Model Based Test Generation: A Case Study

Sidney C. Nogueira[1,3], Emanuela G. Cartaxo[2,3], Dante G. Torres[1,3], Eduardo H. S. Aranha[1,3], Rafael Marques[1,3]

[1]Centro de Informática (CIn) – Universidade Federal de Pernambuco (UFPE) Caixa Postal 7851 – 50732-970 – Recife – PE – Brazil

[2]Departamento de Sistemas e Computação (DSC) – Universidade Federal de Campina Grande (UFCG) Caixa Postal 10.106 – 58.109-970 – Campina Grande – PB – Brazil

[3]Mobile Devices R&D Motorola Industrial Ltda Rod SP 340 – Km 128,7A 13820 000 – Jaguariuna/SP – Brazil

scn@cin.ufpe.br, emanuela@dsc.ufcg.edu.br, {dgt,ehsa,rm2}@cin.ufpe.br

Abstract. *Aiming at improving software quality, as well as reducing the costs of testing process, we developed TaRGeT, a Model-Based Testing (MBT) tool that automatically generates test cases from use case scenarios written in natural language. In this paper we present TaRGeT and show quantitative and qualitative results of the tool's usage in two case studies inside Motorola Brazil Test Center (BTC).*

1. Introduction

Testing activity is crucial to the success of a software project. In general, this activity requires about 50% of software development resources. Model-based testing appears as a promising approach to control software quality as well as to reduce the inherent costs of a test process [El-Far and Whittaker 2001].

In this paper, we will report real use of a tool for test case generation. This tool generates test cases from use case scenarios written in natural language. Then, it is not necessary any additional expertise in formal notations to write the application's model.

This work was developed by the Test Research Project, a partnership between Motorola Brazil Test Center (BTC), CIn-UFPE and DSC-UFCG. The project's goal is to automatize the test case generation, selection and evaluation of Motorola mobile phone applications. This paper is organized as follows. Section 2 presents the tool, Section 3 shows the case studies, Section 4 discusses the use of MBT and TaRGeT in practice and, finally, Section 5 presents our conclusions.

2. TaRGeT Tool

Test and Requirements Generation Tool (TaRGeT) is a tool for automatic test case generation from use case scenarios written in Natural Language (NL). TaRGeT automates a systematic approach for dealing with use cases scenarios and test artifacts in an integrated way. The tool adapts the approach presented in [Cabral and Sampaio 2006] in order to translate use case specifications to LTS models.

The possible scenarios are specified as use cases using NL and a template that supports automatic processing. An example of a use case is showed in Figure 1. The use

case is called “Moving a message from inbox to the Hot Messages folder”. The main flow (Figure 1a) describes a scenario in which the user selects his favorite message and moves it to the hot message folder. The alternative flow (Figure 1b) describes a scenario in which the message is not moved to hot message folder because the message storage is full. The flows are described through steps, identified by a **Step Id**, and composed by a **User Action**, the respective **System Response** and **System State** (the necessary conditions to occur the system response).

Main Flow

Description: The message is moved to Hot Messages folder

From Step: START

To Step: END

Step Id	User Action	System State	System Response
1M	Go to "Message Center"		"Hot Messages" folder is displayed
2M	Go to "Inbox"		All inbox messages are displayed
3M	Scroll to a message		Message is highlighted
4M	Go to "Context Sensitive Menu"		"Move to Hot Messages" option is displayed
5M	Select "Move to Hot Messages" option	Message storage is not full	"Message moved to Hot Messages folder" is displayed

(a)

Alternative Flow

Description: The message is not moved to Hot Messages folder because message storage is full

From Step: 4M

To Step: END

Step Id	User Action	System State	System Response
1A	Select "Move to Hot Messages" option	Message storage is full	"Memory required" dialog is displayed
2A	Confirm memory information dialog		Message content is displayed

(b)

Figure 1 - Template

The TaRGeT strategy for test cases generation consists in obtaining a formal LTS (Labeled Transition System) behavioral model from the input use cases, and then generating test cases from it. The LTS is a graph obtained from the use case steps. Figure 2 shows the LTS model generated from the use case presented in Figure 1. Each LTS transition represents an use case step. The use case flows may be branched, resulting in different paths in the LTS. For instance, Figure 1b describes an Alternative Flow that branches the Main Flow after step *4M*. That branch can be seen in Figure 2, in which the scenario may go to step *1A* or *5M* after step *4M*.

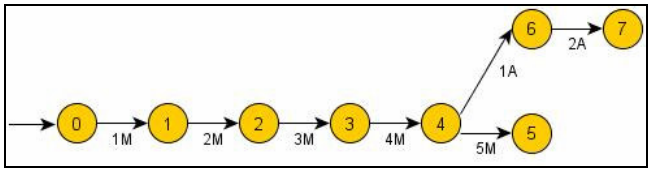

Figure 2 – LTS behavioral model

Each path of the LTS model generates a test case. A path can be obtained from the LTS behavioral model using the Depth First Search method (DFS), by traversing an LTS starting from the initial state [Cartaxo, Neto and Machado 2007]. As a general coverage criterion, all labelled transitions need to be covered by at least one test case. For instance, the LTS model showed in Figure 2 results in the generation of two test cases.

