious since specific test suites, in some cases, present more steps than the generic version, requiring more time for executing the specific test cases. To evaluate these statements, we compare the GT and the ST using the point of view of the test execution process. Likewise, it is important to compare these techniques using the point of view of the test design process since the gain on test execution might not compensate the effort to design product specific test suites. Initially, if we use an SPL test derivation technique, there would be an increase of effort to design test suites with variability representation compared to the generic ones. However, once this initial step is done, not only the test execution could benefit from it but also the maintenance of test suites would be easier.

However, we cannot evaluate these two processes (design and execution) in a single study for a number of reasons. First, the team that designs the test suites is usually different from the team that executes them. This would essentially separate this study into two. Second, while the test design is done once and then maintained, the test execution is done several times so that it would be difficult to interpret the results in a realistic way. Finally, different companies focus differently on these two processes. Some focus more on execution than on design, whereas others do the contrary or even focus equally on both. This leads to problems on the generalisation of the results.

Because we cannot evaluate the process of designing and executing SPL test cases in a single experiment, and also because we have experienced the problems from the test execution point of view, making it our area of expertise, we first focus on the test execution process, considering that the test cases are already specified as generic or specific. We consider this work as the first step towards a deeper understanding on the benefits and disadvantages of adopting product specific test suites. Our results here can be particularly interesting for companies that focus more on test execution.

3 Evaluation Studies

In this paper we empirically evaluate both the GT and the ST techniques from the point of view of the SPL test execution process. Figure 2 illustrates this comparison. On the left side, the GT provides a single generic suite that testers will use to test two different products, P1 and P2. Differently, on the right side, the ST provides two different suites: P1 Suite and P2 Suite, each one specifying its respective product.

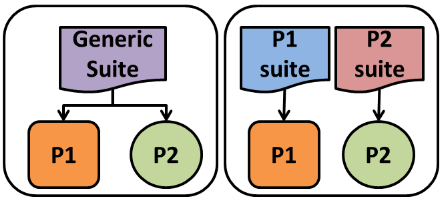

Figure 2: The GT uses one test suite for all the products an the ST uses one suite per product.

In order to compare these two techniques in terms of test execution, we conducted two controlled experiments where subjects had to test different products from the same SPL using either the GT or ST techniques and collecting the time taken to execute the test suites. During this activity, students also reported CRs whenever they identified defects.

In the previous paper [Accioly et al., 2012] we presented the first experiment and now we add a second study that brings the following two main contributions: first it consists of a replication of the first experiment with new subjects, which adds evidence to the results; second, it yields new insights about the impact of the different activities done during the test execution process. This time we present a deeper analysis of the impact of reporting CRs during the experiment execution. To get this new insights we changed our time metric collection procedure as discussed in Section 3.2.3. This adjustment was useful because we had no evidence about how much time the subjects took to report CRs, and how this task would increase the total time of the tests execution. In the following subsections we describe both experiments, explain their differences and their results. Also, all the material used in the experiments is available online [Accioly, 2013].

3.1 Experiments Definition

We have structured the experiments definition using the goal, question, metric (GQM) approach in order to collect and analyze meaningful metrics to measure the proposed process. Our **Goal** was to analyse the test execution process, for the purpose of evaluating two different SPL test case design techniques (GT vs. ST), with respect to the mean time to execute the test suites as well as the number of invalid CRs reported during the test execution process. We have used the point of view of test engineers and researchers in the context of controlled experiments done with graduate and undergraduate students.

To achieve our goal, we elaborated two research questions. **Research Question 1** (**RQ_1**), which investigates whether the ST reduces the test execution effort compared with the GT and **Research Question 2** (RQ_2) which investigates whether the ST reduces the number of invalid CRs compared with the GT. These questions should be answered with the **test execution time** and the **number of invalid CRs** metrics because, from what we have observed, the GT may decrease test execution productivity since testers might take longer to execute test cases besides report defects that do not exist. We also collect other metrics such as valid CRs —CRs that report real product defects– because we need to analyse all reported CRs in order to identify the invalid ones.

3.2 Experiments Planning

To evaluate the elements involved in our experiments planning, first we describe our statistical hypotheses that is later confronted with the collected data.

3.2.1 Hypothesis

To answer RQ_1 concerning the average time to execute the test suites, we present our null hypothesis as H_0 ($\mu_{TimeST} = \mu_{TimeGT}$) and our alternative hypotheses as H_1 ($\mu_{TimeST} < \mu_{TimeGT}$) and H_2 ($\mu_{TimeST} > \mu_{TimeGT}$). Likewise, in order to answer RQ_2, we present the null hypothesis as H_0 ($\mu_{InvalidCRsST} = \mu_{InvalidCRsGT}$) and the alternative hypotheses as H_1 ($\mu_{InvalidCRsST} < \mu_{InvalidCRsGT}$) and H_2 ($\mu_{InvalidCRsST} > \mu_{InvalidCRsGT}$).

