Comparing Different Test Strategies for Software Product Lines

Paola Accioly
Informatics Center
Federal University of Pernambuco
Recife, Brazil
Email: prga@cin.ufpe.br

Paulo Borba
Informatics Center
Federal University of Pernambuco
Recife, Brazil
Email: phmb@cin.ufpe.br

Rodrigo Bonifácio
Computer Science Department
University of Brasília
Brasília, Brazil
Email: 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, several techniques for deriving product specific functional test cases have been proposed. However, this research area still lacks empirical studies showing the benefits of using such techniques. This paper presents a study that empirically compares two different black box manual test design techniques: a generic technique that we have observed in a industrial test execution environment and a product specific technique whose functional test cases could be derived using any SPL technique that considers variations in functional tests. We evaluated their impact from the point of view of the test execution process, achieving results that indicate that executing product specific test cases is faster and generates fewer errors.**

I. Introduction

Efficient testing strategies are important for achieving software quality and reliability. Since an SPL can generate a range of different products, it is challenging to write test cases based on product specific use case scenarios. This happens because of the potential large number of products and because of the variation points scattered through different SPL features scenarios.

In order to derive SPL test cases, different methodologies such as PLUTO [1] and ScenTED [2] have been proposed. These techniques present constructs that represent product variability and provide a 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 software product line testing in industry [3], [4].

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 might use test documents with use case scenarios that usually describe family behavior as a whole, describing most commonalities and abstracting the fact that some steps are optional or alternative, sometimes even omitting such steps. For example, one 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 test case to test all SPL products, even in the cases where some of them are not configured with all of these options.

However, such test suites may hamper manual execution because these abstractions, as often happens when testing is executed by an organization (or team) and development and test design are carried on by another organization, can mislead testers that have to strictly follow test scripts. This can lead to unwanted consequences, such as escaped defects —when testers don't find an error prior to product release. In addition, testers may take longer to execute test cases and report defects that don't exist, decreasing test execution productivity.

Alternatively, with the adoption of an SPL technique that manages variability in test suites, it would be possible to derive different versions of the same test suite customized for the different configurations in the product line. This way, testers wouldn't get confused during test execution process and the problems above mentioned might be solved. However, organizations cannot decide to introduce new techniques or change their usual methods based only in assumptions. We can't simply assume that specific test cases will in fact bring such benefits.

To bring evidence that might help decision making in such context, in this paper we discuss the differences of the Generic Technique (GT) that uses all variants specifications together without variability representation, and the Specific Technique (ST) that uses product customized test suites (Section II) and present an empiric evaluation (Section III) that compares the GT and the ST and evaluate the impact of the use of these two techniques from the point of view of the test execution process by measuring the effort of test cases execution. With this contribution we seek to investigate the benefits and disadvantages of both techniques. We present related work in Section IV and our final considerations in Section V.

II. Generic and Specific Test Cases

To better understand the two techniques, we describe some practical examples of how test cases may turn up to be generic (describing inaccurate family overall behavior) or product specific (showing the specific steps and data values suitable to each product). Besides presenting this difference, we also explain what are the consequences of using generic test cases in a test execution environment. As explained over Section I, we have observed the use of the GT in test teams that focus only on the test execution process of a multinational

TABLE I
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

TABLE II
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.

telecommunications company that sells mobile phones all over the world and outsources test execution activities to test centers located in different countries. To illustrate the techniques, we use a toy 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 that we will call as Blue Carrier (BC). These examples represent what we have observed in practice.

A. Test Case: User Sends MMS with Picture Attached

Our first example considers the scenario of an user sending an MMS with a picture attached as detailed in Table I. 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 the BC. This carrier specifically requires that, before any data transfer, a message pops up asking the user if he really wants to send that message– since data transfer will be further charged. Differently, Table II describes the real product behavior when the product follows BC feature requirements. On steps 7 and 8 we can see the differences from Table I.

In the GT, the test case detailed in Table I would serve to test all SPL products and the tester would be confronted with an inaccuracy while testing products configured with the BC feature. On the other hand, when using the ST, there would be two different test cases. The first one, detailed in Table I, would serve to test the products not configured with the BC feature, whereas the second one, detailed in Table II, would serve to test those products that were configured with the BC feature.

