An Experimental Evaluation of Approaches to Feature Testing in the Mobile Phone Applications Domain

Laisa H. O. do Nascimento, Patricia D. L. Machado
GMF/DSC - Universidade Federal de Campina Grande (UFCG)
Caixa Postal 10.106 - 58109-970
Campina Grande - PB - Brasil
{laisa, patricia@dsc.ufcg.edu.br}

ABSTRACT

Software engineering is a discipline that cannot be applied based solely on elegant theories. Real software production requires software solutions that may apply a mixture of engineering and ad-hoc practices in a systematic way. In this sense, experimentation is staple for identifying the best practices and solutions for a given software development problem. This paper presents results from an experimental evaluation of the use of model-based testing and the use of exploratory testing in the context of feature testing for mobile phone applications. The study is based on the Goal/Question/Metric paradigm. From the results obtained and conclusions reached, an approach to feature testing is proposed.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing strategies*; D.2.8 [**Software Engineering**]: Metrics/Measurement—*Process metrics*

General Terms

Experimentation, Measurement, Software Testing

Keywords

Feature Testing, Model-Based Testing, Exploratory Testing, GQM Paradigm

1. INTRODUCTION

In recent years, the development of mobile phone applications has become more and more complex: at the same time that time-to-market decreases the development time it increases the demand for the quality level of products. The necessity of quality assurance intensifies the use of software testing.

In this paper, we focus on the domain of interactive features. This domain is characterized by applications, composed of a number of features, that are highly interactive, having their flow of execution guided mostly by external input. These applications are often tested manually. Also, the gap between specification and program is narrow, since the logics are captured by external interactions. In this context, feature testing (FT) is a crucial testing activity. A feature is a set of individual requirements that describes some functionality. Alarm Clock, Phonebook and Messaging are example of features usually found in mobile phone applications. A functional understanding of these applications is more effectively achieved by investigating user interface tasks behaviour.

Because features are usually developed and tested either in isolation or within the context of a particular service [10], FT is very important to help to reduce the number of defects that escape from one phase to other during the development and testing processes.

Along with the usual challenges to functional testing, FT for mobile phone applications has some particularities:

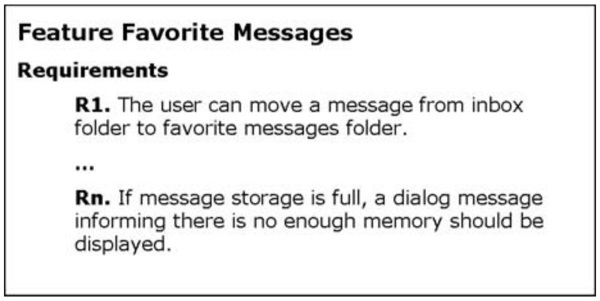
- FT needs to be extensively executed - Due to the way a feature is developed and integrated with different applications, it is fundamental that its behavior is in accordance with requirements. Thus, the feature has to been thoroughly tested, maximizing defect detection.
- FT needs to be rapid - Time-to-market demands development time to be minimized and, consequently, feature testing needs to be executed with minimum time requirements.
- FT depends on deep requirements knowledge - FT is a kind of functional testing, so it is necessary to acquire an adequate level of knowledge of requirements and also of the application domain to devise and execute test cases. Moreover, it is usually necessary to know potential applications behavior because the feature can interact with others features.
- FT cases are executed several times - Feature test cases can be re-executed several times during its development cycle. Normally there is more than one test cycle with variants of the same test suite.

Considering these aspects, what functional testing approach is more appropriate to be used in FT for mobile phone applications? In attempt to answer this question, we conducted an experimental evaluation of the use of Model-Based Testing and the use of Exploratory Testing to test mobile phone features. Both approaches present advantages that make them potential candidates for FT. Nevertheless, it is unclear which one is better suited for FT and what are the forces that guide their successful adoption in the mobile phone applications domain. The experimentation is based on the Goal/Question/Metric paradigm. An evaluation model is proposed for comparing the approaches. Also, it applies the testing approaches to real features. From the experimentation results obtained and conclusions reached, an approach to feature testing is suggested.

