Evaluating the Impact of Different Testers on Model-based Testing

Henrique Neves da Silva
Universidade Tecnologica Federal do Parana (UTFPR)
Cornelio Procopio, PR, Brazil
hen123neves@gmail.com

Guilherme Ricken Mattiello
Universidade Tecnologica Federal do Parana (UTFPR)
Cornelio Procopio, PR, Brazil
mattiello@alunos.utfpr.edu.br

Andre Takeshi Endo
Universidade Tecnologica Federal do Parana (UTFPR)
Cornelio Procopio, PR, Brazil
andreendo@utfpr.edu.br

Érica Ferreira de Souza
Universidade Tecnologica Federal do Parana (UTFPR)
Cornelio Procopio, PR, Brazil
ericasouza@utfpr.edu.br

Simone do Rocio Senger de Souza
Universidade de São Paulo (ICMC)
São Carlos, SP, Brazil
srocio@icmc.usp.br

ABSTRACT

Context: Model-Based Testing (MBT) is an approach that allows testers to represent the behavior of the system under test as models, specifying inputs and their expected outputs. From such models, existing tools might be employed to generate test cases automatically. While MBT represents a promising step towards the automation of test case generation, the quality of the model designed by the tester may impact, either positively or negatively, on its ability to reveal faults (i.e., the test effectiveness). *Objective:* In this context, we conducted a preliminary experiment to evaluate the impact caused by different testers when designing a test model for the same functionality. *Method:* In the experiment, the participants used Event Sequence Graphs and its supporting tool `FourMA` to create test models for two mobile apps: *arXiv-mobile* and *WhoHasMyStuff*. From the test models, test cases were generated using `FourMA` and concretized by means of the `Robotium` framework. In order to measure the impact of different testers, we employed code coverage (namely, instruction and branch coverage) as an estimation of test effectiveness. *Results:* Based on the results obtained, we observe high variation of code coverage among the testers. No tester was capable of producing a test model that subsumes all other testers' models with respect to code coverage. Moreover, factor learning seems not to reduce the code coverage variation. The relation between model size, modeling time, and code coverage were inconclusive. *Conclusion:* We conclude that further research effort on the MBT's modeling step is required to not only reduce the variation between testers, but also improving its effectiveness.

SAST '18, September 17–21, 2018, SAO CARLOS, Brazil,

ACM ISBN 978-1-4503-6555-0/18/09…$15.00
https://doi.org/10.1145/3266003.3266012

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**;

KEYWORDS

Model-Based Testing, Automated Tests, Event Sequence Graph, Android, Mobile Apps, Empirical Study

ACM Reference Format:
Henrique Neves da Silva, Guilherme Ricken Mattiello, Andre Takeshi Endo, Érica Ferreira de Souza, and Simone do Rocio Senger de Souza. 2018. Evaluating the Impact of Different Testers on Model-based Testing. In *III Brazilian Symposium on Systematic and Automated Software Testing (SAST '18), September 17–21, 2018, SAO CARLOS, Brazil*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3266003.3266012

1 INTRODUCTION

Software testing is a widely used practice for evaluating the quality of a system; it might account for more than 50% of the total cost of software development [15]. According to Garousi and Elberzhager [8], 79 billion euros were spent on testing activities in 2010 while an increase to 100 billion euros was expected for 2014. Such facts motivate the improvement of testing approaches to reduce the costs involved [2].

There exists a plethora of research on how to find means to reduce the costs of software testing, while keeping its ability to reveal faults. Some part of the community has pushed for the systematic use of models, in an approach called *Model-Based Testing* (MBT). MBT consists of the elaboration of models that define the expected behavior of the *System Under Test* (SUT). From these models, test cases are generated and then executed in the SUT [19]. While MBT can be applied in different software levels, practitioners have performed it as a black-box technique for system testing. The MBT approach has had considerable success in industry, where it is used to enable the automation of large-scale testing [12]. Moreover,

there are a growing number of open source MBT tools, such as `fMBT`, `ATS4`, `AppModel`, and others [14].

The use of models brings several benefits. For instance, it can help in the traceability between test cases and requirements [15], models are easy to update due to software changes, and it can be used as a test oracle by defining the expected behavior. Nevertheless, the presence of test models has its own limitations. Orso and Rothermel [15] highlight three difficulties in applying MBT: *(i)* modeling which is a complex task performed mostly by humans; *(ii)* problems related to the model size as the number of characteristics of the SUT increases; and *(iii)* the lack of empirical understanding of its fault detection capabilities.

While models are central in MBT and bring several benefits, the quality of the model and other test artifacts have an immediate impact on requirements coverage and fault detection [13]. As models are designed mainly by testers, it is important to understand how the human factor may impact, either positively or negatively, on the test effectiveness. Given the human factor and difficulties highlighted, there are opportunities to empirically investigate which aspects, in the context of modeling for MBT, can influence the testing activity effectiveness.

This paper aims to evaluate the impact on the test effectiveness when different testers create models for the same functionality in an MBT context. Specifically, we conducted a preliminary experiment to answer the following research question: *"What is the impact on code coverage considering that different testers design test models for the same functionality?"*. The participants used an MBT tool named `FourMA` [7] to design test models for two real-world mobile apps: *arXiv-mobile* and *WhoHasMyStuff*. `FourMA` adopts Event Sequence Graphs (ESGs) to represent the expected behavior of the SUT; ESG is simple and intuitive, being a reasonable choice to model the event-driven systems like the mobile applications used in the experiment. From the models created, test suites were generated, concretized and executed in the applications, measuring the code coverage achieved as a proxy for effectiveness. We also observe the impact of other metrics like modeling time and model size in the code coverage obtained.