3.2.2 Design, Instrumentation and Subjects

Bringing the elements evaluated in our context into empirical terms, the metrics evaluated were the execution time (in the second experiment we collected the execution time plus the CR report time as discussed in Section 3.2.3) and the number of invalid CRs. In addition, the treatments (the independent variables) that we compare are the GT and the ST. Furthermore, in order to avoid *bias*, there are some factors that we need to control during the experiment execution as discussed next.

The first factor under control is the subjects selected to execute the experiment, since different background knowledge and testing skills can influence directly on our metrics. The second factor under control is the number of variation points existent in the test cases, since test suites with more variation points might benefit more from the ST than test suites with fewer variation points. In a test execution environment, like the one that we have observed, test suites are usually related to SPL features. They explore flows related to the main functionality of a feature and its interaction with other features. Thus, we control the variation points factor by choosing test suites related to two different features from the same SPL.

Since we have two treatments, two test suites and many subjects, we opted to use a Latin Square Design [Wohlin et al., 2000] to control both the subjects and the number of variation points (related to features). To apply this design, we randomly dispose the subjects in the rows and the features in the columns of the Latin square (see Table 3). Each pair of subjects sets one instance of a Latin square, and each instance is called a replica. Each replica contains two rows and two columns. In each given row and column, the treatment, the GT or the ST, appears only once, meaning that, in each activity, subjects used one technique and one feature. In each activity using one of the techniques, the subjects executed two test suites in two products. With the ST, they execute specific suites for each product. With the GT, they execute the same test suite in the two products as illustrated in Figure 2. We randomly selected the execution order of each product.

Table 3: Layout of experiment design.

	Feature 1	**Feature 2**
Subject 1	GT	ST
Subject 2	ST	GT

To evaluate the treatments, we considered products and test suites from the Research Group Management System (RGMS) SPL,[2] a Java desktop product line with the purpose of managing members, publications and research lines from a research group. The products and the test cases are written in Portuguese since the differences among subjects English skills could affect the execution time.

RGMS most important features are *Members*, *Publications* and *Research Lines*. These features are responsible for the query operations with these entities (inserts, search, updates and deletes). The *Reports* feature is responsible for reports generation in two different formats: PDF or Bibtex. Also the *Research Line Search by Member* feature makes it possible to know which research lines associates with each member registered on the system. Similarly, *Publication Search by Member* retrieves the publications associated with the members of the research group. Lastly, *Global Search* retrieves members, publications and research lines. This feature model is available on the online appendix [Accioly, 2013]. To execute the experiment we chose the two following product configurations:

P1: RGMS, Members, Publications, Research Lines, Reports (PDF, Bibtex), Research Line Search by Member

[2] http://rgms.rcaa.cloudbees.net/

P2: RGMS, Members, Publications, Research Lines, Reports (PDF), Publication Search by Member, Global Search

The features chosen to design the test suites were *Publications* and *Research Lines* because they were rich and contained different flows to explore. They are also sufficiently independent from each other, generating test suites with separate flows. We did not choose the *Members* feature because it is tangled in both products and each feature should present different flows of execution, in order to avoid participants from learning the tool from one feature execution to the other. In other words, if we choose features with similar flows, participants might learn how to use the product when executing the first feature suites and then execute the second feature suites faster.

We manually wrote the test suites instead of using an SPL test derivation technique because we do not focus on studying the benefits of one test derivation technique in special, but on the test design techniques in general. In total, there were 6 test suites. This number was a consequence of our experiment design. First, we produced the generic test suites for F1 and F2 (GS-F1 and GS-F2) trying to simulate the issues discussed in Section 2. To simulate the scenario where the test case has more steps than necessary, some test cases presented steps that did not apply for both products under test. Likewise, to simulate the scenario where the test case has less steps than necessary, other test cases had some missing steps depending on the product configuration. And finally some test cases presented wrong/missing data values trying to simulate the scenario where the test case fails to specify some data value.

Next, GS-F1 and GS-F2 were manually adjusted to attend P1 and P2 specificities, generating the suites SP1-F1, SP2-F1, SP1-F2 and SP2-F2. Each test suite comprises 6 test cases. We chose this scope because the subjects would have two hours to execute two test suites, one for each product, giving a total of 12 test cases in two hours. We learned from earlier experiment executions that this was a reasonable amount of tests to execute in this interval of time.

To illustrate the test suites differences, Table 4 describes a step of the generic scenario that asks the tester to check whether the options for generating reports appear correctly. However, not all products contain these two formats (PDF and Bibtex) so these values are wrong for a set of products with this configuration. We adjusted this step to specify the behaviour of a product configured to support only PDF reports as described in Table 5.

Table 4: Generic test case.

User Action	**System Response**
Verify the options for report generation format	The options (PDF, Bibtex) are available.

Table 5: Specific test case.

User Action	System Response
Verify the options for report generation format	The option (PDF) is available.