In manual black-box testing, when the test specification fails to agree with the product behavior, it probably means that the test case has revealed a defect. However, when testing a BC product, that’s 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 fit different SPL members. In this context, when the tester is not familiar with the products specificities prior to test execution, he might likely 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. He can do that by talking to a requirements analyst or reading the products specification. However, if he can’t find this evidence, he will report an invalid product defect, wasting his and other people’s time to analyze the inaccuracy.

Besides reporting invalid defects, a different type of issue might happen. Let us imagine that the product configured with the BC feature 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 the tester needs to report so that the development team can fix it. However, the tester won’t 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 requirement made by the BC. This will likely lead to an *escaped defect*, that is, defects that are found after the product release. This scenario might be worse than reporting invalid defects because it directly affects products quality, whereas reporting invalid defects only affects testing productivity.

B. Test Case: User Checks Icon and Label on Mobile Phone Main Menu

Our second example considers the test case detailed in Table III. The tester has to check the icon and the label for web navigation on the mobile main menu. Again, this scenario applies for the majority of products except the ones configured with the BC feature, which exposes the carrier brand logo in a different way. In practice, in a real test execution environment, this test case would contain a link leading to a document that describes the icons that should be displayed, but here, we simplified this specification by representing the mobile phone screen in Figure 1. This figure displays the proper behavior of the product with the BC logo and the label “Blue Web” instead of “Web”. A specific test case for a product configured with BC would have the same steps described in Table III except for the differences on the logo and the label that would expose the BC brand logo. Again, while executing the generic test case, testers confronted with this inaccuracy might waste time, report invalid defects or even let a defect escape.

TABLE III
TEST CASE: CHECK ICON AND LABEL ON MAIN MENU

Step ID	User Action	System Response
1	Go to Main Menu	Main Menu Appears
2	Check the browser navigation option	World icon appears with the title “Web”

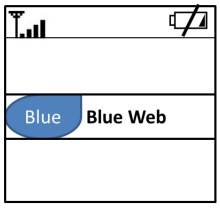

Fig. 1. Correct behavior for BC products.

TABLE IV
TEST CASE: USER ATTACHES VIDEO TO MMS.

Step ID	User Action	System Response
1	Go to Main Menu	Main Menu appears
2	Open Camera Application	Camera App opens
3	Make a 5s video	Video is correctly saved to phone memory
4	Select Options	Option Menu appears
5	Select “Send as MMS”	Dialog appears: “Video is too large to attach. Do you want to resize it?”
6	Hit “Yes”	Video is correctly resized and attached
7	Add recipient	Recipient is added
8	Select “Send Message”	Message is correctly sent

C. Test Case: User Attaches Video to MMS

Our third and last example contemplates the scenario detailed in Table IV. This time, the user makes a 5 seconds video and then try to send it with an MMS. This test case aims to test a feature that limits the MMS size so that, if the user tries to attach a file that exceeds this limit, a message appears asking if the user wants to resize the file in order to send the MMS. The inaccuracy occurs when some products are configured with a feature that gives the option to make videos using low resolution. A 5 seconds video taken with low resolution would not exceed the MMS limit size, so, steps 5 and 6 don’t apply to these products.

In this scenario, the generic test case fails to specify the interaction between the MMS limit size and the camera low resolution features. This probably happened because, when the low resolution feature was added to the SPL, the interaction between these two features was not specified correctly considering this situation. A specific scenario for products configured with both features should specify, on step 5 of Table IV, the following system response “Video is correctly attached”. Then, the next step would be to attach the message recipient. Additionally, the specific test case initial conditions should mention that the low resolution mode is activated.

In summary, generic test cases may differ from specific test cases in three different ways. First, they might present less steps than the product requires, like discussed in our first example. Alternatively, they might present different data values such as icons and labels, as we showed in our second example. Lastly, they might describe more steps than necessary for some products, like presented in our third example. 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.

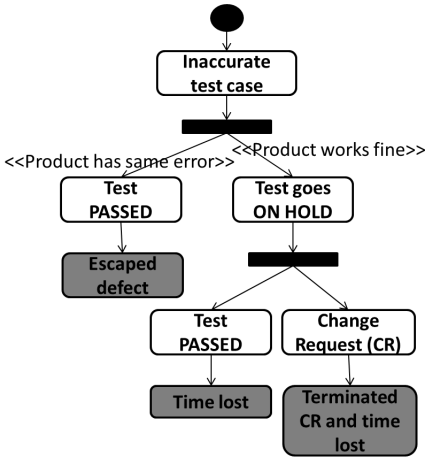

Fig. 2. Possible consequences of generic test cases.

