The Use of Experimentation Packages for Evaluating the Quality of Mobile Software Products

Auri Marcelo Rizzo Vincenzi
Instituto de Informática
Universidade Federal de Goiás, UFG
Goiânia-GO, Brazil
e-mail: auri@inf.ufg.br

Gilcimar Divino de Deus
Departamento de Computação
Pontifícia Univ. Católica de Goiás, PUC-GO
Goiânia-GO, Brazil
e-mail: gyngil@gmail.com

João Carlos da Silva
Instituto de Informática
Universidade Federal de Goiás, UFG
Goiânia-GO, Brazil
e-mail: jcs@inf.ufg.br

Plínio de Sá Leitão-Júnior
Instituto de Informática
Universidade Federal de Goiás, UFG
Goiânia-GO, Brazil
e-mail: plinio@inf.ufg.br

José Carlos Maldonado
Inst. de Ciências Matemáticas e de Computação
Universidade de São Paulo, USP
São Carlos-SP, Brazil
e-mail: jcmaldon@icmc.usp.br

Márcio Eduardo Delamaro
Inst. de Ciências Matemáticas e de Computação
Universidade de São Paulo, USP
São Carlos-SP, Brazil
e-mail: delamaro@icmc.usp.br

***Abstract*—Mobile devices are becoming more and more common. Embedded in these devices are different mobile applications, making the devices more useful and popular. The quality of such applications is increasingly becoming a problem. Several techniques have emerged to assess software quality. In this paper, an experimentation package is proposed to evaluate some of the well-known software testing criteria on detecting faults in mobile software. This paper presents the results obtained after three replications of the proposed package. Based on statistical analysis, it was possible to arrive at statistical equivalences and differences between the evaluated criteria. This can help people concerned to establish testing strategies for mobile software.**

Keywords-experimental package; software testing; ubiquitous application

I. INTRODUCTION

When a customer orders an information system, he/she lists some characteristics or requirements and he/she searches for quality in each item of the list. Software engineers must aim at quality during all the development process. According to IEEE, software product quality is defined as: "The degree to which a system, component or process meets the specified requirements and the needs or expectations of the client" [1].

Standards such as ISO 9000, 9001, and 9002 deal with quality management. One of the requirements of these models is Verification and Validation (V&V). In other words, it is necessary to determine if the product is being produced correctly, if this product meets its requirements and if it responds as expected. Software testing is largely responsible for ensuring the quality of a software product and it is one of the most common activities in software validation.

Several techniques have been adopted to expose faults in software products. *Ad-hoc* testing is based on the experience of the tester that executes a set of test cases he/she believes enough to ensure quality. A more systematic way of carrying out testing is to employ the best known functional and structural techniques.

With the functional technique, a program is tested from the user's point of view. The component being tested is considered as a black box, whose implementation details are not known, inputs are supplied, and results are compared against the expected ones. On the other hand, the structural technique, also known as a white box test, determines test cases based on implementation aspects and helps detect logical and programming faults.

In 2012, there were around 256 million cell phones in Brazil [2]. Their processing power, transmission speed, and other technological characteristics allow information handling by systems in mobile devices. It is very important for projects to be developed, which focus on improving testing strategies and applying them to the mobile environment.

One problem with mobile devices is the difficulty in testing applications in the device itself (real environment). Development and testing phases, in general, take place using emulators on desktop computers. It is extremely important for applications to be tested in their real environment, since errors may occur and be camouflaged by emulators due to their memory and processing limitations. Java Bytecode Understanting Testing/Micro-Edition (JaBUTi/ME) is a tool developed in this context, which supports the testing of Java ME software in both emulators and real devices [3].

 ISBN: 978-1-61208-304-9

This paper presents the results collected after three replications of an experimentation package created with the purpose of analyzing and comparing three testing techniques for mobile devices: *ad-hoc*, functional (focusing on boundary analysis and equivalence partitioning), and structural (mainly All-Nodes and All-Edges criteria [4]).

An experimentation package is a controlled and systematic way of carrying out experiments in several stages, making it possible to incrementally obtain a quantity of statistically significant data. In addition, the availability of an experimentation package allows the same study to be carried out by different people, in different places, with different cultures. This makes it possible to update these data over time, increasing the statistical database and increasing the confidence on the quality of obtained results.