This paper is organized as follows: Section 2 presents the background on MBT, ESGs and the supporting tool `FourMA`. Section 3 brings the experiment configuration. Sections 4 and 5 synthesize and discuss the main results of the experiment, presenting guidelines that can be explored in future work. Section 6 presents the related work. Section 7 concludes and sketches future directions.

2 BACKGROUND

MBT can be applied using a wide variety of modeling techniques, including scenario-based models (message sequence or use case diagrams), and other state-oriented notations, such as: finite state machines and event flow graphs [15]. According to Utting et al. [19], a model simpler than the SUT needs to be produced, or at least easier to check, modify and maintain. The model must generate significant test cases that capture the essence of SUT, being accurate and unambiguous to represent the behavior in a way that can be processed by software tools. The MBT process can be divided into the following main steps. *(1) Modeling:* based on the knowledge of the SUT, a model is designed for test purposes. The test model might represent requirements, features, and the environmental characteristics in which the system will run. *(2) Test Case Generation:* this step is usually done using a tool that will derive test cases from the test model. The tool receives as input a test model and produces as output a set of abstract test cases. *(3) Concretization:* it consists of transforming the abstract test cases into concrete test cases, executable in the SUT. *(4) Test Execution:* in the last step, the concrete test cases are executed in the SUT.

Event Sequence Graphs. In this paper, the test models are specified as ESGs [3] that represent the interactions between a SUT and its users. An ESG model is a directed graph; each vertex represents a user and/or system interaction (namely, events) and edges represent valid sequences of events [3]. Vertices "[" and "]" represent the entry and exit vertices, respectively. Two assumptions must hold for a valid ESG: every vertex v must be reachable, by a path, from "[", and exit vertex "]" must be reachable from v. Figure 1 illustrates an ESG model.

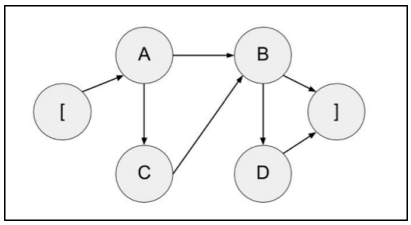

Figure 1: ESG model example

From the model in Figure 1, different algorithms and test selection criteria can be applied to generate test cases in the form of a Complete Event Sequence (CES). A CES is a connected sequence of events beginning at entry vertex "[" and ending at exit vertex "]". Examples of test cases generated from the test model in Figure 1 are shown as follows:

TC 1. `[, A, B, ]`
TC 2. `[, A, C, B, ]`
TC 3. `[, A, C, B, D, ]`

Supporting Tool. To support the MBT process, we employed the `FourMA`[1] tool [7]. `FourMA` is an open source MBT tool for mobile Android apps. It supports the ESG modeling, generation of templates for test scripts, and implements a greedy algorithm, named GETAP [5], to derive test cases covering the all-edges criterion. Figure 2 shows an overview of the tool; the tester can visually edit the test model, validate it, and generate test artifacts.

[1] https://github.com/andreendo/FourMA

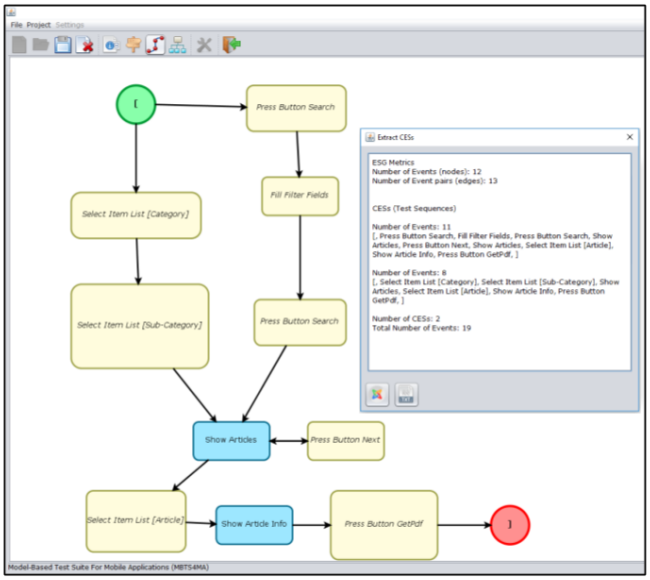

Figure 2: Test model in the FourMA tool and extraction of the respective test cases.

The tester may also bind parameters to any event in the model. When accessing the parameters screen of a given event (Figure 3), the user can manage parameters relevant to the tests. A set of parameters' valuation results in an event instance. Event instances bring more information for the test generation step; currently, FourMA uses event instances to support the generation of test scripts and reduces the effort of concretization.

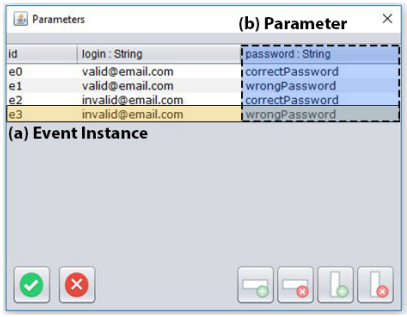

Figure 3: Parameters screen of FourMA.

3 STUDY CONFIGURATION

In order to focus the study and guide the research and analysis stages, the following research question was defined: *"What is the impact on code coverage considering that different testers design test models for the same functionality?"*. The scenario describing the research question is represented in Figure 4, in which given the same specification and test objective, different testers apply MBT. As each tester has its way of interpreting, it is natural to think that the model constructed by one tester will be different from the one designed by another.

