AUTOMATED REFINEMENT OF MODELS FOR MODEL-BASED TESTING

A Dissertation

by

Ceren Şahin Gebizli

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in the
Department of Computer Science

Özyeğin University
July 2017

AUTOMATED REFINEMENT OF MODELS FOR MODEL-BASED TESTING

Approved by:

Assoc. Prof. Hasan Sözer (Advisor)
Department of Computer Science
Özyeğin University

Prof. Lionel Briand
Department of Software Engineering
University of Luxembourg

Asst. Prof. Barış Aktemur
Department of Computer Science
Özyeğin University

Asst. Prof. Cemal Yılmaz
Department of Computer Science and Engineering
Sabancı University

Date Approved: 14 July 2017

Assoc. Prof. Fatih Uğurdağ
Department of Electrical and Electronics Engineering
Özyeğin University

ABSTRACT

Model-Based Testing (MBT) enables automatic generation of test cases based on models of a system. It has been successfully applied in various application domains, each of which might introduce specific challenges. In this dissertation, we introduce methods and tools for addressing some of these challenges for the consumer electronics domain. In particular, we focus on the testing of Digital TV systems as our case study. We identified the following 3 problems in this context: *i)* Models of the system are created based on requirement specifications, which are often incomplete and imprecise. Therefore, these models are subject to accidental omissions of certain system behavior. As a result, critical faults can be left undetected by the generated test cases. *ii)* Resources are extremely limited in the consumer electronics domain. It is not feasible to attain an extensive coverage of test models. *iii)* A product family in consumer electronics often includes hundreds of systems. The set of features can highly differ among these systems. Therefore, the MBT process and modeling must be flexible to systematically manage variability and increase the amount of reuse for test models.

To tackle the first problem, we introduce an approach and tool for automatically extending test models based on a set of collected execution traces. These traces are collected during Exploratory Testing (ET) activities. Several critical faults were detected in 3 case studies after generating test cases based on extended models. These faults were not detected by the initial set of test cases. They were also missed during the ET activities. As a solution for the second problem, we iteratively update test models in 3 steps to focus the test case generation process only on execution paths that are liable to highly severe failures. We use Markov Chains as test models,

in which transitions among states are annotated with probability values. First, we update these values based on usage profile. Second, we perform an update based on fault likelihood that is estimated with static code analysis. Our third update is based on error likelihood that is estimated with dynamic analysis. We generate and execute test cases according the updated values after each iteration of updates. New faults can be detected after each iteration. To address the variability problem, we document variations among tested systems explicitly with a feature model. We map optional and alternative features in the feature model to a set of states in the test model. Transition probabilities in the test model are updated according to the selected features so that the generated test cases focus only on these features. This approach facilitates the reuse of a test model for many systems.

ÖZETÇE

Model-bazlı test (MBT), test edilen sistemin modelleri ile otomatik olarak test senaryoları oluşturulmasını sağlamaktadır. MBT, her biri kendine has zorlukları beraberinde getiren çeşitli uygulama alanlarında kullanılmaktadır. Biz bu tezde, tüketici elektroniği alanındaki zorlukları adreslemek için metot ve araçlar öneriyoruz. Özellikle vaka çalışması olarak Dijital TV sistemlerinin testine odaklanmaktayız. Bu bağlamda 3 problem belirledik; *i)* sistem modelleri gereksinim analizleri baz alınarak oluşturulmuştur ve bu gereksinimler genellikle tam ve açık değildir. Bu sebeple, bu modellerde bazı sistem davranışlarının eksik olma ihtimali vardır. Sonuç olarak da oluşturulan test adımları ile kiritik hatalar bulunamayabilir. *ii)* tüketici elektroniği alanında kaynaklar çok kısıtlıdır. Test modellerinin kapsamını ve böylece oluşturulan test adımlarının sayısını sürekli arttırmak elverişli değildir. *iii)* tüketici elektroniği alanındaki bir ürün ailesi genellikle yüzlerce sistem içerir. Bu sistemlerin barındırdıkları özellikler birbirlerinden farklıdır. Bu sebeple, MBT süreci ve test modelleri, çeşitliliğin sistematik olarak yönetimini ve test modellerinin tekrar kullanılabilirliğini sağlayacak şekilde esnek olmalıdır. İlk problemin üstesinden gelebilmek için, test modellerini, toplanan çalıştırma izlerini baz alarak, otomatik olarak güncelleyecek bir yaklaşım ve araç öneriyoruz. Bu izler araştırmaya yönetilik test aktiviteleri sırasında toplanmaktadır. Yapılan 3 vaka çalışması ile, güncellenen modeller üzerinden üretilen test adımları ile birçok kritik hata bulunmuştur. Bu hatalar daha önce araştırmaya yönetilik test aktiviteleri sırasında bulunamamıştı. İkinci hataya çözüm olarak, test modellerini, hataya neden olabilecek yollara odaklanarak

test adımı üretilebilecek şekilde, 3 adımda iteratif olarak güncelliyoruz. Test modelleri olarak, durumlar arası geçişlere olasılık bilgilerini işleyebileceğimiz Markov Zincirlerini kullanıyoruz. İlk olarak, modeldeki bu olasılık bilgilerini kullanıcı profiline göre güncelliyoruz. İkinci olarak, statik kod analizlerinden çıkartılan hata ihtimaline göre modeldeki olasılık bilgilerini güncelliyoruz. Üçüncü güncelleme işlemimizi ise dinamik analizlerinden çıkartılan hata ihtimallerine göre gerçekleştiriyoruz. Her bir güncelleme iterasyonu sonrası, güncellenen olasılık bilgilerine göre, test adımlarını tekrar üretip çalıştırıyoruz. Vaka çalışmalarında, bu yaklaşım ile her bir iterasyon sonrası yeni hatalar bulunmuştur. Çeşitlilik problemini adreslemek için ise, test edilen sistemdeki varyasyonları açıkça özellik modelleri ile kayıt altına alıyoruz. Test modelindeki durumların tümünü, özellik modellerinde opsiyonel ve alternatif olarak işaretliyoruz. Üretilen test adımlarının bu özelliklere odaklanabilmesi için, test modellerindeki geçiş olasılıklarını seçilen özelliklere göre güncelliyoruz. Bu yaklaşım sayesinde birçok sistem için test modellerinin tekrar kullanılabilmesi sağlanmıştır.

ACKNOWLEDGMENTS

It was 2010 when I graduated from Computer Engineering Department of Dokuz Eylül University. Then, I started working as a Design Verification and Test Engineer at Vestel Electronics R&D, 3 weeks after my graduation. After I started working, it was announced that Vestel and Özyeğin University had a special collaboration for engineers that want to do M.Sc. or Ph.D. It was a very good opportunity to continue academic studies besides the full-time job in parallel. Before starting M.Sc. studies, first I attended the "Software Testing and Analysis" course where I met my supervisor Assoc. Prof. Hasan Sözer. I got inspired by listening to his talks and how he was making students think about the problems and solutions. After that, I became an official M.Sc. Computer Engineering student as of Fall 2012 semester and started studying with a big ambition of thinking about the approaches to find solutions to the problems in my daily work life in the company. I am extremely grateful to my supervisor, Assoc. Prof. Hasan Sözer for his guidance, patience, motivation, and all the useful discussions. He encouraged me to continue to Ph.D after M.Sc. under his supervision, and I was very lucky to have a supervisor that cared so much about my studies, and that responded to my questions quickly and with detail, and always brought different point of views to the problems. He was actively interested in my work and always lead me to improve my studies, encouraged me about writing good papers, participating various conferences with effective presentations, and also joining competitions.

I am very proud of being a student of Özyeğin University and thankful for the collaborative program with Vestel Electronics. I have learned many things from great instructors during M.Sc. and Ph.D.

I would like to thank my committee members; Asst. Prof. Barış Aktemur, Assoc. Prof. H. Fatih Uğurdağ, Asst. Prof. Cemal Yılmaz and Prof. Lionel Briand for their guidance and spending time to review my thesis. Asst. Prof. Barış Aktemur, Assoc. Prof. H. Fatih Uğurdağ always gave constructive feedback during the thesis progress meetings and helped me to improve my thesis.

In addition, I want to thank Prof. Fevzi Belli and Prof. Atıf Memon for providing me feedback and ideas to improve my thesis. I also thank Ali Özer Ercan for his contributions.

I also thank Vestel Electronics very much for the great opportunity and support for pursuing my academic studies. My Ph.D. studies were also supported by Turkish Ministry of Science, Industry and Technology which provided funding as part of the joint grant together with Vestel Electronics (909.STZ.2015).

Special thanks to R&D Design Verification and Test Group who helped me when I was studying and executing my case studies on the test environment. They were very helpful and always ready to share their domain and testing experiences with me.

I want to thank my friend Duygu Metin who was always very supportive, helpful and excited about all the studies that I was doing.

I also thank Burcu Ergun and Abdulhadi Kırkıcı who are M.Sc. students and also my colleagues in Vestel Electronics. They contributed to this thesis by developing tools and performing case studies with me.

Lastly, I would like to thank my family for all their love and encouragement. My parents, Aslan-Betül Şahin, raised me as an ambitious person and supported me in all my pursuits. I thank most of all for my loving, supportive, encouraging, and patient husband M. Çağlar Gebizli who provided me support during my Ph.D. studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Products in consumer electronics domain have been transformed from electromechanical systems to complicated software systems. This trend has two main causes. First, the implementation of the provided features is shifting from hardware to software. Second, the number of these features and their variety are increasing. For instance, current Digital TV (DTV) systems include web browsing, on-demand streaming, home networking and many other features in addition to traditional TV functionalities [2]. As a result, the size and complexity of software systems adopted in these products are continuously increasing. This makes it harder to ensure software reliability due to an amplified number of potential faults[1]. Unlike safety-critical systems, these faults might not have catastrophic consequences in consumer electronics domain. However, their impact on user perception is very critical [4] due to tough competition. Moreover, systems being developed for such a high-volume market are highly cost sensitive. Especially, time and human resources are very limited. This constraint makes it prohibitive to exhaustively test systems. Available resources are often not even sufficient to remove all the detected faults from the system [5]. Only those faults that have the highest impact on user-perceived reliability [4] can be removed before the release, if they are detected. Hence, testing of software-intensive consumer electronics products have to be effective and efficient. Effectiveness is required to capture the most critical faults in terms of their impact on user perception. Efficiency is required for the optimal use of available resources.

[1]Throughout this dissertation, we adopt the terminology introduced by Avizienis et al. [3]. We use the term fault (synonymous with defect or bug) as the cause of an error, which can lead to a failure. We consider error as an internal system state, whereas failure as an event that can possibly be observed by the user in the form of an unexpected output or system behavior.

Traditional testing processes, which by and large involve manually performed activities, fall short in terms of efficiency and effectiveness. Therefore, there have been several techniques proposed to improve these activities and automate them if possible. Model-based Testing (MBT) is one of such techniques [6, 7, 8, 9]. It enables automatic generation of test cases by using the models of the system under test [9, 10, 11]. MBT has been successfully applied in various application domains [12, 13]. Each application domain introduces specific challenges and opportunities for the utilization of MBT. In this dissertation, we introduce methods and tools for addressing some of these challenges for the consumer electronics domain. In particular, we focus on the testing of DTV systems as our case study.

We applied the so-called industry-as-laboratory approach [1] for conducting our research. This approach focuses on industry-relevant problems and evolutionary improvement of research results by exploiting continuous feedback from real-life applications (See Appendix A for details). In the following, we clarify the scope of our studies and provide motivation for the identified challenges regarding the adoption of MBT. Then, we introduce our research questions. We conclude the chapter by summarizing our contributions and providing an overview regarding the organization of the dissertation.

1.1 Thesis Scope and Motivation

The work presented in this dissertation has been carried out as part of a project that is co-funded by the Turkish Ministry of Science, Industry and Technology and Vestel Electronics. The goal of the project is to develop methods and tools for increasing the effectiveness and efficiency of MBT as applied for testing DTV systems. Vestel[2] is one of the largest DTV manufacturers in Europe. It produces DTV systems for 157 different brands from 145 different countries. There are hundreds of test suites being

[2]www.vestel.com.tr

used in the company for testing these systems. These test suites are used for testing various features and they are created for different types of tests such as performance tests, certification tests, connectivity tests and functional tests. In our project, we focus on functional tests only.

An independent software test group within the company performs all the tests for all the projects. The group has 4 weeks to complete all the tests per product development project. Then, additional tests are performed for about 2 weeks by experienced testers. These testers perform Exploratory Testing (ET) [14] to reveal the possibly missed faults. It has been observed that critical faults are mostly detected during ET (See Appendix B for details).

The amount of resources is extremely limited for the test group, considering hundreds of different products being subject to regular regression tests. As a solution approach, test execution is automated as much as possible. Vestel uses an in-house developed tool, namely Vestel Test Automation (VesTA) [15, 16], for this purpose, which also captures and evaluates snapshots of the DTV screen for verifying the system behavior (See Appendix A and [17] for further details regarding test oracle automation in our environment). The tool automatically sends a sequence of remote controller key signals to DTV systems based on a predefined set of test scripts. Some of the functional tests are automated by defining them in the form of these scripts. There are also test cases that involve test steps to be manually followed and executed by testers. These are mainly defined in the form of check-lists.

The manual preparation of test scripts and check-lists is error-prone and time consuming. Moreover, requirements specifications continuously change. As a result, maintenance becomes costly for ensuring consistency and coverage of scripts and check-lists with respect to these specifications. More often than not, manually defined test steps soon turn out to be outdated due to updated requirements. MBT is adopted

by the company to address these problems. A commercial tool called MaTeLo[3] has been used as the MBT tool. The usage behavior for some of the developed DTV systems are modeled with this tool in the form of Markov chains [18]. Transitions among states are annotated with executable scripts or manual actions (i.e., checks) that define the corresponding test steps. MaTeLo can take these models as input and generate various sequences of test steps by a variety of test case generation algorithms.

At first sight, MBT has been proven to be beneficial from several aspects:

- *models of system behavior have served as documentation and they have helped to detect internal inconsistencies in requirements specifications.*
- *it has enabled the automated generation of test scripts.*
- *maintenance costs have been reduced since changes in requirements can be reflected to models instead of dozens of various scripts and check-lists.*

However, there have also been several problems observed. We identified the following 3 problems to be addressed in the scope of this dissertation:

i) ET has continued to be the phase, where the most of the critical faults were detected. There are many features to test for every product and ET is manually performed by a limited number of testers. Hence, limited resources prohibit the extension of the ET phase to find critical faults. We investigated why these faults are missed even when MBT is applied. It was observed that some of the execution paths were omitted in test models (i.e., missing some of the transitions and states). As a result, generated test cases did not exercise the corresponding usage scenarios. There are two main reasons for such omissions. The first reason is human error since model creation is a manual process. Secondly, requirement specifications are used as the main information source for model creation. These specifications are often

[3]http://www.all4tec.net

incomplete and imprecise. As a result, created test models can be incomplete as well. Experienced testers explore the behavior of the system during ET without following any specification but just their domain knowledge and insights. The experience gained during ET is not documented to be exploited later on. Testers do not usually have time and the necessary expertise to document their knowledge/insight in the form of formal models.

ii) The second problem is regarding the size of the models, which include thousands of states and transitions. This size is also reflected to the number of generated test cases. Test execution can not be completely automated for all of these test cases. Hence, the available resources are usually not sufficient to apply all of them to the system. This is especially the case if test case generation aims at exhaustively covering the test model for each regression test. Hence, the generated test cases have to focus only on execution paths that are liable to highly severe failures that can be directly observed by users.

iii) The third problem is variability among hundreds of different DTV systems. The set of features, broadcast specifications and user interfaces can differ among these systems. MBT is actually better than manual test case specification with respect to handling this variability. Variations can be better managed at the abstraction level of test models. However, MBT still falls short to address systematic variability for large scale product families with high number of variations that cross-cut test models [19, 20]. Software Product Line Engineering (SPLE) approach is required for facilitating systematic and scalable reuse [21].

In the following section, we define our research questions aligned with these observations.

1.2 Research Questions

We performed research activities to address the problems that are listed in the previous section. During these research activities, we have addressed several sub-problems and evaluated alternative and/or complementary solutions. We defined 3 main research questions that focus our research.

> ***RQ1: Is it possible to automatically extend test models and as such, increase the effectiveness of MBT by exploiting feedback from ET?***
>
> ***RQ2: Is it possible to employ random-stochastic test case generation based on risk of failure to increase the efficiency of MBT?***
>
> ***RQ3: Is it possible to automatically adapt and reuse test models for large product families with high number of variations?***

Our first goal was to increase the effectiveness of the MBT process, which is measured in terms of the number of detected faults. During the initial attempts in applying MBT to a set of pilot projects, fault detection rate turned out to be low due to omitted execution paths in test models. On the contrary, ET was observed to be highly effective in practice to detect critical faults. ET activities were performed manually by experienced test engineers. However, the experience gained during these activities were not documented and exploited. We defined *RQ1* based on this observation.

Our second goal was to increase efficiency such that new faults can be detected while the number of test steps and the test duration is reduced. To achieve this goal, we needed a non-uniform test case generation method that focus on failure-prone usage scenarios rather than covering the whole model. This also requires additional refinements of test models for augmenting them with information regarding estimated failure risks. *RQ2* is defined to employ such a method and evaluate its impact on

MBT efficiency.

RQ3 is defined to investigate the re-usability of test models for a large family of products by adopting principles from SPLE.

In the following section, we briefly explain solution approaches that we propose to address these questions. We summarize our contributions and provide an outline of the remainder of this dissertation.

1.3 Thesis Contributions and Overview

To answer *RQ1*, we introduced an approach and a tool called Automated Refinement of Models for Model-Based Testing Using Exploratory Testing (ARME) for automatically extending test models based on a set of execution traces. These traces are collected during ET activities performed by experienced testers. ARME compares the recorded execution traces with respect to the possible execution paths in test models. Then, test models are automatically extended to incorporate any omitted system behavior. The extended models can be used for generating various test cases to capture new faults. We performed 3 case studies to evaluate the effectiveness of ARME . Several critical faults were detected after generating test cases based on extended models. These faults were not detected by the initial set of test cases. They were also missed during the ET activities.

We were inspired from the principles of Risk-based Testing (RBT) [22] for addressing *RQ2*. Our test models comprise a set of states and a set of state transitions that are annotated with probability values. These values can steer the test case generation process, which aims at covering the most probable paths. We developed an approach and a tool called Risk-based Model Adapter (RIMA) to update these values in 3 steps for augmenting information regarding the risk of failure. A failure occurs if a faulty program location is reached (i.e., the corresponding feature is used by an end user)

and if the fault infects the program state propagating to an incorrect output/behavior [23]. Hence, we first update transition probabilities based on a collected usage profile to focus on the mostly used DTV features. Secondly, we update the resulting models based on fault likelihood at each state, which is estimated based on static code analysis. Finally, we perform updates based on error likelihood at each state, which is estimated with dynamic analysis. We generate and execute test cases after each refinement step. In our case study on DTV systems, we were able to detect several new faults after each refinement step of our approach. These faults mainly lead to stability issues in the platform such as crashes and lack of response for remote controller commands. We performed a second case study on smart phones, where we were able to detect new faults as well. These faults lead to failures such as missing contacts in contact list and lack of response for the stop call button.

To address *RQ3*, we introduced a product line testing approach and a tool called Feature Oriented Model Adaptation Tool (FORMAT) for systematic reuse of test models for testing a large family of products. Hereby, we document variations among tested systems explicitly and separately with a feature model. We map optional and alternative features in the feature model to a set of states in the test model. Transition probabilities in the test model are updated according to the selected features so that the generated test cases focus only on these features. This approach facilitates the reuse of a test model for many systems. We performed a controlled experiment with 10 participants to evaluate the effectiveness of our approach and tool. Results show that the investment for SPLE pays off already if the test model is reused for 13 products.

This dissertation is organized as follows.

Chapter 2 provides background information on MBT, ET, RBT, and SPLE.

Chapter 3 focuses on extending test models for MBT by exploiting feedback from ET. It introduces the ARME tool and its evaluation. This chapter is a revised

version of the work described in [24, 25].

Chapter 4 introduces the three-step model refinement approach for applying RBT and presents its evaluation. This chapter is a revised version of the work described in [26, 27].

Chapter 5 explains the adopted SPLE approach and FORMAT to facilitate the reuse of test models for a family of products. This chapter is a revised version of the work described in [28].

Chapter 6 provides the conclusions. The evaluations, discussions, and related work for the particular contributions are provided in the corresponding chapters.

Appendix A explains the industry-as-laboratory approach [1] and our experiences in the application of this approach, which is proven to be highly effective for industry-academia collaboration and technology transfer. This section is a revised version of the work described in [25].

