An Experimental Study to Evaluate a SPL Architecture Regression Testing Approach

Paulo Anselmo da Mota Silveira Neto^{1,3}, Ivan do Carmo Machado^{2,3}, Yguarata Cerqueira Cavalcanti^{1,3}
Eduardo Santana de Almeida^{2,3}, Vinicius Cardoso Garcia^{1,3}, Silvio Romero de Lemos Meira^{1,3}

¹Federal University of Pernambuco (UFPE) – Recife, Pernambuco, Brazil

²Federal University of Bahia (UFBA) – Salvador, Bahia, Brazil

³Reuse in Software Engineering (RiSE)

{pamsn, vcg, ycc, srlm}@cin.ufpe.br {ivanmachado, esa}@dcc.ufba.br

Abstract—In Software Product Line Engineering, where products are derived from a common platform, the reference architecture should be considered the main asset. In order to maintain its correctness and reliability after modifications, a regression testing approach based on architecture specification and code was developed. In this paper, we evaluate it in two different scenarios, the corrective scenario, in which it was performed after a corrective change in the code, and in the progressive scenario, executed after a evolution or enhancement, when some specification changes. The results showed that the progressive scenario was less costly than the corrective. Our evaluation also highlights the importance of a code-based technique to select a set of efficient and effective test cases.

Keywords—Software Testing; Software Product Lines; Software Engineering; Regression Testing;

I. INTRODUCTION

Software architectures are becoming the central part during the development of quality systems [1], being the first model and base to guide the implementation [2] providing a promising way to deal with large and intensive systems [3]. Moreover, it evolves over time to meet customer needs, environment changes, improvements or corrective modifications. To be confident that these modifications are in conformance with the architecture specification, do not introduce unexpected errors, and the new features work as expected, regression tests are performed [4].

There are some approaches [2], tools [5] and methodologies in the literature, delving into the issues on how to perform regression testing on software architectures, and the most suitable ways to compare two different code versions. From an industry point of view, with the growing Software Product Line Engineering (SPL) adoption, more efficient and effective testing methods and techniques are needed [6], since the available ones make testing a very difficult, expensive and challenging process [7]. More specifically, industry seeks effective SPL regression testing techniques to reduce the amount of retesting [8].

Even with the importance given to regression testing techniques, there are few reports and empirical studies available in the SPL area, and in general, they have been

conducted as informal case studies without sufficient attempt to scientific rigor from empirical research methods [6]. Thus, additional experimental studies are necessary to provide more evidences about the use of regression testing techniques for SPL development.

In this context, this study describes the evaluation of a regression testing approach [9] aimed to reduce the testing effort, by reusing test cases, execution results, as well as to select and prioritize a more effective set of test cases, in terms of performance improvements for fault detection. Taking advantage of SPL architectures similarities, this approach can be applied among product architectures and between the reference and product architectures.

II. A REGRESSION TESTING APPROACH FOR SPL ARCHITECTURES

According to McGregor [10], regression testing is a technique rather than a testing level. Considering this point of view, regression testing can be performed after any testing level. In our context, it was performed after integration testing, since the purpose of the approach is to verify the integration among modules and components, which compose the SPL architectures.

It aims to check if new defects were introduced into previous tested architecture and if it continues working properly. To be confident that the architecture is working properly, its specification can be used as a test oracle to identify when the tests pass or fail.

The required inputs for the SPL regression testing approach [9] are both SPL architecture versions, the previous executed integration tests and scripts. The *architectural specifications and views* (structural and behavioral), and the *feature model*, *product map* and *feature dependency diagram* can be useful assets from which architecture-related information can be extracted, serving as a guide to identify elements that need to be retested. By using the *architecture structural view*, the relationship between classes and components are easily specified and identified. This is the process of building the integration tests to be used in the regression approach. The *feature model* and *feature dependency diagram* are

Figure 1. The Regression Testing Approach.

used to provide information about the relationship among features, showing their dependencies and hierarchy.

Specific product architectures can be instantiated based on *product maps*, which describe the instance behavior in terms of *mandatory*, *optional* and *variant* features for every product. Test architects use this information to instantiate an architecture by selecting specific features and components. When components or modules are modified, regression testing should be performed on the application architecture, as a means of assessing its compliance [11].

A. Approach Steps

The approach workflow can be viewed in Figure 1, where it is detailed the activities, inputs, outputs, tasks and roles [9]. Although they are presented as sequentially initiated, this process represents an incremental and iterative development step, since feedback connections enable refinements. We following describe every aspect of the approach.