The rest of the paper is organized as follows. In the next section, we present examples of a feature requirement and a test case. Sections 3 and 4 present the testing approaches used in this study. Section 5 describes the experimentation methodology used, while Section 6 presents the results obtained. Finally discussions and conclusions are presented in Sections 7 and 8.

2. FEATURE

Mobile phone applications are composed of features. A feature is a clustering of individual requirements that describe a cohesive, identifiable unit of functionality [21]. These requirements are described in documents that specifies the behavior of a particular feature. For example, consider a feature named “Favorite Messages” that consists in move some message from inbox folder to favorite messages folder. Figure 1 shows a fraction of a requirements document for this feature.


Feature Favorite Messages

Requirements

R1. The user can move a message from inbox folder to favorite messages folder.

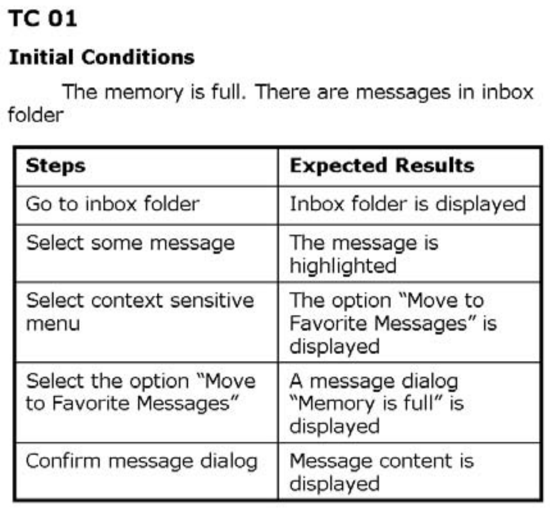
...

Rn. If message storage is full, a dialog message informing there is no enough memory should be displayed.

Figure 1: A sample of feature requirements

An important characteristic of a feature is that its development happens in an evolutionary manner. In practice, a feature can evolve to another feature or it can have some specific functionality described by other feature as a result of interaction. This characteristic requires a good knowledge of the feature requirements and the likely requirements of the features to be affected. Because features are developed in isolation and in an evolutionary way, FT is so important.

As mentioned before, FT is a kind of functional testing that has some particularities. A feature test case is a traditional test case, with initial conditions, steps and expected results (Figure 2). However, a FT test case needs to be abstract since FT execution is usually manual. This is due to the fact that mobile phone applications are highly interactive. Also, most of the functionalities cannot be properly checked by automated test cases as they require human intervention. Moreover, their execution is not feasible, mainly, due to technological barriers.


TC 01

Initial Conditions

The memory is full. There are messages in inbox folder

Steps	**Expected Results**
Go to inbox folder	Inbox folder is displayed
Select some message	The message is highlighted
Select context sensitive menu	The option “Move to Favorite Messages” is displayed
Select the option “Move to Favorite Messages”	A message dialog “Memory is full” is displayed
Confirm message dialog	Message content is displayed

Figure 2: An example of a feature test case

3. MODEL-BASED TESTING

Model-based testing (MBT) is a testing approach where common testing tasks are based on a model of the application under test [16]. Basically, MBT is composed of the following phases (Figure 3):

1. Model building - In this phase, a mental representation of the system’s requirements is formed and mapped to the model.

2. Test cases generation - Test cases are derived from the model. The automation of this phase depends on the nature of the model.

3. Test cases execution - Consists in running the test cases generated previously.

4. Results analysis - Execution time and test cases results are analyzed for model improvement.

One of the advantages of MBT is the possibility to automatize test case generation because, as test cases are generated automatically, the consistency between test cases and requirements can be maintained effortless [12]. This is crucial for mobile phone applications due the fact that feature requirements are continuously changing to meet new demands of integration with different applications of product families. Besides, the automation can reduces the cost of the testing process since it reduces the time of test cases creation [17]. Other advantage of using MBT is the possibility of reusing development artifacts, particularly requirements documents [4].

In the context of feature test for mobile phone applications, MBT, as it seems, is a promising approach, once the time of the testing process can be reduced over automatic test case generation, and feature test needs to be rapid. For instance, Figueiredo et al [14] presents a model-based testing approach to feature interaction testing where features and interactions are specified as use cases that are translated into models for test case generation.