Appendix B provides details and results regarding an industrial case study that we conducted to evaluate the impact of education and experience level on the effectiveness of ET. This section is a revised version of the work described in [29].

CHAPTER II

BACKGROUND

In this chapter, we provide background information regarding MBT, ET, RBT, and SPLE.

2.1 Model-Based Testing

MBT relies on test models to automatically generate test cases [10, 11, 30]. This technique involves processes and techniques for the automatic derivation of abstract test cases from abstract models, the generation of concrete test cases from abstract test cases, and execution of concrete test cases [9].

It has been employed in the industry for more than a decade to increase the efficiency of the testing process and for improving the software quality [7]. It relies on a model of the System Under Test (SUT) and/or its environment that is created mainly by analyzing system requirements.

The content and the structure of the employed models can differ. Regarding terminology, a distinction is made between *system models* and *test models* in the literature [31, 32]. System models describe internal behavior of the system [31] and they are used for system creation [32], whereas test models define the interaction of the system with the user/environment [31] and they are used for test case description/generation [32]. From this perspective, models that we employ for MBT can be considered as test models. These models define user-observable system behavior with respect to a set of inputs and actions of the user. The test model is provided as an input to MBT tool, which automatically generates a set of test cases by traversing the possible behavioral scenarios on the model. Hereby, tool configuration parameters may include the selected test case generation algorithm, coverage criteria, and

the maximum number/length of test cases. Figure 1 provides a simplistic overview of MBT.

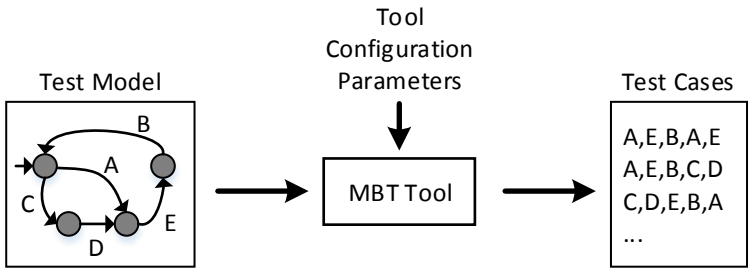


Figure 1: The overall MBT approach.

There are several types of formalisms that are used for expressing the test model. These include Unified Modeling Language (UML) models [33], Finite State Automata (FSA) [34], Event Sequence Graph (ESG) [35], state charts [36], Markov chains [18] and Labeled Transition Systems (LTS) [37].

The use of UML models for MBT focuses in particular on the utilization of use case diagrams, sequence diagrams, collaboration diagrams, and class diagrams [33]. Hereby, test scenarios are mainly specified by sequence and collaboration diagrams. Use case diagrams are used for organizing these scenarios. There are one or more sequence or collaboration diagrams for each use case. Class diagrams are used for documenting the set of classes, methods, and attributes that are involved in the scenarios.

FSA is a commonly utilized formalism for MBT, especially to represent state-based behaviors of a system. Informally, it involves a set of inputs, a set of states, and a transition function that maps pairs of inputs and states to next states. A FSA model is scanned by a MBT tool for executable paths to generate test cases. Each possible sequence of states defines an execution path that can be specified as a test case. There exist several methods for generating such sequences including the

Transition Tour (T) method [38], the Distinguishing Sequence (D) method [39] and the Characterizing Set (W) method [34]. In this work, we did not focus on test case generation; we focus on the refinement of test models instead. Test case generation is performed by the employed MBT tool using two different algorithms. One of these algorithms generates test cases in a stochastic manner. The other one satisfies full transition coverage.

ESG [35, 40] constitutes a more abstract representation compared to FSA. Hereby, inputs and states are represented together as *events*, each of which corresponds to a user-observable action. Informally, ESG involves a set of events and a transition function that maps events to next events.

State charts [36] are similar to FSA models, but they can be hierarchically composed. In addition, states and transitions can be accompanied with boolean guards and a set of actions.

A Markov chain is also basically a FSA, in which probabilities are defined for state transitions [18]. The system may change its state from the current state to another state, or remain in the same state, according to a probability distribution. In our approach, we employ a form of extended Markov chains to represent test models (See Section 3.1.1). We exploit state transition probabilities to focus the scope of the generated test cases.

LTS models also comprise a set of states and transitions between different pairs of these states. In addition, they include a countable set of input labels and a countable set of output labels [37]. These two sets are disjoint. Each transition can be annotated with input and output labels. These labels are used for synchronization [37] and modeling the input/output behavior of reactive systems.

MBT approaches were previously categorized with a taxonomy [9]. In the following, we position the MBT approach adopted in this work according to this taxonomy.

Our model specification scope is input-output. We employ two different types of models. The first one is untimed, deterministic, and discrete. The second one is untimed, stochastic, and discrete. We employ two different test case generation algorithms. The first one uses structural model coverage as test selection criterion. The second algorithm is random & stochastic. Test execution is offline. Concrete test steps are embedded in the test models in the form of test scripts. Hence, we directly generate concrete test cases from the model that can be executed on the system, rather than generating abstract test cases first.

Intended benefits of MBT was previously listed [41] as:

- Explicit specification, modeling, and review of system behavior, which helps in the detection of inconsistencies and faults in requirement specifications [42].

- Use of test models as a means of documentation [43], which enhances communication between developers and testers.

- Automated generation of test cases, while being able to measure and optimize test coverage.

- Use of test models for evaluation and selection of regression test suites.

- Improved maintainability due to the easier management of requirements changes at the model level.

- Increased test quality through model-based quality analysis.

- Lowered costs by shortening test cycles [44].

We observed all these benefits during our studies. In particular, the first and the last benefits were the main driving forces leading to an increased adoption of MBT in the company, where we performed the case studies.

In terms of industrial applications, MBT has been mainly utilized for mission-critical and/or safety-critical systems [13]. For these systems, any failure that can pose a threat to the mission or cause harm to people and environment must be prevented. In that context, the additional cost due to the required hardware/software and human resources is a minor issue. In our work, we focus on the consumer electronics domain, in particular, DTV systems. For these systems, which are supposedly not subject to catastrophic failures, the cost, time-to-market and the perception of the user become the primary concerns, instead. These systems are very cost-sensitive and failures that are not directly perceived by the user can be accepted to some extent, whereas failures that can be directly observed by the user require a special attention. On the other hand, resources are restricted and faults must be detected in a very limited amount of time.

2.2 Exploratory Testing

ET [45, 46] is characterized by a continuous learning and adaptation process, where the tester iteratively learns about the product and its failures, plans the testing activities, designs, and executes the tests, and reports the results [14]. As the main difference from traditional software testing, ET is not based on a set of predesigned test cases. Instead, testers use their creativity and experiences to steer the process dynamically. Test design, execution and learning are all concurrent activities in ET, which aims at using human effort efficiently by utilizing human intuition and experience such as domain knowledge, system knowledge or software engineering knowledge.

There exist various definitions of ET [47, 48, 49]. Bach briefly defines ET as *simultaneous learning, test design and test execution* [47].

There are no formal descriptions or detailed methodologies defined for ET yet. Neither there exist strictly described procedures that should be followed by testers during test execution. That is why, ET has been mainly considered to be an ad-hoc approach. Nevertheless, it is one of the mostly applied and one of the most successful approaches [50], also based on our observations in the industry[1]. ET activities are performed manually and their success depends on the experience and skills of test engineers. Therefore, observations regarding ET cannot be generalized and ET cannot be considered as an alternative for formal and automated techniques such as MBT. However, it was previously showed that ET can be highly effective in practice to detect critical faults [51]. Hence, it can be considered as a promising complementary approach for other testing techniques. This fact also provided the motivation for the first problem addressed in our work (See Chapter 3).

We also reported a case study on ET that is performed in an industrial context. The goal of the study was to evaluate the impact of educational background and experience of testers on the effectiveness of ET. 19 practitioners, who have different education and experience levels, were involved in applying ET for testing a DTV system. The details of this case study is provided in Appendix B.

2.3 Risk-Based Testing

RBT is a testing approach which uses risk evaluations to help on optimizing the testing efforts [22]. With RBT, most important features are focused within limited time by considering the highest priority tests [52].

The main role of RBT is to reduce testing time without affecting the quality of the product. In RBT, critical requirements are evaluated before the creation of test

[1]We discuss our observations based on the industrial case studies in Chapter 3 and Appendix B.

cases. The generated test cases are supposed to increase test efficiency by focusing the critical paths.

RBT can involve test prioritization and test selection activities [53]. Test selection is performed to select a subset of the test cases to be executed. On the other hand, test prioritization is performed to change the execution order of the selected test cases. The selection and ordering of test cases should be optimized because of lack of time and cost during test activities [53]. Test case selection is especially useful for regression tests for identifying the test cases that are relevant to some set of recent changes. For example, a design-based approach for test selection [54] is previously proposed for RBT.

RBT can be used as complementary for other testing approaches. In our studies we combined RBT and MBT. We incorporated information regarding failure risks as part of test models. This information is used for steering test case generation to focus on critical paths.

2.4 Software Product Line Engineering

SPLE [55] is a commonly used approach for managing commonalities and variabilities in a product family. This approach facilitates systematic reuse of software development artifacts, including test artifacts [21]. A key property of this approach is the explicit management of variability among these artifacts. There have been various types of models introduced for modeling variability such Orthogonal Variability Model (OVM) [55] and feature model [56, 57]. In our studies, we used feature models to document variability among products.

Feature models are created by using a standard notation to show mandatory and optional features regarding a product and inter-dependencies among these features [56]. Selection of various features on the model brings various feature combinations. Each combination is considered as valid product that can be derived from a

product family.

Feature models have a tree structure, with features forming nodes of the tree. Feature variability is represented by the arcs and groupings of features. In Figure 2, *Hybrid Broadcast Broadband TV (HBBTV)*, *Digital Live Networking Alliance (DLNA)*, *FOLLOW TV*, *SMART TV*, *PORTAL MODE*, *WIRELESS DISPLAY*, and *INTERNET LOCK* features are the base features. Each feature may have more than one child feature. The relation between a parent feature and its child feature can be: *mandatory*, *optional*, *inclusive-or*, and *exclusive-or* [58]. These reation types are defied as follows:

> ***mandatory:*** These features have to be selected when their parent is selected. (e.g., *SMART TV*).
>
> ***optional:*** These features may be selected if their parent is selected. (e.g., *HBBTV*, *DLNA*, *FOLLOW TV*, *PORTAL MODE*, *WIRELESS DISPLAY*, and *INTERNET LOCK*).
>
> ***inclusive-or:*** At least one feature of this group needs to be selected if their parent is selected. (e.g., *Digital Media Renderer (DMR)* and *Mirror*).
>
> ***exclusive-or:*** Exactly one feature of this group needs to be selected if their parent is selected.(e.g., *Vestel*, *Toshiba*, and *Foxum*).

MBT by itself is helpful in managing variability. Variations can be better managed at the abstraction level of test models rather than test scripts and check-lists. However, the use of MBT falls short to address systematic variability for large scale product families. These families are subject to a high number of variations that cross-cut test models that might be developed for various products [19, 20]. MBT process and test models must be flexible to systematically manage variability and increase the amount of reuse for testing artifacts. SPLE is required for facilitating such a

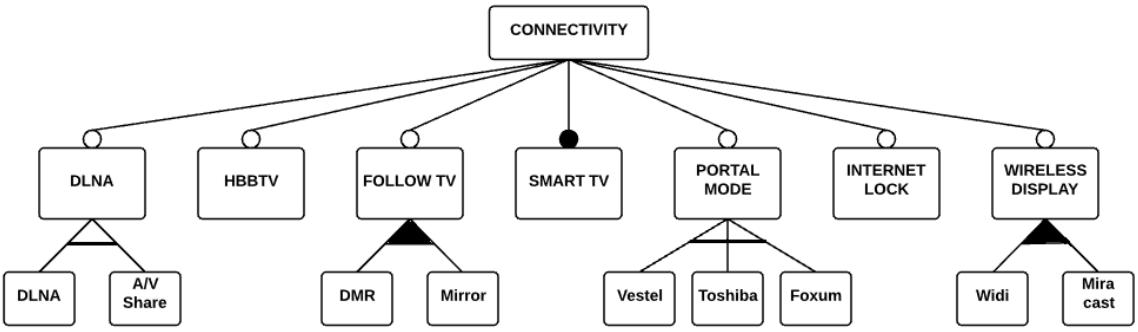


Figure 2: A sample feature model for DTV systems.

systematic and scalable reuse [21, 59].

In the following chapters, we explain our studies in detail and discuss the results.

CHAPTER III

FEEDBACK-DRIVEN MODEL BASED TESTING

In this chapter, we introduce an approach and a toolset, ARME[1] for automatically refining test models based on recorded activities of test engineers.

ARME analyzes the execution traces that are recorded during ET activities. These traces are compared with respect to the possible execution paths in test models to identify omissions. Then, these models are automatically extended to incorporate any omitted system behavior. The models are used for (re)generating test cases that cover the extended parts of these models as well. The recorded traces are also used for updating model parameters. Test models are represented as Markov chains [18], in which transition probabilities are defined for switching among system states. These probability values are updated according to the frequency of visits to system states during ET activities. As a result, the generated test cases focus on the most frequently visited system states. Model refinement continues in an iterative manner as long as ET activities continue.

In all of these case studies, several critical faults were detected after generating test cases based on the refined models. These faults were not detected by the initial set of test cases. They were also missed during the exploratory testing activities.

The chapter is organized as follows. In the following section we present the overall process and methods. The evaluation of the approach is discussed in Section 3.2 in the context of 3 industrial case studies. We conclude the chapter after Section 3.3, where we position our approach with respect to related work.

[1] https://github.com/csgebizli/ARMETool

3.1 Overall Approach

The overall approach is depicted in Figure 3. It consists of an iteration cycle among 3 basic processes: *Exploratory Testing*, *Model Refinement*, and *Model-Based Testing*.

The iteration starts with an initial model developed for MBT. This model is created based on requirements specifications, independent of the ET activities. MBT is followed by ET. We do not interfere with the ET activities, where a test engineer interacts with the system. A built-in tool logs the sequences of events within the SUT throughout ET.

The collected *event sequences* are provided as input for the *Model Refinement* process, which is performed by the ARME toolset. ARME updates the existing test model that is used for MBT in the previous iteration cycle. The model involves states and transitions that represent possible user actions. Transitions represent possible user actions. On the other hand, execution traces are collected as a sequence of low level events (function calls) that take place in the SUT. Hence, ARME takes a third input, namely the *event mapping specification*, which defines a mapping between the abstract states in the model and the events taking part in the collected traces. This mapping is specified in the form of regular expressions. ARME analyzes the execution traces based on the provided mapping and compares them with respect to the possible execution paths in the model of the SUT. Then, the test model is automatically extended to incorporate any omitted system behavior. In the mean time, state transition probabilities are also updated according to the frequency of visited states calculated based on the collected event sequences.

After *Model Refinement*, the *Model-Based Testing* process is repeated. The updated test model is provided to the MBT tool for re-generating test cases. These test cases are executed on the SUT. This iteration cycle can be repeated as long as ET activities continue and new execution traces are obtained. Hence, the number of cycles depends on the reserved test budget and available resources.

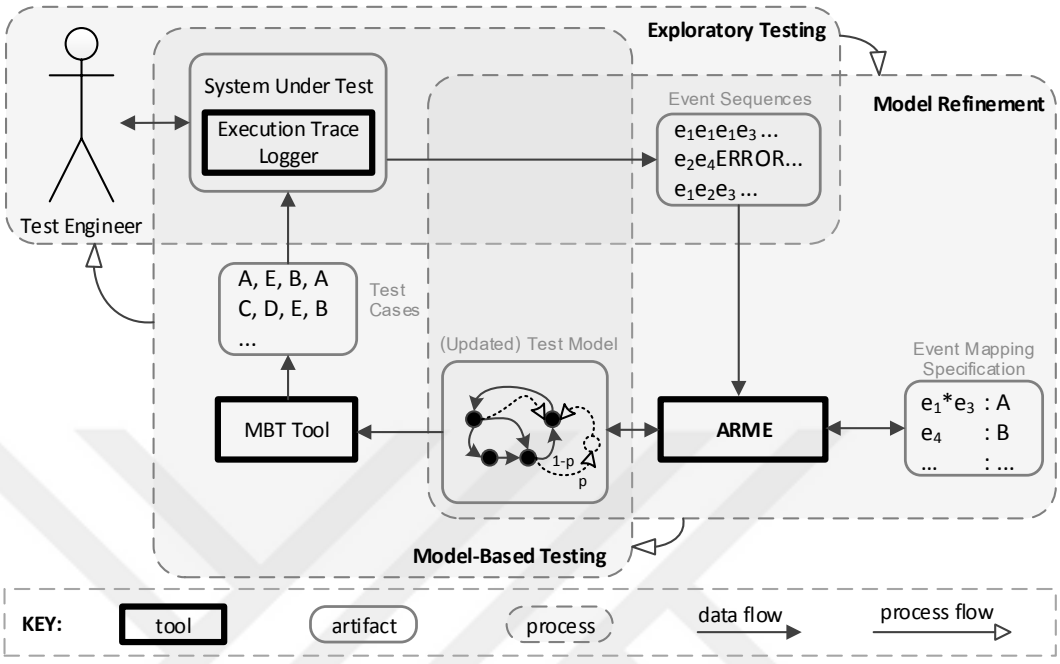


Figure 3: The overall approach.

In the following, we explain our modeling and event mapping approach first. Then, we describe the two types of refinements performed by ARME *i)* adding missing states and transitions to test models, and *ii)* updating state transition probability values.

3.1.1 Modeling Approach and Event Mapping

In this work, we are focusing on event-based, reactive systems in particular. This is mostly the case with MBT approaches in general [30]. In our case, we totally ignore input (value) dependent faults; inputs are basically abstracted away as a sequence of remote controller key press events for a TV system. In principle, our overall approach is agnostic to the type of tool and formalism that is used for creating test models for such a system. In fact, we previously applied our approach on test models that are expressed in ESG formalism as well [24]. However, the application of the approach was manual. In this work, we used MaTeLo[2] as the MBT tool. The ARME tool set

[2]http://www.all4tec.net

is compatible with this tool for automated model refinement. The first reason for employing MaTeLo is to comply with the tools used for industrial case studies. The second reason is that this tool supports the use of Markov chain formalism. Markov chains provided us the ability to perform test case generation in a random-stochastic manner by utilizing transition probabilities. Otherwise, there is no particular reason for choosing this formalism or MaTeLo as the MBT tool. These were already employed by the software test department within the company. All the models that we used for case studies were also already developed with MaTeLo.

In fact, we do not use standard Markov chains but so called Extended Markov chains [60] that are developed with and utilized by the MaTeLo tool. In this formalism, transitions can be labeled with input and output vectors. Hereby, inputs and outputs are observable actions exchanged with the system and they are described with typed variables. Multiple input vectors can be associated with a transition. Each of these input vectors is associated with a probability value such that the sum of these probability values is equal to the transition probability. As such, this modeling formalism includes two levels of probabilities; *i)* the probability to select the transition and *ii)* the probability to select an input vector given that the transition occurs [60]. In our modeling approach, we defined only a single input vector for each transition. This vector includes the sequence of events (i.e., remote controller key presses) necessary to perform the corresponding transition. The sum of all the probability values for all outgoing transitions of a state has to be equal to 1. If there is only one outgoing transition for a state, then the probability value for that transition is 1 by default.

A sample test model is depicted in Figure 4. The legend provided for this figure is also valid for other test models presented in the rest of the dissertation. Hereby, we can distinguish a start state and a terminal state on the top and the bottom of the figure, respectively. Test models have an hierarchical structure and this one is

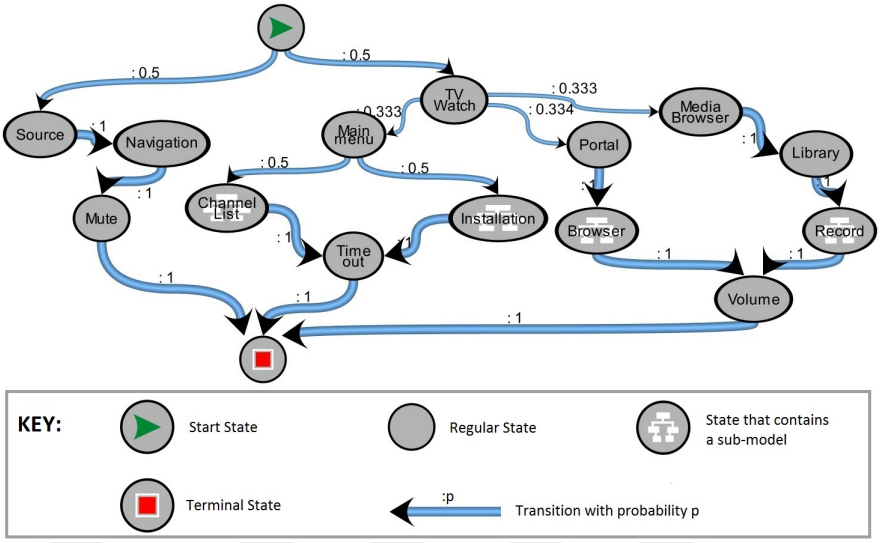


Figure 4: A sample top-level model.

a top level model. States *Channel List*, *Installation*, *Browser*, and *Record* include sub-models corresponding to these states. Each sub-model includes a set of states and transitions just like the top level model. There can also be other sub-sub-models under the sub-models depending on the depth of the hierarchy.