3. Cases Studies

The main objective of the performed cases studies was to assess the impacts and benefits of using TaRGeT in test design activities inside Motorola. Two case studies were run with different mobile phone applications from the messaging domain, which deal with short-messages, multimedia messages, and so on. From now on, we refer to

them as Case Study 1 and Case Study 2. In both case studies, we compared the current manual test design process (MTD), with the test design supported by TaRGeT (TTD). The comparison takes into account the number of test cases, the requirements coverage and the demanded effort.

In the case studies execution, some barriers were found since we could not allocate resources (people) from BTC teams. As these teams perform a great volume of daily work, it was not possible to allocate full time test designers from BTC to work on the case studies. This way, the testing design activities were performed by resources from our research project, which were supervised by experienced test designers from BTC.

The current MTD process can be split in three main stages: analysis, design and technical review. Analysis comprises understanding the application's requirements, writing a draft of the application's behavior and estimating the number of tests to be written for each requirement. Design comprises writing setup, initial conditions and procedures for the tests. Technical review includes all the activities concerning the inspection and rework of the test cases.

The TTD process follows a similar structure, but it splits the design stage in writing use cases and generating tests with TaRGeT. Since the current version of the tool does not generate setup information, manual writing of setup is required. Similarly, this process splits technical review in validation of use cases and inspection of setup values. We did not inspect the procedures of the test cases, since they are automatically generated from validated use cases.

During our case studies, we verified that the analysis stage was not influenced by TaRGeT's usage. For this reason, only the design and technical review stages are discussed in the following sections. Next, we discuss about the artifacts and metrics of the case studies.

3.1 Use Case Documents

Before writing the use cases, we sketched the application's behavior model based on requirement and graphical user interface documentation. The sketches improved the reasoning about the application's behavior, and guided the writing of the use case scenarios. In Case Study 1, the use cases were validated in a simplified way, however in Case Study 2, the use cases were validated by a rigorous technical review. Then, the use cases were reworked and approved based on the observations of the reviewer. Table 1 summarizes the information about the written use cases.

Table 1 - Written Use Cases in TTD approach.

	Case Study 1	**Case Study 2**
Total pages	11	12
N^{o} of Flows	48	31
N^{o} of UCs	4	3
Effort (h)	10:10	14:44*

* includes review and rework tasks

3.2 Number of Test Cases and Requirements Coverage

In Case Study 1, 92 test cases were created using MTD and 268 were generated using TTD. Comparing the two test suites, we can notice a significant increase in size (291%). This increase was a result of the detailed use case specifications, where several scenarios not too relevant were explored exhaustively. This problem was also a consequence of the lack of experience of use case's author and the absence of a validation of the use case scenarios by a BTC test designer.

Furthermore, Figure 3 shows the coverage of the test suites produced. The horizontal axis shows the requirements ids and the vertical axis shows the number of tests that cover a specific requirement. In average, the TTD process generated twice the number of tests when compared to MTD for each requirement. For some documented requirements (S8 and S10) the TTD test suite was able to cover requirements neglected by MTD test suite. The input documentation of TTD guided the use case author to specify scenarios for all documented requirements, which leads to at least one test case for each requirement.

On the other hand, some requirements (S14, S16 and S17) were not specified in the input documentation. This means that those requirements were only covered by the MTD process, since the manual test designers had expertise knowledge about them.

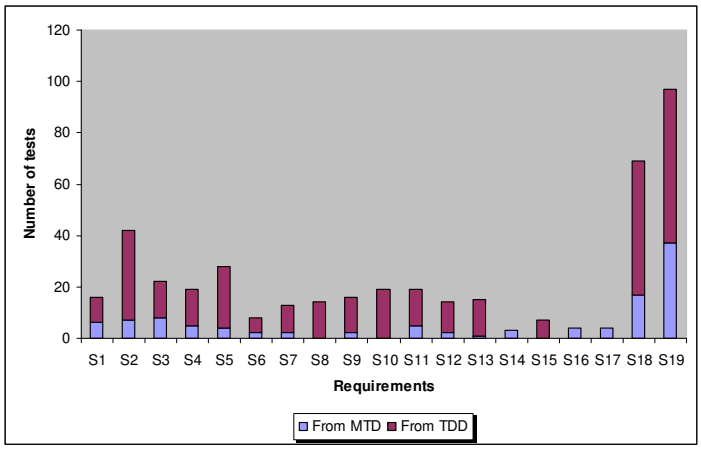

Figure 3 - Requirements coverage.

In Case Study 2, the MTD team wrote 38 tests and other 41 tests were generated using the TTD process. The similarity in size is a result of the technical revision performed by BTC test designers, plus the use case author's previous experience in test case design. Such experience contributed to produce an optimized specification used to test the most relevant behaviors of the application.