In order to measure the effort to execute the different test suites, we developed an application called TestWatcher, which collects test time execution (in seconds), reported CRs ids, some possible observation about the execution and the test case result (passed or failed). TestWatcher recorded all this information in a spreadsheet.

A total of 20 subjects engaged in the first experiment. They were all Computer Science graduate (PhD or MSc) students from the Federal University of Pernambuco, Brazil. They had different levels of knowledge in software testing. The participants were randomly assigned in pairs to form 10 replicas of the Latin square. To form the first Latin square replica, the features (*Publications* and *Research Lines*) were randomly assigned to Feature 1 and Feature 2 and then, finally, the treatments were randomly assigned within the first square replica. Then we replicated the first square configuration to the 9 other replicas.

Likewise, in the second experiment we had the participation of 22 undergraduate Computer Science students from the University of Brasília. This time, to form the replicas, we randomly paired the subjects, the features and the treatments to for each one of the 11 replicas.

3.2.3 Metrics Collection

Using the TestWatcher, we simulate a test environment with some simplifications. Remembering Section 2, when the tester puts the test case on hold to investigate whether there is a defect on the product or just a test issue, he/she carries out different tasks depending on the environment structure. To consider and simulate all those situations we worked on the following approach in both experiments: while executing tests, whenever the tester had to investigate if there was a defect on the product, he/she paused test execution, using the TestWatcher, and asked the experiment conductor, who knew previously the issues present on the test suites. If there was a test inaccuracy, the conductor instructed the tester to ignore the issue resuming test execution and writing an observation on the TestWatcher, explaining what was wrong with the test, for example, if a step did not apply for the product under test. Otherwise, if there was a product defect, he would tell the tester to create a CR. After that, the conductor would take note of the investigation interruption. By the end of the experiment, the conductor would have a report on how many times testers interrupted execution because of test inaccuracies.

The purpose of having the number of interruptions is that we could run a first analysis considering just the execution time, without the investigation time. If our analysis indicated that there was a significant difference on the test execution effort just by measuring the execution time, this means that, in a industrial scenario, considering all the interruptions for investigation, the effort to execute generic test cases would be even greater. On the other hand, if the average execution time for both techniques did not differ significantly for both techniques we could then analyse different scenarios by adding time intervals for each noted interruption; as explained before, the interruption time can widely vary depending on how the tester will act to investigate the problem.

As for the reported CRs, every tester received a text document with a template to report defects. In the first experiment we asked the subjects to pause the time while reporting CRs and resume it to return to the test cases execution. As a result, we had the time for executing test cases and the number of reported CRs, but we did not have any evidence about the impact that reporting invalid CRs had on the time metric. Nevertheless, in a industrial test execution environment, when the tester reports an invalid CR due to test inaccuracies there is an associated waste of time, but we did not have evidence to analyse that. This motivated us to improve our time metric collection procedure to include time for reporting CRs and check what would happen in practice. This time, when the subject had to report a CR he/she did not need to pause time.

3.3 Experiments Operation

Each experiment lasted three days, each day with a two hours session. We divided day 1 session in two phases. The first phase had the purpose of giving some training about black-box testing, giving a demonstration of how subjects should proceed while executing the test cases using the TestWatcher and filling out the CR template. Since exploratory testing is not the focus of this paper, the main concern was to instruct the students to follow the test script correctly, otherwise they might be tempted to explore the tool trying different flows from the ones described in the script.

The second phase of day 1 was a dry run using the RGMS. We asked the subjects to download and install the test environment on their computers and execute a test suite with three test cases, collecting metrics and asking questions. The conductors monitored the whole process. After finishing these activities, participants sent their results (the spreadsheet generated by the TestWatcher and the CRs reported) to the conductors email. We did not use this data to run any analysis because there would be a lot of interruptions caused by participants doubts on how to proceed with the tasks.

On day 2 and day 3 we ran the Latin square first and second columns respectively, following the layout of Table 3. During this process, the subjects were not aware of the differences between the techniques neither which technique they were working with. We made this decision because instructing subjects about the two techniques could cause *bias* since they could infer that one technique was better than the other. Like on the dry run activity, the participants sent by email the results achieved. In the first experiment one subject missed day 1 activity, so we decided to exclude this results since this subject did not attend the training. Because of that, we analysed in the first experiment the results of 18 subjects, completing 9 Latin square replicas.

In the second experiment, from the 22 subjects who initiated the activities, 2 missed some of the activities and 7 had some problems to complete the activities. Some of them executed exploratory tests, spending time exploring the tool with steps and input values that were not specified in the tests or had trouble to report time. For instance, one subject managed to complete one test case in 1 second which is highly unlikely. For this reason, we decided to exclude these results and ran our analysis using the result of 10 remaining students, completing 5 Latin square replicas. Next we present our results.

3.4 Experiments Results

This section describes the results achieved by our data analysis. First we present the time execution results and then the number of reported invalid CRs.