Planning. Performed as a means to guide the test cycle execution. In this phase, the *Test Plan* is created by gathering information about the goals, schedule, adequacy criteria, the coverage measure, resources and associated risks.

Analysis. Carried out in order to understand how corrections and evolutions impact the architecture. By manually analyzing its specification and the modified class, methods are identified, and the relevant tests can be designed or selected. It may support the next activities, reducing the coverage of the modified version that should be examined.

Test Selection and Design. It aims to design and select test cases to be further executed against the new version of the architecture. This activity is composed of the following steps:

- **Graph Generation:** After performing the analysis, the test architect may need to generate graphs to catch code behaviors. Thus, a graph representation for both versions of modified portions of the architecture (the new and old versions) are generated. It can be supported by tools, e.g. JDiff [5].
- **Graph Comparison:** In order to identify critical edges and understand how the code has changed, the graphs are then compared. A good knowledge in control flow graphs analysis is required during this step, since the test architect will easier check the code behavior.
- **Specification Comparison:** When performing progressive regression testing, the last two steps (graph generation and comparison) are replaced by specification

comparison. This step aims to compare the original specification with the modified one, identifying added, deleted or changed components, classes, and features.

- **Test Selection:** After the graphs, specification or textual (code) comparison, the critical edges and paths are analyzed aiming to find test cases that exercise a modified portion of the architecture. In this step, the paths should be analyzed to classify the previous designed tests (integration tests) from the repository.
- **Instrumentation:** To assess the test cases efficiency and coverage, the identified paths and new code can be instrumented. Hence, *test designers* will be confident that the tests indeed exercise the desired paths/code. If the selected test case did not cover the required path, a new test case is to be designed.
- **Test Suite Composition:** After test case selection and design, a test suite is assembled. The *test designer* creates test suites grouping tests based on different information. The test suite will be used to build the regression test cycle to be further executed.
- **Test Case Prioritization:** Aims to order the test cases and scripts from the test suite, executing tests with highest priority, for example, based on criticality and complexity criteria, earlier than lower priority test cases. Prioritization techniques may be used, some of them take advantage of some information about previous executed test cases to order the test suite [12].

Execution. Test suites are executed against the modified version in a regression testing cycle. Thus, the *test engineer* exercises the architecture, executing the previous selected test cases and scripts. In case where some problem is found, he should search in the repository for a change request (CR) that reports the problem. When no CRs are found that can be associated to such problem, a new registry is created. The execution results and the new and associated CRs are recorded and an investigation starts in order to precisely identify which components, modules, versions and modifications caused the failure.

Reporting. All these information is assembled to compose the *Test Report*. This report is important for the *test manager*, to organize and schedule activities and serve as basis for next test plans and regression cycles.

In this evaluation, the subjects were asked to apply the approach after some modifications due to a corrective and enhancement performed in the architecture.

III. EVALUATION

This section describes the evaluation of the proposed approach, comprising three subsections, namely *Planning* section, which describes the plan used to perform the evaluation and further analyze the results; *Execution* section, describing what has been done to prepare the execution of the evaluation as well as its execution, and finally the *Analysis* section, which details the analysis of the gathered data. These represent the sequence of tasks to be performed in this evaluation.

This evaluation is characterized by four dimensions: (i) It is **off-line** since it was performed out of the semester classes; (ii) The **M.Sc. Students** are subjects using the approach; (iii) The evaluation addresses **an academic project** (without commercial pressure), in which it considers the approach using test cases for the integration of classes, rather considering components and modules; and (iv) This investigation is focused on a **specific issue**, namely when the code is modified due to a corrective or evolutionary action. Furthermore, this evaluation is a **multi-test within object study**, since it examines one *object* (Regression testing approach) and more than one *subject*.

A. Planning

Goal. The goal of this evaluation was to analyze the regression testing approach for the purpose of evaluation with respect to understandability, usability, completeness, applicability and effectiveness from the point of view of SPL researchers and test engineers in the context of a SPL project. To achieve the goal previously stated, quantitative and qualitative questions were defined and described as follows.

Effort: Q1. *How much effort does it take to apply each step defined in the approach?*

Usability and Understandability: Q2. *Do the subjects have difficulties to understand/apply the approach?*

Completeness: Q3. *Is there any missing activity, roles or artifact?*

Effectiveness: Q4. *How many defects were detected using the approach?* Q5. *How many tests were correctly classified (Re-testable, Reusable, Obsolete and Unclassified).*

After defining the set of questions, these needed to be mapped to a measurement value, in order to characterize and manipulate the attributes in a formal way. The metrics are quantitative ways to answer the questions, and they are detailed in Table I.