D. 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 [5]. Only by reading test scripts, 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 behavior, testers must investigate whether this inconsistency is a defect. In this context, using generic test cases for SPL products may hamper test execution because when they fail to specify a certain product behavior, 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 2 considers the scenarios that typically happen when a test case is not accurate. The activity flow starts when the test case doesn’t correctly specify the behavior of the system under test, presenting an issue similar to those described earlier in this section. 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 won’t 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 example described in Subsection II-B, the mobile company would release the BC product without the BC brand logo on the Web Navigation option.

In the right branch, the product works fine, so the incon-

sistency is found and the tester puts the test case on hold to start investigating whether there's an issue in the test case or in the product. He 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 he finds evidence that the product works as expected by costumers, he 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 organization, this kind of investigation can take a small amount of time, 1 or 2 minutes 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 more time if, for instance, the tester needs to look throughout requirements documents to find out about the expected system behavior. Finally, the tester last resource is to contact the development team, having to wait for an answer. For instance, in the organization we mentioned before, the development team worked in a different time zone, so the questions took longer than one day to be analyzed. Meanwhile the test case remained on hold and the tester moved on with the test suite.

Either way, if the tester can't find evidence about the expected product behavior, he will assume that there is a product defect. He will create a change request (CR) and, when the development team gets to analyze 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 a terminated CR which is a negative metric indicating that the tester reported a failure that didn't exist.

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

E. Product Specific Test Cases for SPL

To derive product specific test cases for a given SPL there are different possibilities. One 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 really appropriate because the more complex the SPL gets, the harder it is to maintain each product 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 [1] and ScenTED [6]. The second alternative is to structure use cases using modularization mechanisms so that it is possible to generate specific use cases for SPL products. One existing technique for this matter is MSVCM [7]. After deriving product specific use cases, these specifications can be used as input to an automatic test suites generation tool such as TaRGeT [8].

Either way, we believe that having product specific test cases might help to solve SPL test execution problems. To evaluate this statement we propose a study to compare generic and product specific test suites 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. We believe that, 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.

Unfortunately, we can't evaluate these two processes (design and execution) in one 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 generalization of the results.

Because we can't evaluate the process of designing and executing SPL test cases in one 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. For the future we intend to extend this study to consider the test design process separately. We consider this first study 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 like the organization that we have observed.

III. Evaluation Studies

In this paper we empirically evaluate both the the GT and the ST techniques from the point of view of the SPL test execution process. Figure 3 illustrates this comparison. On the figure left side, the generic technique provides one single generic suite that testers will use to test two different product: P1 and P2. Differently, on the right side, the specific approach provides two different suites: P1 Suite and P2 Suite, each one specifying its respective product.

In order to compare these two techniques in terms of test execution, we conducted a controlled experiment where students had to test different products from the same SPL using either the GT and ST techniques while collecting the time taken to execute the test script. During this activity, students also

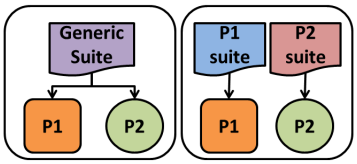

Fig. 3. Techniques.

collect reported CRs whenever they identified defects. By the end of the experiment, we analyzed and discussed the achieved results. All the experiment material is available *online*.[1] The next subsections describe the experiment definition, planning, execution, results and, finally, we consider threats to validity.

A. Experiment Definition

We have structured the experiment definition using the goal, question, metric (GQM) approach in order to collect and analyze meaningful metrics to measure the proposed process, described as follows:

Goal: analyze the test execution process, for the purpose of evaluating two different SPL test case design techniques (GT vs. ST), with respect to their efficiency regarding time to execute the test suites as well as the number of terminated CRs reported during the test execution process. Using the point of view of test engineers and software engineering researchers in the context of controlled experiments done with graduate and undergraduate students.

Research Question 1 (RQ_1): Does the ST reduce the test execution effort compared with the GT?

Metric: Test execution time

Research Question 2 (RQ_2): Does the ST reduce the number of terminated CRs compared with the GT?

Metric: Number of terminated CRs.