This paper shows the database status after the third replication of the experimental package (investigating *ad-hoc*, functional, and structural testing techniques), and the adaptation of the JaBUTi/ME tool to support one of the testing techniques. Related works are described in Section II. In Section III, the main characteristics of the JaBUTi/ME tool are described, along with the testing criteria it supports. The experimentation package used in the replications is detailed in Section IV. Section V presents the experiment description, including the statistical data analysis. Section VI presents the conclusion of this study and future research directions.

II. Related Studies

Some studies in the literature have discussed mobile applications testing and the majority of them applies black-box testing technique without comparisons with structural testing.

Malevris [5] presented a method to effectively perform structural testing in Java programs. The proposed methods intend to generate a set of feasible paths and automatically generate test data to traverse such paths. Symbolic execution is used to identify feasible paths and the results show that, in general, the proposed methods avoid the generation of infeasible paths and ensure high coverage of the generated paths. No comparison with additional testing criteria is provided.

Pocatilu [6] focuses on the aspects related to unit testing in mobile applications based on Java ME. Emulators are used to run test cases written according to the JUnit testing framework. The author concludes that unit testing does not have to be limited to the JUnit framework, and other methods and techniques shall be used, such as the ones proposed in our evaluation.

Hu and Neamtiu [7] propose an approach for automating the testing process for Android applications. The first step was to understand the nature and frequency of bugs affecting graphical user interface (GUI) of Android applications. Following, they proposed an automated test generator for detecting these GUI bugs. The approach is based on feeding the application with random events, instrumenting the Android Virtual Machine, registering log/trace files, and analyzing them post-run. In that work, no structural testing criteria was employed to evaluate the quality of the generated test data.

In our work, we evaluate three different testing criteria considering the coverage and fault detection capability of their generated test set.

III. JaBUTi/ME and Mobile Devices

Testing without a tool increases the chance of human mistakes, and lowers productivity in test execution and analysis of results. Many tools have been produced, and each is focused on the use of one or more criteria. Java Bytecode Understanding and Testing (JaBUTi) [8] is one such tools. It explores structural testing criteria, which help creating test cases that exercise specific parts of the code.

Among the various resources offered by JaBUTi, one of the most important is the support in the coverage of bytecode-based Java programs. In others words, JaBUTi performs all computations for the Java structural test directly on bytecode, not on program source.

Java Bytecode Understanting Testing/Micro-Edition (JaBUTi/ME) is a version of JaBUTi that supports the structural testing of Java ME programs [3]. It explores the same resources as the original version and complements the original version with resources that allow program test in real mobile devices or emulators. Among the customizable resources in this version are the different code instrumentation mechanisms offered, which make it possible for the real application to communicate with the test server in accordance with memory and connectivity restrictions imposed by the different types of mobile devices, as shown in Figure 1.

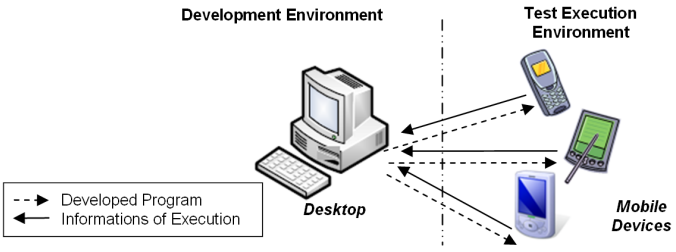

Figure 1. Environment cross platform

Program instrumentation is an essential activity for applying structural testing, making it possible to capture information about code coverage during test case execution. When a code is being instrumented, a call to a method responsible for identifying and storing information about which section of code has been executed is added to the bytecode. This information is later sent to the test server which computes the resultant coverage with respect to each testing criterion.

Since the development of JaBUTi/ME, a series of experimental studies was carried out aiming at evaluating whether its characteristics help the test of Java ME products. In this context, the focus of this study is to execute tests of programs developed for mobile devices using JaBUTi/ME. The creation of an experimentation package allows the experiments to be executed in a controlled environment and to be replicated by other researchers also interested in this research area.