3.4.1 Time Analysis

In order to interpret data, we first carried out a descriptive analysis to observe data tendency based on some characteristics such as dispersion and median. The box-plots [Jedlitschka and Pfahl, 2005] in Figure 3 compares execution time in both techniques for the two experiments. The first box-plot shows that the execution time tends to smaller values on the ST compared to the GT. The average to fulfil the activities using the GT was 975s while the average for the ST was 824s. The ST had an average decrease of 15.5% in time execution. Also we could not detect any outliers in our data.

In the second box-blot we have similar results. We observe that the ST tends to present smaller values than the GT. Also we can see that the GT observations seem to be more dispersed than the ST values. The mean time for completing the activities using the GT was 1251s while the mean time for the ST was 1061s, providing a decrease in average time of 15.2%. We believe that this increase in the execution mean time compared with the execution mean time in the first experiment was due to the fact that in the second experiment we count the time to report CRs as well.

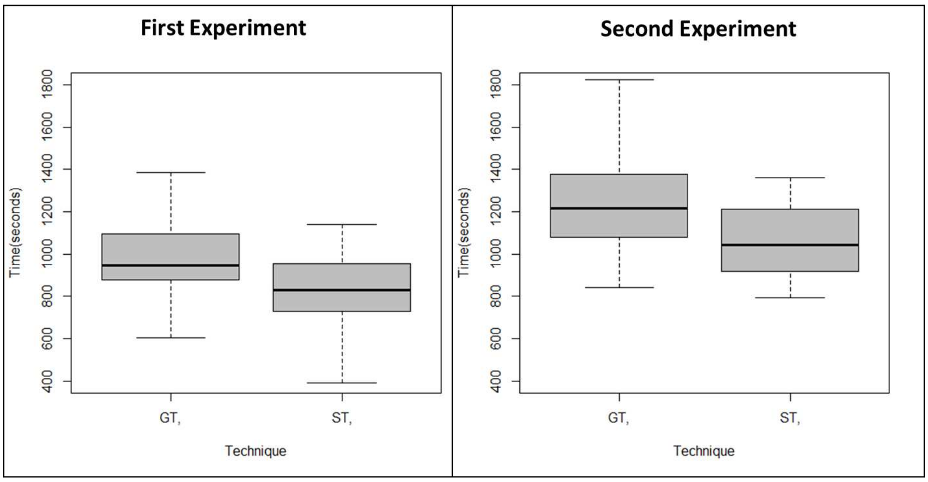

Figure 3: Box-Plot graphics comparing techniques

In addition to the median values, we compared the observations according to each subject results. Looking at the subjects individual responses in both techniques we realized that, in spite of the feature used to execute the test cases, almost the totality (94% in the first experiment and 80% in the second one) of the subjects finished the activities using ST in less time than using GT. From the 28 subjects analysed in both studies, only 3 took more time to execute the ST than the GT.

Moving on with the statistical analysis, we investigated whether the tendency observed in our samples was indeed significant by running a hypothesis test. To do that, we chose the ANOVA test [Wohlin et al., 2000] to compare the effect of both treatments on the response variable (mean time). But before running the ANOVA, we ran some tests to check whether the effect model satisfies the assumptions to run the ANOVA test. First we ran the Box Cox test to verify the constance of residuals variance, then we used the Tukey Test of Additivity to check whether our model was indeed additive and, finally, we ran the Shapiro-Wilk tests to check for residuals normality, all the results of theses tests are available on the online appendix [Accioly, 2013].

After testing the necessary assumptions, we ran the ANOVA test reaching a p-value for the technique factor of approximately 0.0001 in the first experiment and 0.01 in the second one, which gives us significant evidence that the ST reduce the time necessary to execute the experiment activities. We will discuss more general conclusions on Section 3.6.

Since we were able to gather evidence showing that the technique has influence on execution time without considering the investigation time, we did not run the second analysis adding extra time for pauses in test execution since differences would only increase but the scenario would not change. For instance, there were 19 interruptions in the first experiment caused by generic test suites inaccuracies during the experiment execution. Considering 60s as the shortest amount of time for investigation, we would have an average on approximately 1038s for the GT execution which gives a gain of 21% compared to the ST execution. Considering that, in a test execution company, these intervals can take much more than 1 minute and usually more than 2 products from an SPL are tested, this time increase would have a bigger impact on test execution. We had similar results on the second experiment.

3.4.2 Change Request Analysis

Remembering our second research question (RQ2), we investigate if the GT would increase the number of invalid CRs. This might happen because the tester would get confused interpreting a test inaccuracy as a defect and reporting a CR that, in an industrial test environment, would be invalidated. To evaluate the status of the CRs that the subjects reported, we first read all reported CR descriptions and classified them into the following categories: valid, invalid, duplicated and irreproducible.