To evaluate the differences between the GT and the ST we chose to collect test time execution and the number of terminated 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 don't exist. We will also collect other metrics such as valid CRs —CRs that report real product defects— because we need to analyze all reported CRs in order to identify the terminated ones.

B. Experiment Planning

To evaluate the elements involved in our experiment planning, first we describe our statistical hypotheses.

1) Hypothesis: To answer RQ_1 concerning the average of time consumed to execute the test suites:

$$H_0 \quad : \quad \mu_{TimeST} = \mu_{TimeGT} \tag{1}$$

$$H_1 \quad : \quad \mu_{TimeST} \neq \mu_{TimeGT} \tag{2}$$

To answer RQ_2 concerning the number of terminated CRs reported during test execution:

[1]http://goo.gl/Brx3r

TABLE V
LAYOUT OF EXPERIMENT DESIGN.

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

$$H_0 \quad : \quad \mu_{TerminatedCRsST} = \mu_{TerminatedCRsGT} \tag{3}$$

$$H_1 \quad : \quad \mu_{TerminatedCRsST} \neq \mu_{TerminatedCRsGT} \tag{4}$$

2) Design, Instrumentation and Subjects: Bringing the elements evaluated in our context to empirical terms, the metrics evaluated are execution time and terminated CRs and the treatments 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 in the same SPL.

Since we have two treatments, two test suites and N students, we opted to use a Latin Square Design [9] to control the subject and the test suites (related to features) factors. To apply this design to our considered factors, we dispose subjects in the rows and the features in the columns of the latin square (see Table V). In this way, we systematically use the three principles of experiment design [9]. Also, in each given row and column, the treatment, the GT or the ST, appears only once. Additionally, in each treatment activity, the subjects execute 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 3. The order of the products execution is also randomized.

To evaluate the test techniques, we considered products and test suites from the Research Group Management System (RGMS) SPL. It's 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.

Figure 4 exhibits the RGMS feature model [10]. *Members*, *Publications* and *ResearchLines* features are responsible for managing these entities insertion, search and deletion from the system. Meanwhile, the *Reports* feature is responsible for publication reports in two different formats: PDF or Bibtex. Also *RLSearchbyMember* feature makes it possible to know

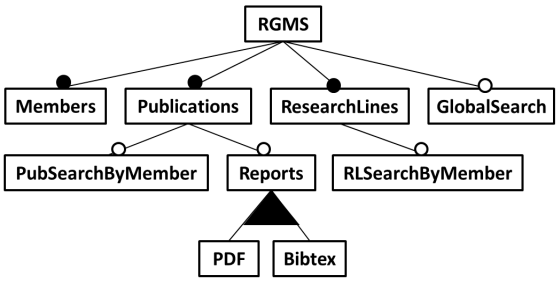

Fig. 4. RGMS feature model.

TABLE VI
GENERIC TEST CASE.

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

TABLE VII
SPECIFIC TEST CASE.

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

which research lines associates with each member registered on the system. Similarly, *PubSearchbyMember* retrieves the publications associated with the members of the research group. Lastly, *GlobalSearch* retrieves members, publications and research lines. To execute the experiment we chose two configurations as described:

P1: RGMS, Members, Publications, ResearchLines, Reports (PDF, Bibtex), ResearchLineSearchbyMember

P2: RGMS, Members, Publications, ResearchLines, Reports (PDF), PublicationSearchbyMember, GlobalSearch

The features chosen to design the test suites were *Publications* and *ResearchLines* because they were rich and contained different flows to explore and also sufficiently independent from each other generating test suites with separate flows. We didn't choose the *Members* feature because it was really 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 don't focus on studying the benefits of one test derivation technique, but on the test design techniques in general. In total, there were 6 test suites. First, we produced the generic test suites for F1 and F2 (GS-F1 and GS-F2) trying to simulate the issues discussed over Section II. Some test cases presented steps that didn't apply for both products under test to simulate the scenario where the test case has more steps than necessary. Other test cases had some missing steps depending on the product configuration to simulate the scenario when the test case has less steps than necessary. And finally some test cases presented wrong/missing data values trying to simulate the scenario when 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 quantity because the participants 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 exemplify the test suites differences, Table VI describes one 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 one of the configurations. We adjusted this step to specify the behavior of a product configured to support only PDF reports generation as described in Table VII.

In order to measure the effort to execute the different test suites, we developed an application called TestWatcher to collect 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.