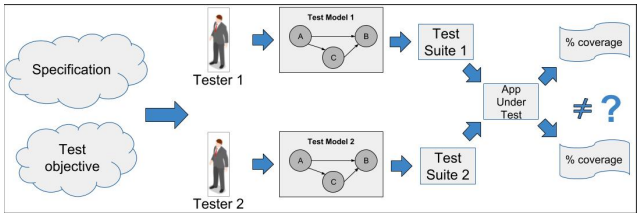

Figure 4: Two different testers designing a test model

3.1 Hypotheses Formulation

Null Hypothesis. The null hypothesis (H_0), which represents an assertion that there is no relation between the phenomena studied, is defined as: "There is no meaningful impact on code coverage when different testers design test models for the same functionality".

Alternative Hypothesis. The alternative hypothesis (or research hypothesis) confronts null hypothesis H_0. For this experiment alternative hypothesis H_1, which represents the statement that H_0 rejects, is defined as: "There is a meaningful impact on code coverage when different testers design test models for the same functionality".

3.2 Variable Selection

To better define the effects and influences observed throughout the experiment, it is necessary to list two types of variables: dependent variable and independent variable.

Dependent variables. Dependent variable or response variable represents a value that changes according to how the independent variable is manipulated. We can list as dependent variables: modeling time, model size (number of nodes, edges, and model elements) and level of code coverage reached by the test model (instructions and branches of the application code).

Independent variables. Independent variables or predictive variables are factors used to measure its influence on the dependent variables. In the current experiment the independent variables are: the test models designed by the participants; the two mobile apps tested in the experiment; functionalities (test goals) defined for participants to be verified.

3.3 Experimental Design

The participants were separated into two groups; each participant created a test model using the FourMA tool for two, open source, Android apps: arXiv-mobile[2] and WhoHasMyStuff[3]. Both apps are available for end users on Google Play and their

[2]https://play.google.com/store/apps/details?id=com.commonsware.android.arXiv.
[3]https://play.google.com/store/apps/details?id=de.freewarepoint.whohasmystuff.

Table 1: Mobile apps size

	# LoC	# Activities	# instructions	# branches
arXiv-mobile	2,093	8	12674	578
WhoHasMyStuff	806	4	3702	258

source code are found in GitHub[4]. The size and complexity of the selected apps can be observed in Table 1.

`ArXiv-mobile` is a "Books and References" app, with an average rating of 4.5 stars and more than 50,000 downloads. It provides a search interface for scientific papers published on the arXiv preprint platform. `WhoHasMyStuff` is a "Productivity" app, with an average rating of 4.3 stars and approximately 5,000 downloads. It provides a management interface for borrowed personal items.

The experiment was divided into two iterations. In the first iteration, Group 1 created the model for the `arXiv-mobile` app and Group 2 for the `WhoHasMyStuff` app. In the second iteration, the opposite happened, according to Table 2. The app chosen in the first iteration diverges between the groups, so that factor learning is taken into account. In both iterations, the participants receive some test goals and created the test models by exploring the apps. The modeling time of each participant was also measured.

Table 2: Experiment design

	1^{st} iteration	2^{nd} iteration
Group 1	`arXiv-mobile`	`WhoHasMyStuff`
Group 2	`WhoHasMyStuff`	`arXiv-mobile`

Next, from the test models, one of the researchers generated, through the `FourMA` tool, the abstract test cases. Then, the test cases were concretized and executed, observing and analyzing the code coverage achieved.

We applied a profile form and a post-experiment questionnaire with the participants. The profile form aims to collect the level of experience with software development and testing. At the end, a questionnaire in the Likert scale was applied to collect the participants' perception of the MBT approach adopted in the experiment. So, three closed questions are about the interaction with the main features of the tool, and an open question for the participants to give suggestions.

3.4 Selection of Subjects

The experiment was carried out at the Federal University of Technology – Paraná (UTFPR), Cornélio Procópio. We selected 12 students that were enrolled in the advanced topics in software testing course. We divided the participants into two random groups, as previously mentioned.

[4]https://github.com/jdeslip/arxiv-mobile and https://github.com/stovocor/whohasmystuff.

3.5 Operation

We used our previous experience on conducting empirical studies and applying MBT to establish the needed time for training and experiment's tasks. The whole experiment was carried out in 250 minutes. Initially, the training session took 100 minutes. Two researchers provided an introduction to topics related to the experiment, namely MBT, ESGs, and the `FourMA` tool. Then, the participants practiced with three examples. The first two examples were the modeling of common systems: a calculator with basic arithmetic operations and a login system. The third example focused on modeling some functions of a more complex system, a mobile app named `AnkiDroid Flashcards`[5].

The experiment was conducted in the remaining 150 minutes. Participants started with the profile form. The participants had at most 75 minutes to design the test model of each app, as defined in Table 2. The models were not created to cover the entire app, but rather only certain features defined in the experiment instructions. At the end of the first iteration, the designed model was validated by `FourMA` (by searching for inconsistencies in the ESG) and submitted on the `Moodle` platform. Subsequently, the participants created the test model for the second app and followed the same procedure. The remaining time was available for the post-experiment questionnaire.

After the experiment, all the models constructed by the participants were collected and validated by one of the researchers. The validation involved verifying if the model was consistent with the behavior of the app tested and covered the established test goals. Among the models, we found only one model whose representation did not matched with the actual app behavior. The participant outlined in his/her model a sequence of events that could not be replicated directly in the app. In this case, we ignored the invalid path, but still considered the valid paths conceived by the participant. In this way, the model was still used for the following analyses.

We also observed that some participants did not use the parameters feature (Section 2); there were eight models without parameters. For such models, we established default values in the concretization phase.