Another point that favours the application of MBT in this context is requirements coverage. Considering the hypothesis that the model cover 100% of the requirements, the test

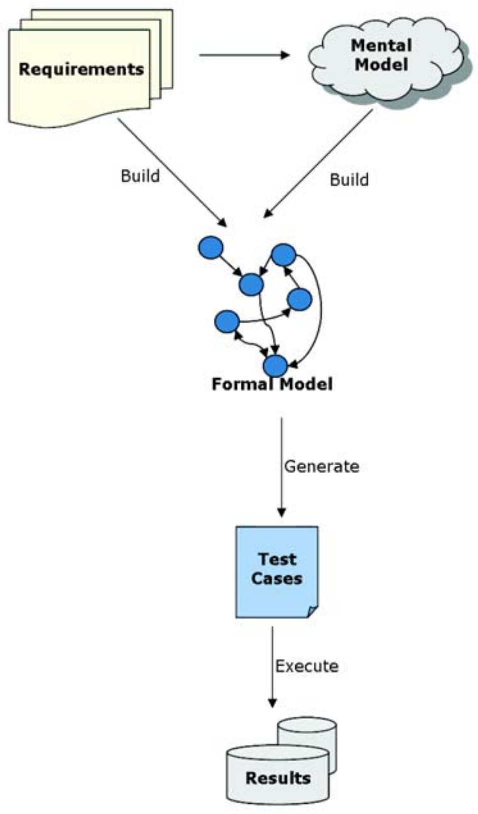


Figure 3: Model-Based Testing

cases generated automatically will probably cover too. This fact guarantees that there are test cases for all requirements described on requirements documents.

4. EXPLORATORY TESTING

The term exploratory testing (ET) was first published by Kaner in his book *Testing Computer Software* to distinguish exploratory testing and ad hoc testing. The ad hoc testing is like a subtype of exploratory, where there are not annotations to re-execute the tests [1]. ET can be defined as a testing approach where design and execution of test cases occur simultaneous. In other words, ET is any testing where the tester controls the design while tests are performed and uses the knowledge acquired to design better tests [3].

Exploratory testing complements other testing approaches [18]. Some works have pointed out the advantages of its use, like effectiveness, efficiency, and rapid feedback [2, 3]. However there are no scientific evidences about this [18].

Bach [2] presents a methodology to use exploratory testing in test sessions. A test session can be defined as a predetermined time period where a tester tests the application according to an objective, also pre-determined, and makes a report with information about bugs, tester, and test time among other details.

Considering the advantages of using ET, it seems that it is a good approach to use in FT. By Using ET the tester uses his skills. This fact increases the probability of detecting hidden faults more easily than using other testing approaches, which can reduce the number of defects and improves the effectiveness of FT.

Moreover, as mentioned before, feature test cases execution is usually manual due to the interactive nature of mobile phone applications. This is particularly suitable for strategies such as ET.

Finally, another characteristic of ET is that the tester acquires knowledge about the system during test execution. As FT demands deep requirements knowledge, this characteristic increases our feeling that ET is a good approach for feature test.

5. EXPERIMENTAL METHODOLOGY

In this section, we present an experimental methodology to evaluate testing approaches. The focus is on evaluating the use of exploratory testing and the use of model-based testing in feature testing for mobile phone applications.

Experimentation in Software Engineering is important to discover how some techniques performs, discover its limitations and understand how to improve them [6]. When we wish to evaluate some technique or process, it is necessary to follow some measurement model that provides the mechanisms to conduct this evaluation. Some mechanisms for defining measurable goals have appeared in the literature: the Software Quality Metrics Approach (SQM) [8], the Quality Function Deployment Approach (QFD) [19], and the Goal/Question/Metric Paradigm (GQM) [5, 7]. Because GQM allows the evaluation of the object of study [5], we choose its for defining our measurement model.

The GQM paradigm is a mechanism for defining and evaluating a set of operational goals using measurements [5, 7]. A measurement model is defined into three levels: conceptual (goal), operational (question), and quantitative (metric).