A valid CR describes a defect on a product. Whenever the subject described the same issue with more than one CR, the second one was considered duplicated. An invalid CR represents the scenario that we investigate —the subject reported a CR that did not exist because of test case inaccuracy. Lastly, we considered three reported CRs on the first experiment to be irreproducible because the two students who reported them used their personal MacBooks during the activity and the RGMS product line did not have yet a version with full support to the OS X operational system. Consequently, for them, the GUI presented some problems such as labels that were too short for its text or fields that were too bright for reading. We did not have this problem on the second experiment.

Because we had a data set with many observations whose value was equal to zero, that is, during the test suite execution, the participant did not report any CR, we didn't run a hypothesis test. It is difficult to analyze patterns in a set of observations containing this considerable amount of zeros, so, we simply compared the total number of CRs reported in both techniques.

Table 6 presents the results of the CR analysis. As we can see, there is not a significant difference concerning valid CRs. Almost every CR that participants reported on the ST were also reported on the GT. The ST detected more valid defects in the first experiment than in the second one. We believe that one possible explanation for this is that we observed in the second experiment that some subjects were more judicious than others. For example, one of them tried to insert a research line with a very long name and then the system did not register the entity correctly. To mitigate this risk we could fix the inputs on the test cases. Another subject reported a CR because the string described in the test case did not contain capital letters and the ones presented by the product did. So the test suites should probably be revised more carefully. On the other hand, there was a significant difference in the number of invalid CRs reported on the GT compared to the ST for both experiments. We present the individual numbers on our online appendix [Accioly, 2013].

	GT	ST
Valid	15	18
Invalid	20	1

	GT	ST
Valid	13	9
Invalid	20	1

Table 6: CRs reported on the first experiment (left side table) and on the second experiment (right side table).

3.5 Interpretation of Results

Before the first experiment execution, we did not expect that, on the first analysis, without considering pauses for investigation, the technique would have such a significant influence on time. Perhaps, one of the causes is that, when subjects executing the GT met a difference between the test case and the system behaviour, they tried a workaround, looking for missing steps and pressing the return button in order to repeat some steps.

We believe that in an industrial test execution environment, this would happen too. When we observed subjects acting like that, we thought that this might cause an increase on the number of valid CRs found using the GT because the subjects who did that would go beyond the test case scenario in comparison with the ones executing the specific version of the same test case. Indeed in the second experiment there were more valid CRs reported from subjects using the GT, on the other hand, on the first experiment there were more valid CRs reported in the ST. Because of that we cannot be sure if this is a significant effect or it happened because some subjects were more judicious than other.

In general, we suspect that generic test suites may trigger a more exploratory behaviour on the subjects. There is a testing approach called exploratory, where testers have the freedom to explore different flows at the same time testing more than one feature and performing the steps back and forth. However, when executing scenario based test cases, the focus should be on the steps described.

With the second experiment execution, we now have evidence of the impact that the CRs report activity has on the time metric. The second experiment data presents an increase of the mean execution time for both techniques (around 250s for each techniques) but the average decrease from the ST to the GT remained almost the same of the first experiment —around 15% in both experiments. This was an interesting result because we expected that the ST gain would be bigger in the second experiment since the subjects had to report much more CRs in the GT. However the total number of executed tests (240) is much bigger than the number of reported CRs (43). Also, from what we have observed, the time taken to report a CR is short, perhaps half the time than the time taken to execute a test case. Because of that we believe that the CR reporting activity did not have a significant impact on our data. This means that the test execution and the interruptions for investigation are more significant to the observed improvements. Nevertheless, it is important to notice that, even if reporting invalid CRs do not impact significantly the time, it causes a negative impact on the development team productivity that needs to analyse every reported CR before concluding which ones are invalid. That is why invalid CRs are a negative metric in the test execution industry.

In addition, the second experiment helped us to add more evidence to our problem since we used subjects with less testing experience and achieved results that are consistent with the previous experiment. The ability to replicate and confirm the results of a controlled experiment is important because it reduces the chance that the results previously achieved did not represent a real trend, but instead resulted from different sources of biases such the subjects sample choice.

As we can see, based on our studies, SPL test execution can benefit from product specific test cases. These benefits are reduction on test execution time and on invalid CR rates. However, we must make some considerations on the validity of this experiment, as discussed next.

3.6 Threats to Validity

This section describes concerns of these studies and other aspects that one must take into account in order to generalize our results. To organize this section we classified our threats using the *Internal*, *External*, *Construct* and *Conclusion* categories.

In the *Internal* threats category, during both experiments execution, some students performed the activity using their personal laptops. This resulted in a heterogeneous environment during the experiment operation and the RGMS did not have versions that support different operational systems. So, when two subjects used MacBooks to execute the activity on the first experiment, they reported 3 defects, related to the RGMS interface that had labels that were too short for the text in it, that subjects who worked with Windows could not find. Because we could not reproduce those defects using Windows, we considered these 3 defects as irreproducible and did not consider them in the CR analysis. Nevertheless, we ran a second time analysis removing these two subjects results and the analysis remained practically the same. Thus, we believe that this situation did not affect significantly the time metric collection. We did not have this problem on the second experiment execution.