During the ET activities, *Execution Trace Logger* collects execution traces in the form of sequences of function calls. For a DTV, for instance, these function calls usually correspond to the calls that are triggered when the user presses a key on the remote controller. On the other hand, test models include states and transitions that are represented at a higher level of abstraction. Sequences of events trigger transitions and they are specified as inputs for the corresponding transitions. In our approach, a sequence of events that are specified as input for a transition is mapped to the target state of that transition. As such, an *event mapping specification* is created manually to map a sequence of low-level events to states in the model. The relevant sequences of low-level events are defined in the form of regular expressions.

There are three basic assumptions regarding the *event mapping specification*. First, we assume that this specification is correct and complete. It maps each state of the model to a sequence of events (function calls).

Second, we assume that sequences of events that map to states are disjoint. That is, a sequence that is mapped to a state cannot be a sub-sequence taking place in another mapping. In fact, all sequences start with a different event in our case. This makes it scalable to check if a sequence matches to a mapping definition.

Third, we assume that a whole sequence of events that cannot be mapped refers to a new (omitted) state. It might be the case that such sequences represent more than one state. However, it is not possible to decompose such unrecognized sequences of events to multiple states without the domain knowledge. A domain expert can possibly control new states added by the tool and modify/decompose them manually, if deemed necessary.

3.1.2 Extending Test Models

In this sub-section, we describe the model extension procedure in detail. The procedure is outlined in Algorithm 1, which takes different cases into account. We explain the conditions and refinement operations for each of these cases. We also illustrate these cases based on the sample model that was provided in Figure 4. Note that Algorithm 1 refers to a single model refinement step that is iteratively executed as part of the overall approach.

As a running example for describing the model refinement on sample model in Figure 4, consider an event mapping specification listed in Listing 3.1. Assume that a sequence of events is recorded during ET as: *"..., a, b, x, y, c, d, ..."*, *"..., e, f, c, d, ..."*, *"..., h, l, i, ..."*, *"..., kl, lm, n, pr, ..."* where "x" and "y" are not defined in the specification.

The refined model with sample extensions regarding the different cases is depicted in Figure 5. We refer to the corresponding parts of this model, while explaining different cases considered by the algorithm.

Listing 3.1: An example event mapping specification.

```
1  // <Event Sequence>: <States>
2        <a><b>       : Navigation;
3        <c><d>       : Mute;
4        <e><f>       : Source;
5        ...
6        <h><l>       : Playing Record under "Record";
7        <i>          : Youtube under "Browser";
8        ...
9        <kl><lm>     : Antenna Settings under "Installation";
10       <n><pr>      : Advanced Channel List under "Ch. List"
11       ...
12 //
```

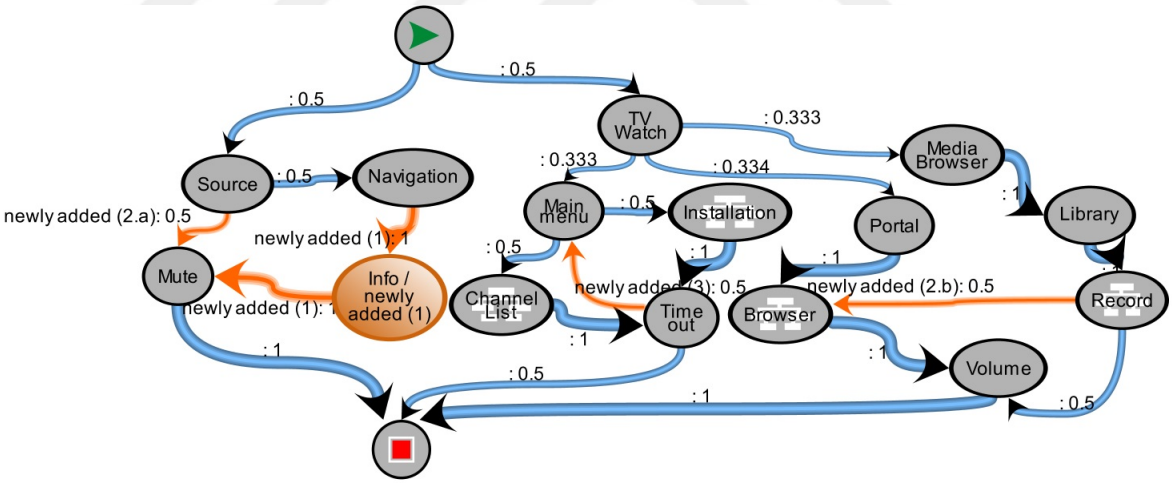


Figure 5: Sample model view after refinements.

The procedure (Algorithm 1) has one main loop (Line 4-35). In this loop, the recorded event sequence is consumed until the end and transitions from one state (S_P) to another (S_F) is traced on the model one at a time. S_P is initialized as the *start state* before the loop (Line 3). In each step, the next sequence of events is mapped to S_F (Line 5). If such a state can be mapped, and if there is a transition defined between S_P and S_F, then S_F becomes the new S_P (Line 34) and the process

Algorithm 1 Model Extension Procedure

```
1: Input: event sequence, test models
2: procedure ExtendModel
3:     S_P ← start-state
4:     while sequence is not consumed do
5:         S_F ← get next mapped state
6:         if S_F is not defined then
7:             create event sequence E
8:             while S_F is not defined do
9:                 e ← skipped event from sequence
10:                append e to E
11:                S_F ← get next mapped state
12:            end while
13:            create new state S_N
14:            define mapping S_N : E
15:            define transitions S_P to S_N and S_N to S_F
16:        else
17:            if transition is not defined from S_P to S_F then
18:                if S_P and S_F are in the same model then
19:                    S_1 ← S_P
20:                    S_2 ← S_F
21:                else
22:                    S_1 ← upper level state of S_P
23:                    S_2 ← upper level state of S_F
24:                end if
25:                if direct transition between S_1 and S_2 is not allowed then
26:                    S_Pre ← common predecessor state for S_1and S_2
27:                    S_Suc ← common successor state for S_1and S_2
28:                    define transition S_Suc to S_Pre
29:                else
30:                    define transition S_1 to S_2
31:                end if
32:            end if
33:        end if
34:        S_P ← S_F
35:    end while
36: end procedure
```

continues with the next sequence of events. A model extension is considered for the following cases.

1. If an event sequence cannot be mapped to a state after S_P, a state is deemed missing in the model (Lines 6-16). This state should take place between S_P and the next state S_F that can be mapped in the event sequence. In this case, a new event sequence, E is created first (Line 7). Events are skipped one by one (Line 9) as long as the remaining sequence cannot be mapped to a state (Line 8). In the mean time, E is appended with all the skipped events (Line 10). Once S_F is found, a new state, S_N is defined and a new mapping is added for this state to be associated with the event sequence E (Line 14). Then, two new transitions are defined from S_P to S_N and from S_N to S_F (Line 15).

   Based on the running example, ARME recognizes that "x" and "y" are not defined, so it builds a regular expression from these traces and maps them to a state called "newly added (1)" and connects the transitions to this new state to complete the model. This case is illustrated with the set of extensions marked with orange color and labeled as *newly added (1)* in Figure 5.

2. If S_F is mapped (Line 16) but there exists no transition from S_P to S_F (Line 17), then a transition is missing. We consider two sub-cases for this case:

   (a) If S_P and S_F are part of the same model, then we can directly add a transition from S_P to S_F. As such, they are recorded as S_1 and S_2, respectively (Lines 19-20).

   Based on the running example, there is a path from states *Source* to *Mute*, so ARME adds a new transition (newly added (2.a)) between these states. This case is illustrated with the extension marked with orange color and labeled as *newly added (2.a)* in Figure 5.

(b) If not, we find their *upper-level* states. A state S_U is an upper-level state of S if S_U is associated with a sub-model and S takes place in that sub-model. If a state S takes place in the top-level model, then the upper-level state of S is itself. Let's say the top level model includes states A and B such that S_P takes place in the sub-model of A and S_F takes place in the sub-model of B. We cannot directly add a transition from S_P to S_F in this case. They reside in two different sub-models. Instead, we can add a transition from state A to B, which are defined as part of the same model (i.e., the top level model). As such, states A and B are recorded as S_1 and S_2, respectively (Lines 22-23).

Based on the running example, ARME recognizes the execution traces *"..., hl, i, ..."* which are mapped to two states: *Playing Record* and *Youtube*. Since these states are not part of the models at the same level, it cannot directly add a transition between them. Therefore, it finds their upper-level states; *Record* is upper level state of *Playing Record* and *Browser* is upper level state of *Youtube*. Then it adds a new transition (newly added (2.b)) from *Record* to *Browser*. This case is illustrated with the extension marked with orange color and labeled as *newly added (2.b)* in Figure 5.

3. It might not be possible to make a direct transition from S_1 and S_2 for the 2.b case above. In this case (Line 25), we create a cycle by adding a transition from the common successor back to the common predecessor of these states (Line 28). Then, it becomes possible to first visit S_1, then follow the cycle to visit the common predecessor of S_1 and S_2 again, and then visit S_2.

   Based on the running example, ARME recognizes execution traces *"..., kl, lm, lm, m, n, pr, ... "*, which are mapped to two states: *Antenna Settings* and *Advanced Channel List*. These states are not at the same level. Therefore it

finds their upper-level states. A direct transition between these states are not allowed. So, ARME adds a new transition (newly added (3)) from the common successor back to the common predecessor. This case is illustrated with the extension marked with orange color and labeled as *newly added (3)* in Figure 5. We directly add a transition from S_1 and S_2 as determined in the second case (*2.a* or *2.b*) if this constraint does not apply (Line 30).

Items 1 and 2 above are generic, whereas item 3 is related to a (possible) constraint regarding the application domain. A direct transition might not be allowed among a group of states due to a domain-specific or application-specific constraint. In our case studies, for instance, we employed test models that are designed for DTV systems. These models are designed such that direct transitions among the sub-menu items are not possible. The user has to go back to the main menu first. This can happen automatically after a timeout period; as such, this change might not be visible in the sequence of user events. There exists a timeout state for each menu group and one should go through this state and the state that represents the main menu before switching to another menu. If such constraints exist, one must explicitly specify (tag) groups of states, among which there can be no direct transitions. In our case, we can automatically identify and tag states that represent TV sub-menu items since they follow a pattern: a group of states represent sub-menu items if these states have in-degree and out-degree values equal to 1, if they have the same predecessor and if they have the same successor, which is not the terminal node. In the running example, for instance, the upper-level states of *Antenna Settings* and *Advanced Channel List* have in-degree and out-degree values equal to 1. They have also the same predecessor and the same successor, which is not the terminal state.

In the following sub-section, we describe refinements that are performed regarding state transition probability values in the model.

3.1.3 Updating State Transition Probability Values

In our approach, we not only extend models for missing paths, but we also refine probability values according to the usage profiles collected during ET. These profiles do not reflect real user behavior; however, they have helped us to cover error prone scenarios. We assume that ET activities focus on such scenarios. The numbers of visits to different system states are counted for subsequent iterations of ET activities. For each iteration i, a probability value, $P_{calc}(t)^i$ is calculated for each transition t, according to Equation 1.

$$P_{calc}(t)^i = \frac{\text{\# of visits for target(t)}}{\sum_{\forall t', source(t)=source(t')} \text{\# of visits for target(t')}} \tag{1}$$

Hereby, the number of visits to the target state of the transition is calculated as the nominator. The denominator is calculated as the sum of the numbers of visits for all the alternative transitions. These are the transitions, which have the same source states as t. The calculated $P_{calc}(t)^i$ values in each iteration are not directly assigned as the probability values on transitions. Instead, they are reflected to the calculations in subsequent iterations according to Equation 2.

$$P(t)^i = \begin{cases} \frac{1}{\text{\# of alternative transitions}} & i = 0 \\ \left| P(t)^{i-1} - \frac{P(t)^{i-1} - P_{calc}(t)^{i-1}}{k} \right| & i \geq 1 \end{cases} \tag{2}$$

Initially, default probability values are assigned by the MaTeLo tool for the first iteration. Let's assume for a state s that there exist n outgoing transitions to n different states. Then, for each of these transitions, the probability value is calculated as $P(t) = 1/n$ when $i = 0$. In subsequent iterations, this value is updated according to the $P_{calc}(t)$ calculated in the previous iteration. Hereby, k is a coefficient ($k > 0$) that allows us to adjust the weight of the $P_{calc}(t)$ value on calculating $P(t)$. If $k = 1$, the

calculation of $P(t)$ does not depend on the previous $P(t)$ value but is completely determined by $P_{calc}(t)$ that is calculated in the previous iteration. Previously calculated $P(t)$ value has more influence on the calculation as k increases.

In our case studies, we fixed the value of k as 2.

We also proposed another approach about updating the state transition probabilities. The details about that study is in Chapter 4.

In the following section, we explain these studies with research questions and we will present the experimental setups.

3.2 Industrial Case Study

In this section, we introduce three industrial case studies for the application of our approach to DTV systems developed by Vestel. There are hundreds of test suites being used for testing these systems. Some of these test suites are manually created. Some others are automatically generated by employing MBT. Many models have been created for this purpose. They are represented as extended Markov chains, some of which include thousands of states and transitions. MaTeLo is used for model creation and test case generation. All the test cases are automatically executed with VesTA (See Chapter 1). In parallel, manual exploratory tests are performed by experienced test engineers.

We performed case studies with 3 different models. The same software version was used throughout the studies and no faults were fixed to keep the results consistent. The first model is related to the channel installation, named as Digital Video Broadcasting - Terrestrial Channel Installation (DVB-TCI). The second one is for testing the Media Browser (MB) module, which allows the users to open video, music, or picture files on TV via external devices. The third one is created for Random Torture Test (RTT). RTT is actually *stress testing*, which aims at discovering how the system behaves under sustained use. The term *torture* is adopted within the company

instead of stress testing and it was reflected to the naming of test models. The RTT test model involves the use of several features that are expected to be mostly used by the end users.

3.2.1 Research Questions

Our goal is to refine test models iteratively and to be able to generate more effective test cases/steps by using the refined models for detecting more faults. As such, we defined the following 4 research questions:

> ***RQ1:*** To what extent models are refined by ARME in terms of updated or newly added states and transitions?
>
> ***RQ2:*** To what extent successive iterations of the approach contribute to finding new faults?
>
> ***RQ3:*** To what extent ARME contributes to finding new faults compared to MBT and ET?
>
> ***RQ4:*** How does ARME affect test suite size in terms of the number of test steps?

In the following, we explain the experimental setup we used for evaluation. Then, we present and discuss the obtained results for addressing the research questions. We also illustrate steps of our approach using the case studies as a running example.

3.2.2 Experimental Setup

Test Models: We used 3 different test models that were previously created for DVB-TCI, MB, and RTT based on requirement specifications documents. All the 3 models were already being used in production. They were developed by the software test department within the company prior to our studies. We learned that the test models regarding DVB-TCI and MB were developed by manually analyzing the

corresponding requirements specification documents. In addition, usage profiles that are collected from real users were analyzed for the development of the RTT model. Mostly used DTV features were identified from these profiles and the model was developed to define the behavior just for these features. Figure 6 depicts the top level RTT model.

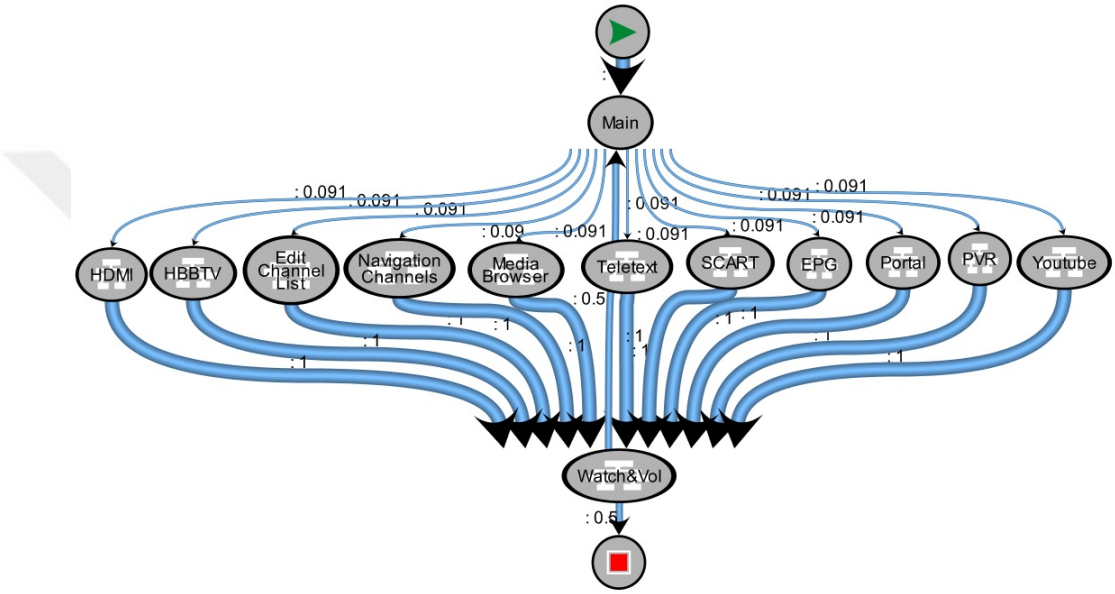


Figure 6: A snapshot of the initial test model for RTT.

Mapping Specifications: We created a mapping specification for each of the test models. As an example, Listing 3.2 shows a part of the mapping specification created for the MB model[3]. Hereby, each state is represented by a regular expression. For instance, we can see at Line 5 of Listing 3.2 that a function call named p, followed by zero or more function calls named v maps to the state *Playing Video*. These expressions are defined manually to map a sequence of low-level events to states in the model. There is usually a specific function call that handles a remote controller command. For this reason, a single function call is mapped to a state in most cases.

[3]Due to confidentiality, we do not disclose the real function names used in the implementation.

Listing 3.2: Part of the event mapping specification for the MB model.

```
1 // <Event Sequence>            : <States>
2       <mansr><f>*<snd>         : Tune to DVB-S channel;
3       <mb><mn>                 : MB menu;
4       <v><mn>                  : Video option of MB menu;
5       <p><v>*                  : Playing Video;
6       <sub>                    : Press Subtitle;
7 //
```

Test Case Generation: Based on the developed models, test cases are generated automatically with two different test case generation algorithms that are provided by the MaTeLo tool. For the first two case studies, test models were designed to cover all the features as documented in requirement specifications. We employed the so called *Minimum Arc Coverage* algorithm [61]. This algorithm is also known as *Chinese Postman* [61]. We used this algorithm with default parameters, in which the coverage criterion is basically transition coverage. Test case generation stops when all the transitions in the model are visited at least once.

For the third case study, where updated transition probability values are utilized, test models were designed for stress testing. Hereby, the goal was not to cover all the features; on the contrary, testers were focusing on a set of specific features, while ignoring some other features. Hence, we exploited execution traces to obtain a usage profile and refine transition probabilities accordingly. We employed the so called *User Oriented* algorithm [61] to generate test cases based on these refinements. This algorithm has two parameters: *i)* the maximum number of test steps within a test case and *ii)* the maximum number of test cases. We set the maximum number of test steps to 5000 (a limit that we never reached). We set the maximum number of test cases to 1. As a result, this algorithm starts from the start state and traverses the model by choosing a transition at each state non-deterministically based on transition

probabilities. It does not guarantee transition coverage. It generates test cases by using the probability values annotated on the transitions of the model. If a probability value of a transition which is going out of a state is higher than other transitions going out of the same state, then the algorithm will chose that transition stochastically. The algorithm terminates when the final state is reached. The generated test cases for all the 3 case studies are used in production.

ET Activities: ET activities are performed by three test engineers by focusing on the features in the scope of these case studies. The number of engineers and test durations were not deliberate choices made by us. ET activities were already being performed regularly in the company. We just collected data without interfering with the process. During the ET activities, all the execution traces are logged. We asked test engineers to save the records that were logged during tests. We also asked them to record each failure they found.