 ISBN: 978-1-61208-304-9

The replications of one particular package help to improve the collected data, to increase the sample size and to allow more reliable conclusions. One reason for carrying out the replications was the physical impossibility of executing the entire experimentation package with a large sample of subjects. This was due to the limited number of places in software laboratories available. The replications in this study serve to increase the level of confidence in the collected data. They also help to show which technique, *ad-hoc*, functional or structural is more efficient in detecting faults in Java ME programs for the creation of new test cases and program code coverage. Another intention was to evaluate whether the resources offered by the tool are useful for testing Java ME programs in real devices and emulators. Due to the unavailability of a sufficient number of real devices, that replication was carried out using mobile device emulators.

IV. Experimentation Packages

This section describes how the experimental study using the JaBUTi/ME tool in replications was conducted. The purpose was to evaluate the three techniques mentioned earlier and their suitability for testing mobile device applications. This also made it possible to evaluate the benefits the criteria supported by the tool offer to the tester.

The goal of this study is to contribute to the development of an incremental test strategy with the support of a testing tool that can be used to improve the quality of software products and information systems used in mobile devices. Considering the increasing demand for mobile device software, the results of this study may significantly contribute to evaluating of testing techniques and to increasing in the quality of mobile software products.

A. Experimentation Package for JaBUTi/ME

The experimental study follows the process described by Wohlin et al. [9]. This experimentation package is defined and organized in the following way:

- **Definition**: Structural Test of Java ME Software in Mobile Devices Using JaBUTi/ME.
- **Context**: This experiment is an example of software engineering and, more specifically, of software testing. A specific tool, JaBUTi/ME, which was created for the structural testing of Java ME programs.
- **Hypotheses**: The following hypotheses may or may not have been valid after the experiment has been carried out.
  - Null Hypotheses:
    - $H_{0,1}$ - The structural technique, supported by JaBUTi/ME tool, detected the same number of faults as the *ad-hoc* or functional techniques;
    - $H_{0,2}$ - The structural technique, supported by JaBUTi/ME tool, obtained the same percentage of coverage as the *ad-hoc* or functional techniques;
    - $H_{0,3}$ - The structural technique, supported by JaBUTi/ME tool, did not contribute to the creation of new test cases.
  - Alternative Hypotheses:
    - $H_{1,1}$ - The structural test, supported by JaBUTi/ME tool, detected different number of faults obtained when compared to the *ad-hoc* or functional technique;
    - $H_{1,2}$ - The structural technique, supported by JaBUTi/ME tool, obtained a different percentage of coverage when compared to the *ad-hoc* or functional techniques;
    - $H_{1,3}$ - The structural technique, supported by JaBUTi/ME tool, contributed to the creation of new test cases which had not previously been identified by either the *ad-hoc* or functional test.
- **Dependent Variables**:
  - Program complexity;
  - Number of defects revealed;
  - Coverage percentage;
  - Number of new test cases.
- **Independent Variables**:
  - *Ad-hoc* technique;
  - Functional technique;
  - Structural technique;
  - Selected programs.
- **Participants**: Sixty people with computer science and Java programming knowledge participated in the experiment as subjects. The only prerequisite to participate in the experiment is a basic knowledge of Java programming. Participants should be able to recognize commands, programming structures, loops, and so on. No software testing knowledge was required.
- **Experimental Project**: Four Java ME programs were selected for the experiment. The factorial-fractional randomized technique [10] was used to assign to each subject a particular testing technique and a program to be tested. One of these programs was used for teaching functional and structural techniques. Participants used the other three to run the experiment. The participants identification by their names was not relevant for the object of the experiment. Participants were grouped merely as a way of dividing the same program among a given number of students. The information was collected and evaluated individually. It is important to mention that the programs were divided equally among the groups.

The experiment was carried out over three non consecutive days. An hour of training was provided for each technique. Later, the participants had an hour and a half to apply "hands on" the technique in one of the selected programs. The laboratory had 20 desktop computers with the Linux operating system, Java 6.0, Eclipse, Wireless Tool Kit 2.5, EclipseME, and the JaBUTi/ME tool.

The programs were selected from software repositories such as http://www.sourceforge.net and http://code.google.com. Twenty programs were pre-selected based on the availability of source code and program complexity, of which the four most complex were chosen.

 ISBN: 978-1-61208-304-9

All these programs were previously instrumented using JaBUTi/ME to make it possible to collect trace data during the program execution, even when the *ad-hoc* or functional technique is used to generate test cases. The execution was monitored and code coverage could be evaluated later in relation to the structural criteria implemented by JaBUTi/ME. It is important to point out that the same tool was used to evaluate the three techniques. Figure 2 shows the process of executing instrumented software and how coverage information was collected. Additionally, each subject should also fill out a form indicating when a given test case detects a fault.