The following tools were adopted. The modeling makes use of `FourMA`, the concretization employs `Robotium` and `JUnit4`. The `Robotium` platform is responsible for simulating user interactions with the app's GUI. Finally, `JaCoCo` was adopted to measure instruction and branch coverage. The data analysis was carried out with the software `RStudio`. Our laboratory package is available to download at:

https://experiment.page.link/mbt2018

3.6 Threats to Validity

This section identifies the threats to validity, categorizing them using the terminology presented by Wohlin et al. [20].

[5]AnkiDroid is an open source app that allows the study by means of memory cards. It is in the "Education" category, with an average rating of 4.5 stars and 32,190 users. Available at: https://play.google.com/store/apps/details?id=com.ichi2.anki

3.6.1 Internal Validity. A possible threat was the allocation of a researcher to concretize the abstract test cases, adding a possible bias to the obtained results. We chose this configuration because the current study scope is the modeling stage of MBT. The presence of a unique individual aimed to decrease the variation in how the test cases are implemented. Some decision on the experiment design might have impacted the results. For instance, the training session could be longer in order to increase the quality of models and reduce the variation.

3.6.2 External Validity. During the modeling process the participants used only the `FourMA` tool together with the ESG technique. The exclusivity in question may include the bias that the current study is not representative of an MBT approach. Other threat is that the participants are students and a big part do not have experience on MBT. While most of them have experience with software development and testing, the results cannot be generalized to professional testers.

3.6.3 Conclusion Validity. An important point to note was the fact that the experiment had 12 participants. This limited the confidence in the relations found and their generalization. The sample size also influences the search for correlations between the data collected, since the main statistical methods demand from a population of considerable size. Based on this, we did not perform statistical tests.

3.6.4 Construct Validity. One relevant construct validity threat is the fact that we chose just mobile apps, with specifics characteristics, possibly introducing a bias. It is possible that this results may not be generalized to other software domains. However, these apps were selected because they are free and open source software, available in F-Droid[6], contain trivial functionalities (e.g. simple items search, insert and update) and are part of other relevant studies [4].

In this paper, we chose, as test effectiveness measure, the code coverage. The relationship between code coverage and fault detection capabilities is a controversial topic and there is no consensus in literature [17]. Despite this, coverage is a widely used metric and it is adopted as a common proxy to estimate test effectiveness [4]. Moreover, we measured the overall coverage, though only some parts of the apps were aimed in the tests. Results based on the features' code coverage could show differences.

4 ANALYSIS OF RESULTS

This section analyzes the results of the experiment.

4.1 Profile of Participants

The experiment had 12 participants. The profile of each participant was drawn from a questionnaire, summarized in Figures 5 and 6. Most participants (75%) said they have a strong experience with mobile apps. In software testing, 58.3% have medium experience and 25% noticeable experience. As graduating students of Software Engineering, they have experienced at least 425 hours on courses and projects involving software modeling (mainly UML).[7] It is important to emphasize that 11 out of 12 participants have already taken a previous 85-hours course on software testing. Notice that no participant has high familiarity with MBT or with mobile development.

Figure 6 shows the software development experience (in years). In this aspect, the participants were similar, where nine of them have 2 or 3 years of experience.

Despite the limited number of participants, the experiment had a sufficient diversity so that the test models could represent the variability of experience and profiles that testers would have *in situ*.

4.2 Quantitative Analysis

Table 3 shows the coverage of instructions and branches[8] achieved by putting together the test suites derived from the 12 test models and executing them in the apps. For the `arXiv-mobile` app, all test suites reached 68% of instruction coverage and 40% of branch coverage. For the `WhoHasMyStuff` app, all test suites achieved 63% of instruction and 40% of branch coverage. The obtained coverage did not reach 100% since not all apps' functionalities were modeled.

Table 3: Total coverage reached when running all test suites derived from the test models.

Mobile Application	Instruction Coverage	Branch Coverage
`arXiv-mobile`	68%	40%
`WhoHasMyStuff`	63%	40%

None of the 12 participants in the experiment were able to achieve, with their test models, the total coverage. This shows that each elaborate model "ran" different paths in the applications, executing different instructions.

This difference between the total coverage and the coverage of each subject provides support for the research hypothesis: "There is a meaningful impact on code coverage from the time different testers design a test model for the same functionality". Besides the total coverage, the research hypothesis can be supported by analyzing other aspects.

> **Finding 1:** No tester was capable of producing a test model that subsumes all other testers' models with respect to instruction and branch coverage.

Figure 7 represents the distribution of instruction and branch coverage obtained by individual test models. For instruction coverage, observe a high variation in the range of values[9]: 17% for `arXiv-mobile` and 18% for `WhoHasMyStuff`. For both apps, the mean, represented by a small square, is

[6] https://f-droid.org/en/.

[7] Estimation based on the Software Engineering curriculum of UTFPR.

[8] In the literature, instruction coverage and branch coverage can be related with the all-nodes and all-edges criteria, respectively.

[9] It represents the dispersion of values, denoted by the difference between the highest value and lowest value.