In total, 20 persons engaged in the experiment. They were all computer science graduate (PhD or MSc) students with 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 inside the first square replica. Then we replicated the first square configuration to the 9 other replicas.

3) Metrics Collection: Using the TestWatcher, we simulate a test environment with some simplifications. Remembering Section II, 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 does different tasks depending on the environment structure. To consider and simulate all those situations we worked on the following approach to simulate the investigation time: whenever the tester had to investigate if there was a defect on the product, he 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 continuing with the test execution and writing an observation on the TestWatcher, explaining what was wrong with the test, for example, if a step didn't 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 real 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 didn't differ significantly for both techniques we could then analyze different scenarios by adding time intervals for each noted interruption.

As for the reported CRs, every tester received a text document with a template to report defects. Whenever the tester found a defect, he had to pause test execution, since defect reporting time was not relevant for our purposes, fill out the template, report the CR id on TestWatcher, and then fail the test case.

C. Experiment Operation

We executed the experiment in three days, each day with a two hour session. The experiment took place in a computer laboratory with one experiment conductor. We divided day 1 session in two phases. The first phase had the purpose of giving some training about manual black-box testing, giving a demonstration to explain 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. 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 conductor monitored the whole process. After finishing these activities, participants sent their results (the spreadsheet generated by the TestWatcher and the CRs reported) to the conductor email. We didn't 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. We used this data to analyze if the metrics were properly collected.

On day 2 and day 3 we ran the latin square first and second columns, respectively following the layout of Table V. During this process, the subjects weren't 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 impacting on the activity. Like on the dry run activity, participants sent to the conductor email the results achieved in each round. Because one student missed day 1 activity we decided to exclude her results since she didn't attend the training. So, in total, we analyzed the results of 18 students, completing 9 latin square replicas.

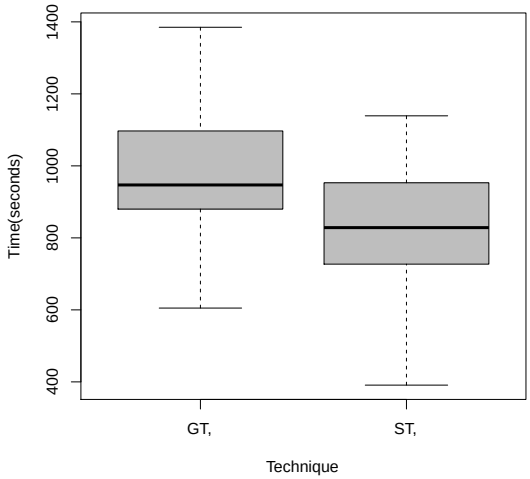

Fig. 5. Box-Plot graphic comparing techniques

D. Experiment Results

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

1) Time Analysis: In order to interpret data, we first carried on a descriptive analysis to observe data tendency based on some characteristics such as dispersion and median. We plotted the box-plot [11] illustrated in Figure 5 comparing execution time in both techniques. It shows that the execution time tends to smaller values on the ST compared to the GT. The average to fulfill the activities using the GT was 975s while the average for the ST was 824s. The ST had a average decrease of 15% in time execution. Also no outliers appeared in the graphic.

In addition to the median values, we wanted to compare the observations according to each subject results. Looking at the subjects individual responses in both techniques we saw that, no matter the feature used to execute the test cases, almost the totality (94%) of the students finished the ST in less time than the GT. From the 18 students analyzed only one took more time to execute the ST than the GT. We weren't able to find out if something went wrong to that particular subject.

Moving on with the statistical analysis, we wanted to see if the tendency observed in our samples was indeed significant by running a hypothesis test. To do that, we chose the ANOVA test [12] to compare the effect of both treatments on the response variable. But before running the ANOVA, we ran the Box Cox test to check if our data was normally distributed and then we used the Tukey Test of Additivity to check whether our linear model was additive [9]. After ensuring these properties, we ran the ANOVA test reaching a p-value for the technique factor of approximately 0.0001 which gives us significant evidence that the specific test suites can reduce time for execution activities.

Since we were able to gather evidence showing that the

technique has influence on execution time without considering the investigation time, we didn't run the second analysis adding extra time for pauses in test execution since differences would only increase but the scenario wouldn't change. In total there were 19 interruptions 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 an gain of 21% compared to the ST execution.