Faults Found by Test Cases Generated from the Initial Test Models and Faults Discovered during the Initial ET Activities: For the DVB-TCI case, with the initial model, 22 test cases were generated, which include 261 test steps in total. After executing the generated test cases, 3 faults were found. During the ET activities, in total 5 faults were found including the 3 faults which were found with the test cases from the initial model.

For the MB case, with the initial model, 107 test cases were generated, which include 809 test steps in total. After executing these test cases, 36 faults were found. During the ET activities, in total 44 faults were found including the 36 faults which were found with the test cases from the initial model.

For the RTT case, with the initial model, 1 test case was generated which includes 950 test steps in total. After executing the test case, 1 fault was found. During the ET activities, in total 2 faults were found including the 1 fault which was found with

the test case from the initial model.

These results show that ET activities are highly effective in detecting faults. In all the case studies, ET found all the faults revealed by MBT. Besides, test engineers found some additional faults which had not been found with MBT before. These results support the motivation for our approach.

In the following section, we show that these results can be further improved by refining existing test models with ARME based on ET. We can detect further new faults, which were missed by both ET and MBT before refinements of ARME.

3.2.3 Results and Discussions

In this section, we evaluate the results obtained for the 3 different case studies and discuss our experiences from the perspective of the 4 research questions.

Refinement of Models by ARME (RQ1) ARME has introduced many extensions to test models based on the collected execution traces during ET activities. For instance, Figure 7 depicts the top level DVB-TCI model previously created with the MaTeLo tool. The original model did not consider the fact that after going into one of the sub-menu items, the user can go back to the main menu directly. There was no such a transition defined. ARME added this missing transition to the model. The corresponding transition introduced by ARME is marked by painting it in orange color. For this model, there was a missing transition between the states of two different sub-models. These sub-models were representing sub-menu alternative options. Therefore, a direct transition between them was not allowed. As a result, a transition was added from the common successor state back to the common predecessor state in the top level model.

For the MB model, ARME added several new states and transitions. Some of the refined models are depicted in Figure 8, Figure 9, and Figure 10, where modifications are marked with orange color. The first two rows of Table 1 show the overall updates

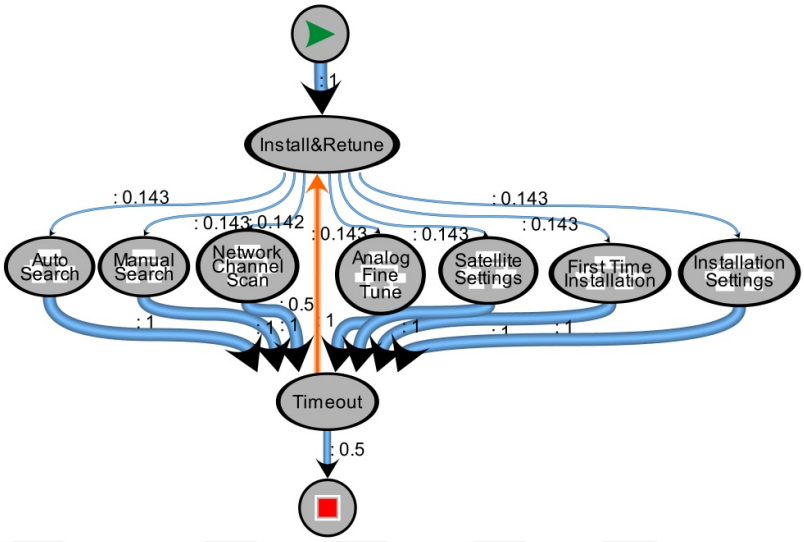


Figure 7: The refined top level DVB-TCI model.

performed on test models after all the refinement iterations for the DVB-TCI and MB case studies. For the DVB-TCI case study, the number of added states is 3 and the number of added transitions is 6. For the MB case study, the number of added states is 15 and the number of added transitions is 23. We did not update the transition probabilities for these two case studies. On the other hand, we can see that there are no new states or transitions added to the test models for the RTT case study. However, all the transition probabilities in the top level model are updated.

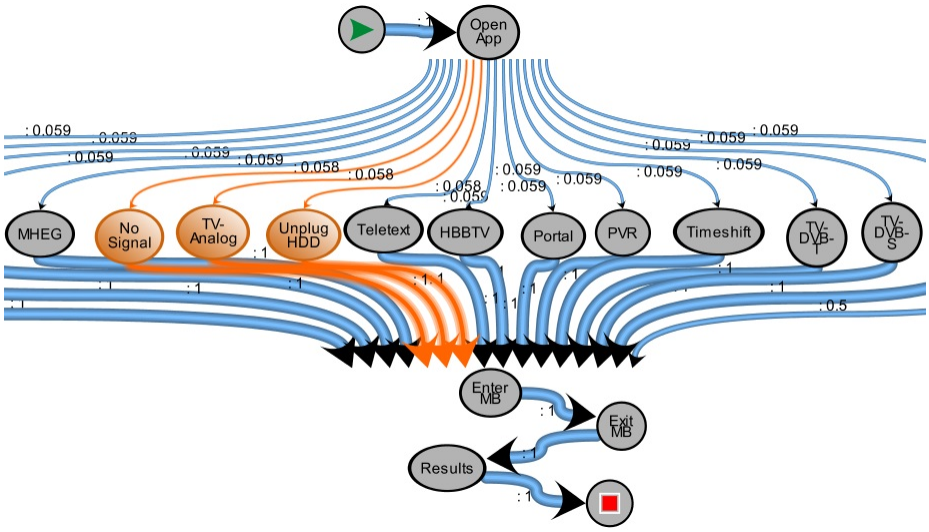


Figure 8: The refined Enter/Exit MB top level model.

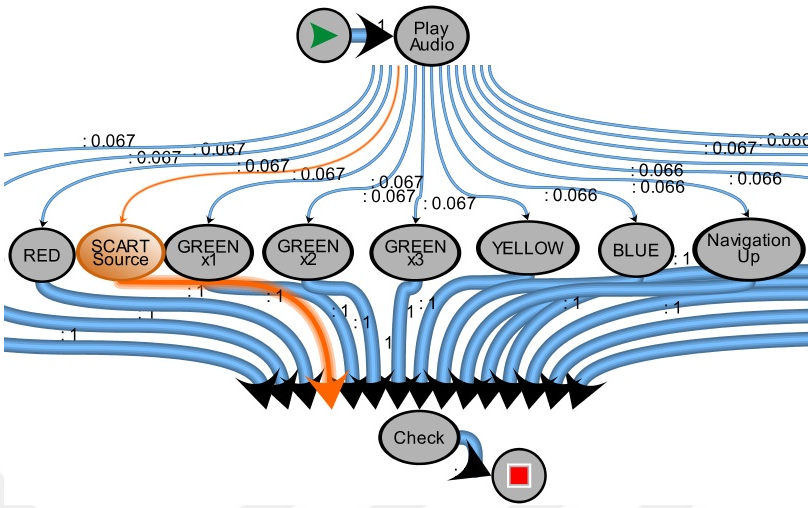


Figure 9: The refined Playing Audio sub-model.

Case Study	# of Added States	# of Added Trans.	# of Updated Trans.
1. DVB-TCI	3	6	0
2. MB	15	23	0
3. RTT	0	0	11

Table 1: Refinement of test models throughout iterations.

Updates of probability values on state transitions are performed iteratively as explained in Section 3.1.3. For instance, Table 2 shows the updates performed on the top level RTT model after the first iteration. Hereby, the first column lists the alternative state transitions from the *start* state. The second column shows the initial probability values before the update. They are all equal to each other by default. The third column lists the number of visits to the corresponding states during manual testing activities. The fourth column shows the calculated probability values (i.e., P_{calc}) according to the number of visits. Finally, the last column lists the updated probability values calculated based on Equation 2. Hereby, the states that are listed in the first column are those states that are directly reachable from the start state in the top level model. Hence, the updated probability values as listed in the last column are assigned to transitions, where the source of the transition is the start state and

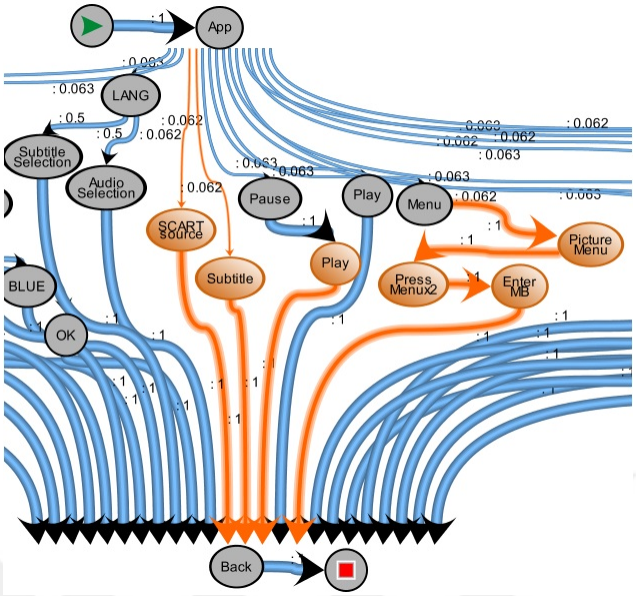


Figure 10: The refined Playing Video sub-model.

the target of the transition is the state that is listed in the first column of the same row. These values are re-calculated and replaced throughout iterations. The number of iteration cycles depends on the reserved test budget and available resources. In our case studies, we did not have any control over these. We repeated the cycle until new faults are not detected by model refinement. That is, we used the absence of new faults as the stopping criterion.

New Faults Detected by ARME in Successive Iterations of Model Refinement (RQ2) To address *RQ2*, for each case study, we first recall the initial results that were obtained before any model refinement was performed. Then, we list the results obtained after each successive iteration of model refinement performed by ARME.

For the DVB-TCI case, 22 test cases were generated from the initial model which was created based on requirement specifications. After executing these test cases, 3 faults were found. Independent from this, ET activities were performed on DVB-TCI, and 5 faults were found. These 5 faults include the 3 faults which were found with the

State-Module	Init. Prob.	# of Visits	Calculated Prob.	Updated Prob.
EPG	0.091	7	0.093	0.092
Portal	0.091	4	0.053	0.072
Youtube	0.091	12	0.16	0.125
HBBTV	0.091	0	0	0.045
Media Browser	0.091	3	0.04	0.066
PVR	0.091	6	0.08	0.086
HDMI	0.091	1	0.013	0.052
SCART	0.091	3	0.04	0.066
Teletext	0.091	2	0.026	0.059
Edit Channel List	0.091	3	0.04	0.066
Navigation	0.091	34	0.45	0.27

Table 2: Calculated probability values for the RTT case study.

test cases generated from the initial model before. That is, 2 new faults were found with ET activities. Then we refined the initial model by taking the execution traces from ET activities. We regenerated the test cases from the refined model and we executed this new set of test cases. This time, 8 faults were found in total. 5 of them were found before. However, 3 new faults were found. These faults have not been found before either with MBT using the initial model or with ET activities. Moreover, these 3 faults were highly critical; one of them caused the DTV to reset itself after a channel search was performed; another one was the reason for duplicate channels added at the end of channel search; the activation of the third fault resulted in a crash of the system. We continued to take execution traces from new ET activities that focused on the DVB-TCI feature. Each time, we refined test models with ARME based on new traces obtained. Then, we applied MBT based on the refined model. We performed these steps iteratively until there was no new fault detected. The results of the DVB-TCI case study are summarized in Table 3. The first row lists the initial results obtained with the original model. The following rows show the results after successive iterations. We can see that new faults were discovered after every

Model	# of States	# of Trans.	# of Test Cases	Total # of Test Steps	# of New Faults Detected by ET	# of New Faults Detected by ARME	Total # of Faults
Initial	85	106	22	261	-	-	3
Iter. 1	86	109	1	252	2	3	8
Iter. 2	86	110	1	265	1	1	10
Iter. 3	88	112	1	267	0	1	11
Iter. 4	88	112	1	267	0	0	11

Table 3: Properties of the DVB-TCI model and the number of faults found throughout successive iterations of model refinement.

iteration of model refinement. We can also notice that the number of test cases was decreased to 1 after the first iteration. This is due to the loop introduced by ARME (See Figure 7), which makes it possible to cover many execution paths in a single test case.

Initially, 107 test cases were generated for the MB case, which include 809 test steps in total. After executing the generated test cases, 36 faults were found. However, 8 more faults were discovered during the ET activities. After the refinement of the test models based on these activities, 853 test steps were generated. 48 faults were found when these test cases were executed. 36 of these faults had been already detected before with the initial model. 12 new faults were discovered, 8 of which had been discovered during the ET activities. 4 of the new faults had not been detected before neither with the initial model nor with the ET activities. These 4 faults were highly critical as well; the first one made the audio/video output disappear; the second one corrupted remote controller key buffers; the third one made user commands undetectable; the last one caused the TV to reset itself. These faults were detected after the first model refinement only. Overall results of this study are summarized in Table 4. The first row lists the initial results obtained with the original model. The remaining rows list the results obtained after successive iterations of model refinement. We can see that new faults were discovered after every iteration

Model	# of States	# of Trans.	# of Test Cases	Total # of Test Steps	# of New Faults Detected by ET	# of New Faults Detected by ARME	Total # of Faults
Initial	171	298	107	809	-	-	36
Iter. 1	181	314	113	853	8	4	48
Iter. 2	183	316	113	861	1	2	51
Iter. 3	186	321	115	873	0	1	52
Iter. 4	186	321	115	873	0	0	52

Table 4: Properties of the MB model and the number of faults found throughout successive iterations of model refinement.

of model refinement. We can also observe that the number of test cases increases by 6 after the first iteration. However, 12 new faults were discovered. This is because, more than one fault can be found with one test case.

For the RTT case, initially, 1 test case which includes 950 test steps was generated. After executing the generated test case, 1 fault was found. However, 1 more fault was discovered during the ET activities. After the first model refinement based on these activities, 940 test steps were generated. 4 faults were found when the new test case was executed. 1 of these faults had been already detected before with the initial model. 3 new faults were discovered, one of which had been discovered during the ET activities. 2 of the new faults had not been detected before neither with the first model nor with the ET activities. These 2 faults were highly critical as well; the first one caused a crash of an application; the second one corrupted remote controller key buffers after HBBTV[4] operation. We can see these results in the second row of Table 5 (Iteration 1). In Table 5, we can also see the results obtained with the initial model in the first row (Initial) and results that are obtained after successive model refinements. These refinements are continued until the fifth iteration after which no new faults were detected. We can see that new faults were discovered after each

[4]HBBTV (http://www.hbbtv.org/). It is an initiative for harmonizing the broadcast/broadband delivery of entertainment services for TVs and set-top boxes.

Model	# of States	# of Trans.	# of Test Cases	Total # of Test Steps	# of New Faults Detected by ET	# of New Faults Detected by ARME	Total # of Faults
Initial	87	116	1	950	-	-	1
Iter. 1	87	116	1	940	1	2	4
Iter. 2	87	116	1	929	1	2	7
Iter. 3	87	116	1	923	0	1	8
Iter. 4	87	116	1	936	0	1	9
Iter. 5	87	116	1	927	0	0	9

Table 5: Properties of the RTT model and the number of faults found throughout successive iterations of model refinement.

of the first four iterations of model refinement. Note that the number of test steps decreases while the number of test cases remains same. At the same time, the number of detected faults increases. This is because, at each iteration, test case generation focuses on different execution paths according to the updated probability values on state transitions.

Overall Contribution of ARME in Finding New Faults Compared to MBT and ET (RQ3) Results of the case studies show that we can identify critical faults by refining test models based on ET with the help of ARME. Our approach and the tool support enabled the transfer of knowledge from manual ET activities to MBT. Table 6 shows the comparison of the number of faults found with the initial test models, with ET activities and with ARME after all the successive iterations for each of the 3 case studies.

For the DVB-TCI case study, 3 faults were found with the initial test model. Then, 3 additional faults were revealed during the ET activities. In addition to these 6 faults in total, 5 more new faults were detected with ARME. Hence, 45.45% of the overall faults were detected only by ARME. Likewise, 7 and 6 new faults were detected only by ARME in the second and the third case studies, respectively. These faults

Case Study	# of Found Faults with Initial MBT	# of Found New Faults with ET	# of Found New Faults with ARME	Ratio of New Faults Found by ARME
1. DVB-TCI	3	3	5	45.45%
2. MB	36	9	7	13.46%
3. RTT	1	2	6	66.66%

Table 6: Comparison of the number of faults found with the initial test models, with ET activities and with ARME.

represent 13.46% of the overall faults for the second case study, whereas the ratio is 66.66% for the third case study.

We observed that the effectiveness of the models used for MBT can be significantly increased by exploiting feedback from ET. Integration of the two testing techniques is highly effective in detecting faults and reusing domain knowledge. We showed that our approach helps to detect critical faults and increases the efficiency of the testing process. As an initial investment, the approach requires additional effort for defining the mappings of events to states. However, this is a one-time effort. Without ARME we would only detect the faults that are revealed with the initial MBT and the ET activities only. We could still refine test models manually, at least whenever a new fault is discovered during ET activities; however, for each such case, we would have to debug the model and the execution traces to reveal why the fault was not detected and we would have to manually identify missing edges and states.

Effects of ARME on Test Suite Size (RQ4) We compared the test suit sizes that are obtained with the initial test models and those that are obtained after successive iterations of model refinement with ARME. Table 7 shows the difference in terms of the total number of test steps. In the first case study for the DVB-TCI model, the number of test steps was increased by 2.3%. In the second case study for the MB model, the number of test steps was increased by 7.91%. In the third case study for the RTT model, the number of test steps was decreased by 2.42%. Results show

Case Study	Total # of Test Steps with Initial MBT	Total # of Test Steps after ARME	Difference in the Size of Test Steps
1. DVB-TCI	261	267	+2.3%
2. MB	809	873	+7.91%
3. RTT	950	927	-2.42%

Table 7: The difference of test suite sizes in terms of the total number of test steps after successive iterations of model refinement with ARME.

that ARME does not always lead to significantly larger test suites compared to those obtained with the initial test models. On the contrary, refinements of ARME can even decrease the number of test steps although the execution of these steps enables the detection of new faults. Hence, test suites become more effective in terms of the number of detected new faults per executed test step.

3.2.4 Threats to Validity and Limitations

We have two basic assumptions regarding our approach. First, we assume that the collected traces always reflect correct, possible user behavior. Second, we assume the availability of test engineers who regularly perform ET activities. The general applicability of our approach is constrained by our assumptions regarding the application domain and availability of experienced test engineers for performing ET activities in the working context.

The second assumption is a crucial one since the main purpose of our approach is to exploit feedback received from ET activities to refine testing models. Our approach is not usable without the availability of such feedback.

The first assumption limits the set of possible subject systems since they have to be event-based systems in which all user allowed actions are correct. That is, the system should not crash or lock due to an unexpected sequence of user actions. Although this assumption brings in a limitation, most of the consumer electronics products

such as smart phones, refrigerators, washing machines, dishwashers, air conditioners, and cookers can be considered as candidate subject systems that conform to our assumption. In fact, we performed another preliminary study for a camera application of smart phones produced by the same company. We detected a new fault after we refine its test model based on execution traces obtained from ET activities.

Our model refinement algorithm considers multiple cases for deciding on how to extend the given test models (Recall Section 3.1.2). One of these cases considers an application-specific constraint, in which a direct transition is not allowed among a group of states in the test model. If such a constraint exists, one must explicitly specify (tag) the corresponding groups of states. In our case, we could automatically identify and tag such states since they conform to a pattern in the test models. This might not be possible for all types of systems. In principle, one can remove this case from the algorithm or add other types of constraints to make it more effective for other systems. The assumptions regarding the event mapping specification (See Section 3.1.1) can also be altered. We observed that incorporation of more domain knowledge leads to possibly less generic but more efficient model refinement processes.

Our approach requires multiple iterations. Model update cost is negligible (takes couple of minutes), however, we need to generate test cases and execute them after each iteration. We ignored this additional cost, because these tasks are automated.

3.3 Related Work and Our Contributions

There exist a large body of work on MBT techniques [62], tools [44] and different types of models employed like finite state machines [63], [64], and Markov chains [18]. There also exist surveys on MBT [62] and several case studies [44, 65] where the effectiveness of MBT is evaluated. The surveyed studies and the applied MBT techniques are mainly focusing on the modeling approach and test case generation methods. The evaluation is performed by comparing the number of faults detected by MBT

with respect to the number of faults exposed by traditional testing approaches. However, iterative refinement of the developed models and the incorporation of domain knowledge have not taken much attention in the literature.