Selection of Participants. This evaluation involved eight participants. All of them had completed a post-graduate course in the software testing area prior to this evaluation. The subjects were either upper-level computer science majors or graduate students. They were selected by *convenience sampling*, which means that the nearest and most convenient persons were selected as participants [13]. Participants were

informed that we would like to investigate the outcome of the approach execution. However, we did not inform them about what aspects we intended to study, i.e. they were not aware of the hypotheses stated.

Evaluation Material. In this study, the Regression testing approach documentation was available to the participants so that they could perform the requested activities, strictly following the defined steps. Required support tools were also available to them. All participants received a training on the proposed approach. The training consisted of two sessions, respectively comprising the subjects: (i) concepts regarding SPL, variability and software testing; (ii) the proposed regression testing approach flows, activities, steps and support tools.

Training. The subjects were trained in several aspects of SPL, and control flow graphs, besides the use of the following tools: Junit(<http://www.junit.org/>), Eclemma plugin(<http://www.eclemma.org/>), and JDiff tool [5]. The analyzed approach was also another training topic. Next, they performed the regression testing approach in the code provided. Most of students had previous experience in industrial projects. However, the subjects had low or none industrial experience in reuse activities, such as component development and SPL engineering. On the other hand, all of the subjects were members of the RiSE Labs, and their research area involve these aspects, which gives them theoretical knowledge. Regarding software testing, all of them had a post-graduation on it, and medium industrial experiences. Despite the experience reported regarding regression testing, they have no or low experience in control flow graph analysis and most of them had never used a test selection technique beforehand.

After that, the training on the regression testing approach took place, where the corrective and progressive scenarios were presented. Next, all subjects received the documentation and artifacts required to execute the evaluation. The corrective scenario was applied, followed by the progressive scenario. At the end, a feedback questionnaire was applied in order to improve the interpretation and to get rid of any inconsistency found in the reported artifacts and documentation.

The basis for the statistical analysis of an experiment is the hypothesis testing. If the hypothesis can be rejected then conclusions can be drawn based on the hypothesis testing under given risks [14]. Hence, five different hypotheses were formally defined to address each of previous established metrics (see Table I).

Null Hypothesis. It determines that there is no benefit of using the proposed approach, i.e. there is no difference in terms of effectiveness and effort when using the approach. The null hypothesis is specified as follows: $H0_1 : \mu_{EAA} \geq 20\%$, $H0_2 : \mu_{AUAD} \geq 40\%$, $H0_3 : \mu_{ARAM} > 3$; $H1_1 : \mu_{ND} \leq 20\%$; $H2_1 : \mu_{NTCC} \leq 40\%$.

Alternative Hypothesis. It determines that the proposed

Metrics	Description
M1 - Effort to Apply the Approach (EAA). Related to Question Q1, this metric measures the amount of time spent in order to understand and follow the Regression Testing approach and produce the artifacts proposed.	$EAA_{step} = \frac{TotalTimeSpentApplyingEachStep}{TotalTimeSpentInTheApproach}$
M2 - Approach Understanding and Application Difficulties (AUAD). Related to Question Q2, this metric aims to identify possible misunderstandings in the approach usage. It is necessary to identify and analyze the difficulties found by users when applying the approach.	AUAD = Number of subjects with difficulties raised during the approach learn and application.
M3 - Activities, Roles and Artifacts Missing (ARAM). Related to Question Q3, it intends to identify the activities, roles and artifacts considered absent from the regression testing approach in order to calibrate or even include them, depending on the analysis.	ARAM = Number of missing activity/steps/role/artifact identified during the approach execution.
M4 - Number of Defects (ND). Related to Question Q4, it intends to identify the total number of defects in a given time period/activity/step in the software.	ND = The number of seeded defects identified, during the approach execution.
M5 - Number of Tests Correctly Classified (NTCC). Related to Question Q5, it aims to identify the correct classification of the test cases used and designed during the approach execution.	NTCC = The number of tests correctly classified (Re-testable, Reusable, Obsolete and Unclassified). It is important to select the test cases that need to be executed in the new version.