In the *External* threats category, some conditions limit the generalisation of our results. First, the subjects involved in these experiments were not all testers. They were Computer Science graduate and undergraduate students with different skills on software testing. Some of them worked as testers in different companies but others did not have the same experience, particularly, the undergraduate students had little or no experience in software testing. We believe that, with the execution of the second experiment, we diminished the impact of the subjects threat since we could reproduce similar results using a new group of students with different profiles.

Some studies have already addressed the question of the feasibility of conclusions drawn from results of experiments performed with students and some suggest that, for some software engineering areas, using students as subjects in experiments is often perceived as a good surrogate for using industry professionals [Buse et al., 2011, Staron, 2007]. Furthermore, Runeson compared the results achieved in an experiment using three different groups, undergraduate and graduate students and industry people. His results indicated that there was no significant difference between the three groups results [Runeson, 2003].

Besides that, from what we have observed in the executed experiments, we believe that if we had used testers to replicate these studies we would perhaps notice a decrease on the number of invalid CRs for the GT. This happens because some subjects with less experience in software testing tended to report more invalid CRs than the more experienced ones, even if they were all encouraged to investigate the defects with the conductor. However, it is noteworthy that, in the industrial contexts we had access to, manual black-box test suites are often executed by testers with less experience.

Another issue about the experiments subjects is that testers with more experience testing the same SPL will tend to have fewer problems while executing generic test suites since they are already familiar with each product specificities. However, when the SPL incorporates new features, new configurations are possible and the tester again has to take some time to get used to the new features.

The results achieved by these experiments also depend on the selected SPL. Perhaps SPLs with more variation points might benefit more in adopting the ST than SPLs with fewer variation points. The more inaccuracies, the more time is spent to investigate and execute test cases. In our studies, we injected 2 or 3 inaccuracies in each generic test suite.

Considering the *Construct* threats category, an interesting consideration to be made is that since we are collecting the time taken to report the CRs, we need to be careful to control the size and complexity of the CRs reports because more judicious subjects may take more time to report a CR with more details while others would describe the defect using only one line or two. To report CRs we provided the same template for both experiments which had 4 information fields to complete as follows: CR ID, test case ID, number of the step where the defect was found and a brief issue description. After assessing the CRs we observed that they had in average the same size, the students didn't use more than one paragraph to report CRs.

Finally, in the *Conclusion* threats, during our studies we chose to work with an academic SPL, the RGMS. Some might see this as a potential threat to our results. However, this system has been evolved for some years in the discipline of Software Reuse in the Informatics Center of the Federal University of Pernambuco, and it represents situations that are commonly seen in industrial SPLs.

Besides that, some researchers believe that empiric evaluations are not limited to industrial systems. Buse, for example, claimed in his paper about benefits and barriers in user evaluations in software engineering [Buse et al., 2011] that these artefacts can often allow researchers to easily translate research questions into successful experiments. It may reduce training time, simplify recruiting, and allows greater control over confounding factors. We believe that, by choosing RGMS, we gained all these benefits.

In the second experiment we decided to maintain the same experiment material (SPLs, features, test suites) because of our motivation to achieve reproducibility with different subjects. In the first experiment we had few subjects (18) and we wanted to add evidence to these results. We also analyse the impact of the CRs report activity and compare the numbers of reported CRs in both experiments. If we changed the subjects, the metric collection procedure and the material used it would be difficult to compare the results from both experiments. Nevertheless, in a future replication of this study we can change the material.

Another consideration to be made is, because we had a limited amount of time to apply our experiment, from the 32 possible configurations of the RGMS product line, we chose the 2 instances containing more features and that were the most different from each other considering alternative features so that we could represent all RGMS features and its different variation points. This reduced set of products cannot be considered unrealistic using the point of view of test companies. Ideally, testers would validate all possible combinations in a product line instance. Unfortunately, the space of possible combinations in a realistic product line might be enormous and exhaustive. What happens in practice is that companies follow some criteria to focus the test execution on a small subset of the products.

Lastly, as discussed in Section 3.2.2, in the first experiment, to form the Latin square replicas, we randomly assigned the subjects in pairs to form the rows of each square, then we randomly assigned *Publications* to Feature 1 and *ResearchLines* to Feature 2 to form the columns of the squares. Then, we raffled the techniques arranging them to form the first replica the same way that Table 3 describes. Lastly, we replicated this arrangement to form the other replicas. A different approach, randomly assigning the treatments for each replica, could give more solid results because it would be a full randomised configuration. Nevertheless, we believe that this consideration did not compromise our results because we had significant differences in individual results and also because we ran tests that excluded non-additivity and non-normality anomalies. In the second experiment we fixed this issue and obtained similar results.