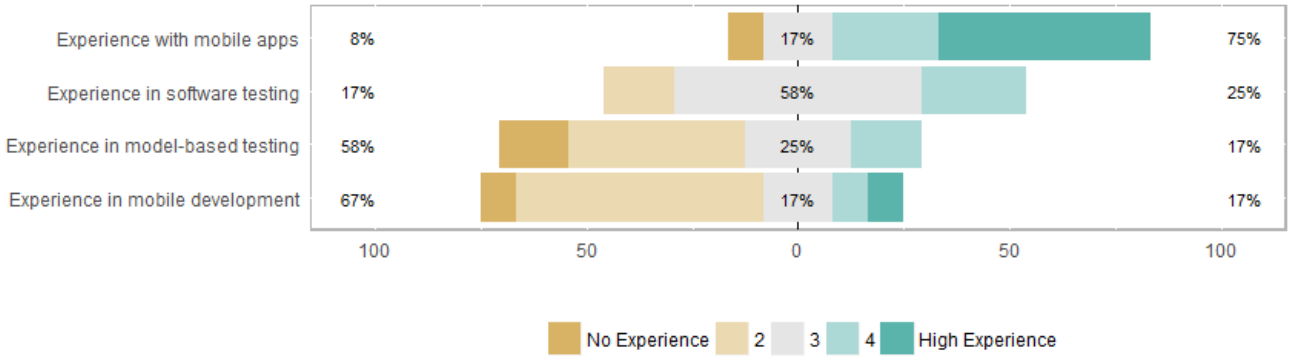

Figure 5: Profile of participants.

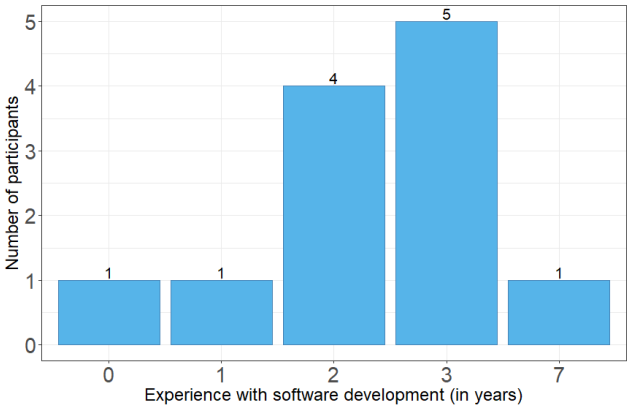

Figure 6: Experience with software development.

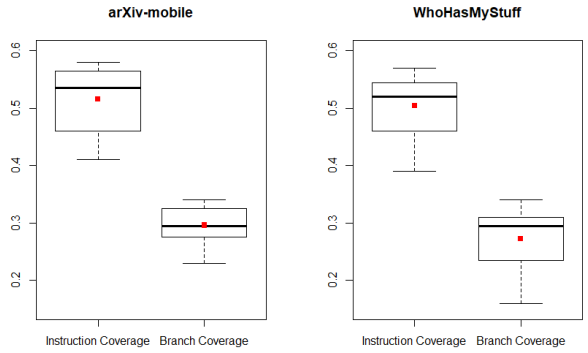

Figure 7: Instruction and branch coverage for `arXiv-mobile` and `WhoHasMyStuff` apps.

in the lower quartile. The effect on branch coverage differs; for `arXiv-mobile`, the mean is in the upper quartile with a variation range of 11%; for `WhoHasMyStuff` the mean remains in the lower quartile, having 18% as variation range. The way the coverage values are scattered denotes once again the research hypothesis, in which different testers imply different test results.

Finding 2: The execution of test suites derived from test models created by each participant results in different coverage values ranging from 11% to 18%.

Figure 8 illustrates how coverage values were dispersed for each iteration. In both iterations, mean values were below the median. For `arXiv-mobile`, the variation for the first iteration was 12% of instruction coverage and 6% of branch coverage; for the second iteration it was 14% and 9%, respectively. For `WhoHasMyStuff`, the variation in the first iteration was 15% of instruction coverage and 10% of branch coverage; the second iteration resulted in 15% for instruction and branch coverage. Notice that the learning factor between the two iterations was not noticeable; the coverage values did not undergo any major increase.

Finding 3: The learning factor of the participants, among the iterations, did not imply an increase in code coverage.

Figure 9 represents the relation between code coverage and modeling time. After plotting the points, we drew a line[10] that indicates the linear function based on the instruction or branch values (y-axis) and modeling time of each participant (x-axis). As for `arXiv-mobile`, there seems to be a trend that the longer the time spent with modeling, the greater the instruction and branch coverage. On the other hand, this behavior was not generalized since the tendency with `WhoHasMyStuff` has shown inverse: the longer the modeling time, the smaller is the instruction and branch coverage.

[10] R language instruction: `abline(lm(y-axis~x-axis))`

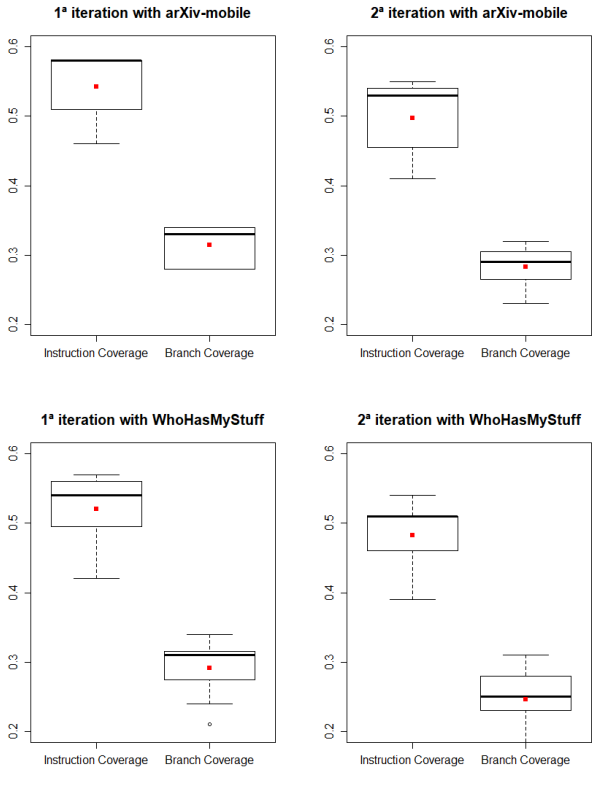

Figure 8: Instruction and branch coverage for `arXiv-mobile` and `WhoHasMyStuff` per iteration.