Table I
EXPERIMENT METRICS.

approach can be more effective, and efficient to find faults in the program. The alternative hypothesis is specified as follows: $H_{\alpha_1} : \mu_{EAA} < 20\%$, $H_{\alpha_2} : \mu_{AUAD} < 40\%$, $H_{\alpha_3} : \mu_{ARAM} \leq 3$, $H_{\alpha_{11}} : \mu_{ND} > 20\%$, $H_{\alpha_{21}} : \mu_{NTCC} > 40\%$.

An arbitrary value was chosen to each metric, based on practical experience and common sense, since it was not found well-known values in the literature that could fit to our purpose. On the other hand, this arbitrary fashion to define and apply values may serve as a basis for new evaluations. As new evaluations will be performed, the values will be calibrated.

Design. The design type to be used in this evaluation is the *one factor with one treatment* [13], in which we want to analyze the subjects and approach performance. In this sense, the factor was the regression testing approach and the treatment is the application of the approach in both scenarios, the corrective and progressive.

B. Execution

This evaluation was performed with a set of classes simulating the project of a SPL architecture. It has two versions of a banking system, which manages accounts, saving accounts, customers and companies. The first version (V1) was developed with eighteen (18) classes and one (1) interface, and fifty-eight (58) integration test cases were used to test the conformance of the system against the specification. A second version was developed with new functionalities (as a way to simulate evolution) and a set of seven (7) faults seeded in the code. This new version (V2) is composed of twenty-four (24) classes and three (3) interfaces. These changes aimed to evaluate the regression testing approach in both scenarios during an evolution and correction scenario.

The seeded faults were inserted based on four sources: (i) McGregor's SPL fault model [15], where the most common faults found in SPL projects are summarized; (ii) based on the mapping study previously performed [6]; (iii) knowledge in the application domain; (iv) the most common Java development faults extracted from the Internet.

Firstly, both code versions, three change requests and a

set of previous designed integration test cases were provided, so the participants could validate the approach considering the correction scenario. The participants needed to apply the approach aiming to find the faults previously seeded, as well as to classify the integration test cases. They also had to apply all steps and answer the questionnaire. The steps referring to graph generation and comparison are optional in the approach, but the subjects were asked to use them at least one time. After reporting the first results, the class diagrams (from both versions) were provided to characterize an evolutionary scenario. Thus, the subjects had to evaluate the specification changes, correctly classify the existing integration tests and create new test cases. The test cases had to be designed following the same coverage criteria used in the previous designed integration test cases. Figure 2 summarizes all scenarios.

Figure 2. Evaluation Scenarios.

Before performing the evaluation, two pilot projects were conducted with the same structure defined in this planning phase. The first pilot was performed by a single subject, aiming to detect problems and calibrate the evaluation process before its real execution. An issue regarding how the approach deals with specification changes (evolution) was detected during this first pilot, and in order to solve it, a new step (Specification Comparison) was added to address this problem.

The second pilot was also performed by a single subject, who has a minimal experience in industrial projects performing test execution and design to regression, integration and exploratory testing. Problems such as code faults (not purposely seeded) and absence of new-structural test cases were detected. Modifications on both code versions (new

and old) were performed in order to solve these issues. A problem with the background questionnaire was also identified, namely some important questions used to extract the subjects profile were absent. Hence, we decided to correct it by adding a new question. During this pilot, three new threats were discovered: the code size, the provided CRs and the injected faults. These are detailed in section V.

The results of the evaluation were collected using measurement instruments. Thus, time-sheets were used to collect the time spent in each activity. Moreover, all subjects received a questionnaire to evaluate their educational background, participation in software development projects, experience in testing and reuse. In addition, the subjects received a second questionnaire for the evaluation of satisfaction and difficulties to use the proposed approach.

IV. ANALYSIS AND INTERPRETATION

Q1- Effort to Apply the Approach. This aspect was evaluated in two scenarios: corrective and progressive. While the first spent 94.78 hours to be performed, the second was executed in 56.06 hours. These numbers are concerned to the total number of worked hours of the members in each step.

Corrective Scenario. Table II shows the raw data collected after the experiment execution, where “not considered (NC)” means that the step was not reported correctly and “not reported (NR)” means that no time was reported by the subject. Before analyzing the collected data, some issues were observed: *Test Design* and *Test Selection* steps needed to be refined, subjects ID 1 and 3 were removed from the analysis since they did not correctly report the data regarding to this item. In addition, *Graph Generation*, *Graph Comparison* and *Reporting* steps were not reported completely, whereas steps *Graph Generation* and *Graph Comparison* were defined as optional by the approach and some subjects did not report it. Only the subject ID4 performed the *Reporting* step.