4 Related Work

SPL Testing has been considered a challenging task [Pohl and Metzger, 2006, Tevanlinna et al., 2004, Käkölä and Dueñas, 2006, Engström and Runeson, 2011], not only due to the huge number of products that might be generated from reusable assets [Pohl and Metzger, 2006, Jaring et al., 2008], but also motivated by the lack of recommendations and best practices for testing product lines –actually, most of the research on SPL testing focuses on proposing new approaches and techniques [Engström and Runeson, 2011, Neto et al., 2011], instead of empirically assessing their benefits.

For instance, Bertolino proposed a text based use case extension tailored for product line functional testing [Bertolino and Gnesi, 2003], whereas other works detail how to derive product specific test cases from activity and sequence diagrams [Nebut et al., 2003, Kamsties et al., 2003, Olimpiew and Gomaa, 2005, Reuys et al., 2005]. Our work complements these works, since it shows evidences about the benefits of product specific test cases, which might be generated from any derivation approach.

Regarding empirical studies on SPL testing, in 2010 Neto [Neto et al., 2011] conducted a systematic mapping study with the purpose of investigating the state-of-the-art of SPL testing practices and identifying possible gaps in existing techniques. This study illustrated a number of areas in which additional investigation would be useful, specially regarding evaluation and validation research. This work also served as a basis to propose a novel process for supporting testing activities in SPL projects, the RiPLE-TE [Machado et al., 2010]. In addition, they conducted two experimental studies to evaluate the proposed process.

Ganesan compared the costs and benefits of two approaches for SPL quality assurance: one that does not consider reusable assets (and test each SPL member as an independent product), and another that considers reusable assets among different components [Ganesan et al., 2007]. Their conclusions, based on *Monte-Carlo* simulations, points that it is worth to test the reusable assets of an SPL during domain engineering (using code inspection and static analysis), and test just product specific parts during product engineering (using not only code inspection and static analysis, but also functional tests). Our study has only focused on functional testing during the application engineering level.

Denger compared the effectiveness of code inspection and functional testing to find SPL defects [Denger and Kolb, 2006]. Their findings suggest that the two techniques complement each other, finding different types of defects. Differently, our assessment concerns about the level of details in which test designers specify SPL test cases, and its consequences on both productivity and quality of defect reports.

Empirical studies on software testing are not so common as well, as discussed by Juristo, *"over half of the existing knowledge is based on impressions and perceptions and, therefore, devoid of any formal foundation"* [Juristo et al., 2004], and the lack of such an empirical body of evidence is a considerable challenge of the software testing research [Bertolino, 2007, Engström et al., 2008].

Nevertheless, empirical comparisons between testing methodologies have been recently reported. For instance, Itkonen compare the effectiveness to find defects with tests based on exploratory tests [Itkonen et al., 2007]; while Lima compares two test prioritisation techniques ($Manual \times Automatic$) also using a latin square design [Lima et al., 2009].

Finally, in our previous work [Accioly et al., 2012] we presented an empirical study that aimed to bring evidence to help decision making in such contexts. We replicated this study with different subjects and with a novel metric collection procedure to have a better notion of what happens during the task of executing tests and reporting CRs. The new results confirm the results achieved in the first experiment.

5 Conclusions

In this paper we described our first study that compared two techniques used to test SPL products (GT and ST) and extended it by presenting a second experiment done with new subjects and that also brought new insights about the impact that the different activities done during the process of testing can have on productivity. Both experiments yielded similar results. Our conclusions are that the ST can indeed improve the test execution process productivity by reducing test execution time and invalid CR rates. Also, the higher number of invalid CRs reported in the GT had no significant impact on time during the test execution phase, but it can still have a negative impact to the development team productivity.

We consider these studies as a first step towards understanding the impact of adopting product specific test suites. For future works we intend to conduct studies that evaluate these techniques using the point of view of the test design process so that we can measure the effort of designing and maintaining generic and product specific test suites.

Acknowledgments

We would like to thank our reviewers, the SPG members,[3] Cristiano Ferraz, CNPq and CAPES PROCAD Brazilian research funding agencies and INES, grants 573964/2008-4 and APQ-1037-1.03/08, for partially supporting this work.

References

[Accioly, 2013] Accioly, P. (2013). Online appendix available at `http://goo.gl/Brx3r`.

[Accioly et al., 2012] Accioly, P., Borba, P., and Bonifacio, R. (2012). Comparing two black-box testing strategies for software product lines. In *Software Components Architectures and Reuse (SBCARS), 2012 Sixth Brazilian Symposium on*, pages 1–10.

[Bertolino, 2007] Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. In *Future of Software Engineering*. IEEE.

[Bertolino and Gnesi, 2003] Bertolino, A. and Gnesi, S. (2003). Use case-based testing of product lines. In *Proceedings of the 9th European software engineering conference , Finland*. ACM.

[Bonifácio and Borba, 2009] Bonifácio, R. and Borba, P. (2009). Modeling scenario variability as crosscutting mechanisms. In *Proceedings of the 8th International Conference on Aspect-Oriented Software Development, USA*. ACM.