Table 2 shows the number of tests from TTD and MTD test suites that cover common and different requirements in Case Study 2. We can observe that, 27 tests from TTD covered the same requirements of 28 tests from MTD. 14 tests from TTD are variations of behaviors not covered by MTD test suite. On the other hand, 10 tests from the MTD suite covered scenarios not documented by the analyzed documents and consequently not specified in the use cases. The BTC test designers considered the variations (14 tests) generated by TTD very relevant. However, since the TTD suite was

not run against the implementation, we could not measure their effectiveness to detect bugs.

Table 2 - Mapping tests according to covered requirements.

Test Cases	TTD	MTD
Common	27	28
Variation or Differs	14	10
Total	41	38

3.3 Demanded Effort

Initially, the goals of Case Study 1 did not consider effort analysis and few effort metrics were collected. For this reason, only Case Study 2 had enough data to perform effort comparison between MTD and TTD processes. The design and technical review stages of the MTD process were used as the basis for this comparison. Figure 4 shows that the effort of using the TTD process was approximately half of MTD in both stages.

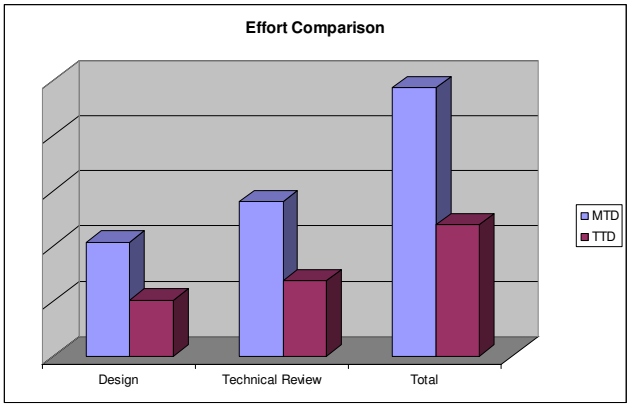

Figure 4 – Effort in hours spent in design and technical review.

4 Considerations About Using MBT and TaRGeT in the Practice

In Case Study 1, TaRGeT generated a huge number of test cases, many of them were considered redundant by expert designers. An alternative solution to reduce the number of generated test cases, apart from specification reduction, is to apply some selection criteria to guide the generation process, like test purposes [Cartaxo 2006], or to avoid similar test cases [Cartaxo, Neto and Machado 2007].

The participants of the case studies provided relevant feedbacks about MBT and TaRGeT tool. The use of NL for use case authoring and the document templates reduced effort to create the behavior model of the application. The generated test cases were also more homogeneous. However, some of the users were not acquainted with the template, as we expected. In addition, requirements documents used to create the use cases must be complete in order to achieve the maximal coverage.

The reduced effort using TTD suggests that it is faster to write and inspect use cases scenarios rather than writing and inspecting test cases. We can also support this

conclusion by observing the significant smaller amount of use cases specifications, compared to the test cases specifications.

5 Conclusions

This paper reports two case studies performed with the TaRGeT tool on mobile applications domain. The achieved results indicate that the use of such tool can reduce effort concerning test design activities. One drawback of the tool is the lack of a better test selection approach. Currently, improvements in this way are being developed, which can lead to better results.

Moreover, TaRGeT use cases specifications were used to automatically generate test cases to test the tool itself. Such tests were applied as the main artifact for validating the releases of TaRGeT. Then, we also conclude that TaRGeT and our use case templates are suitable to generate tests for desktop applications.

The comparison of four MBT tools and a measurement framework are described in [Sinha, Williams and Santhanam 2006]. This comparison shows that the complexity of use of the evaluated MBT tools is a common barrier in order they be extensively adopted. In addition, it achieves that text-based modeling (e.g. abstract state machine language) results in higher modeling efficiency when compared with graphic-based modeling (e.g. UML). We expect that natural language specification is helpful to decrease the modeling complexity level in addition to improve modeling efficiency.

Finally, this work yielded substantial knowledge that will be helpful in the next case studies. In our future experiments, we will include the assessment of test effectiveness of our MBT approach, measuring the number of detected bugs.

References

A. Sinha, C. E. Williams, P. Santhanam (2006). A measurement framework for evaluating model-based test generation tools. IBM Systems Journal, Vol 45, No 3, 2006, pp. 501-514.

Cabral, G. and Sampaio, A. (2006). Formal Specification Generation from Requirement Documents. In Proceedings of the Brazilian Symposium on Formal Methods (SBMF 2006), pp. 217-232.

El-Far, I. K. and Whittaker, J.A. (2001). Model-Based Software Testing. Encyclopedia of Software Engineering (edited by J. J. Marciniak).Wiley, 2001.

Cartaxo, E. G., Neto, F. G. O. and Machado, P. D. L. (2007). Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems. To appear in: IEEE International Conference on Systems, Man and Cybernetics, 2007, Montreal.

Cartaxo, E. G., Neto, F. G. O. and Machado, P. D. L. (2007). Automated Test Case Selection Based on a Similarity Function. Workshop Modellbasiertes Testen (MOTES07), Bremen, 2007. To appear in Lecture notes in Informatics (LNI).

Cartaxo, E. G. (2006). Functional test cases generation for mobile phone applications. Master's thesis, Federal University of Campina Grande, Oct 2006.