Steps/Subj.	ID1	ID2	ID3	ID4	ID5	ID6	ID7	ID8	Total
(a)	120	26	90	5	90	255	180	120	886
(b)	60	10	90	60	180	60	60	60	580
(c)	NR	20	120	NR	NR	140	150	25	455
(d)	NR	30	15	NR	NR	NR	60	60	165
(e)	240	73	240	20	240	90	120	60	1083
(f)	NC	55	NC	10	290	25	30	60	813
(g)	30	20	110	5	30	110	120	30	455
(h)	10	23	76	5	120	180	90	20	524
(i)	10	10	24	8	30	20	30	10	142
(j)	10	60	175	5	60	7	10	120	447
(l)	50	30	160	NR	30	60	60	90	480

Table II
APPROACH EXECUTION EFFORT (MINUTES) CONSIDERING CORRECTIVE SCENARIO.

Furthermore, we performed the outliers analysis using the approach defined in [16]. Based on this analysis, subjects ID4 and ID6 were identified as outliers for the Planning phase. However, we chose to keep all subjects identified as

(a) Effort considering all steps.

Steps	EAA (%)
Planning(a)	15.58
Analyzes(b)	10.20
Graph Generation(c)	8.00
Graph Comparison(d)	2.90
Textual Comparison(e)	19.04
Test Design and Selection(f)	8.26
Instrumentation(g)	8.00
Test Suite Composition(h)	9.21
Test Case Prioritization(i)	2.50
Test Execution(j)	7.86
Reporting(l)	8.44

(b) Effort considering some steps.

Steps	EAA (%)
Planning(a)	21.52
Analyzes(b)	14.08
Textual Comparison(e)	26.30
Instrumentation(g)	11.05
Test Suite Composition(h)	12.72
Test Case Prioritization(i)	3.44
Execution(j)	10.85

Table III
EFFORT TO APPLY THE APPROACH ON CORRECTIVE SCENARIO.

outliers and consider its times in the effort analysis, despite of the limited number of subjects.

The effort to apply the approach is shown in Tables III(a) and III(b). The first table shows the effort to apply each step, whereas the second shows only the steps that were completely and correctly reported. The time spent during the planning step can be justified by the fact that no subjects had performed it previously. Since it was their first time, they needed some time to understand the test plan and collect all information in order to fill it. Besides gathering such information, they should plan the test cycle considering the constraints and information provided by the instruments.

Regarding the textual comparison step, the subjects needed to compare both code versions, and also to understand how the change impacts on the domain application rules. They should identify portions of the code in order to discover critical paths, that will be further exercised by the created and selected test cases. Sometimes a deep analysis is necessary to understand language behaviors. In this case, the graph generation and comparison steps should be performed. Although most of the subjects complained about boredom when executing this step, they agreed with such a significance.

In this evaluation, we adopted the data presented in Table III(a), which **rejects** the null hypothesis since no steps achieved effort values higher than 20%. We chose this data set since we could better understand how all steps in the approach behaved. However, if we considered the data set presented in Table III(b), the null hypothesis could not be rejected. As we can see in Table III(b), the Planning (21.52%) and Textual Comparison (26.30%) steps exceeded the established metric.

Progressive Scenario. Table IV shows the general (raw) data collected after the experiment execution. Subjects ID1 and ID3 were excluded from this evaluation because they did

not correctly report the output of *Test Design and Selection* step. Subject ID4 was removed, since he did not performed the *Reporting* step. This information was collected during an interview, where the subjects explained their questionnaire answers, and also during the experiment data analysis.

Steps/Subj.	ID1	ID2	ID3	ID4	ID5	ID6	ID7	ID8	Total
(a)	60	23	120	2	90	90	120	120	625
(b)	30	10	60	5	150	30	30	60	375
(c1)	30	22	53	20	80	60	90	30	385
(f)	NC	65	NC	15	270	120	120	60	650
(g)	30	25	98	5	30	60	60	30	338
(h)	10	17	20	5	30	130	60	20	292
(i)	10	10	34	5	10	10	15	10	104
(j)	10	25	34	5	60	10	8	60	212
(l)	50	25	98	NR	30	60	60	60	383

Table IV
APPROACH EXECUTION EFFORT CONSIDERING PROGRESSIVE SCENARIO.