- **Instrumentation**: In this stage, the forms, software, and laboratory environment for carrying out the experiment was prepared.

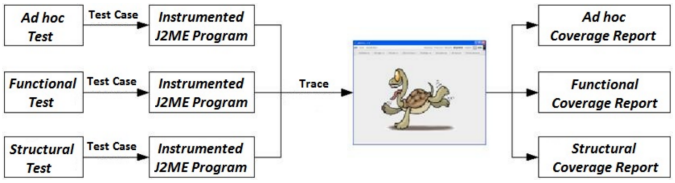

Figure 2. Monitoring scheme outline

Four forms were prepared to be filled out by the subjects: Form 1 – Group Formation; Form 2 – Test Cases; Form 3 – Suggestions; and Form 4 – Course Evaluation. These forms and all the data collected may be obtained by contacting the corresponding author.

The four most complex pre-selected programs were chosen for the experiment. Table I presents the name of the programs used for data collection, the average maximum cyclomatic complexity of their methods, and a brief description of each.

TABLE I. SELECTED PROGRAMS AND COMPLEXITY

Id.	Name	Complexity	Description
P1	AntiPanela	3.87	Registers soccer players and performs team drawings based on the number of players, avoiding favoritism.
P2	CarManager	5.52	Monitors and manages motor vehicle fuel expenses.
P3	CódiceFiscale	6.17	Checks the validity of or generates the Italian "Codice Fiscale" tax ID.

Programs P1, P2, and P3 were used by participants to apply *ad-hoc*, functional, and structural techniques. Their order and distribution are defined in Table II. A fourth program, called BMI, which calculates Body Mass Index based on height and weight and classifies an individual according to obesity level, was used for training participants in functional and structural techniques and tools.

To assess the quality of the resulting test set on detecting faults, ten faults were artificially seeded to each program based on the concept of mutation [11]. Faults were related to variable initialization, computations, control flow, interface, and data structure. After inserting the faults, programs were compiled and instrumented using JaBUTi/ME resources to make it possible to monitor test case execution, and later to analyze their coverage in relation to the criteria supported by the tool.

- **Evaluation**: For all programs, the evaluation was based on Form 2 – Test Cases, which contains information about test case execution (faults found).
- **Preparation**: Materials and instructions for participation in the experiment were distributed. It is important to demonstrate what is really taking place as the experiment was conducted. The BMI software was chosen for teaching all of the techniques and for running programs in mobile device emulators. This software was not used for collecting information from the participants.
- **Execution**: This is the task of executing what was planned in the estimated time and documenting any deviation that could change or affect the objective of the experiment. Program specifications were also explained to the participants, so they could become familiarized with the programs under testing.
- **Data Validation**: At the end of the application of each technique by the participant, the entire project (including the trace file) must be labelled and sent to the organizing commission, ensuring that the generated data of each participant was correct.
- **Analysis and Interpretation**: Immediately after experiment and replication data had been collected, the information was cross-checked and analyzed in order to evaluate the hypotheses defined in the experimentation package.
- **Presentation and Packaging**: This paper intends to group the data of these three replications.

V. EXPERIMENT DESCRIPTION

The proposed experimentation package was replicated three times. The information collected after each replication strengthen and increase the entire experiment's sample size.

An introduction about software testing showing the importance of testing, the role of the tester, the main kinds of tests, unit, integration, and system tests were explained to and discussed with the participants during training. The students were then randomly assigned to 6 groups. Once the groups were defined, a Java ME program together with its respective specification text were distributed to each group. The objective on the first day was to find the largest number of faults in the programs in accordance with each individual's knowledge of software construction and testing, i.e., using the *ad-hoc* technique. The distribution of programs to the groups is shown in Table II.

TABLE II. GROUP, PROGRAM AND TECHNIQUE DISTRIBUTION

Technique/Group	G1	G2	G3	G4	G5	G6
ad-hoc	P1	P3	P2	P1	P3	P2
Functional	P2	P1	P3	P2	P1	P3
Structural	P3	P2	P1	P3	P2	P1

G – Group; P – Program;

After executing the *ad-hoc* technique, on the second day the participants received training concerning functional test

 ISBN: 978-1-61208-304-9

technique criteria. New programs and their specifications were distributed to each group. The groups were again asked to apply the knowledge they had acquired on functional testing to carry out tests on the second program.