We use the Goal Question Metric (GQM) paradigm for defining our measurement model. Firstly, we defined our goal: *Evaluate the use of exploratory testing and the use of model-based testing in feature testing for mobile phone applications*. After that, we defined the questions:

- What is the effort in using each approach?
- How efficient is each approach?
- How relevant are the detected faults when following each approach?
- Do the approaches find the same faults?

Finally, we defined the metrics to be consider in order to find out answers to these questions:

1. **Effort** - The effort in testing T involves the time required to apply the approach. Thus, T is calculated as

$$T = T_s + T_e + T_a,$$

   where T_s is the time for setting up the phone, choosing the feature, looking for requirements documents, generating the model and the test cases when using MBT, and defining test plan when using ET; T_e is the time for test execution; T_a is the time spent on results analysis (not in metrics calculation).

2. **Efficiency** - The efficiency EFF is related to the ratio of detected faults. So, EFF is given by

$$EFF = \frac{N_f}{T},$$

where N_f is the number of faults detected during test execution, T is the effort in testing.

3. **Relevance** - The relevance R of an approach in the context of FT for mobile phone applications is associated with the relevance of faults detected using it. We calculate R as follows:

$$R = \frac{\sum R_f}{N_f},$$

where N_f is the number of faults detected during test execution; R_f is the relevance of a fault according to some criterion. We suggest the criterion bellow:

   - 1 - A transparent problem is invisible to the customer. Example: bad layout.
   - 2 - A minor problem that does not impede the user from accomplishing the desired function. Example: error messages aren't very clear.
   - 3 - A moderate problem that impedes, but does not prevent, the user from accomplishing the desired function. Example: user data must be modified to work.
   - 4 - A serious problem that produces intermittent loss of function or degraded performance. Example: can't use major product function.
   - 5 - A critical problem that renders the work product unfit for use and/or unable to be serviced. Example: system crash.

4. **Faults similarity** - The faults similarity $F_{A \bigcap B}$ is the intersection between the set of faults detected using the testing approach A and the set of faults detected using the testing approach B. $F_{A \bigcap B}$ is calculated as

$$F_{A \bigcap B} = F_A \bigcap F_B,$$

where F_A is the set of faults detected using A, F_B is the set of faults detected using B. Faults similarity can also be defined as:

$$\frac{F_{A \bigcap B}}{N_{fA} + N_{fB}} * 100\%,$$

where N_{fA} is the number of defects detected using A, N_{fB} is the number of defects detected using B.

Finally, we determined the mechanisms to collect the metrics defined above. Since the testing approaches have different tasks, we define a sequence of activities for each one. For model-based testing the tasks are:

1. Feature selection - This task is composed by two subtasks:
   - Requirements documents searching - For choosing a feature to conduct an experiment, it is necessary that it has some characteristics (size, complexity). These characteristics are analyzed by reading requirements documents; however sometimes these documents are not available.
   - Phone setting up - After selecting the requirements documents, it is necessary to verify if there is an available phone with this feature implemented in its software. If there is no phone, another feature has to be chosen.

   At the end of this task, the feature has been selected.

2. Model generation - The behavioral model generation consists in mapping the functional requirements for a formal or semi-formal notation that represents the feature behavior. This task can be automated. For instance, Cabral [9] presents a method to generate formal specification from requirement documents.

3. Test cases generation - From the behavioral model defined previously, the test cases are generated. This task can be automated, depending on the nature of the model. Cartaxo et al and Nogueira [11, 13] present tools to generate test cases automatically from LTSs [1] and CSP [2] models respectively.

4. Test cases execution - Consists in executing the test cases. A logger tool can be used to capture logs that can help in results analysis.

5. Results analysis - This task concerns analysis of test cases execution results. The metrics defined previously are calculated.

Considering that in exploratory testing we do not generate the behavioral model, the following tasks are suggested:

1. Feature selection - This task is performed in the same way as for model-based testing

2. Test plan definition - Consists in reading documents and writing test plans. The test plan defines test sessions (see Section 4 for more details about test session content).

3. Test plan execution - This task consists in executing the test sessions defined in the test plan. A logger tool can be used to capture the traces that can help in test cases re-execution and in results analysis.