Tables V(a) and Y(b) show the effort to apply the steps. The former shows the effort to apply each step, and the latter shows the effort to apply the progressive scenario, considering only the correctly reported steps. If we do not consider the wrongly reported or incomplete steps, the null hypothesis is not rejected. The null hypotheses **is rejected** since the most costly (in time) step did not exceed 20% ($H_{01} : \mu_{EAA} \geq 20\%$).

By analyzing the datasets, we have clear indicators that the time spent in the progressive scenario was lesser than in the corrective scenario. As the subjects firstly applied the corrective scenario, when they performed the next scenario, they had some expertise in the domain and code yet. This previous acquired experience can explain the results but also, as reported by some subjects, the result can be proved by the lower amount of retestable test cases to be performed in the progressive scenario.

Q2- Approach Understanding and Application Difficulties. Analyzing subject's answers regarding the difficul-

(a) Effort considering all steps.

Steps	EAA (%)
Planning(a)	18.57
Analyzes(b)	11.14
Specification Comparison(c1)	11.44
Test Design and Selection(f)	19.32
Instrumentation(g)	10.04
Test Suite Composition(h)	8.68
Test Case Prioritization(i)	3.09
Test Execution(j)	6.30
Reporting(l)	11.38

(b) Effort considering some steps.

Steps	EAA (%)
Planning(a)	26.81
Analyzes(b)	16.08
Specification Comparison(c1)	16.51
Instrumentation(g)	14.50
Test Suite Composition(h)	12.52
Test Case Prioritization(i)	4.46
Test Execution(j)	9.09

Table V
EFFORT TO APPLY THE APPROACH ON PROGRESSIVE SCENARIO.

ties faced during the approach execution, it was identified that 62.5% subjects had some kind of difficulty to understand the approach. Because of the understanding problem, all of them also encountered problems to apply the approach.

Two subjects (ID 1 and 5) claimed that the main problem for the understandability of the approach was the number of steps and tasks, that were seemingly too many and tough, requiring a certain knowledge in both development and testing areas. Another three subjects (ID 3, 4, and 5) reported that one of the understandability issues was the lack of examples during test design and selection, more specifically to help the test classification task. The subjects (ID 3 and 6) stated that the input and output of each step were not clearly presented in the approach. Subject ID1 reported his difficulty in understanding the relation between role and tasks, which tasks each role should perform.

The null hypothesis related to the percentage of subjects with any kind of difficulty in the process defines a percentage of more than 40% ($H_{02} : \mu_{AUAD} \geq 40\%$). Since we had 62.5% of the subjects with at least one difficulty, this null hypothesis was **not rejected**. However, in the same way as the previous hypothesis, this value for the null hypothesis was defined without any previous data.

Correlation Analysis. Based on the collected subjects profile, there is no correlation among the characteristics of subjects profile and the difficulties to understand the approach. Although subjects (ID 3, 4, and 5), which had no experience in applying test selection techniques, reported the absence of examples to help the test classification, other subjects with no experience did not pointed this problem.

Q3- Activities, Roles and Artifacts Missing. By analyzing the data, we noticed that no subjects identified any missing activities, roles, artifacts or steps. Since we had 0 (zero) identified as missing, the ($H_{03} : \mu_{ARAM} < 3$) null hypothesis was **rejected**.

Correlation Analysis. Based on collected subject profile, all subjects have at least 1 year of experience in software testing, they also have some kind of testing course and some of them have been worked with regression testing. It can serve as a clue to indicate that the approach is complete and well structured.

Q4- Number of Defects. By analyzing the faults found during the approach application, the following dataset (Table VI) was structured. As it can be seen, all seeded faults were identified.

Faults	ID1	ID2	ID3	ID4	ID5	ID6	ID7	ID8
Fault1	x	x	x	x	x	x	x	x
Fault2	x	x	x		x	x	x	x
Fault3		x				x	x	x
Fault4		x		x	x			
Fault5		x		x	x			
Fault6		x						
Fault7	x					x	x	

Table VI
DEFECTS PER SUBJECTS.

It is important to highlight that only the root cause was

considered to evaluate this aspect. Besides, the not purposely seeded faults and indentation faults were not considered in this evaluation. Faults wrongly reported in the questionnaire were not considered as well. All of them will serve as lessons learned to avoid in future experiments. Considering these data, since we had 0 faults no identified, the $H1_1 : \mu_{ND} \geq 20\%$ null hypothesis is **rejected**.