2) Change Request Analysis: Remembering our second research question (RQ2), we wanted to investigate if the GT would increase the number of terminated CRs. This might happen because the tester would get confused interpreting a test inaccuracy as a defect and reporting a CR that, in a real test environment, would be terminated. To evaluate the status of the CRs that participants 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 real defect on a product. Whenever the participant described the same issue with more than one CR, the second one was considered duplicated. An invalid CR represents the scenario that we're investigating —the subject reported a CR that didn't exist because of test case inaccuracy. Lastly, we considered three reported CRs to be irreproducible because the two students who reported them used their personal MacBooks during the activity and RGMS product line doesn't have yet a version with full support to the OS X operational system. Consequently, for them, the interface presented some problems such as labels that were too short for its text or fields that were too bright for reading.

This time we didn't run a hypothesis test because there were many observations with values equals to zero, that is, during the test suite execution, the participant didn't report any CR. 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 VIII describes the results of the CR analysis. As we can see, there isn't 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 faults, but not in a significant way. On the other hand, there was a high number of invalid CRs on the GT compared to the ST. Coincidentally the only invalid CR reported on the ST was reported by the same student that spent more time on the ST than on the GT.

TABLE VIII
REPORTED CRS.

	GT	ST
Valid	15	18
Invalid	20	1

E. Interpretation of Results

Before the experiment execution, we didn't 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 that we have observed is that, when participants executing the GT met a difference between the test case and the system behavior, they tried to make a work around, looking for missing steps and pressing the return button in order to repeat some steps. When we observed participants acting like that, we thought that maybe this might cause an increase on the number of valid CRs found using the GT because the students who did that would go beyond the test case scenario in comparison with the ones executing the specific version of the same test case. Nevertheless this wasn't observed in the CR.

We suspect that generic test suites may trigger a more exploratory behavior on the subjects. In Software Testing there is an approach called exploratory testing, 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 script based test cases, the focus should be on the steps described.

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

F. Threats to Validity

This section describes concerns that must be improved for future replications of this study 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 [12]. However, we could not identify any threats considering the *Construct* validity.

1) Internal: During the experiment execution, some students performed the activity using their personal laptops. This resulted in a heterogeneous environment during the experiment operation. However, RGMS doesn't have yet versions that support different Operational Systems (OS). So, when two students used MacBooks to execute the activity, they reported 3 defects, related to RGMS interface that had labels that were too short for the text in it, that students who worked with Windows OS could not find. Because we could not reproduce those defects using Windows, we considered these 3 defects as irreproducible and didn't consider them in the CR analysis. Nevertheless, we ran a second time analysis removing these two students results and the results remained practically the same. Thus, we believe that this situation didn't affect significantly the time metric collection.

2) External: With respect to the external validity, some conditions limit the generalization of our results. First, the subjects involved in this experiment weren't all testers. They were computer science graduate students with different skills on software testing. Some of them really worked as testers in different companies but others didn't have the same experience.

Some studies have already addressed the question of the feasibility of conclusions drawn from results of experiments performed with students and some suggests that, for some software engineering areas, using students as subjects in experiments is often perceived as a good surrogate for using industry professionals [13], [14]. Furthermore, Runeson compared the results achieved in one experiment using three different groups: ungraduate students, graduate students and industry people. His results indicated that there was no significant difference between the three groups results [15].

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

Another matter about the experiment 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 this experiment 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 study, we injected 2 or 3 inaccuracies in each generic test suite.

3) Conclusion: During this study we chose to work with a non-real SPL, the RGMS. Some might see the use of a non-real SPL 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 real SPLs.

Besides that, some researchers believe that empiric evaluations are not limited to real projects. Buse, for example, claimed in his paper about benefits and barriers in user evaluations in software engineering [13] that non-real artifacts can often allow researchers to more 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.

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 is likely to be enormous and exhaustive. What happens in practice is that companies follows some criteria to focus the test execution on a small subset of the products.

Lastly, as discussed over Section III-B2, 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 V describes. Lastly, we replicated the techniques 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 randomized configuration. Nevertheless, we believe that this consideration didn't compromise our results because we had really significant differences in individual results and also because we ran tests that excluded non-additivity and non-normality anomalies.