After the tests using the functional technique were run, on the third day participants were trained in the structural technique and use of the JaBUTi/ME tool. After this, the third and final distribution of programs was carried out and students applied structural technique concepts in running structural tests.

During the execution of the tests using any of the techniques, the participants recorded any nonconformity they found. To conclude the experiment, they were asked to fill out a form with suggestions for improvement and their individual evaluation of the course. It is important to emphasize that all students were required to test all three programs using the three different testing techniques.

A. Data Analysis

The experimentation package was prepared to capture coverage information for the programs under testing, regardless of applied technique. The JaBUTi/ME tool was used to read the data from the executed tests of each subject. The tool supports four testing criteria: All-Nodes, All-Edges, All-Uses, and All-Potential-Uses [8]. The training focused on the first two criteria, known as control flow criteria (All-Nodes and All-Edges). However, all the criteria cited above, including data flow, were analyzed by measuring the coverage of the tests in relation to these test criteria. It is important to point out that non-executable test requirements produced by the above mentioned criteria were not identified. In addition, test execution time was limited to one hour and a half. Therefore, it may be that the maximum coverage of 100% was not achieved due to these requirements and time constraints. However, since the objective was to compare which test set covered more testing requirements, the maximum obtained coverage of any test set is sufficient to establish this relationship. Tables III and IV synthesize these data.

The cumulative data after the third replication shows that the generated test set from the structural technique achieved the highest coverage of all the programs tested, and, for this set of programs, the standard deviations of the three techniques were very close (see Table IV). These data show that the values presented do not cluster around the mean and that the structural technique demonstrates better coverage for software testing in a mobile device context. Figures 3 to 5 show the coverage evolution of each testing criterion supported by JaBUTi/ME for each program under analysis. Observe that structural testing test set achieved the highest coverage in all three programs.

The structural and *ad-hoc* techniques detected more faults (see Table IV and Figure 7). Although the numbers are small in comparison with the number of faults inserted, it is important to point out that program coverage was not complete and that test execution time was a criterion in creating the experimentation package. This suggests that there is a tendency for increasing the number of detected faults as coverage also increases, which would only be possible if test creation and execution time increased. Since the structural technique presented the best results in coverage and number of test cases, it has a chance of revealing more faults than the other techniques, due to its different characteristics , but this should be further investigated.