Correlation Analysis. By observing the subject with the best results in this aspect, we could see that all of them have more than 2 years of experience in software development. It can indicate that a high experience in development is required by the person which will apply the approach.

Regarding to the number of subjects that found a specific fault, we can notice that the faults (1, 2 and 7) were the most found during this experiment, it could be explained by the fact that the CRs provided by the experimenter described these faults. It can indicate that the CRs help the approach execution. In additional, no correlation was found regarding to the type of fault.

Q5- Number of Tests Correctly Classified. The subjects were asked to classify the test cases in five categories (obsolete, reusable, retestable, new-structural and new-specification). Unfortunately, some subjects (ID 1 and 3) were excluded from this evaluation since they did not report anything or wrongly reported the results.

Table VII summarizes the number of test correctly classified by subjects. Where “not reported (NR)” means that the subjects report some test cases but not correctly, and “none” indicates that no test cases were reported.

Subject ID	ID2	ID4	ID5	ID6	ID7	ID8
Obsolete	NR	1	none	1	1	NR
Reusable	13	30	2	3	3	7
Retestable	3	6	9	5	5	NR
New-Structural	NR	NR	none	none	none	NR
New-Specification	14	NR	3	14	14	NR
NTCC (%)	40.54	50	18.91	31.08	31.08	9.45

Table VII
NUMBER OF TESTS CORRECTLY CLASSIFIED.

A set of 58 test cases were provided to the subjects, among these tests, 3 obsolete, 14 retestable, 41 reusable, 2 new-structural and 14 new-specification. Most of the subjects complained about the absence of examples regarding each kind of test. It can explain the bad results in this aspect.

Although we observed that the test classification description should be improved, two subjects achieved more than 40% of correctly classified test cases. For this reason, we consider the null hypotheses ($H2_1 : \mu_{NTCC} \leq 40\%$) was **rejected**.

Correlation Analysis. We noticed that subjects with high experience in software development, had better results. It can indicate that experience in development can also help during the test classification step.

V. THREATS TO VALIDITY

Envisioning a possible replication of this study, we have identified the following aspects:

Maturation: This is the effect that subjects react differently as time passes. Some subjects can be affected negatively (tiring or boring tasks) during the experiment, and their performance may be damaged. In order to mitigate boredom, two different experiments should be performed for each scenario (corrective and progressive).

Instrumentation: This is the effect caused by the artifacts used for experiment execution, such as data collection forms, code, seeded faults, etc. A real-world SPL project should be used, that could provide more reliable data.

Gained Experience: It is the effect caused by the experiment execution order, in our case, the corrective scenario was performed before the progressive scenario. The subject gained a certain experience executing the first scenario, reducing the time needed to perform the second scenario. In a context with more subjects available, each scenario should be performed by different subjects groups.

Experience in Java Development: Subjects with low experience in software development using Java can affect this validity, since it is hard to understand the code and its peculiarities. To mitigate this lack of experience, the language specifications were provided and we chose a small and common domain (Bank System). However, we believe that it is not enough to mitigate this threat.

VI. RELATED WORK

In [17], the authors emphasize that with the advent and use of software specifications, source code no longer has to be the single source for selecting test cases. Their particular interest has been devoted to specification-based conformance testing. The main goal of their work is to review and extend their previous work on Software Architecture (SA)-based conformance testing, to provide a systematic way to use an SA for code testing. They present an conformance testing approach, establishing a set of steps in order to test a C2 style architecture. This work also presents a case study, where the approach is applied in the elevator system’s architecture.

In [2], the authors explore how regression testing can be systematically applied at the software architecture level in order to reduce the cost of retesting modified systems, and also to assess the regression testability of the evolved system. Moreover, this approach addresses two goals: (i) Test conformance of a modified implementation P’ to the initial SA and (ii) test conformance of an evolved software architecture. To achieve these goals, a set of steps and tools were used.

In addition to the contributions of these previous work, our study contributes with a SPL architecture regression testing approach [9, 18] to optimize the test selection/prioritization and reduce the testing cost. Therefore, instead of only analyzing the efficiency on detecting faults, we also investigated the aspects of effort of the approach execution. An evaluation

was conducted with eight subjects in order to validate the approach.

VII. CONCLUSION

In this work, we have presented a description about the evaluation of a SPL regression testing approach [9], including its consequences for software testing and the influencing factors. The evaluated approach can be applied in two scenarios, the corrective scenario in which it was performed after a corrective change in the code, and the progressive scenario executed after a evolution or enhancement, when some specification changes. Furthermore, the approach also use a graph comparison technique in order to compare different code versions.