Recent advances in automatic black-box testing were summarized by [66]. In their literature review, existing approaches are grouped into four categories as random testing, MBT, testing with complex inputs, and combinatorial interaction testing. In the area of MBT, it is confirmed that the effectiveness of this approach is limited by the cost of producing high quality models [66]. Hence, automatically inferring these models is considered to be a promising research direction, which has taken the attention of researchers recently. The other alternative is to apply MBT on specified models [66]. Inferred models are further grouped into two categories: ripped models and learning-based models [66]. Ripped models are automatically extracted from the system and used for MBT. On the other hand, learning-based models are created iteratively in two phases. First, a model is inferred to generate a set of test cases. Second, observations regarding system behavior during the execution of these test cases are used for augmenting the existing model. The process iterates between these two phases. There are several recently proposed approaches [67, 68, 69] that can be considered within this category. Our approach also relies on learning-based models. However, it is distinguished from the existing approaches in two ways. First, these approaches do not exploit ET activities to refine the model. They exploit test cases that are generated from the previous version of the model. Second, model refinement is limited to the augmentation of missing edges and transitions. In our work, we employed Markov chains as the modeling formalism and we also updated probability values associated with transitions.

A detailed literature review regarding specification mining is provided in the Ph.D. thesis of [70]. The common approach among the reviewed studies is summarized as the application of data mining or machine learning techniques on source code or execution

traces to generate specifications in various formats [70]. Collected execution traces and mined specifications are defined at different levels of abstraction. For instance, component interactions are analyzed in [71], whereas object behavior models are mined by TAUTOKO [72] in the form of FSA. In terms of the model abstraction level, our approach is more aligned with the Observe-Model-Exercise* [69] approach, which derives a model of the Graphical User Interface (GUI) in the form of Event Flow Graph (EFG). TAUTOKO [72] and Observe-Model-Exercise* [69] are similar to our approach in the sense that they both employ an iterative model refinement approach. They start with a possibly incomplete model of the application. Test cases are generated and executed while new events or states are identified by monitoring the execution. Then, the model is extended with these events/states. However, these approaches do not exploit ET activities to refine the model. In addition, the adopted formalisms do not involve transition probabilities to focus test case generation on error-prone execution paths.

MBT and capture-replay testing techniques were previously integrated [73]. This integration effort focuses on refining and adapting a capture-replay testing tool based on changes in the user interface due to evolution. In our work, we focus on refining test models based on the captured execution traces instead. We did not consider evolution explicitly; however, it is possible to address changes in the system by updating the event mapping specification.

There exists another approach [74] that proposes the automated refinement of models for testing. The approach employs an EFG [74] to represent test models for testing applications through their GUI. Cause-effect relationships among the GUI elements are observed at run-time, while the generated test cases are executed. The model is iteratively refined based on the inferred relationships. This approach does not rely on execution traces that are collected during ET activities to refine the model. Instead, the system is executed with a seed test suite that is generated using an

existing model of the GUI. The assumption is that this model represents all possible sequences of events that may be executed on the GUI [74].

There exist other so-called *automated feedback-based techniques* introduced previously to refine the models of SUT or input data for testing [75, 76, 77, 78]. Some of these techniques aim at increasing the code coverage [75, 76], whereas some others aim at generating more test cases to increase the coverage of system behaviors [77, 78]. Regardless of their goals, however, all these techniques rely only on the execution of a previously created test suite to obtain feedback. Hence, the feedback that can be obtained is limited by the coverage of the system behavior achieved with this test suite. In our work, we addressed this limitation by coupling MBT and ET processes.

Besides the reported studies on MBT, ET has also been evaluated in the context of industrial case studies [79]. These studies investigate the impact of human personality [80], learning styles [81] and the way that different types of knowledge [82] are utilized like domain knowledge, system knowledge and general software engineering knowledge. Their conclusions are mainly drawn from interviews that are performed at different companies [79].

Exploratory modeling [83] has been introduced as an approach for model development based on the principles of ET. Hereby, a state diagram of the expected behavior of the SUT is developed first. Then, this model is refined by observing the different states and behaviors while interacting with the SUT. However, this approach has no tool support and it was not evaluated in the context of an industrial case study.

In practice, ET is usually performed manually and MBT is applied as a complementary, rather than an integrated approach. To the best of our knowledge, there has been no method supported by a toolset to couple these approaches and utilize the knowledge gained in ET as a feedback for MBT. In this work, we showed that such an approach is effective for detecting faults.

CHAPTER IV

RISK-DRIVEN MODEL BASED TESTING

In this chapter, we introduce a risk-driven three-step model refinement approach and a toolset, RIMA [1] for automatically updating probability values in 3 steps for augmenting information regarding the risk of failure.

We represent test models in the form of Markov chains. These models comprise a set of states and a set of state transitions that are annotated with probability values. These values steer the test case generation process, which aims at covering the most probable paths. RIMA refines these models in three steps:

- *updates transition probabilities based on a collected usage profile.*
- *updates the resulting models based on fault likelihood at each state, which is estimated based on static code analysis.*
- *performs updates based on error likelihood at each state, which is estimated with dynamic analysis.*

We generate and execute test cases after each refinement step. We applied our approach in the context of two industrial case studies for MBT of a DTV system. We observed promising results, in which new faults were revealed after each refinement.

The chapter is organized as follows. In the following section we introduce our approach. In Section 4.2 we explain the application of our approach in the context of two industrial case studies and present the results. We discuss related work in Section 4.3 and conclude the chapter.

[1] https://github.com/csgebizli/RIMA

4.1 Overall Approach

The overall approach is depicted in Figure 11. We assume that a model of the system exists in the form of a Markov chain. First, we generate test cases based on this model by using the MaTeLo tool. The test case generation process focuses on covering the most probable paths on the model according to transition probabilities. Next, the generated test cases are executed on the system. We collect a memory usage profile during test case execution. After test case execution, RIMA updates the model. This cycle of model update, test case generation and test execution is repeated three times. Each time, the model is updated based on a different source of information. In the first update cycle (5.1), usage profile is used for assigning a probability value to each transition. As a result, the generated test cases after this update will focus on execution paths that are mostly visited by the users of the system. In the second update (5.2), we exploit static analysis alerts generated for the source code. We use Klocwork[2] as the static analysis tool and we use alerts generated by this tool to estimate the relative risk of faults [84] being present for different software modules. These modules are associated with different states in the model based on the represented feature and utilization of modules for that feature. As such, RIMA updates the model based on estimated risks. In the third update (5.3), we exploit the memory usage profile that is collected during the previous test cycles. This profile reveals test execution paths that lead to memory leaks, and it is used for estimating the relative risk of errors. RIMA performs a further update on the model based on this estimation. Hereby, we update the transition probabilities for only top level states each of which can be mapped to a particular DTV feature.

In the following sections, we explain how the model updates are performed in each iteration.

[2]http://www.klocwork.com/

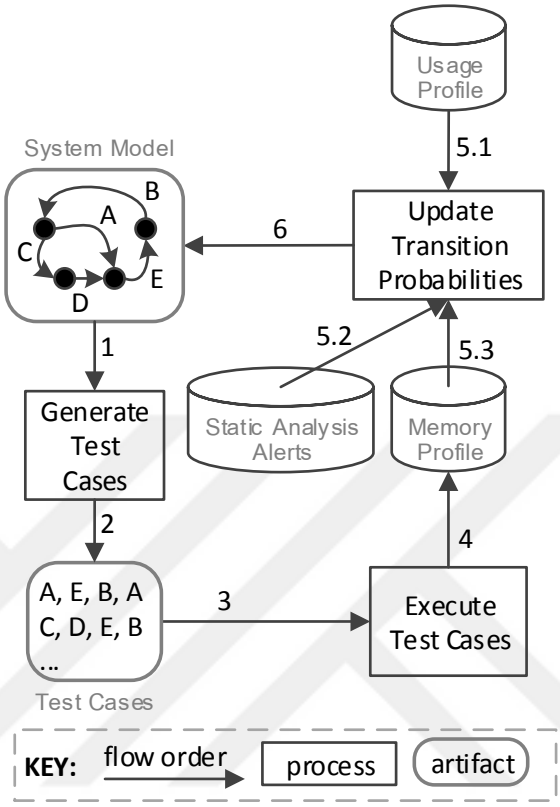


Figure 11: The overall approach.

4.1.1 Model Refinement based on Usage Profile

Initially, we assume that all the probability values for outgoing transitions of a state are equal to each other, adding up to 1. Hence, if there are n transitions going out from a state, the probability value for each of these transitions is $1/n$. In the first update cycle, we utilize a previously collected usage profile (See Section 4.2). Since, the test model represents the usage behavior, we can calculate the number of visits made to each state of this model. Then, RIMA updates the probability values to each outgoing transition based on the ratio of visits made to its target state. Assuming that there are n transitions going out from a state to n different states, we calculate the number of visits made to each of these states i, as v_i. Then, the probability value

assigned for each transition targeting at state i is $v_i / \sum_{i=0}^{n-1} v_i$.

4.1.2 Model Refinement based on Static Analysis

In the second update cycle, we utilize alerts generated by a static code analysis tool working on the system source code. We map source code modules to different system states in the model. Then, for each state, we calculate the ratio of alerts that are associated with this state each of which represents a TV feature implemented by these modules. We consider this ratio as a relative risk of fault, which can lead to an error if the corresponding alerts are not false positives and if the identified potential faults are triggered during the visit to the state. RIMA updates the test model according to this calculated risk as follows. It introduces a new state to the model, namely an *error state*, E. Then it introduces transitions from each of the existing system states to E. If there were previously n outgoing transitions from a state s, the number of outgoing transitions becomes $n + 1$. It assigns the probability value, p to this new transition according to the risk for s, calculated as the ratio of alerts associated with s to the total number of alerts. For each of the other outgoing transitions from s, the probability value is multiplied by $(1 - p)$. Figure 12 illustrates such an update with a simple example. On the left hand side of the figure, we see a part of the model before the update. Hereby, we see a source state s and 3 target states t_0, t_1 and t_2, with transition probabilities 0.2, 0.3 and 0.5, respectively. On the right hand side of the figure, we see the updated model, where a new state E is introduced. The probability value for the transition targeting at E is calculated as 0.2. Hence, all the other transitions are multiplied by 0.8. This update guarantees that the probability values for all the outgoing transitions add up to 1.

4.1.3 Model Refinement based on Dynamic Analysis

In the third update cycle, we utilize a memory profile to calculate another risk [85]. This profile is collected during the previous test cycles. We use a previously developed

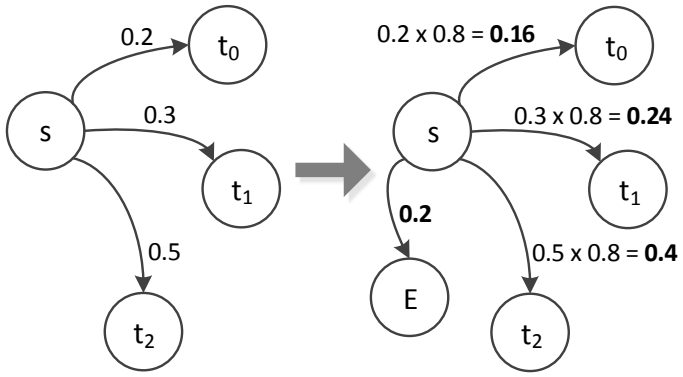


Figure 12: The updated transition probabilities for outgoing transitions of state s after introducing a risk of error with probability 0.2.

tool [85], which measures memory utilization before and after the usage of each feature in the system. The difference between the two measurements reveals memory leakage. Features are mapped to the states of the model and a memory error probability is calculated for each state, which is proportional to the amount of memory leak caused by each feature. Then, the model is updated just like the previous update cycle by RIMA tool. An additional *error state* is introduced. Transitions from all the states to this state are annotated with the calculated error probabilities. Error probability for each state s, is calculated as the ratio of memory leak associated with s to the total amount of memory leak. Finally, all the other transitions are updated to keep probability values for outgoing transitions sum up to 1.

In the following section, we introduce two case studies and illustrate each step of the approach.

4.2 Industrial Case Studies

In this section, we introduce two case studies from the consumer electronics domain. In particular, we illustrate the applications of our approach for MBT of DTV systems and Smart Phone (SP) which are developed by Vestel.

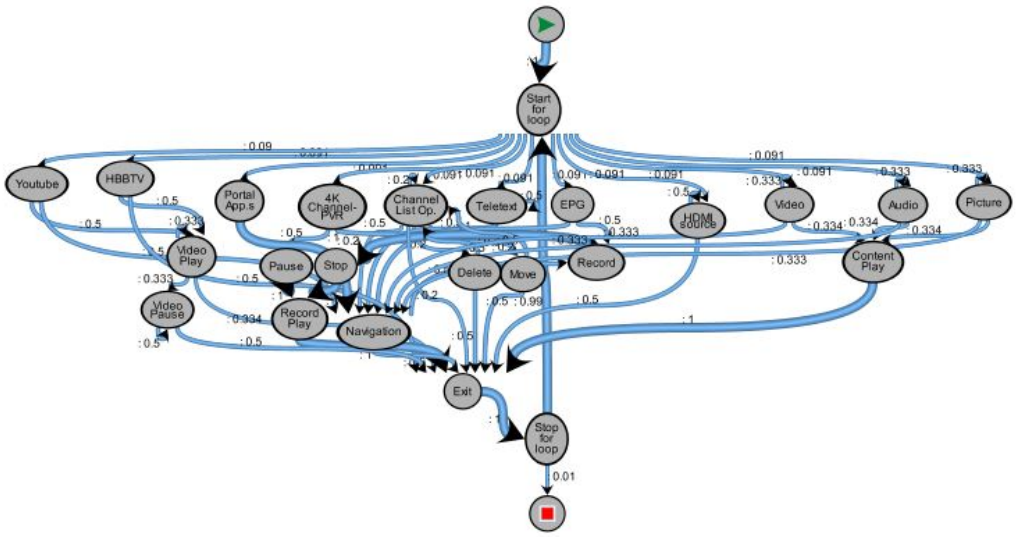


Figure 13: An initial test model used for MBT with default probabilities (DTV).

Figure 13 and Figure 14 depict the models that are used in our case studies. These models were previously developed by the software test group in the company. The models are defined in the forms of a Markov chain with the MaTeLo tool. They constitute a hierarchical structure, in which states can further comprise sub-models.

In Figure 13, we see the top level model, where states mainly represent various features of the system. The list of main features for DTV include Electronic Program Guide (EPG), MB Video, MB Audio, MB Picture, Portal, Youtube, HBBTV, Personel Video Recorder (PVR), Source Switch (HDMI-SCART), Teletext, and Channel List.

In Figure 14, there are states which represent the main features in the form of applications deployed on a SP. In our case study, we focused on those applications that are developed by Vestel only. The list of main features for SP include VCloud, VCam, Photos, Alarm, VMarket, Contacts, Phone Call, Themes, and SMS-MMS.

We used two of the test case generation algorithms that are provided by the MaTeLo tool. We employed the so called Minimum Arc Coverage algorithm [61]

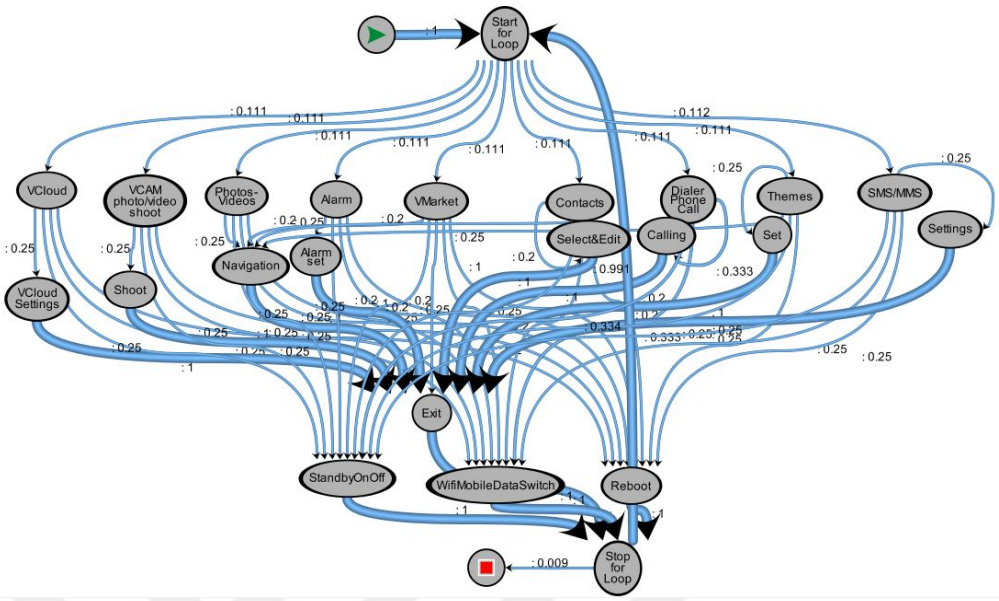


Figure 14: An initial test model used for MBT with default probabilities (SP).

with default parameters before model refinements (*Iteration 0*). This algorithm terminates when all the transitions in the model are visited at least once. After each model refinement step, we employed the so called User Oriented algorithm [61]. This algorithm has two parameters: i) the maximum number of test steps within a test case, and ii) the maximum number of test cases. We set the maximum number of test steps to 5000. We set the maximum number of test cases to 1. As a result, this algorithm starts from the start state and traverses the model by choosing a transition at each state non-deterministically based on transition probabilities. The algorithm terminates when the finish state is reached.

For DTV case study; in total 847 test steps were generated from the initial model before refinements. Execution of these test steps took 4 hours. 7 faults were detected during test execution. 5 of these faults were previously found during manual test activities. Therefore, 2 new faults were identified with MBT.

For SP case study; in total 1416 test steps were generated from the initial model before refinements. Execution of these test steps took 6.5 hours. 3 faults were detected during test execution. 2 of these faults were previously found during manual test

activities. Therefore, 1 new fault was identified with MBT.

In the following subsections, we describe research questions and the experimental setup.

4.2.1 Research Questions

The time is extremely limited compared to the amount of functionality to be tested in the consumer electronics domain. Test models that we use in our case studies have hundreds of states and transitions. Our goal is to update transition probabilities of test models based on usage profile analysis, static code analysis and dynamic analysis results respectively to be able to generate more effective test steps for detecting more faults. As such, we defined the following 3 research questions:

> ***RQ1:*** Is it possible to update transition probabilities in the test models based on usage profile to detect new faults that are more likely to be exposed to the users without compromising from their effectiveness?
>
> ***RQ2:*** Is it possible to update transition probabilities in the test models based on static code analysis to detect new faults by decreasing the total test steps and total time?
>
> ***RQ3:*** Is it possible to update transition probabilities in the test models based on dynamic analysis to detect new faults by decreasing the total test steps and total time?

The generated test cases from test models have to focus only on execution paths that are liable to highly severe failures that can be directly observed by users. Our first goal was cover different parts of a test model to find the faults which are most likely to be seen by the end-users and are not seen with the previous approaches before. User perceived failure severity is important, which is defined as the level of irritation experienced by the user caused by a product failure. Therefore, the testing

process must be optimized to detect faults that lead to user-perceived failures. We defined *RQ1* based on this observation.

The second goal was to refine the test model to find more faults with fewer test steps by considering warnings of modules in the source code with risk analysis performed by a static analysis tool. Therefore, we defined *RQ2* to evaluate static code analysis impact on MBT efficiency.

RQ3 is defined to see the dynamic analysis impact on MBT efficiency by increasing the number of found new faults while reducing the test steps and total time for test execution. We observed that there existed memory-related faults that could not be detected by the generated test cases.

In the following, we explain the experimental setup we used for evaluation. Then, we present and discuss the obtained results for addressing the research questions. We also illustrate steps of our approach using the case studies as a running example.

4.2.2 Experimental Setup

Experimental Setup for DTV

- ***Model Refinement based on Usage Profile***

The first model update is performed based on usage profile. To collect this profile, 30 products were sent to 30 field testers. As soon as they connect their TVs to the Internet, log files are created and sent to record which modules are visited by users. After 30 days, we collected the log files and analyzed them. Then RIMA calculated the probability values for each module as listed in the first part of Table 8. After updating the transition probabilities, 809 test steps were generated. Execution of these test steps took 4 hours. 9 faults were detected during test execution. 7 of them were previously found. Therefore, 2 new faults were identified after the first iteration.

- ***Model Refinement based on Static Analysis***

Software Module	Iteration 1		Iteration 2		Iteration 3	
	# of Visits	Calculated Prob.	# of Warnings	Calculated Prob.	Memory leak (MB)	Calculated Prob.
Portal	1900	0.146	18	0.322	40.855	0.218
Youtube	2250	0.173	18	0.322	89.380	0.477
HBBTV	500	0.038	6	0.108	8.846	0.047
MBR Video	1750	0.134	2	0.036	22.375	0.119
MBR Audio	400	0.03	1	0.017	4.167	0.022
MBR Picture	100	0.007	1	0.017	3.980	0.021
PVR	1000	0.076	3	0.054	9.351	0.05
Channel List	1750	0.134	3	0.054	2.516	0.013
EPG	2000	0.153	2	0.036	3.094	0.017
Teletext	1250	0.096	1	0.017	1.675	0.009
HDMI-SCART	100	0.007	1	0.017	1.002	0.005

Table 8: Collected data and the corresponding probability values used for updating the test model in successive iterations (DTV).