[Buse et al., 2011] Buse, R. P. L., Sadowski, C., and Weimer, W. (2011). Benefits and barriers of user evaluation in software engineering research. *ACM SIGPLAN Notices*.

[Denger and Kolb, 2006] Denger, C. and Kolb, R. (2006). Testing and inspecting reusable product line components: first empirical results. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, Brazil*. ACM.

[Engström and Runeson, 2011] Engström, E. and Runeson, P. (2011). Software product line testing - a systematic mapping study. *Information and Software Technology*, 53.

[Engström et al., 2008] Engström, E., Skoglund, M., and Runeson, P. (2008). Empirical evaluations of regression test selection techniques: a systematic review. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, Germany*. ACM.

[Ganesan et al., 2007] Ganesan, D., Knodel, J., Kolb, R., Haury, U., and Meier, G. (2007). Comparing costs and benefits of different test strategies for a software product line: A study from testo ag. *International Software Product Line Conference*.

[Itkonen et al., 2007] Itkonen, J., Mantyla, M. V., and Lassenius, C. (2007). Defect detection efficiency: Test case based vs. exploratory testing. In *Proceedings of First International Symposium on Empirical Software Engineering and Measurement, Spain*.

[Jaring et al., 2008] Jaring, M., Krikhaar, R. L., and Bosch, J. (2008). Modeling variability and testability interaction in software product line engineering. In *Proceedings of the Seventh International Conference on Composition-Based Software Systems, Spain*. IEEE.

[3] www.cin.ufpe.br/spg

[Jedlitschka and Pfahl, 2005] Jedlitschka, A. and Pfahl, D. (2005). Reporting guidelines for controlled experiments in software engineering. In *International Symposium on Empirical Software Engineering, Australia*.

[Juristo et al., 2004] Juristo, N., Moreno, A. M., and Vegas, S. (2004). Reviewing 25 years of testing technique experiments. *Empirical Softw. Engg.*, 9.

[Käkölä and Dueñas, 2006] Käkölä, T. and Dueñas, J. C., editors (2006). *Software Product Lines - Research Issues in Engineering and Management*. Springer.

[Kamsties et al., 2003] Kamsties, E., Pohl, K., Reis, S., and Reuys, A. (2003). Testing variabilities in use case models. In *5th International Workshop on Product Family Engineering, Italy*.

[Lima et al., 2009] Lima, L., Iyoda, J., Sampaio, A., and Aranha, E. (2009). Test case prioritization based on data reuse an experimental study. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, USA*.

[Machado et al., 2010] Machado, I. C., Neto, P. A. M. S., Santana, E., and Meira, S. R. L. (2010). RiPLE-TE: A process for testing software product lines. In *Proceedings of the 23rd International Conference on Software Engineering Knowledge Engineering, USA*.

[Nebut et al., 2003] Nebut, C., Pickin, S., Traon, Y., and Jèzèquel, J. (2003). Automated requirements-based generation of test cases for product families. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*.

[Neto et al., 2011] Neto, P. A. d. M. S., Machado, I. C., and McGregor, J. D. (2011). A systematic mapping study of software product lines testing. *Information & Software Technology*, 53(5).

[Neves et al., 2011] Neves, L., Teixeira, L., Sena, D., Alves, V., Kulezsa, U., and Borba, P. (2011). Investigating the safe evolution of software product lines. In *Proceedings of the 10th International Conference on Generative Programming and Component Engineering, USA*. ACM.

[Olimpiew and Gomaa, 2005] Olimpiew, E. M. and Gomaa, H. (2005). Model-based testing for applications derived from software product lines. In *Proceedings of the 1st international workshop on Advances in model-based testing, USA*. ACM.

[Pohl and Metzger, 2006] Pohl, K. and Metzger, A. (2006). Software product line testing. *Commun. ACM*, 49.

[Reuys et al., 2005] Reuys, A., Kamsties, E., Pohl, K., and Reis, S. (2005). Model-based system testing of software product families. In *Proceedings of the 17th International Conference Advanced Information Systems Engineering, Portugal*, volume 3520.

[Runeson, 2003] Runeson, P. (2003). Using students as experiment subjects - an analysis on graduate and freshmen student data. In *Proceedings of the 7th International Conference on Empirical Assessment in Software Engineering. Keele University, UK*.

[Staron, 2007] Staron, M. (2007). Using students as subjects in experiments–A quantitative analysis of the influence of experimentation on students' learning proces. In *CSEE&T*. IEEE Computer Society.

[Tevanlinna et al., 2004] Tevanlinna, A., Taina, J., and Kauppinen, R. (2004). Product family testing: a survey. *ACM SIGSOFT Software Engineering Notes*, 29(2).

[Wohlin et al., 2000] Wohlin, C., Runeson, P., and Höst, M. (2000). *Experimentation in Software Engineering*. Kluwer Academic Publishers.