We performed an evaluation following some principles from [13]. The evaluation assessed the approach effectiveness and efficiency in find faults, classify existing test cases, as well as, it addressed the effort to apply it on both progressive and corrective scenarios. Although the approach had not been evaluated in a real scenario, it was important to show some bottlenecks and deficiencies.

Notwithstanding, for future work, the approach can be improved by incorporating a tool. We believe that code visualization techniques, such as graphical representation, can increase the user experience and accuracy.

Finally, we believe that a case study in a real scenario for software testing must be performed, taking in account the lessons learned and the changes to be performed, so more concrete conclusions can be drawn. Admittedly, in no way we imply that results from this study can definitively answer all of the stated research questions for all environments. Rather, the analysis presented lends insights into their answers, that may be either verified or compared in future research.

ACKNOWLEDGMENT

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES(<http://www.ines.org.br>)), funded by CNPq and FACEPE, grants 573964/2008-4, APQ-1044-1.03/10 and APQ-1037-1.03/08 and CNPq grants 305968/2010-6, 559997/2010-8, 474766/2010-1 and FAPESB.

REFERENCES

- [1] M. Shaw and P. Clements, "The golden age of software architecture," *IEEE Software*, vol. 23, no. 2, pp. 31–39, 2006.
- [2] H. Muccini, M. Dias, and D. J. Richardson, "Software architecture-based regression testing," *J. of Systems and Software*, vol. 79, no. 10, pp. 1379–1396, 2006.
- [3] M. J. Harrold, "Testing: a roadmap," in *ICSE: International Conference on Software Engineering*, NY, USA, 2000, pp. 61–72.
- [4] A. Orso, N. Shi, and M. J. Harrold, "Scaling regression testing to large software systems," in *SIGSOFT'04: International symposium on Foundations of software engineering*, NY, USA, 2004, pp. 241–251.
- [5] T. Apiwattanapong, A. Orso, and M. J. Harrold, "JDiff: A differencing technique and tool for object-oriented programs," *Automated Software Engineering*, vol. 14, no. 1, pp. 3–36, 2007.
- [6] P. A. da M. Silveira N., I. d. C. Machado, J. D. McGregor, E. S. de Almeida, and S. R. de L. Meira, "A systematic mapping study of software product lines testing," *Inf. Softw. Technol.*, vol. 53, May 2011.
- [7] R. Kolb and D. Muthig, "Challenges in testing software product lines," in *CONQUEST: Conference on Quality Engineering in Software Technology*, Nuremberg, Germany, 2003, pp. 81–95.
- [8] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *FOSE: Future of Software Eng.*, Washington, DC, USA, 2007, pp. 85–103.
- [9] P. A. da M. Silveira N., I. do C. Machado, Y. C. Cavalcanti, E. S. Almeida, V. C. Garcia, and S. R. de L. Meira, "A regression testing approach for software product lines architectures," in *Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS)*, Sept. 2010, pp. 46–55.
- [10] J. D. McGregor, "Testing a software product line," Tech. Rep. CMU/SEI-2001-TR-022, 2001.
- [11] L. Jin-hua, L. Qiong, and L. Jing, "The w-model for testing software product lines," in *ISCSCT: Inter. Symp. on Computer Science and Computational Technology*, Los Alamitos, CA, USA, 2008, pp. 690–693.
- [12] G. Rothermel, R. J. Untch, and C. Chu, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Eng.*, vol. 27, no. 10, pp. 929–948, 2001.
- [13] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.
- [14] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in software engineering: an introduction*. Kluwer Acad. Pub., 2000.
- [15] J. D. McGregor, "Toward a fault model for software product lines," in *SPLC: Software Product Line Conference*, Limerick, Ireland, 2008, pp. 157–162.
- [16] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*. Boston, MA, USA: PWS Publishing Co., 1998.
- [17] H. Muccini, M. S. Dias, and D. J. Richardson, "Systematic testing of software architectures in the c2 style," in *FASE: International Conference on Fundamental Approaches to Software Engineering*, Barcelona, Spain, 2004, pp. 295–309.
- [18] P. A. da M. Silveira N., *A Regression Testing Approach for Software Product Lines Architectures: Selecting an efficient and effective set of test cases*. Germany: LAP Lambert Academic Publishing, 2010.