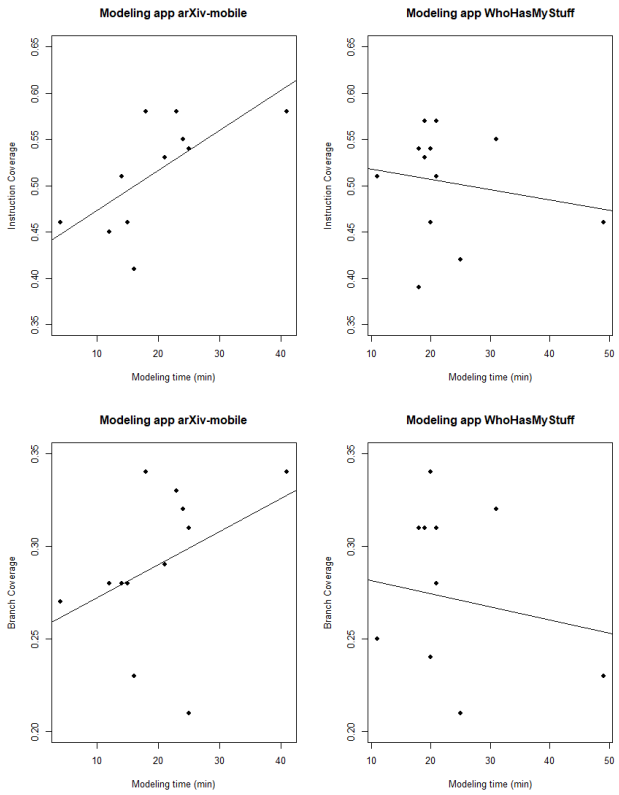

Figure 9: Code coverage and modeling time.

Finding 4: Modeling time had no influence on code coverage.

Figure 10 shows the relation between model size and the branch coverage. As each event in the model mainly represents a function that will be executed in the SUT, it is expected to assume that the coverage will be increase as a function of the number of events, edges, and elements in the model. This behavior is observed by the linear functions drawn in the graphs. The line is more accentuated for `arXiv-mobile` in comparison with `WhoHasMyStuff`. This might be a consequence of the difference in complexity between the apps (see Table 1); the `arXiv-mobile` app has a larger number of lines of code and consequently offers more instructions to execute as the test model grows.

Finding 5: There is a trend that the larger the test model size, the greater the code coverage.

4.3 Post-experiment Questionnaire

The questionnaire had four questions to collect the participants' perception of the techniques and tools used in the experiment. Figure 11 shows the answers to the questions. Most participants approved the use of the `FourMA` tool and the ESG technique for creating test models, categorizing them as appropriate for the activity. Participants also considered as relevant the parameters functionality, making the modeling more complete.

In the open question, the participants gave suggestions on all the concepts involved in the experiment (MBT, ESG, Input Data, `FourMA`). There were nine responses: all suggested improvements in the `FourMA` tool (e.g., add shortcut to undo action, add shortcut to save test template, and so on).

5 DISCUSSION

In this study, we employed code coverage as a proxy to analyze effectiveness. Therefore, we assume that test suites generated from models are supposed to achieve high code coverage in order to detect more faults. Each model tested the mobile app in a nearly unique way, though some parts were covered by all models. The variation from one model to another ranged from 11% to 18%. Considering the controlled context (same modeling technique, tool, SUT, and test goals), we believe these values represent meaningful variations and might harm the systematic adoption of MBT.

We also analyzed other characteristics. First, we observed no significant impact of the learning factor. Our results gave

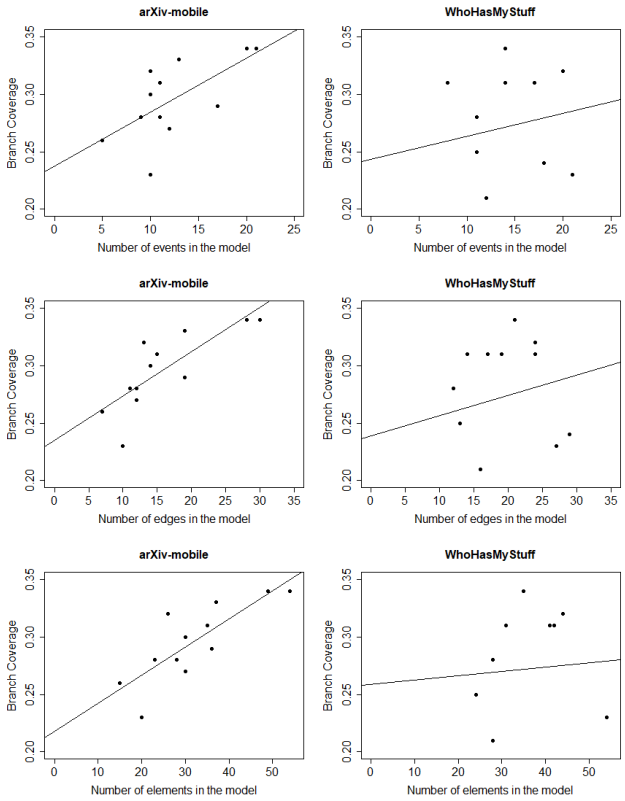

Figure 10: Branch coverage and model size. The model size is presented in three perspectives: *(i)* number of events, *(ii)* number of edges, and *(iii)* number of elements (including vertices, edges, parameters, and event instances).

an initial evidence that the code coverage variation did not seem to reduce with more experience. Nevertheless, we plan to replicate the experiment with senior practitioners. Second, we analyzed the modeling time and model size: modeling time did not seem to be related, while model size had a positive impact on code coverage. By the latter observation, the low code coverage achieved by some testers mainly results from missing elements in the test model. Therefore, one future direction is to help testers to identify missing parts and augment the model in a more accurate and comprehensive test representation.