4. Results analysis - Consists in reading the test sessions annotations and reporting the defects found. The metrics defined previously are calculated

Tables 1 and 2 show the tasks and the metrics that will be collected at the end of the application of each approach.

[1]Labelled Transition Systems (LTSs) provide a formalism to specify, model, analyze and reason about system behaviour [15].

[2]Communicating Sequential Processes (CSP) is a formal specification language primarily designed to describe the behavior of concurrent and distributed systems [20].

Table 1: Tasks when using model-based testing and metrics to be collected.

Task	Metric
Feature selection	T_s
Model generation	T_s
Test cases generation	T_s
Test cases execution	T_e, N_f, R_f
Results analysis	T T_a, EFF, R, $F_{A \bigcap B}$

Table 2: Tasks when using exploratory testing and metrics to be collected.

Task	Metric
Feature selection	T_s
Test plan definition	T_s
Test plan execution	T_e, N_f, R_f
Results analysis	T, T_a, EFF, R, $F_{A \bigcap B}$

6. EXPERIMENTAL EVALUATION

In this section, we describe an experiment to evaluate two testing approaches: model-based testing and exploratory testing, in the context of feature testing for mobile phone applications. The goals of this experiment were to validate and to improve the measurement model used and to quickly evaluate the use of these testing approaches in feature testing. Therefore, as a preliminary experiment, we chose a small sample of features. The experiment was performed by one person with knowledge about how MBT and ET are used in feature testing.

The methodology defined in Section 5 was used to conduct this experiment. First, we applied the tasks suggested to model-based testing and collected the necessary data. After, we applied the tasks suggested to exploratory testing and also collected the data. Finally, we analyze the results and calculated the metrics defined previously.

We chose two small features, each one having two requirements documents. These requirements documents have about twenty pages each. The chosen features are common in mobile phone applications, but due to their secrecy, we do not show their details.

Tables 3 and 4 show the data collected. Once the annotations obtained applying ET need to be analyzed and validate (verify if the detected faults were really faults for example), T_a is not equal to 0. Using MBT, there was generated 11 test cases for feature A and 7 test cases for feature B.

Table 3: Data collected using MBT.

	T_s (min)	T_e (min)	T_a (min)	N_f
Feature A	127	10.44	0	0
Feature B	146	22.15	0	0

Tables 5 and 6 summarize the obtained results. For feature A (Table 5), the testing approaches did not detect problems, however we can see that the effort applying ET is smaller than applying MBT. Analyzing the results for feature B (Table 6), we can notice that ET detected one fault with associated relevance equals to one. Moreover, in this case the effort in using ET was smaller than using MBT.

Table 4: Data collected using ET.

	T_s (min)	T_e (min)	T_a (min)	N_f
Feature A	78	37	13	0
Feature B	77	39	12	1

Table 5: Experiments results Feature A.

	MBT	ET
Effort (min)	137.44	128
Efficiency ($\frac{N_f}{h}$)	0	0
Relevance	$\perp$	$\perp$
Faults similarity	$\varnothing$	$\varnothing$

Table 6: Experiments results Feature B.

	MBT	ET
Effort (min)	168.15	128
Efficiency ($\frac{N_f}{h}$)	0	$\simeq 0.46$
Relevance	$\perp$	1
Faults similarity	$\varnothing$	$\varnothing$

During the execution of the tasks, we used internal Motorola tools for getting log files and generating test cases automatically from the model. Table 7 shows which tasks were automated.

It is important to remark that for MBT we cover 100% of the requirements. Regarding the features implementation coverage, ET covered more states than MBT. Actually, this leads to uncover one fault. The feature implementations considered in the experiment had already been tested by other teams and the defects had been fixed.

This explains why only one fault has been detected. Our goal when choosing stable features was to access the ability of the approaches in uncovering escaped defects from their usual testing cycle, once escaped defects are a real problem in feature testing.

7. ANALYZING RESULTS

Although the study was performed on a small sample, our feeling is that, in an initial phase, exploratory testing is more indicated than model-based testing for feature testing, considering the mobile phone application domain. We had many difficulties to formalize the requirements when applying MBT because, most of the time it is necessary to have a previous knowledge about the application behavior and the requirements documents did not have information about this behavior. So, as ET can increase the knowledge about the application behavior, we believe that its use in an initial phase can be more effective and feasible.