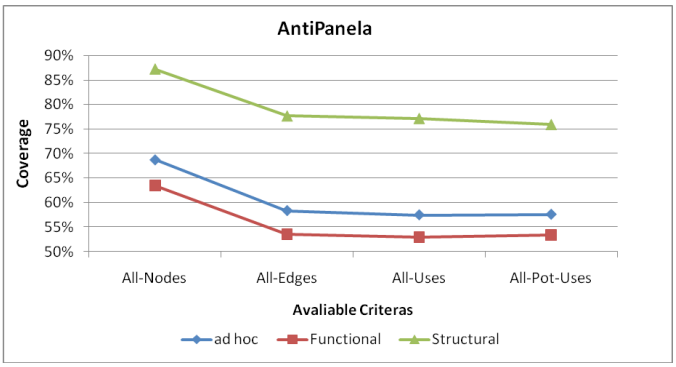

Figure 3. Coverage by program: AntiPanela

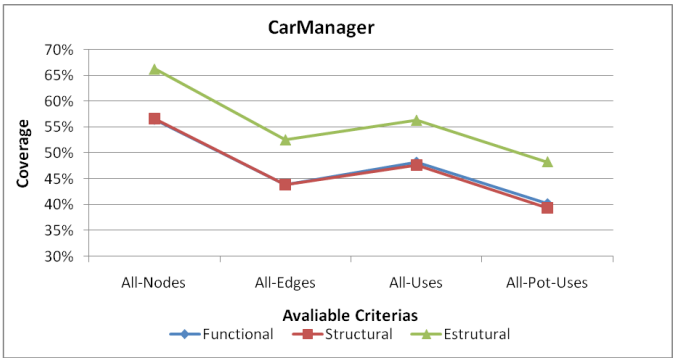

Figure 4. Coverage by program: CarManager

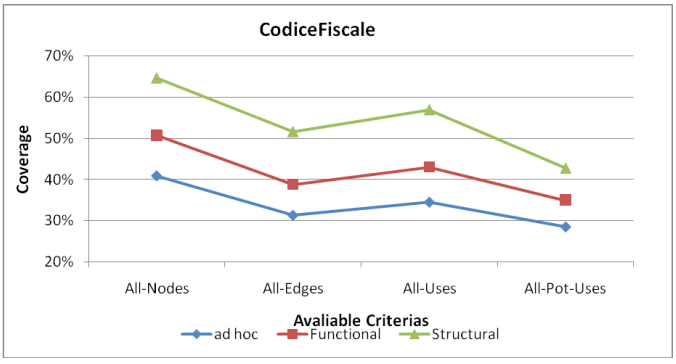

Figure 5. Coverage by program: CodiceFiscale

Thus, the more complex the program, such as CarManager and CodiceFiscale, the greater the time required to test it. In addition, the structural and functional techniques with JaBUTi/ME were used in actual practice by the majority of participants for the first time. All this information can be found in Tables III and IV, and Figures 6 and 7.

 ISBN: 978-1-61208-304-9

TABLE III. AVERAGE OF COVERAGE (%), TEST CASES AND FAULTS BY PROGRAM

	AntiPanela			CarManager			CodiceFiscale		
Criteria/Technique	*ad-hoc*	Functional	Structrual	*ad-hoc*	Functional	Structrual	*ad-hoc*	Functional	Structrual
All-Nodes	68,76	63,50	87,25	56,56	56,63	66,27	40,88	50,71	64,69
All-Edges	58,35	53,50	77,67	43,89	43,84	52,53	31,29	38,79	51,69
All-Uses	57,47	52,89	77,17	48,22	47,63	56,33	34,47	43,00	57,00
All-Pot-Uses	57,59	53,33	75,92	40,17	39,32	48,27	28,47	34,93	42,81
Number of Test Cases	11,13	10,56	13,33	7,75	9,16	10,62	7,94	7,29	11,56
Faults Found	3,73	2,67	3,33	1,75	1,79	1,00	1,59	1,69	2,50

TABLE IV. STATISTICS OF COVERAGE (%), TEST CASES AND FAULTS BY TECHNIQUE

	ad-hoc			Funcional			Structrual		
Criteria/Technique	Av	SD	Median	Av	SD	Median	Av	SD	Median
All-Nodes	55	20	56	57	18	61	72	18	73
All-Edges	45	18	44	46	17	50	59	18	59
All-Uses	47	19	48	48	17	50	62	18	63
All-Pot-Uses	42	19	40	43	18	44	54	20	54
Number of Test Cases	8,9	6,6	8,0	9,1	5,0	8,0	11,8	5,4	11,0
Faults Found	2,3	2,0	2,0	2,1	1,5	2,0	2,3	1,8	2,0

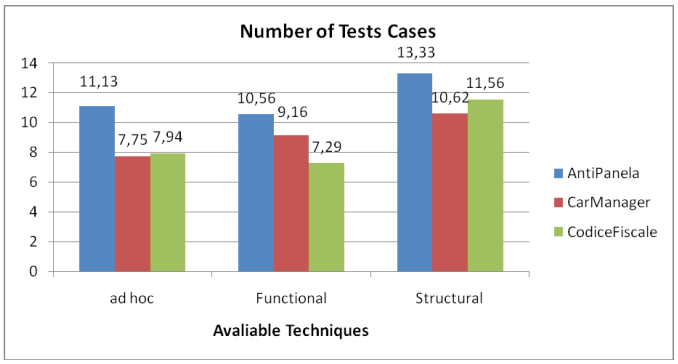

Figure 6. Number of test cases by program

Some data were lost during the experiment. The most common causes were: a) the participant did not save Form 2 – Test Case files correctly and was unable to send them to the course organizers; b) the participant did not initialize the programs correctly. This made it impossible to capture coverage information. Information loss reached about 20% for AntiPanela, 12% for CarManager, and 20% for CodiceFiscale.