IV. Related Work

SPL Testing has been considered a challenging task [2], [3], [16], [4], not only due to the huge number of products that might be generated from reusable assets [2], [17], but also motivated by the lack of well known recommendations and best practices for testing product lines— actually, most of the research on SPL testing focuses on proposing new approaches and techniques [4], [18], instead of empirically assessing their benefits.

For instance, Bertolino and Guinesi proposed a text based use case extension tailored for product line functional testing [1], whereas other works detail how to derive product specific test cases from activity and sequence diagrams [19], [20], [21], [6]. Our investigation 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 Ivan et al. [18] 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 [22]. In addition, they conducted two experimental studies to evaluate the proposed process.

Ganesan et al. 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 [23]. 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 and Kolb compared the effectiveness of code inspection and functional testing to find SPL defects [24]. 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 is not so common as well, as discussed by Juristo et al., *"over half of the existing knowledge is based on impressions and perceptions and, therefore, devoid of any formal foundation"* [25], and the lack of such an empirical body of evidence is a considerable challenge of the software testing research [26], [27].

Nevertheless, empirical comparisons between testing methodologies have been recently reported. For instance, Itkonen et al. compare the effectiveness to find defects with tests based on exploratory tests [28]; while Lima et al., compare two test prioritization techniques ($Manual \times Automatic$) also using a latin square design [29].

V. Conclusions

In this paper we described the differences between two techniques used to test SPL products (GT and ST) and discussed some problems observed in a real test execution environment that aroused from the use of the GT that might be mitigated with the use of the ST. In order to analyze this statement, we conducted a controlled experiment with students simulating a test execution environment. While executing test suites, they collected time and reported CRs. This data was later used for analysis and we concluded that the ST can indeed improve the test execution process productivity by reducing test execution time and terminated CR rates.

We consider this study 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.

Acknowledgment

We would like to thank CAPES and INES for partially supporting this work.

References

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

[2] K. Pohl and A. Metzger, "Software product line testing," *Commun. ACM*, vol. 49, 2006.

[3] A. Tevanlinna, J. Taina, and R. Kauppinen, "Product family testing: a survey," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 2, 2004.

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

[5] B. Beizer, *Black-box testing - techniques for functional testing of software and systems*. Wiley, 1995.

[6] A. Reuys *et al.*, "Model-based system testing of software product families," in *Proceedings of the 17th International Conference Advanced Information Systems Engineering, Portugal*, vol. 3520, 2005.

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

[8] L. Neves *et al.*, "Investigating the safe evolution of software product lines," in *Proceedings of the 10th International Conference on Generative Programming and Component Engineering, USA*. ACM, 2011.

[9] G. E. P. Box *et al.*, *Statistics for experimenters : design, innovation, and discovery*. Wiley-Interscience, 2005.

[10] K. C. Kang *et al.*, "Feature-oriented domain analysis feasibility study," *Software Engineering Institute, USA*, 1990.

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

[12] C. Wohlin *et al.*, *Experimentation in Software Engineering*. Kluwer Academic Publishers, 2000.

[13] R. P. L. Buse, C. Sadowski, and W. Weimer, "Benefits and barriers of user evaluation in software engineering research," *ACM SIGPLAN Notices*, vol. 46, 2011.

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

[15] P. Runeson, "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*, 2003, pp. 95–102.

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

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

[18] P. A. da M. S. Neto *et al.*, "A systematic mapping study of software product lines testing," *Information & Software Technology*, vol. 53, no. 5, 2011.

[19] C. Nebut *et al.*, "Automated requirements-based generation of test cases for product families," in *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, 2003.

[20] E. Kamsties *et al.*, "Testing variabilities in use case models," in *5th International Workshop on Product Family Engineering, Italy*, 2003.

[21] E. M. Olimpiew and H. Gomaa, "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, 2005.

[22] I. C. Machado *et al.*, "RiPLE-TE: A process for testing software product lines," in *Proceedings of the 23rd International Conference on Software Engineering Knowledge Engineering, USA*, 2010.

[23] D. Ganesan *et al.*, "Comparing costs and benefits of different test strategies for a software product line: A study from testo ag," *International Software Product Line Conference*, 2007.

[24] C. Denger and R. Kolb, "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, 2006.

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

[26] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Future of Software Engineering*. IEEE, 2007.

[27] E. Engström, M. Skoglund, and P. Runeson, "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, 2008.

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

[29] L. Lima *et al.*, "Test case prioritization based on data reuse an experimental study," in *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, USA*, 2009.