In the second iteration, we utilized static code analysis alerts. The second part of Table 8 lists the number of alerts reported for each module and the calculated probability values accordingly. RIMA updated the model as depicted in Figure 15. 136 test steps were generated based on this model. Execution of these test steps took 1.5 hours. 3 faults were detected during test execution. 2 of them were previously found. Therefore, one new fault was identified after the second iteration.

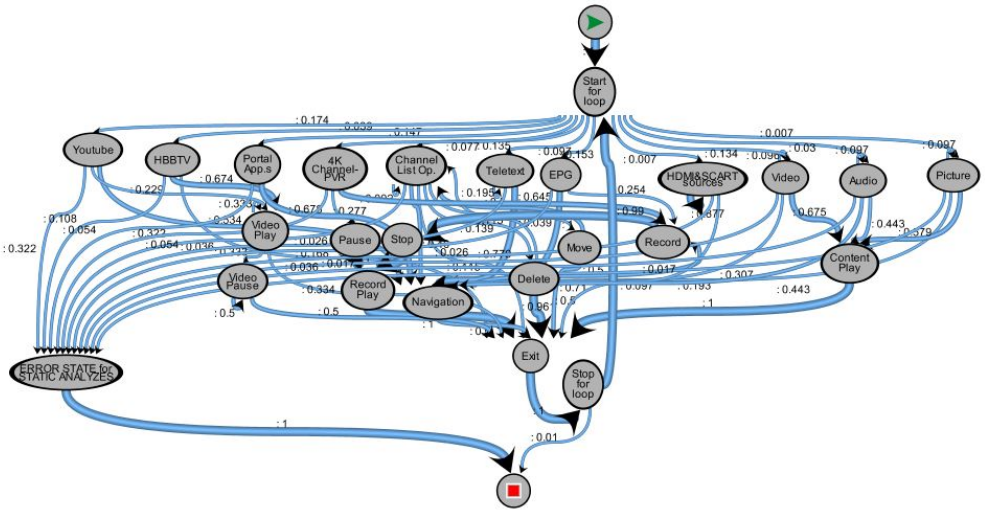


Figure 15: The test model updated based on static analysis after the second iteration (DTV).

- ***Refinement based on Dynamic Analysis***

In the third iteration, we utilized memory profiles that are collected in the previous iterations. The third part of Table 8 lists the amount of memory leak associated with each module and the calculated probability values accordingly. The updated model can be seen in Figure 16. 117 test steps were generated based on this model. Execution of these test steps took 1.5 hours. 3 faults were detected during test execution. 1 of them were previously found. Therefore, 2 new faults were identified after the third iteration.

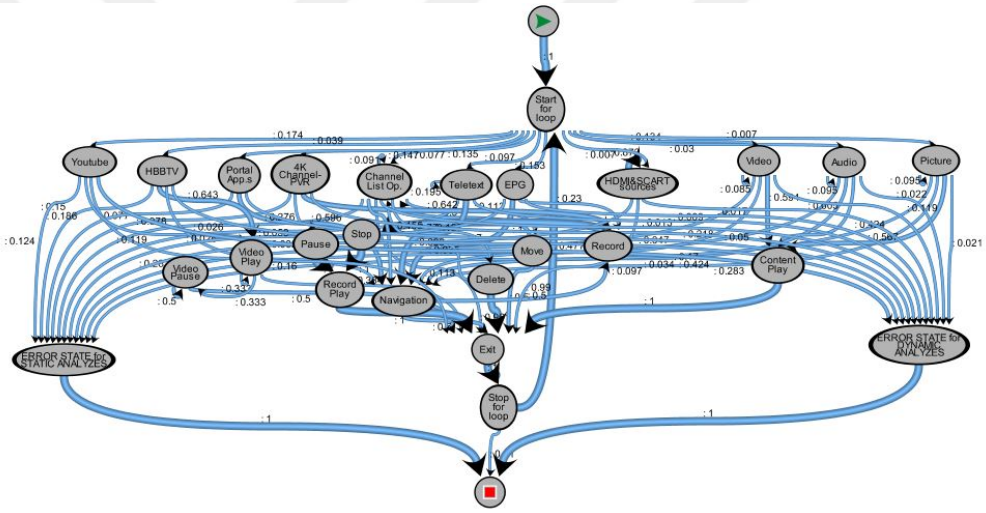


Figure 16: The test model updated based on memory profile after the third iteration (DTV).

Experimental Setup for SP

- ***Model Refinement based on Usage Profile***

The first model update is performed based on usage profile. To collect this profile, 50 products were sent to 50 field testers. There was an application which collected the usage counts for each application on the phone. After 30 days, we analyzed the usage of every users. Then RIMA calculated the probability values for each module as listed in the first part of Table 9. After updating the transition probabilities, 1205

test steps were generated. Execution of these test steps took 5.5 hours. 6 faults were detected during test execution. 3 of them were previously found. Therefore, 3 new major faults were identified after the first iteration.

Software	Iteration 1		Iteration 2		Iteration 3	
Module	# of Visits	Calculated Prob.	# of Warnings	Calculated Prob.	Memory leak (MB)	Calculated Prob.
VCloud	250	0.009	9	0.095	0.268	0.005
VCam	6000	0.222	11	0.116	1.9	0.038
Photos-Videos	4000	0.148	11	0.116	28.55	0.571
Alarm	3500	0.130	7	0.074	2.5	0.05
VMarket	1500	0.056	9	0.095	7.38	0.148
Contacts	2750	0.102	8	0.084	0.601	0.012
Dialer Phone Call	5000	0.185	16	0.168	1.2	0.024
Themes	1000	0.111	7	0.074	6.88	0.016
SMS-MMS	3000	0.037	17	0.179	0.811	0.136

Table 9: Collected data and the corresponding probability values used for updating the test model in successive iterations (SP).

- ***Model Refinement based on Static Analysis***

In the second iteration, we utilized static code analysis alerts. The second part of Table 9 lists the number of alerts reported for each module and the calculated probability values accordingly. RIMA updated the model as depicted in Figure 17. 1014 test steps were generated based on this model. Execution of these test steps took 5 hours. 4 faults were detected during test execution. 3 of them was previously found. Therefore, one new major fault was identified after the second iteration.

- ***Refinement based on Dynamic Analysis***

In the third iteration, we utilized memory profiles that are collected in the previous iterations. The third part of Table 9 lists the amount of memory leak associated with each module and the calculated probability values accordingly. The updated model can be seen in Figure 18. 857 test steps were generated based on this model. Execution of these test steps took 4 hours. 3 faults were detected during test execution. 2 of

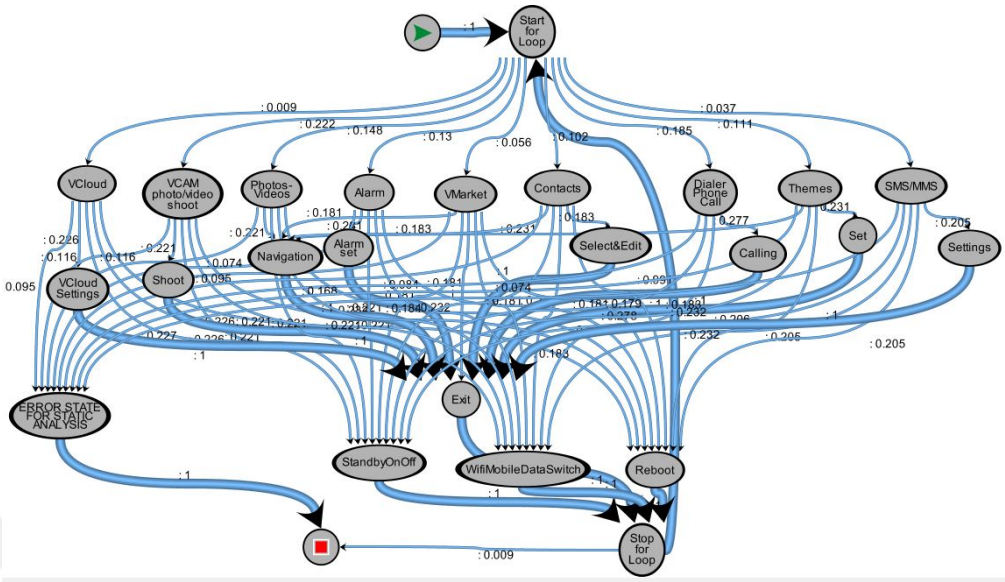


Figure 17: The test model updated based on static analysis after the second iteration (SP).

them were previously found. Therefore, 1 new minor fault was identified after the third iteration.

4.2.3 Results and Discussion

Iteration #	# of Test Steps	Test Execution Time (hr)	# of Faults Detected	# of New Faults Detected
0	847	4	7	2
1	809	4	9	2
2	136	1.5	3	1
3	117	1.5	3	2

Table 10: Test results after each iteration (DTV).

The overall results for DTV case study are summarized in Table 10. We were able to detect 9 faults with the original model, out of which only 2 were actually detected with MBT for the first time. In successive iterations, we were able to detect 5 more new faults. All these faults were critical. They mainly lead to stability issues in the platform such as crashes and lack of response for remote controller commands.

The overall results for SP case study are summarized in Table 11. We were able to

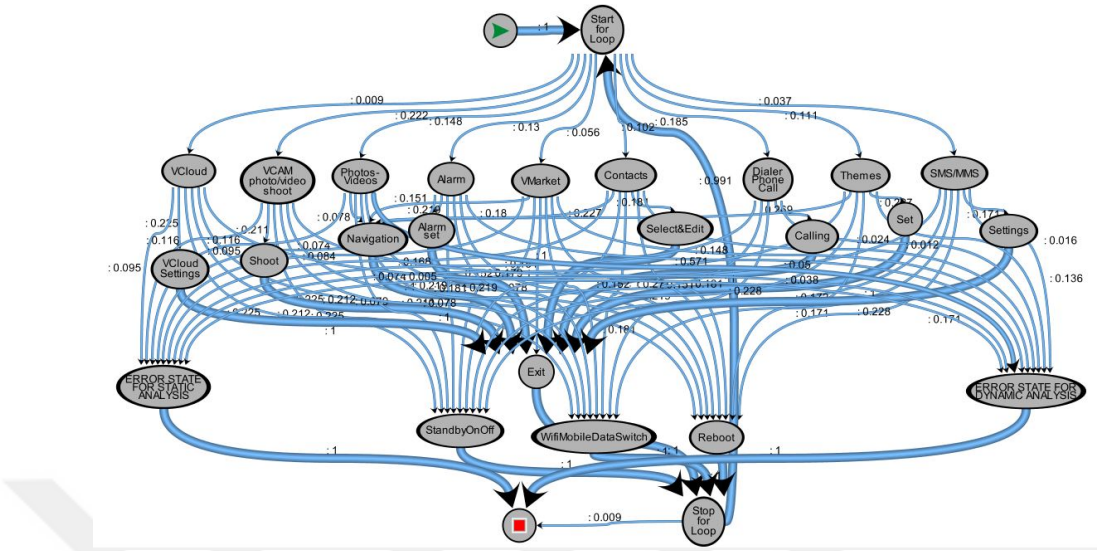


Figure 18: The test model updated based on memory profile after the third iteration (SP).

Iteration #	# of Test Steps	Test Execution Time (hr)	# of Faults Detected	# of New Faults Detected
0	1416	6.5	3	1
1	1205	5.5	6	3
2	1014	5	4	1
3	857	4	3	1

Table 11: Test results after each iteration (SP).

detect 3 faults with the original model, out of which only 1 was actually detected with MBT for the first time. In successive iterations, we were able to detect 4 more major new faults and 1 minor new fault. These faults are mostly related with functionality breakdowns problems such as missing contacts in contact list, dis-functionality of stop call button after some time, wrong timing for the alarm function.

In our approach, RIMA introduces two new *error states* to the test model throughout the refinement iterations. In principle, different types of *error states* can be added each of which represents a different type of error. The probability (risk) of causing each of these types of errors can be calculated for system states with static and/or dynamic analysis.

Another alternative approach would be to combine all the analysis results and

perform an update once, instead of performing updates in several iterations. The advantage of this approach would be the decreased completion time of tests. However, as a drawback, one can be able to detect less number of faults. This is because, test steps generated in each iteration mainly focuses on different paths on the model although there can be occasional overlaps. Systematic avoidance of these overlaps can help to reduce overall test duration.

For DTV case, the total test execution time, including all the refinement steps took 11 hours. This duration can be reduced by skipping *Iteration 0*, which does not contribute much for detecting user-perceived failures. We can observe that the durations for the second and third iterations are reduced dramatically (more than 50%). The total time spent for successive test executions is 3 hours. Yet, 3 new faults were detected. Hereby, we should note that the number of test steps is not directly proportional to test duration. The execution of some of the test steps can not be fully automated and as such these steps take more time than others to execute. In these steps, a test engineer has to manually control a set of properties regarding audio and video. Therefore, elimination of such steps can further decrease the overall test duration.

4.2.4 Threats to Validity and Limitations

We assume that the first versions of the test models exist in the form of a Markov chain. States are determined with the help of software engineers in the company.

One of the limitations is regarding the manual effort that has to be invested to identify risks. For instance, in our first case study, 2 domain experts analyzed the source code of a DTV for 1 week to detect resource (memory and processing time) consuming modules. After detecting such 11 modules, 1 test engineer completed test modeling with MaTeLo in 2 days. Similarly, 1 domain expert analyzed the source code of SP for 3 days to detect resource consuming modules. 1 test engineer completed

test modeling of 9 modules with test scripting in the model in 2 days.

Antoher limitation is the dependency on tools for static code analyzes and collection of memory profiles. We assume that these tools provide reliable feedback.

We performed two case studies focusing on two products, DTV systems and SPs. In principle, our approach can be also applied for other consumer electronics products such as refrigerators, washing machines, dishwashers, air conditioners, and cookers.

4.3 Related Work and Our Contributions

MBT techniques are extensively studied in the literature [62]. There exist tools applied in practice [44] and various formalisms for model specification like finite state machines [63] [64] and Markov chains [18]. There also exist case studies [44, 65] for evaluating the effectiveness of MBT. In this paper, we evaluate the effectiveness of MBT in consumer electronics domain. In particular, we evaluate the effectiveness of a successively refined model.

Recent studies on MBT [86, 87] take the timing behavior and concurrency into account. Petri nets are used for specifying test models [87] and test oracles are automatically generated to enforce timing requirements [86]. The generated test cases reflect the timed and concurrent nature of inputs for the system. In our approach, we did not take the timing behavior and concurrency into account. However, we make the failure behavior explicit in test models.

There exist recent studies [69, 74, 88] on refinement of test models for MBT. These studies aim at extending existing models by adding missing states and transitions. Missing model elements are discovered by analyzing run-time states and events collected from the executing system. In our approach, we assume that the model is complete; however, we update transition probabilities to focus the generated test cases on different parts of the model. These updates are performed not only based on dynamic analysis, but also usage profile and static code analysis.

This approach is also inspired from the principles of RBT [22]. However, we complement dynamic analysis with usage profile and static analysis for estimating risks.

CHAPTER V

MODEL-BASED SOFTWARE PRODUCT LINE TESTING

In this chapter, we introduce an approach and a toolset, FORMAT that is integrated this tool with a set of existing tools for adopting SPLE together with MBT.

A *reference test model* is developed for capturing all possible usage scenarios for a family of systems rather than a single system. This model is represented in the form of hierarchical Markov chains. Variations among tested systems are documented in the form of a feature model. A separate specification maps optional and alternative features in the feature model to a set of states in the test model. Transition probabilities for these states are modified according to (selected) system features such that test cases focus only on these features. This approach facilitates the reuse of a test model for many systems.

We developed a tool called FORMAT and integrated this tool with a set of existing tools to automate our SPLE approach for MBT. The whole process is automated except the initial (one-time) development of the feature model, mapping specification and the reference test model. Once this initial investment is made, the reference test model can be reused for different systems by just selecting the relevant features. Feature selections are performed on the feature tree via a graphical user interface. Then, transition probabilities are automatically updated on the test model.

We performed a controlled experiment in an industrial context to evaluate the effectiveness of our approach and tool. 10 participants were involved in testing 10 different DTV systems. A reference test model has to be adapted for each of these systems based on their features. The effort spent by each participant was measured both when FORMAT is used and when it is not. Results indicate that FORMAT

reduces the effort significantly. The use of FORMAT and the adoption of the SPLE approach requires an initial investment. However, results show that this investment pays off already if the reference test model is reused for 13 products.

In the following section, we explain our approach and tool. In section 5.2, we present the controlled experiment for evaluating our approach in an industrial setting.

5.1 Overall Approach

The overall SPLE approach for MBT [8] is depicted in Figure 19. 3 input models are necessary to apply the approach: *i)* a *reference test model* that captures the possible behavior for a family of products, *ii)* a *feature model* that documents the commonalities and variations among these products, and *iii)* a *mapping specification* that maps feature model elements to states of the test model.

5.1.1 Developing Feature Diagrams

The creation of feature diagrams requires one-time effort if features are not subject to change. The feature diagram is developed with the online SPLOT tool[1]. An example feature diagram of a product is shown in Figure 20(a). One can select or deselect optional features on a feature diagram to specify a particular product configuration. This is the first step of the approach and it is also performed via the user interface of the SPLOT tool. Figure 20(b) shows an example product configuration, where the first optional feature named *Connectivity* is not selected as part of the product. As a result, its sub-features including *Netflix* and *Wifi* are also deselected. Hence, the generated test cases are not supposed to cover these features.

[1]http://www.splot-research.org/

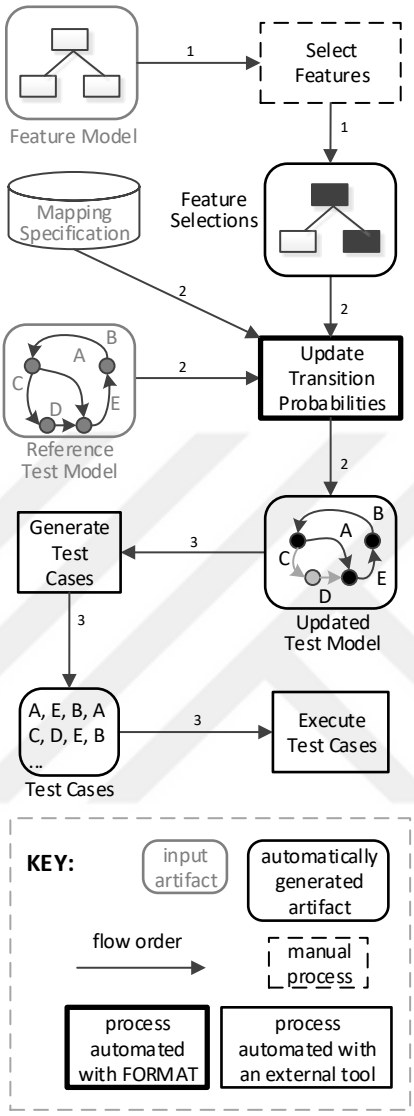
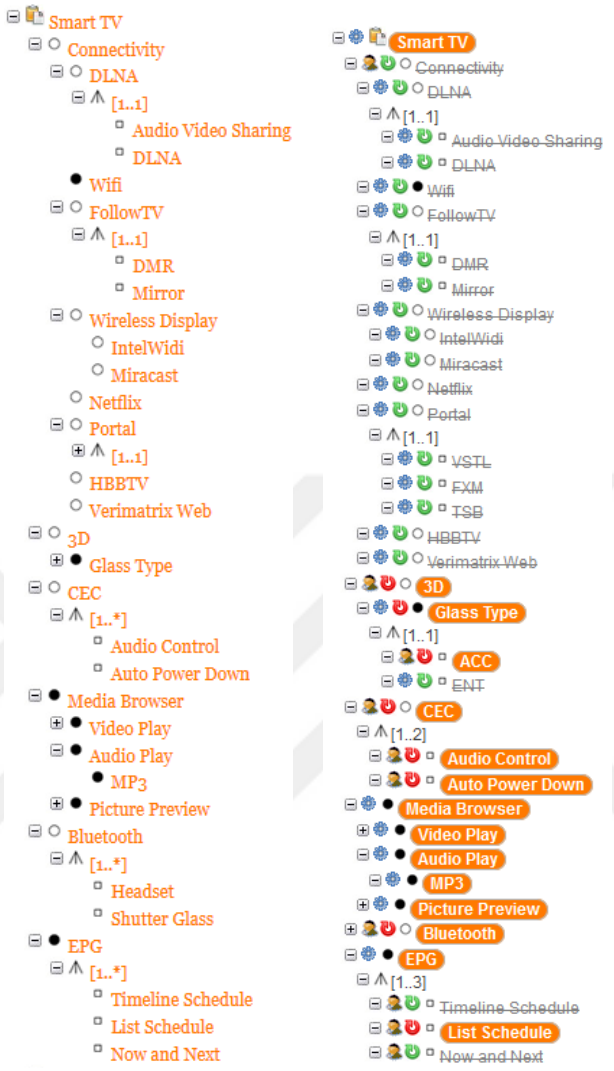


Figure 19: The overall approach.