The effort in applying exploratory testing is clearly smaller than the effort in applying MBT. In our experiment, we use the template presented in Cabral [9] to generate the model

Table 7: Task automation applied in the experiment.

Model-Based Testing	
Task	**Automatic**
Feature selection	No
Model generation	Yes
Test cases generation	Yes
Test cases execution	No
Results analysis	No
Exploratory Testing	
Task	**Automatic**
Feature selection	No
Test plan definition	No
Test plan execution	No
Results analysis	No

automatically when applying MBT. This way, we needed to specify the requirements again using the presented template. If the requirements were already defined using the template, the effort in applying MBT will decrease.

Actually, a single fault was detected by applying ET only. Although this detected fault does not offer basis for generalization, our feeling is that ET is more suitable for FT than MBT. However, ET does not support properly the re-execution of test cases, since they are not documented as for MBT.

Therefore, in a second moment, the use of model-based testing seems to be more interesting due to the many cycles that compose a feature testing. By using MBT, the test cases can be automatically generated and updated, decreasing the costs of the testing process and making it easier for specific test cases to be selected based on certain criteria.

Figure 4 summarizes our idea for a feature testing approach. In an initial phase, exploratory testing is applied and execution logs are saved automatically. These logs can be used to help to construct a model of the feature. Then, during further test cycles, model-based testing is applied. The contributions of exploratory testing are basically the execution logs and the knowledge acquired during test sessions. The former can contribute to complete or to improve the formal model, whereas the latter can contribute in the description of requirements using a controlled natural language (CNL), that makes it possible for an automatic mapping of requirements to a formal model [9]. Because feature testing needs to be rapid, automatic test cases generation provided by model-based testing is an useful characteristic along test cycles.

This approach is a junction of two testing approaches, so probably there is no major costs associate with its use, once the steps are applying the activities from MBT and from ET. Besides this, some phases can be automated. While the tester is doing exploratory testing, a logger tool gets the log. This log contains information about the system state during test execution and can complement the formal model. The model generation can be automatic using a CNL to specify the requirements. The test cases can be generated automatically using some tool that receives the formal model as input.

The evaluation of this approach can be conducted through experiments to verify if its use increases the number of detected faults and decreases the effort needed in a testing process. The results can be compared with the results of using other approaches.

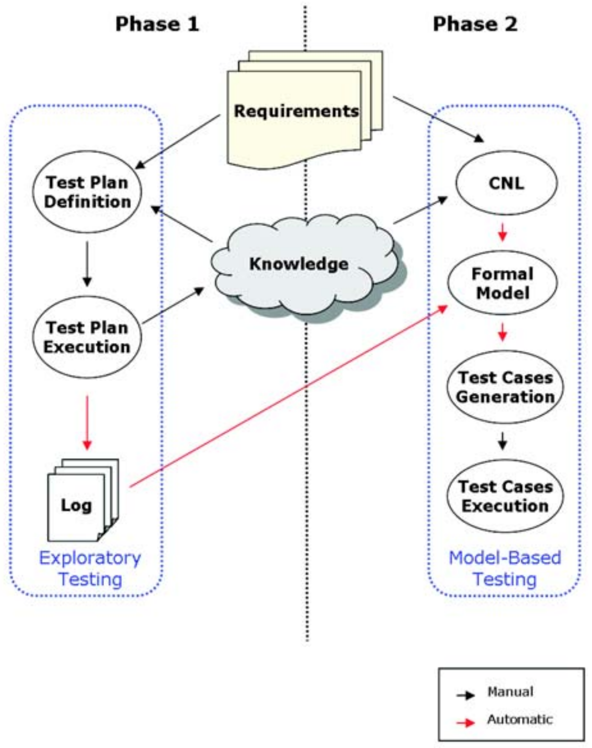


Figure 4: Feature testing approach

8. CONCLUSIONS

This paper presented the results of an experimental evaluation of two testing approaches for feature testing: model-based testing and exploratory testing. We presented an experimental methodology based on GQM paradigm to define a measurement model to evaluate the two testing approaches. After that, we conducted an experimental evaluation using the presented methodology. At first instance, exploratory testing produced better results than model based testing. Finally, based on metrics and initial impressions collected from the experiment, we proposed a combined approach to feature testing by using exploratory testing and model-based testing.