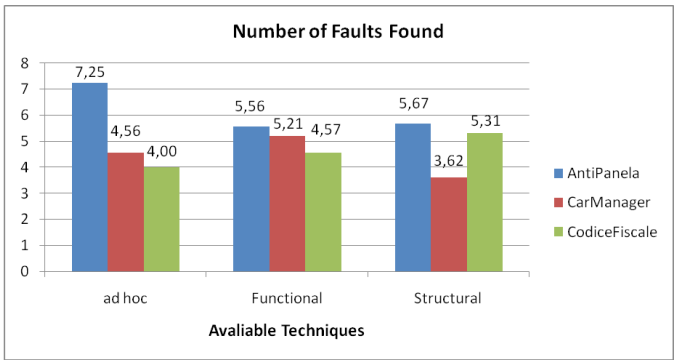

Figure 7. Number of faults found by program

Despite our emphasis on the importance of correctly following all the steps and executing the experiment, unfortunately deviations happen, the simultaneous supervision of around 20 participants per replication is very complex, and losses of data are inevitable. On Form 3 – Suggestions, 45% of the subjects asked for the presentation of other tools, including other languages, to give them more options for carrying out the tests. Thirty percent (30%) said that they would need more time to learn and practice the techniques. In other words, they assumed that they did not find more faults in the programs because of time constraints. Fifteen percent (15%) suggested not using Java ME programs.

In Form 4 – Course Evaluation, 100% of the participants said that the course had increased their knowledge of testing. Eighty-eight percent (88%) indicated that they felt confident in applying presented techniques. On the form, participants were asked to grade the level of knowledge acquired during the course. The average was 7.9 and the general grade for the course was 8.6, considering a 0 to 10 scale. Thus, the majority of participants approved and praised the initiative because testing techniques are not widely disseminated and it is difficult to find a free course on testing.

The participants made a number of comments about the course. The most important were: 1) that there is a lack of trained testing personnel; 2) that testing software is difficult; 3) that there is a shortage of testing tools. Many participants were interested in further studying JaBUTi/ME and in applying it in academic and professional programs.

The first step in the statistical analysis was to group the data by technique (*ad-hoc*, functional, and structural) rather than by program (AntiPanela, CarManager, and CodiceFiscale). The Shapiro-Wilk Test showed that the sample did not present a normal distribution. That is, it was necessary to use non-parametric statistical methods. The Kruskal-Wallis Test is robust for normality and its use makes it possible to check if there are relevant differences between the techniques evaluated in this paper. Its application showed that there are relevant differences between the three techniques for the criteria of coverage and number of test cases, as shown in Table V.

TABLE V. KRUSKAL-WALLIS TEST – RANK SUM TEST

Crit./Tech.	Ad-Hoc	Functional	Structural	p-value	Diff
All-Nodes	55,5	73,0	61,0	0,000019	Yes
Test Cases	8,0	11,0	8,0	0,009148	Yes
Faults	2,0	2,0	2,0	0,930200	Yes

 ISBN: 978-1-61208-304-9

Having discovered that there are differences between the techniques, it is necessary to find out what these differences are and then display the most effective technique for carrying out Java ME software tests on mobile devices. The Kruskal-Wallis Multiple Comparison Test is robust for normality and number of samples. It was used to compare pairs of techniques for each criterion. The result of this comparison is shown in Table VI.

TABLE VI. KRUSKAL-WALLIS MULTIPLE COMPARISON TEST

All-Nodes	Diff Observed	Crit Diff	Diff
ad-hoc X Structural	37,531753	20,86848	**Yes**
ad-hoc X Functional	3,86463	19,95248	No
Structural X Functional	33,667123	20,96088	**Yes**
Test Cases	Diff Observed	Crit Diff	Diff
ad-hoc X Structural	25,485814	20,69736	**Yes**
ad-hoc X Functional	5,804322	19,56142	No
Structural X Functional	19,681492	20,51164	No
Faults	Diff Observed	Crit Diff	Diff
ad-hoc X Structural	4,3252033	20,500141	No
ad-hoc X Functional	3,8666667	19,48089	No
Structural X Functional	0,4585366	20,31164	No

VI. CONCLUSION

Together, the three replications of the experiment highlight the importance and complexity of software testing in software engineering. All the different techniques and criteria focus on finding faults in types or parts of applications. The best known criteria include value limit analysis, equivalence partitioning, all-nodes, and all-edges.