The variation observed might represent an issue for MBT, making it an approach with low repeability. This has an initial impact on results reported in literature, mostly the empirical studies with a very few testers. The effectiveness with respect to MBT could be under- and/or over-estimated due to the tester's performance at the modeling step. On the other hand, such scenario is also undesirable in practice. A poorly designed model and test cases generated from it might fail to reveal critical faults.

All in all, more empirical studies on MBT (namely, controlled experiments) are required. Essential parts of MBT lack experimental understanding to set out better tools, modeling techniques, and test strategies.

6 RELATED WORK

Gudmundsson et al. [9] validate the MBT adoption by answering the following research question: "Can MBT be used effectively and efficiently in mobile system testing, using the same approach as testing in non-mobile systems?" In their case study, the authors applied MBT in the mobile app `QuizUp`. The MBT steps followed were the same as described previously, while the adopted tools were different. The results show that MBT can be used effectively and efficiently to test mobile applications, using the same approach that is applied to other types of systems. However, a meaningful amount of time was spent understanding the modeling technique, learning about the mobile application environment, and being aware of the functionalities it can offer.

Farto and Endo [7] introduced an MBT tool, named `FourMA`, which was used in this study. They conducted an experimental study to evaluate the modeling strategies of the `FourMA` tool in an industrial context.

Janicki et al. [11] survey the obstacles to applying MBT in mobile applications in industrial settings. In short, the authors argue that more researches should be done to simplify the modeling process. As the biggest obstacle in the research is the design of test models, they suggested to develop more tools that can make the creation and maintenance of the test models as easy and fast as possible.

Takala et al. [18] evaluate a set of supporting tools called `TEMA Toolset`, which aims to automate the GUI testing in Android applications. Through a case study carried out with the BBC News app, 14 bugs were found: eight were discovered during the modeling process and six during test execution.

Entin et al. [6] report the challenges discovered by applying, for several years, MBT in a software for the energy company OMICRON, along with the agile methodology SCRUM. The study defines a process that involves all stakeholders in the definition of the model. The process aims to increase the accuracy and integrity of MBT. In addition, from a simple formal notation to the definition of requirements, less experienced software test engineers were able to generate well-structured parts of the model, with reduced required maintenance effort.

Schulze et al. [16] performed an experiment comparing the effort and efficiency of tests conducted by two testers, on the same two versions of the same SUT. The first tester used a completely manual approach, with no automation of any kind, and the second tester used the MBT approach, automating the generation and execution of test cases. The authors observed that manual testing took less preparation time and detected more inconsistencies in the GUI. MBT spent more time in the preparation (due to the modeling activity), but it was more systematic and detected more functional problems. The results show that MBT detected

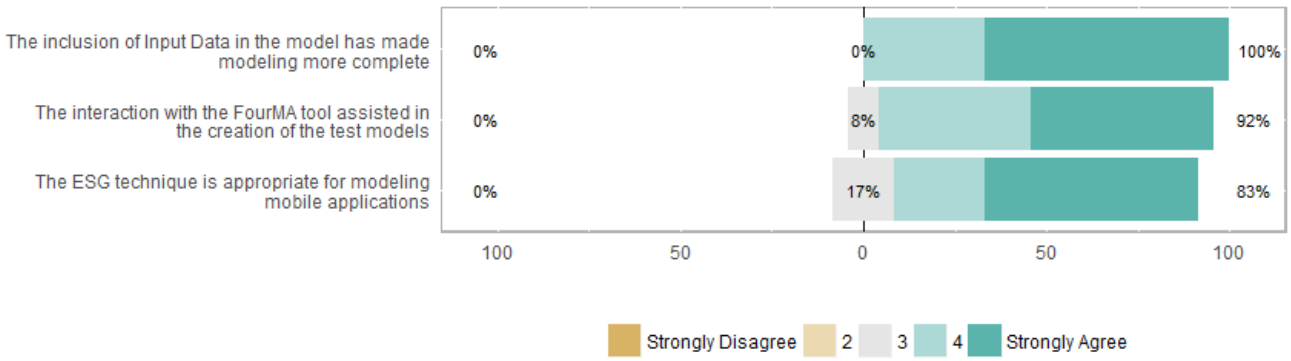

Figure 11: Post-experiment questionnaire results.

more faults with a high severity degree, obtaining a score of severity 60% higher than that obtained by manual tests.

Iqbal et al. [10] conducted a systematic literature review to evaluate the quality and contexts of empirical studies for MBT. After the application of the selection criteria, 87 papers were analyzed, and none of them compare the impact of test models produced by different testers. Although they did not strictly follow the guidelines of a systematic review, the authors were able to map the most important papers in the subject. Based on the literature and to our best knowledge, this is the first experimental study that analyzes the impact of different testers on the MBT's modeling step.

7 CONCLUSION

The objective of this paper was to evaluate the impact of different test models, generated by different individuals, on the MBT effectiveness. Thus, an experiment was conducted with 12 participants using the `FourMA` tool to model two mobile apps. The experiment produced a total of 24 test models; from them, test suites were generated, concretized and executed in the SUTs. The data collection consisted of code coverage reports, and metrics related to the modeling step. The results provide initial evidence to support the alternative hypothesis (H_1) *"There is a meaningful impact on code coverage when different testers design test models for the same functionality"*. We observed variation in code coverage among the participants from 11% to up to 18%. The results gave initial evidence that modeling time and factor learning did not seem to influence code coverage, while model size looks to be positive related to code coverage.

