Model Based Test Generation: An Industrial Experience

Sidney Nogueira[1,3], Emanuela Cartaxo[2,3], Dante Torres[1,3], Eduardo Aranha[1,3], Rafael Marques[1,3]

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

[2]Departamento de Sistemas e Computação – 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. *Software testing is a crucial activity to a successful software project that demands a significant effort. Aiming at improving software quality as well as to reduce 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 of Motorola Brazil Test Center. Such studies point that the use of such a MBT tool reduces the effort of test generation process comparing with the current process that is entirely manual.*

1. Introduction

Testing activity is crucial to the success of a software project. In general, this activity requires about 50% of software development resources [Beizer 1990]. 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 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/Brazil and UFCG/Brazil. The project's goal is to automate the test case generation, selection and evaluation of Motorola mobile phone applications. This paper is organized as follows. Section 2 presents the used tool and Section 3 shows the case studies. Finally, Section 4 discusses the use of MBT and TaRGeT in the practice and 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 requirements and test artifacts in an

integrated way. The tool is based on the approach presented in [Cabral and Sampaio 2006] and it is discussed next.

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 is “The message is moved to hot message folder” (Figure 1(a)), where the user selects his favorite message and moves it to the hot message folder. The alternative flow (Figure 1(b)) represents a situation that the message is not moved to hot message folder because the message storage is full. The flows are described through steps (**Step Id**) that include 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 behavioral model from use cases. Then, test cases are generated using a test case extraction algorithm. The LTS model is built by following the steps id. Figure 2 shows the LTS model generated from the use case presented in Figure 1. For each “step id” is created a transition with user action, system state and system response.

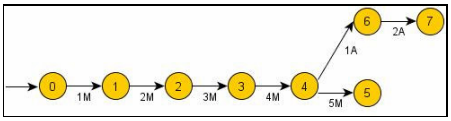

Figure 2 – LTS behavioral model

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 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. For comparison purpose, it was decided to execute the current manual test design process (MTD) and the test design supported by

TaRGeT (TTD) in both case studies. The comparison takes into account the number of test cases, the requirements coverage and the demanded effort.

In the case studies execution, some barriers was been found due to the impossibility of allocating resources (people) by teams test partners. As these teams execute a great volume of daily work, then they can not be able to allocate resources full time for the case studies execution. This way, we allocate resources of our team to execute testing design activities during the case studies. However, the participants of our research team were trained and supervised by the experienced test designers. In this way, we reduced the cost of our partners.

The current MTD process can be split in three main stages: analysis, design and technical review. Analysis comprises understanding of the application requirements, writing a draft of the behavior to be tested 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. Because the available version of the tool at that moment misses 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 rest of this paper. Next, we discuss about the artifacts and metrics of the case studies.

3.1 Use Case Documents

Before writing the use cases, we used a graphical tool to sketch a behavior application model based on requirements and graphical use interface specifications. The sketches drove and simplified 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 produced use cases.

Table 1 - Produced 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 see a significant increase in size (391%). 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 cases documents by a MTD team.

Also, 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 the same requirements. This is an effect of the use case specifications, as explained before. For some documented requirements (S8 and S10) the TTD test suite was able to cover requirements neglected by MTD test suite. The input specification of TTD helped the use case author to specify scenarios for all documented requirements and to generate at least a test for each requirement. However, some of the requirements (S14, S16 and S17) were not in the analyzed documents and they were covered only by the MTD process due to the test designers' expertise.

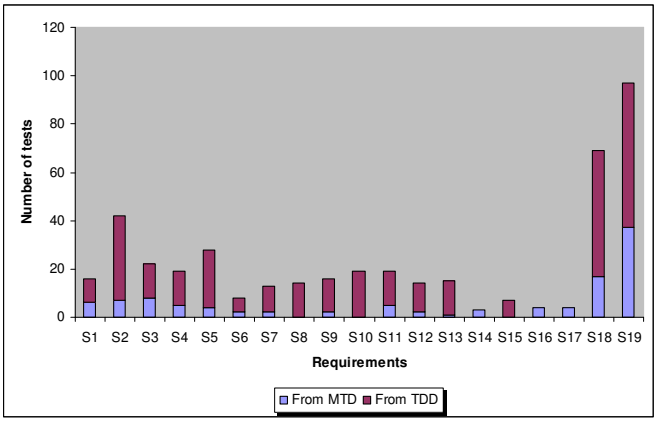

Figure 3 - Requirements coverage.

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

Table 2 shows the number of tests from TTD and MTD test suites that cover common and different requirements. We can observe that, 27 tests from TTD covered the same requirements of 28 tests from MTD. Fourteen tests from TTD are variations of behaviors not covered by MTD test suite. On the other hand, ten tests from the MTD suite covered scenarios not documented by the analyzed documents and consequently not specified in the use cases. The MTD team considered the variations (14 tests)

generated by TTD very interesting. 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.

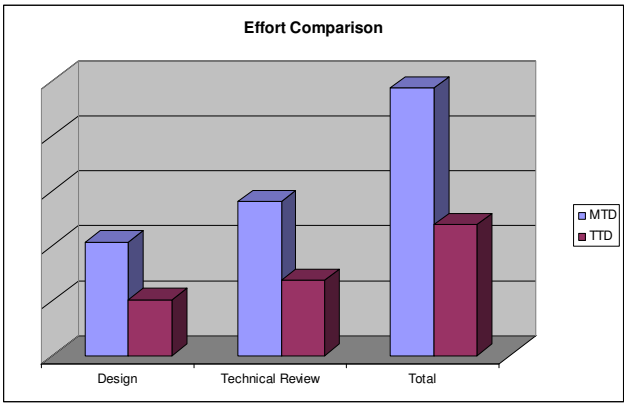

Figure 4 – Effort in hours spent in design and technical review.

4 Considerations About Using MBT and TaRGeT in the Practice

We presented a high number of generated test cases. An alternative solution to cut out the big number of generated test cases, apart from specification reduction, is to apply some selection criteria as test purposes [Cartaxo 2006] or to avoid similar test cases [Cartaxo, Neto and Machado 2007].

The participants of the case studies gave some feedbacks about what they thing about MBT and TaRGeT tool. The use of natural language and document templates resulted in a 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 so comfortable with the template as we were expecting. Also, 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 significantly smaller number of pages of use cases specifications when compared to the test cases specifications.

5 Conclusions

This work reports some experiments performed using the TaRGeT tool for mobile applications. The achieved results lead indicates that the use of such tool can reduce effort concerning test design activities. One limitation of the used version of the tool concerns the lack of a better test selection approach. Improvements in the tool are in progress and they can lead to the achievement of even better results.

In addition, we observed that the use of a controlled natural language (CNL) can motivate a larger audience to use TaRGeT tool not only due to test design effort reduction. CNL can improve the quality of generated test by implanting a standard for vocabulary and syntax of test cases.

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

Finally, the performed yielded important information used to prepare next case studies that can include the assessment of test effectiveness of our MBT approach (number of bugs found, etc.).

References

Beizer, B. (1990) Software Testing Techniques. Van Nostrand Reinhold, second edition, 1990.

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. IEEE International Conference on Systems, Man and Cybernetics, 2007.

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.