SPLOT can export feature selections in XML format as shown in Listing 5.1.



(a) A partial feature diagram of a family of DTV products.

(b) A set of sample feature selections for one of the DTV products.

Figure 20: A snippet from a sample feature diagram for a DTV product family and a set of feature selections on this model for a sample product.

5.1.2 Updating Models

In the second step, selected feature configuration is used for updating the reference test model based on its mapping to the feature model. Mapping specification relates feature names to top-level states of the test models. The creation of this specification

requires one-time effort if feature model and test model elements are not subject to change. However, it can also be generated automatically in principle, if both models follow certain naming conventions.

Listing 5.1: A snippet from the specification exported by SPLOT.

```
<feature_model name="SmartTV"><meta>...</meta>
<feature_tree>
:r Smart TV(_r)
  :o Connectivity(_r_2)
    :o DLNA(_r_2_9)
      :g (_r_2_9_31) [1,1]
        : Audio Video Sharing(_r_2_9_31_32)
        : DLNA(_r_2_9_31_33)
    :m Wifi(_r_2_10)
    ...
    :o Wireless Display(_r_2_12)
      :o IntelWidi(_r_2_12_99)
      :o Miracast(_r_2_12_100)
    :o Netflix(_r_2_29)
  ...
  :m EPG(_r_7)
    :g (_r_7_24) [1,*]
      : Timeline Schedule(_r_7_24_25)
      : List Schedule(_r_7_24_26)
      : Now and Next(_r_7_24_27)
</feature_tree><constraints>
</constraints></feature_model>
```

Our tool, FORMAT[2] automates the second step of the approach, where the model is updated. Initially, the model includes default transition probability assigned by the MaTeLo tool. Algorithm 2 outlines the update procedure followed by the tool. Hereby, probability values for transitions to states that represent unselected features are set as 0 (Line 7). These values are summed up (p) for the source state (Line 6). Then, probability values of all the transitions originating from this state are multiplied by $1/(1-p)$ (Lines 10-14). As such, their relative priority remains the

[2]https://github.com/csgebizli/FORMATTool

same and the sum of probabilities of these transitions becomes equal to 1.

Algorithm 2 Model Update Procedure.

```
1: S ← set of states in the test model
2: for all s ∈ S do
3:     p ← 0
4:     for all t ∈ S|∃ transition (s, t) do
5:         if t maps to a deselected feature then
6:             p ← p + p(s, t)
7:             p(s, t) ← 0
8:         end if
9:     end for
10:    for all t ∈ S|∃ transition (s, t) do
11:        p(s, t) ← p(s, t) × 1/(1 − p)
12:    end for
13: end for
```

For instance, Figure 21 depicts a test model, where probability values for transitions targeting at *Netflix* and *Wifi* states are updated to be 0. These updates are consistent with the feature selections shown in Figure 20(b). The probabilities of all the other transitions outgoing from the *Settings* state add up to 1 after the update. We should note that Figure 21 shows the top-level model only. Many of the states in this model hierarchically include further sub-models.

In the third step, the updated test model is used for generating a set of test cases and executing them on the product. Test execution is automated with VesTA, whereas test case generation is automated with the MaTeLo tool. This tool-chain implements a variety of test case generation algorithms.

In our approach, we employed the so called "Most Probable" algorithm [61]. This algorithm traverses the model by selecting the most probable transition at each state and it terminates after a limited number of steps. As a result, *Netflix* and *Wifi* states in the example test model will never be visited for instance.

In the following section, we present a controlled experiment in an industrial setting to evaluate our approach and tool.

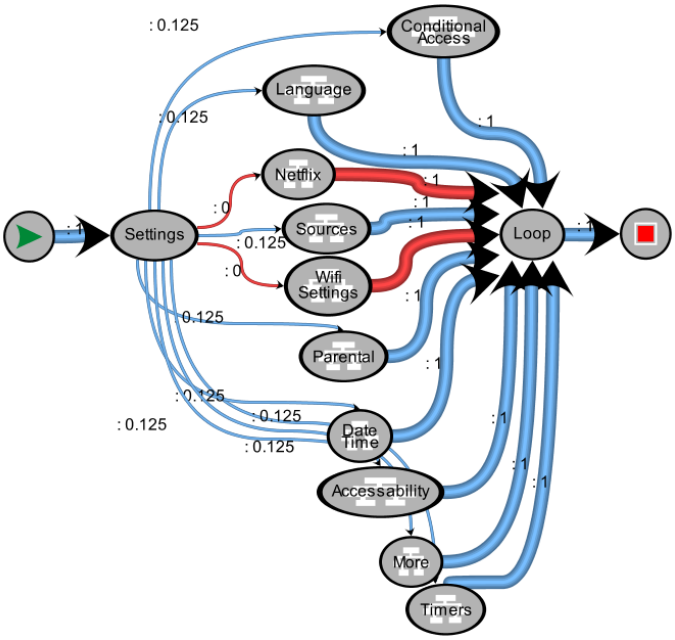


Figure 21: The updated test model.

5.2 Industrial Case Study

We conducted a controlled experiment in an industrial setting to evaluate our approach. Hereby, our goal is to measure the effort reduction provided by FORMAT. We also aim at evaluating the effort reduction with respect to the effort necessary to adopt our approach. In traditional approach, one must update a test model manually for each different product by considering its features. FORMAT performs these updates automatically by taking the corresponding feature selections as input. However, there is a need for initial investment to specify the feature diagram and a mapping of its elements to the states in the test model. Hence, we want to know to what extent FORMAT pays off this investment. First, we introduce our research questions in the following. Second, we describe the experimental setup. Third, we present and elaborate on the results. Finally, we discuss validity threats and limitations.

5.2.1 Research Questions

We defined the following research questions:

RQ1: How much effort is required for manually updating test models for each product?

RQ2: How much effort is required for using FORMAT to update test models?

RQ3: How much effort is required for developing a feature diagram and a mapping specification between a feature diagram and a test model?

The first research question, *RQ1* is defined for understanding the cost of the traditional method for updating test models. *RQ2* aims at understanding the cost of using FORMAT for the same purpose. *RQ3* is defined for understanding the cost of initial investment that has to be paid to be able to use FORMAT. We also would like to test the significance of difference in effort reduction when FORMAT is used. Hence, we formulated our null hypothesis as follows:

- H_o FORMAT does not have any effect on the amount of effort spent by test engineers/technicians to adapt test models for different products.

We choose 0.01 as the significance level for rejecting the hypothesis.

5.2.2 Experimental Setup

Hereby, we explain the experimental setup we used for addressing the research questions and testing our hypothesis.

Factors: **Tool support** is the only factor of our experiment, i.e., whether FORMAT is used for updating test models or not. Hence, this factor is measured in nominal scale, at two levels: with FORMAT, without FORMAT.

Independent Variables: There is one independent variable that we kept at fixed levels in the experiment. That is the **product family** used. Each participant was treated with the same set of **randomly** selected 10 products from a DTV product

family. Participants were provided with a reference test model that captures all possible usage scenarios for this family of systems. They were also provided with a feature diagram, mapping of features in this diagram to states of the test model, and the set of features included in each of the 10 products. Table 12 lists the number of selected and deselected features for the products. There exist 96 features in total. The number of deselected features is associated with the number of updates that has to be performed on the test models. Hence, these numbers are directly associated with the expected effort.

Table 12: The number of (de)selected features for 10 products that are used in the experiment.

Product ID	# of Selected Features	# of Deselected Features
P1	47	49
P2	43	53
P3	34	62
P4	24	72
P5	54	42
P6	53	43
P7	63	33
P8	44	52
P9	44	52
P10	44	52

Dependent Variables: **Effort** is the only dependent variable in the experiment. It is measured in ratio scale in terms of seconds.

Participant Selection: There is a dedicated software testing group in the company who is performing tests of various consumer electronics products. The testing group is composed of either test engineers who studied at a college/university or test

technicians who completed two-year educational programs. We selected **10 participants** for our experiment **randomly** from this group. Table 13 provides information regarding the participant profile. Participants are enumerated as *S1*, *S2*, ... *S10* in the first column. The second column shows their job title in the company. We see that 8 of the participants are test engineers and two of them are test technicians. The third column lists their testing experience in terms of the number of months.

Table 13: Participant information and provided training during experiment preparation.

Participant ID	Job Title	Testing Experience (months)	FORMAT Training (hours)	MaTeLo Training (hours)	SPLOT Training (hours)
S1	Test Engineer	60	1	1	0.5
S2	Test Engineer	12	1	1	0.5
S3	Test Engineer	36	1	1	0.5
S4	Test Technician	144	1	2	0.5
S5	Test Technician	180	1	2	0.5
S6	Test Engineer	12	1	1	0.5
S7	Test Engineer	24	1	1	0.5
S8	Test Engineer	6	1	4	0.5
S9	Test Engineer	6	1	4	0.5
S10	Test Engineer	6	1	4	0.5

Experiment Design: Our design has one factor and two treatments as shown in Table 14. All the participants were assigned to both of the treatments. First, they update the test model for each of the products manually. Then, they used FORMAT to perform the same task. They have to work directly on the test models for manual updates. On the other hand, they have to only work on the feature diagram when using FORMAT. Therefore, the order of applying the treatments does not affect the effort.

Table 14: The experiment design with one factor that has two levels. The same set of (all of the) participants are assigned to each of the two treatments.

	Factor: Tool Support	
	Level: with FORMAT	Level: without FORMAT
Experiment 1	10 participants	10 participants

Preparation: An experienced test engineer prepared the reference test model, the feature diagram and the mapping specification, which are provided to the participants before the experiment. The reference test model was already available and being used in the company and it was being manually adapted for different products. It took 5 hours to create the feature diagram for the product family. It took 3 hours to create the mapping specification that maps features to states of the test model.

We also provided training regarding 3 tools to all the participants. These tools are FORMAT, MaTeLo and SPLOT. We can see the training durations for these tools in the third, fourth and fifth columns of Table 13, respectively.

FORMAT takes 3 inputs from the user via its user interface. The first one is the selected product configuration, which is exported by SPLOT. The second one is the reference test model. The third input is the mapping specification. The user should provide the full path of the directories, where each of these 3 input files are located. Then the model update can be performed by pressing a single button. The tool first checks the consistency of the feature list and the states of the test model based on the mapping specification. Then, it automatically updates the relevant transition probabilities on the test model. The updated model can be saved as a separate model dedicated for a particular product. We explained the user interface and these steps to all the participants.

MaTeLo is the tool that is being used in the company for test case generation. Test models can be created and edited with this tool. Models are specified in the form of Markov chain formalism.

The training for MaTeLo takes the longest time (4 hours at maximum) since the tool has a complicated user interface. Moreover, participants should also be knowledgeable on MBT and the Markov chain formalism. In fact, some of the participants have already had knowledge and experience regarding the usage of MaTeLo.

However, some of them used the tool for the first time. Therefore, durations vary a lot for the training regarding this tool.

We provided a training on SPLOT so that participants can open the provided feature diagram with this tool, they can select or deselect features on the diagram, and finally export these selections as a feature configuration of a particular product. We provided participants with the relevant documentation and performed an exercise that involves creating a product configuration based on a given feature diagram.

Execution: During the experiment, each participant was provided with the test reference model, feature diagram and mapping specification. They were also provided with the feature list regarding the 10 products listed in Table 12. Then two tasks were assigned one after the other.

The first task involved using MaTeLo for updating the reference test model for each of the 10 products based on their features. Hereby, they had to locate states in the model that are associated with the deselected features. Then, they had to update the probability values for transitions targeting at these states with 0 values. MaTeLo automatically updates the probability values for the remaining transitions to keep the model consistent.

The second task involved using SPLOT to deselect features on the feature diagram for each of the 10 products and supplying the resulting diagrams to FORMAT, which automatically updates the reference test model. This task is basically the implementation of the process depicted in Figure 19. We measured the effort for 3 activities during the completion of the tasks:

F: Effort required for (de)selecting features of a product on a feature diagram with SPLOT.

M: Effort required for updating the test model manually with MaTeLo.

T: Effort required for updating the test model with FORMAT by supplying the feature configuration created with SPLOT.

The effort required for the first task (without FORMAT) is basically equal to M. The effort required for the second task (with FORMAT) is equal to $F + T$. In the following, we present and discuss the results.

5.2.3 Results and Discussion

The overall results can be seen in Table 15. Hereby, F, M, and T values are listed for each participant (*S1* through *S10*) and for each product (*P1* through *P10*). For instance, the first participant (*S1*) spent 2977 seconds for (de)selecting features to define 10 products in total. The time required for the same participant to update test models for all the products manually is 18240 seconds. The feature configuration of the seventh product (*P7*) took the longest time (420 seconds). On the other hand, updating the test model required the longest time (1950 seconds) for *P9*.

We can see from Table 15 that T values are negligible compared to F and M values. As such, the effort required for using FORMAT is dominated by the effort necessary for performing feature configurations. Obviously, effort varies for different products and different subjects. However, the amount of variation is not high. This can be observed with box plots of F values and M values in Figures 22 and 23, respectively. Hereby, the x-axis lists the 10 participants involved in the experiment.

We take the average effort required per product to compare the effort spent by the participants. The average of M values for a participant indicates the mean effort per product without FORMAT. On the other hand, The average of $(F + T)$ values for

Table 15: Participant ID (**S#**) Effort for manually updating test models (**M**) performing feature selections on a feature model (**F**) and effort for automatically updating test models(**T**). All measures are provided with the units of seconds.

	S1			S2			S3			S4			S5		
Product	F	M	T	F	M	T	F	M	T	F	M	T	F	M	T
P1	290	1800	10	390	2810	12	235	2160	11	307	3340	10	435	2750	11
P2	310	1750	7	270	2770	13	376	1850	11	350	3150	11	370	2360	1211
P3	270	1780	9	310	2120	11	275	2890	9	410	2780	9	398	2270	11
P4	298	1900	8	370	2340	12	337	2160	8	315	2450	8	417	2900	10
P5	268	1860	8	280	1860	12	417	3550	13	320	2850	8	328	2550	8
P6	223	1650	9	307	2650	13	265	2160	12	415	2430	9	385	2470	9
P7	420	1800	9	310	1970	11	287	1850	11	385	2130	9	390	1850	11
P8	300	1860	10	340	2660	9	374	2890	10	370	2660	10	428	2290	10
P9	350	1950	6	290	2950	12	457	2160	11	420	2950	11	360	2580	12
P10	248	1890	10	405	2330	13	378	2550	9	335	2330	10	417	2860	11
Total	2977	18240	86	3272	24460	118	3401	24220	105	3627	27070	95	3928	24880	105

	S6			S7			S8			S9			S10		
Product	F	M	T	F	M	T	F	M	T	F	M	T	F	M	T
P1	369	3180	14	280	1920	9	420	2890	10	510	3480	15	390	3160	10
P2	357	3270	11	310	2150	11	380	2730	11	505	3570	13	420	3340	11
P3	448	3150	12	295	2280	10	390	2290	9	410	3290	17	380	2730	13
P4	435	2670	9	325	1840	10	320	2370	12	460	3470	14	375	3370	8
P5	425	3120	13	265	1970	8	425	2670	11	380	3680	15	460	2950	10
P6	375	3480	12	320	2050	9	410	2830	15	420	3550	11	525	2760	9
P7	427	3450	11	285	1930	8	380	2250	13	445	2950	16	420	2950	9
P8	268	2370	10	290	1980	10	370	3170	10	460	3280	17	395	2970	8
P9	345	2870	13	385	2250	8	440	2940	11	390	3190	16	450	3180	11
P10	463	2920	10	340	1950	10	360	3180	13	430	3360	13	490	2980	10
Total	3912	30480	115	3095	20320	93	3895	27320	115	4410	33820	147	4305	30390	99

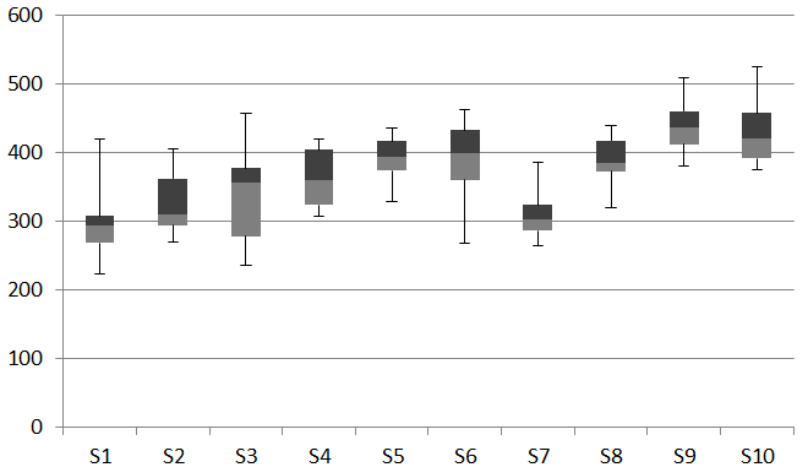


Figure 22: Variation of F values for the 10 products.

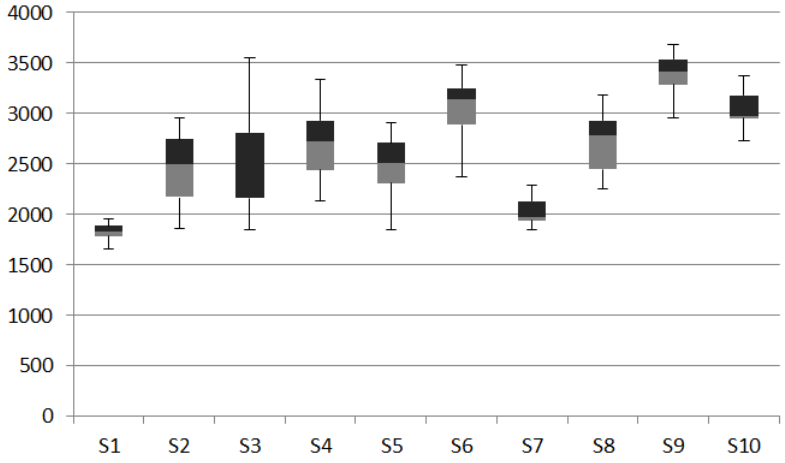


Figure 23: Variation of M values for the 10 products.

a participant indicates the mean effort per product with FORMAT. These values are listed in Table 16. We can see that the effort is significantly reduced when FORMAT is used. We performed a t-test to validate this observation. In particular, we performed a paired t-test since the same group of participants was assigned to both of the treatments. Results[3] are listed in Table 18. We can see that *t Stat* is much smaller than *(- t Critical)* values. We can also see that P(T¡=t) one-tail and P(T¡=t) two-tail values are very low. They are well below the threshold (0.01), which means that the

[3]We used Microsoft Excel (2010) to obtain the results.

Table 16: Average effort per product with and without FORMAT.

Participant ID	with FORMAT	without FORMAT
S1	306.3	1824
S2	339	2446
S3	350.9	2422
S4	372.2	2707
S5	403.3	2488
S6	402.7	3048
S7	318.8	2032
S8	411	2732
S9	455.7	3382
S10	440.4	3019
Average:	380.03	2610

difference is significant. That also means that the null hypothesis H_o can be rejected and the effort reduction achieved with FORMAT can be confirmed.

We also performed Wilcoxon test [89, 90]. Like the paired t-test, the Wilcoxon test also includes comparisons of differences between measurements. However, it is a non-parametric test that can be used when the data are not normally distributed.

Results[4] are listed in Table 17. We obtained the value of Wilcoxon's test statistic as -55. Critical value is set as 8 according to critical values table which is based on the number of subjects. We compared these two values and saw that our obtained value is statistically significant since it is much smaller than the critical value.

Therefore, the null hypothesis H_o can be rejected and again the effort reduction achieved with FORMAT can be confirmed.

The variation of effort can be observed in the box-plot depicted in Figure 24. The plot shows that the variation of effort for a particular treatment is very low; however, there is a significant difference when we compare the required effort for the

[4]We used Microsoft Excel (2010) to calculate the results.

Table 17: Wilcoxon test results. (**S#**); Participant ID , (**M**); effort for manually updating test models, (**F**); for performing feature selections on a feature model and (**T**); effort for automatically updating test models.