The variables discussed in this paper had their scope limited to the modeling step. However, other dependent variables, present in different steps, can also be analyzed to foment new discussions. For instance, future work may investigate the impact caused when different testers perform the concretization step. We plan to increase the sample size in future replications, as well as investigate forms to reduce the variation at modeling level. Some possible directions are the identification of test modeling guidelines, better tooling, and definition of domain-specific modeling techniques. Finally, an empirical comparison with automated test input generation tools like MobiGUITAR [1] and ORBIT [21] could be defined. Since these tools focus on app crashes while MBT focuses on domain knowledge, it would be interesting to evaluate the impact of those different types of oracle.

REFERENCES

[1] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon. 2015. MobiGUITAR: Automated Model-Based Testing of Mobile Apps. *IEEE Software* 32, 5 (Sept 2015), 53–59. DOI:https://doi.org/10.1109/MS.2014.55

[2] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. Mcminn, A. Bertolino, and others. 2013. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software* 86, 8 (2013), 1978–2001.

[3] F. Belli, C. J. Budnik, and L. White. 2006. Event-based modelling, analysis and testing of user interactions: approach and case study. *Software Testing, Verification and Reliability (STVR)* 16, 1 (2006), 3–32. DOI:https://doi.org/10.1002/stvr.335

[4] S. R. Choudhary, A. Gorla, and A. Orso. 2015. Automated Test Input Generation for Android: Are We There Yet? (E). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Washington, DC, USA, 429–440. DOI:https://doi.org/10.1109/ASE.2015.89

[5] A. T. Endo and A. Simao. 2017. Event tree algorithms to generate test sequences for composite Web services. *Software Testing, Verification and Reliability* (2017). DOI:https://doi.org/10.1002/stvr.1637

[6] V. Entin, M. Winder, B. Zhang, and A. Claus. 2015. A process to increase the model quality in the context of model-based testing. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 1–7. DOI:https://doi.org/10.1109/ICSTW.2015.7107471

[7] G. C. Farto and A. T. Endo. 2017. Reuse of Model-based Tests in Mobile Apps. In *Proceedings of the 31st Brazilian Symposium on Software Engineering (SBES'17)*. ACM, 184–193. DOI:https://doi.org/10.1145/3131151.3131160

[8] Vahid Garousi and Frank Elberzhager. 2017. Test automation: not just for test execution. *IEEE Software* 34, 2 (2017), 90–96.

[9] V. Gudmundsson, M. Lindvall, L. Aceto, J. Bergthorsson, and D. Ganesan. 2016. Model-based Testing of Mobile Systems – An Empirical Study on QuizUp Android App. In Proceedings First Workshop on *Pre- and Post-Deployment Verification Techniques*. 16–30. DOI:https://doi.org/10.4204/EPTCS.208.2

[10] Muhammad Zohaib Iqbal, Salman Sherin, and others. 2017. Empirical studies omit reporting necessary details: A systematic literature review of reporting quality in model based testing. *Computer Standards & Interfaces* (2017).

[11] Marek Janicki, Mika Katara, and Tuula Paakkonen. 2012. Obstacles and Opportunities in Deploying Model-based GUI Testing of Mobile Software: A Survey. *Software Testing, Verification and Reliability (STVR)* 22, 5 (Aug. 2012), 313–341. DOI: https://doi.org/10.1002/stvr.460

[12] A. Kramer, B. Legeard, and R. V. Binder. 2017. 2016 / 2017 Model-based Testing User Survey: Results. (2017). http://www.cftl.fr/wp-content/uploads/2017/02/2016-MBT-User-Survey-Results.pdf

[13] Bruno Legeard. 2010. Model-based testing: Next generation functional software testing. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[14] Zoltán Micskei. 2014. Model-based testing (MBT). *Department of Measurement and Information Systems Budapest University of Technology and Economics http://mit. bme. hu/˜ micskeiz/-pages/modelbased_testing. html", Accessed* (2014), 12–08.

[15] A. Orso and G. Rothermel. 2014. Software Testing: A Research Travelogue (2000–2014). In *Proceedings of the on Future of Software Engineering (FOSE)*. ACM, 117–132.

[16] C. Schulze, D. Ganesan, M. Lindvall, R. Cleaveland, and D. Goldman. 2014. Assessing model-based testing: an empirical study conducted in industry. In *The 36th International Conference on Software Engineering Companion*. ACM, 135–144.

[17] A. Schwartz and M. Hetzel. 2016. The Impact of Fault Type on the Relationship between Code Coverage and Fault Detection. In *2016 IEEE/ACM 11th International Workshop in Automation of Software Test (AST)*. 29–35. DOI:https://doi.org/10.1109/AST.2016.013

[18] Tommi Takala, Mika Katara, and Julian Harty. 2011. Experiences of system-level model-based GUI testing of an Android application. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*. IEEE, 377–386.

[19] Mark Utting, Alexander Pretschner, and Bruno Legeard. 2006. A taxonomy of model-based testing. (2006).

[20] C Wohlin, P Runeson, M Host, MC Ohlsson, B Regnell, and A Wesslen. 2000. Experimentation in software engineering: an introduction. 2000. (2000).

[21] Wei Yang, Mukul R. Prasad, and Tao Xie. 2013. A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications. In *Fundamental Approaches to Software Engineering*, Vittorio Cortellessa and Dániel Varró (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 250–265.