Each technique has a particular focus, and techniques should be used together to find more faults in programs. The presented techniques help the tester select entry domain values systematically and may optimize the creation of test cases and increase fault detection.

The data collected in the replications of this experiment by Deus et al. (2008) show that the use of JaBUTi/ME and the structural technique help create test cases and consequently, provide greater coverage in mobile device programs. A statistical analysis showed that all techniques work equally well in detecting faults. In other words, the number of faults found using the evaluated techniques in this study did not differ significantly. However, it is important to point out that there are other characteristics besides fault detection that add value to software, which include the coverage of the software's internal structure, mainly important for program maintenance.

Thus, due to the techniques' similar performance, it is necessary to evaluate other criteria to choose the most efficient technique for ensuring mobile software product quality. Statistical analyses showed that among the evaluated techniques, there are significant differences in the criteria of code coverage and the number of test cases. Statistically, the structural technique performs better with respect to both of these aspects. More test cases were created and, consequently, greater coverage was achieved. Therefore, this initial study was not conclusive and should be replicated more times to increase its knowledge database.

Lessons were learned with each replication. This will help to improve the quality and objectivity of future studies that assess the results of experimentation packages.

Future research into mobile device software quality may include replication of this experimentation package using real mobile devices instead of emulators, creation of an effective method for mobile software quality control, and the evaluation of these or other techniques for conventional (non-mobile) software.

Smartphones are becoming more and more common and a large number of applications are created and freely distributed in different software repositories. Another option for future research is to use this package or to create a new package for Android environment that uses Java, that is a prerequisite for execution in JaBUTi/ME.

ACKNOWLEDGMENT

The authors would like to thank the Instituto de Informática – INF/UFG, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES – Brasil, and Fundação de Amparo à Pesquisa do Estado de Goiás – FAPEG – Brasil, which support this work.

REFERENCES

[1] IEEE, "IEEE standard glossary of software engineering terminology," International Standard, IEEE Computer Society Press, Standard 610.12-1990 (R2002), 2002.

[2] G1, "Mobile phones reach 256 milion of lines in july on brazil," Web page, Aug. 2012, [retrieved: Sep., 2013] (in Portuguese). [Online]. Available: http://g1.globo.com/tecnologia/noticia/2012/08/telefonia-movel-alcanca-256-milhoes-de-linhas-em-julho-no-brasil.html

[3] M. E. Delamaro, A. M. R. Vincenzi, and J. C. Maldonado, "A strategy to perform coverage testing of mobile applications," in *I International Workshop on Automation of Software Test – AST'2006*. New York, NY, USA: ACM Press, May 2006, pp. 118–124.

[4] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge University Press, 2008.

[5] N. Malevris, "On structurally testing Java programs effectively," in *Proceedings of the 3rd international symposium on Principles and practice of programming in Java*, ser. PPPJ'04. Trinity College Dublin, 2004, pp. 21–26, [retrieved: Sep., 2013]. [Online]. Available: http://dl.acm.org/citation.cfm?id=1071565.1071570

[6] P. Pocatilu, "Testing Java ME applications," *Informatica Economica*, vol. 12, no. 3, pp. 147–150, 2008.

[7] C. Hu and I. Neamtiu, "Automating gui testing for android applications," in *Proceedings of the 6th International Workshop on Automation of Software Test*, ser. AST'11. New York, NY, USA: ACM, 2011, pp. 77–83, [retrieved: Sep., 2013]. [Online]. Available: http://doi.acm.org/10.1145/1982595.1982612

[8] A. M. R. Vincenzi, W. E. Wong, M. E. Delamaro, and J. C. Maldonado, "JaBUTi: A coverage analysis tool for Java programs," in *XVII SBES – Brazilian Symposium on Software Engineering*. Manaus, AM, Brazil: Brazilian Computer Society (SBC), Oct. 2003, pp. 79–84.

[9] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. New York, NY, USA: Springer Heidelberg, 2012.

[10] G. E. P. Box, J. S. Hunter, and W. G. Hunter, *Statistics for Experimenters: Design, Innovation, and Discovery*, 2nd ed. Wiley-Interscience, May 2005.

[11] Y.-S. Ma, J. Offutt, and Y. R. Kwon, "Mujava: an automated class mutation system: Research articles," *STVR – Software Testing, Verification and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.

 ISBN: 978-1-61208-304-9