The next steps will then be to re-apply the methodology to other features to improve our measurement model and validate our reflections about the use of exploratory testing and model-based testing in feature test. Also, we need to conduct an experimental evaluation of the suggested feature testing approach. For this, the evaluation model will be extended to take the existence of a number of testing cycles into account.

9. ACKNOWLEDGMENTS

This work has been developed in the context of a research cooperation between Motorola Inc., CIn-UFPE/Brazil and UFCG/Brazil. We thank the entire group for all the support, criticisms and suggestions throughout the development of this research. This work is also supported by FAPESQ/ CNPq (Projeto 060/03 e Processo CNPq 550466/2005-3).

10. REFERENCES

[1] C. Agruss and B. Johnson. Ad hoc software testing: A perspective on exploration and improvisation. *http://www.testingcraft.com/ad_hoc_testing.pdf*, 2000. Accessed in May 28th, 2007.

[2] J. Bach. Session-based test management. *Software Testing Quality Engineering magazine*, vol. 2(no. 6), 2000.

[3] J. Bach. Exploratory testing explained. *http://www.satisfice.com/articles/et-article.pdf*, 2003. Accessed in May 28th, 2007.

[4] D. L. Barbosa, H. S. Lima, P. D. L. Machado, J. C. A. Figueiredo, M. A. Juca, and W. L. Andrade. Automating functional testing of components from uml specifications. *Int. Journal of Software Eng. and Knowledge Engineering*, 2007. To appear.

[5] V. R. Basili. Software modeling and measurement: the goal/question/metric paradigm. Technical report, College Park, MD, USA, 1992.

[6] V. R. Basili. The role of experimentation in software engineering: past, current, and future. In *ICSE '96: Proceedings of the 18th international conference on Software engineering*, pages 442–449. IEEE Computer Society, 1996.

[7] V. R. Basili, G. Caldiera, and H. D. Rombach. The goal question metric approach. *Encyclopedia of Software Engineering*, 1:528–532, 1995.

[8] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, pages 592–605, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[9] G. Cabral and A. Sampaio. Formal specification generation from requirement documents. In *SBMF 2006: Proceedings of the Brazilian Symposium on Formal Methods*, pages 217–232, 2006.

[10] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. Feature interaction: a critical review and considered forecast. *Comput. Networks*, 41(1):115–141, 2003.

[11] E. G. Cartaxo. Geração de casos de teste funcional para aplicações de celulares. Master's thesis, COPIN - Universidade Federal de Campina Grande, 2006.

[12] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 285–294, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[13] S. de Carvalho Nogueira. Geração automática de casos de teste csp orientada por propósitos. Master's thesis, CIn - Universidade Federal de Pernambuco, 2006.

[14] A. L. L. de Figueiredo, W. L. Andrade, and P. D. L. Machado. Generating interaction test cases for mobile phone systems from use case specifications. *SIGSOFT Softw. Eng. Notes*, 31(6):1–10, 2006.

[15] R. G. de Vries and J. Tretmans. On-the-fly conformance testing using SPIN. 2(4):382–393, Mar. 2000.

[16] I. K. El-Far and J. A. Whittaker. Model-based software testing. *Encyclopedia on Software Engineering*, 2001.

[17] A. Hartman and K. Nagin. The agedis tools for model based testing. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 129–132, New York, NY, USA, 2004. ACM Press.

[18] J. Itkonen and K. Rautiainen. Exploratory testing: A multiple case study. In *ISESE 2005: Proceedings of the International Symposium on Empirical Software Engineering*. IEEE Computer Society, 2005.

[19] M. Kogure and Y. Akao. Quality function deployment and cwqc in japan. Quality Progress, 1983.

[20] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

[21] C. R. Turner, A. L. Wolf, A. Fuggetta, and L. Lavazza. Feature engineering. In *IWSSD '98: Proceedings of the 9th international workshop on Software specification and design*, page 162. IEEE Computer Society, 1998.