Participant ID	F+T	M	Difference	Positive	$\|Difference\|$	Rank	Signed Rank
S1	306.3	1824	-1517.7	-1	1517.7	1	-1
S2	339	2446	-2107	-1	2107	5	5
S3	350.9	2422	-2071.1	-1	2071.1	3	-3
S4	372.2	2707	-2334.8	-1	2334.8	7	-7
S5	403.3	2488	-2084.7	-1	2084.7	4	-4
S6	402.7	3048	-2645.3	-1	2645.3	9	-9
S7	318.8	2032	-1713.2	-1	1713.2	2	-2
S8	411	2732	-2321	-1	2321	6	-6
S9	455.7	3382	-2926.3	-1	2926.3	10	-10
S10	440.4	3019	-2578.6	-1	2578.6	8	-8
						Positive Sum:	0
						Negative Sum:	-55
						Test Statistic:	-55

two treatments.

In the following, we provide answers for the research questions based on the obtained results. We also evaluate the trade-off between the effort reduction achieved with FORMAT and the initial investment required for using FORMAT.

Effort for manually updating test models for each product We observed that this effort might vary depending on the product and the participant. The boxplot in Figure 23 depicts the distribution of this effort for the 10 participants. The minimum effort recorded is the effort spent by *S1* on *P6*, which is 1650 seconds. The maximum effort recorded is the effort spent by *S9* on *P5*, which is 3680 seconds. The average effort spent by each participant per product is listed in the third column of Table 16.

Effort for using FORMAT to update test models The variation in effort is not very high concerning this activity. The average effort spent by each participant

Table 18: t-Test: paired two-samples for means.

	Variable (1)	Variable (2)
Mean	380.0300000	2610
Variance	2573.7334444	223156.2222
Observations	10.0000000	10
Pearson Correlation	0.9215381	
Hypothesized Mean Difference	0.0000000	
df	9.0000000	
t Stat	-16.5496667	
P(T $\leq$ t) one-tail	0.0000000239	
t Critical one-tail	1.8331129	
P(T $\leq$ t) two-tail	0.0000000479	
t Critical two-tail	2.2621572	

per product is listed in the second column of Table 16. This effort is dominated by the effort required for defining product configurations on the feature diagram (i.e., F values). The box-plot in Figure 22 depicts the distribution of this effort for the 10 participants. The minimum effort recorded is the effort spent by *S1* on *P6*, which is 223 seconds. The maximum effort recorded is the effort spent by *S10* on *P6*, which is 525 seconds.

Effort for developing a feature diagram and a mapping specification between a feature diagram and a test model The feature diagram and mapping specification were prepared before the experiment. Participants were not involved in this activity since the models have to be defined only once and they can be reused for all the products of the product family. It took 5 hours to create the feature diagram. This activity is performed by an experienced engineer based on functional requirement specifications. The feature diagram is not modified as long as the set of possible features in the product family do not change. It took 3 hours to map the features in

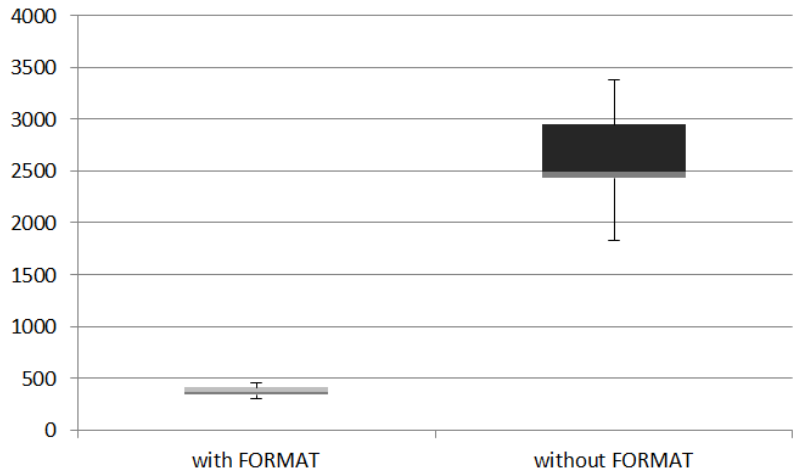


Figure 24: Variation of effort with and without FORMAT.

the feature model to a set of states in the test model. This activity is also performed by the same engineer, who developed the feature diagram. The created specification remains the same as long as the set of features and the system usage behavior (as such, the test model) do not change.

In total, the initial investment, which involves the development of the feature diagram and the mapping specification, required 8 hours (480 minutes) in total. On the other hand, the effort required for the adaptation of a test model is decreased from 2610 seconds to 380.03 seconds on average (See Table 16). That leads to 2229,97 seconds gain per product, which is equal to 37 minutes. Since, $480/37 = 12.915 < 13$, the initial investment can be amortized after the creation of test models for 13 different products. The total number of products that can be defined based on the feature diagram is 90. Hence, the investment can be paid off if it is utilized for 15% of the product family. Based on these results, we conclude that the SPLE approach and FORMAT lead to significant reduction of effort for developing/adapting test models for MBT.

5.2.4 Threats to Validity and Limitations

Our evaluation is subject to external validity threats [91] since it is based on a single experiment. Especially the set of participants has been selected randomly from the test group within a company. The experiment can be replicated in various contexts to be able to generalize the results. Internal validity threats are mitigated by using real systems and involving real engineers/technicians from the industry. Conclusion and construct validity threats are mitigated by observing the activities of participants without interfering with them. We also performed statistical tests to evaluate the significance of the results.

We should note that our conclusions are based on the assumption that the initially developed feature diagram and the mapping specification can be reused without being subject to frequent changes.

5.3 Related Work and Our Contributions

The use of SPLE approach for MBT has been studied before [59]. Existing studies utilize test models that define possible use cases for a family of products. Then, product-specific test models (and test cases) can be obtained based on this specification. From this perspective, our approach and existing studies are alike. However, each study uses different formalisms and methods. For instance, ScenTED [92] and CADeT [93] use UML activity diagrams for specifying use case descriptions. UML sequence diagrams are used for defining test cases. Hereby, variability information is embedded in the UML models and binding variants for specific applications requires manual effort. We use Markov chain formalism to define usage behavior as the test model. A separate specification is used for documenting the variability in the form of a feature model. This approach requires a third specification to define the mapping between the test model and the feature model. However, this requires one-time effort and test model can be transformed automatically. In addition, test model is not

tangled with variability information.

MPLM [94] is an extension of the MaTeLo tool-chain for supporting SPLE together with MBT. This tool also uses hierarchical Markov chains for capturing all possible use cases for a family of products. However, variability is specified in the form of an OVM [55]. This graphical model defines variation points, the set of variants for each variation point and selection constraints for each variant. MPLM removes a set of states and transitions from the usage model based on the selected variants in the OVM. In our approach, we employ feature models instead of OVM for documenting variability. This choice has 3 benefits. First, test model elements inherently represent features of the system and as such, defining a mapping between these elements and the feature model becomes straightforward. OVM defines both external and internal variability. Internal variability concerns with platform related or implementation level variations that are not visible to users [55]. Hence, its scope is unnecessarily broad for our purpose. Second, feature models can be better mapped to our test models due to their hierarchical structure. Third, the use of feature models are more common. This fact is recognized by the established theoretical work [57] and tool support [95].

There also exist other approaches [96, 97] that use feature models for documenting variability. However, they use test models in the form of state machines that cover all the features that can be included in any product. Transitions and states in the models are removed if the corresponding features are not selected for the product to be tested. In our approach, we keep the elements of the model and we do not change the structure of the model. We obtain different test cases by just modifying transition probabilities based on feature selections.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

Consumer electronics devices that are traditionally electromechanical systems has become software-intensive embedded systems. Increasing size and complexity of software systems makes it harder to ensure software reliability. This trend is a major issue for the consumer electronics domain, where the time-to-market is very short and systems are very cost sensitive. Traditional testing processes fall short being effective in detecting critical faults. They also lack efficiency regarding the use of limited resources. Therefore, there have been several techniques proposed to improve the testing process and automate testing activities. MBT is one of such techniques, which enables automatic generation of test cases based on models of a system. MBT has been successfully applied in different application domains. In this dissertation, we introduced methods and tools for effective application of MBT for the consumer electronics domain. In particular, we focused on the testing of DTV systems as our case study. We had identified the following 3 problems in this context: *i)* models have to be updated manually based on incomplete and imprecise information. *ii)* resources are extremely limited for the whole testing process. *iii)* test models are subject to high variability for large family of products.

To address the first problem, we introduced an approach and a toolset for incorporating information from manual ET activities into MBT to increase the effectiveness of models that are used for test case generation. In our approach, the execution traces that are recorded during ET activities are utilized as a feedback for refining test models. We applied our approach in the context of 3 industrial case studies to improve the models for model-based testing of a DTV system. We could detect real

and critical faults. These faults were not detected by MBT based on existing models and they were also missed during the ET activities.

To address the second problem, we introduced an iterative model refinement approach that utilizes usage profiles, static analysis and dynamic analysis. The overall goal was to cover different parts of a test model to effectively detect faults that are more likely to be exposed to the users. We applied our approach in the context of an industrial case study for MBT of a DTV system. We were able to detect new faults in each iteration, which shows that successive refinements of test models from different perspectives is a viable approach for increasing MBT efficiency. We performed a second case study with SP systems and likewise, we were able to detect new faults as well.

To address the third problem, we proposed an approach for using a reference test model for different products by adopting SPLE principles. This model captures all possible usage scenarios for a family of systems. In addition, we document variability explicitly with a feature model and define a mapping between this model and the test model. The preparation of these input models required an initial investment. However, testing effort per product was dramatically reduced since the reference test model can be automatically adapted for various products. We conducted a controlled experiment in an industrial setting to evaluate the effectiveness of the approach. Our experiment yielded significant results regarding the reduction of effort per tested product. We also evaluated the trade-off between the additional effort necessary for applying the approach and the effort reduction that is achieved after this one-time investment. We observed that the initial investment can be already amortized by reusing the reference test model for 13 different products in a product family.

Our future work has two dimensions. The first dimension is concerned with the improvement of solution approaches that we introduced. The second dimension is

regarding the improvement and extension of evaluation methods used for these approaches.

The first dimension is related to the improvement of both the underlying techniques and the implementation of tools. Currently, the tools that we have developed are mainly unmature research prototypes and they are in fact composed of a set of tools each of which performs a complementary task. For instance, ARME is composed of 3 complementary sub-modules: one of these determines new states and transitions to be added based on a given execution trace and event mapping specification; another one takes this information as input and applies the determined extensions on a given model; the third one takes the extended model and execution traces as input to calculate probability values for each transition of the model. In the future, we aim at providing a seamless integration of such complementary tools as part of an integrated development environment.

The improvement of underlying techniques can further reduce effort and risk of error. For instance, we utilize an *event mapping specification* for updating existing test models with ARME. One can consider the automated generation of a test model from scratch by just using this specification and recorded event sequences. A more effective approach would involve automated learning of usage behavior without using any specification that is created manually [68, 98]. That is, a test model can be directly inferred from low level sequence of events.

We have several assumptions for using ARME. These assumptions mainly lead to a more efficient but less generic approach. For instance, it is assumed that a sequence of events uniquely represents a state in an *event mapping specification*. This assumption might not hold for all types of systems but it simplifies the employed algorithm, which performs a single pass over the recorded event sequence. One can tackle any of such assumptions to trade-off efficiency for genericity. For example, mapping rules can be relaxed to allow intersecting sequences represent various states, in which case the

employed algorithm has to resolve resulting ambiguities.

As part of the second dimension of our future work, we plan to apply our solution approaches, the developed methods and tools for other types of systems, in particular home appliances products other than DTV systems. Concerning the evaluation of RIMA in particular, we would like to perform tests with randomly generated test cases to form a baseline for comparison. Such a comparison would reveal if new faults are detected as a result of the application of RIMA or if they can also be detected by just a naive, randomized test strategy. Also, we plan to conduct another controlled experiment with a crossed design regarding our third study with FORMAT. This time, we will study with a different group of subjects and randomize the order of treatments for participants.

Appendices

APPENDIX A

INDUSTRY AS LABORATORY APPROACH

Industry-as-laboratory was introduced as a promising approach for software engineering research [1]. It was proposed to address the drawbacks of the so-called *research-then-transfer* approach, which is more common in state-of-the-practice. Three main issues were observed concerning this conventional approach [1]. First, the research problem is usually identified and formulated according to the solution technology that is of interest to the researcher, not according to the needs of the industry. Second, there is a lack of feedback on interim results while these results are refined in successive research efforts built on top of each other. Third, the evaluation is delayed until the research is deemed mature enough at which point it becomes more difficult to find relevant case studies and apply the results.

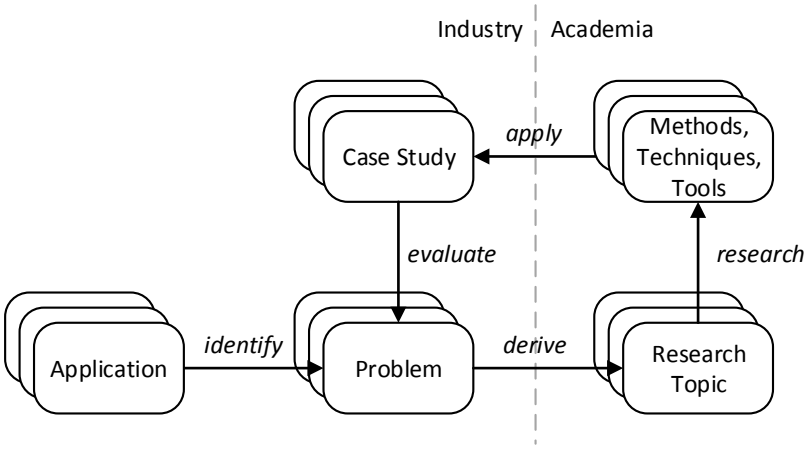


Figure 25: Feedback cycle employed in the industry-as-laboratory approach.

The overall process employed in an industry-as-laboratory approach is presented in Figure 25. Hereby, the research problem is identified based on an evaluation

of the application environment in the industry. Research and technology-transfer activities are not performed in separate, sequential phases. They are interleaved in the form of a continuous feedback cycle, in which interim results are applied in practice. Identification of further problems and research topics are steered by the feedback received from industrial case studies. Successive research efforts become more and more problem-focused (See Figure 26).

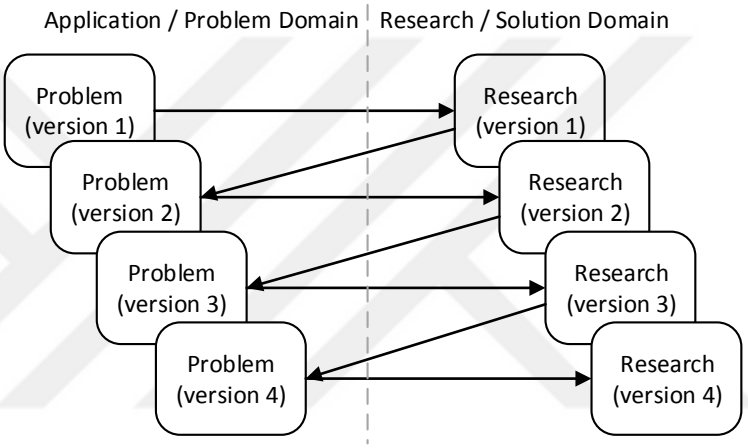


Figure 26: An illustration of the industry-as-laboratory approach [1].

We applied the industry-as-laboratory approach [1] for conducting research in the scope of this dissertation. In this section, we discuss our experiences in the application of this approach, which is proven to be highly effective for industry-academia collaboration and technology transfer. In the following, we introduce the observed challenges regarding the adoption of MBT in consumer electronics domain, for testing DTV systems in particular.

A.1 Research Challenges

In the scope of this dissertation, we focus on functional tests and in particular the adoption of MBT for automating these tests. We observed several practical issues in our research context, as listed in the following.

- *Error-prone Modeling*: We observed that critical faults were being detected during the manual ET activities. These faults were missed by the generated test cases using MBT. These test cases did not exercise the corresponding usage scenarios simply because test models were missing some of the transitions and states. For instance, it was possible to directly go back to a menu screen from a sub-menu after a timeout duration, while this transition was not included in the model at all. There are two main reasons for such omissions. The first reason is human error since model creation is a manual process. Second, requirement specifications are used as the main information source for model creation. These specifications are often incomplete and imprecise. As a result, created test models can be incomplete as well. In fact, the use MBT revealed the missing parts in the specification; however, this happened only after manually investigating the reason for inconsistencies between the set of faults detected by MBT and those detected during ET. Test engineers explore the usage behavior of the system independently and their explorations are generally not documented and systematically compared with respect to requirement specifications.

- *Limited Time*: The time is extremely limited compared to the amount of functionality to be tested in the consumer electronics domain. Test models that we use in our case studies have hundreds of states and transitions. However, some of the existing test models for DTV systems that were developed in Vestel for MBT already included thousands of states and transitions. These models are created from the user perspective. They have a hierarchical structure. The level of abstraction and different levels of hierarchy are aligned with the user interface. At the top level, each node represents a usage mode of the system. For each usage mode, there exist further sub-models. In these models, each node represents a state that the user can be in while using the system. For

instance, for the *video* mode, the user can be in states such as *playing*, *paused*, and *stopped*. In addition, the video can be fast forwarded, rewined, a previously recorded video can be retrieved, and the recording can be made regular in certain time intervals. Each relevant action in the context of various features (playing on Youtube, playing on MB, etc.) is represented as a state and as a result, there can be thousands of states due to the increasing number of features with complicated usage scenarios adopted in new generation DTV systems. Hierarchical modeling helps to manage such large-scale models by focusing on a particular usage mode at a time.

In principle, it is possible to generate infinitely many test cases from test models. Execution time for each test case depends on the number of test steps and the type of test steps involved. It can range from 1 minute to 1.5 hours. Even a single test step can take 10 minutes to execute (e.g., playing video for 10 minutes). In practice, trade-offs have to be made due to limited time. The generated test cases have to focus only on execution paths that are liable to highly severe failures that can be directly observed by users. Time limitation is mainly related to the level of test automation, which is discussed in the following.

- *Test Automation*: Test case generation is automated with an MBT tool[1]. However, automated execution of the generated test cases is not straight-forward. These test cases are defined at an abstract level and they cannot be directly executed. Vestel uses an in-house developed tool, namely VesTA, to automate test execution on DTVs with scripts. The implementation of these scripts requires one-time effort and they are embedded in test models. Such models were already available prior to our case studies. Hence, test execution is automated.

[1]Vestel currently employs MaTeLo (http://www.all4tec.net) for developing test models and generating test cases from these models.

However, test automation also requires the differentiation of correct and incorrect system behavior to evaluate test results. In general, this problem is referred to as the test oracle problem [99]. VesTA can automate test case evaluation by comparing the current DTV screen with respect to a set of previously captured images of the screen. For some of the test cases, this approach is applicable and sufficient, provided that the expected results are defined in the test scripts. For some other test cases, a test engineer has to manually check certain properties on the screen and audio as well. An example for this case could be the test of subtitle mode for people that have hearing impairment. The color of subtitles in this mode should change according to different people speaking in the displayed picture at different times. It is very hard to automate test oracles for such cases due to synchronization issues. We do not consider such issues as a MBT problem in general, but rather practical impediments of the application domain that should be taken into consideration while using MBT.

- *Model Variability*: In addition to the products of its own, Vestel is producing DTV systems for 157 different brands from 145 different countries. The set of features, broadcast specifications and user interfaces can differ from system to system. MBT is actually better than manual testing with respect to handling this variability. Variations can be better managed at the abstraction level of test models. However, MBT still falls short to address systematic variability for large scale product families with high number of variations that cross-cut test models [19, 20]. The problem gets amplified when we utilize embedded test scripts in models to facilitate test execution automation. Although test models are defined at an abstract level, test scripts are not. They comprise low level input events that are subject to more variation for various products. Therefore, the MBT process and test models must be flexible to systematically

manage variability and increase the amount of reuse for testing artifacts. This need can be fulfilled to some extent by parameterizing the lower level scripts in the test model. However, when the number of parameters increases as well as interactions (possible conflicts) among them, a systematic approach and tool support are required to manage them. Otherwise, manual traceability and maintainability becomes infeasible. A SPLE approach is required for facilitating systematic and scalable reuse [21].

There are currently hundreds of mapping definitions in the database. The management of this database becomes a challenge due to high variation among the various types of DTV systems. The model is kept generic by utilizing high level scripts and mapping them to platform-dependent low level scripts. However, this approach shifted the problem of management of *Model Variability* to the management of mapping definitions.

A.2 Application of the Industry-as-Laboratory Approach

In this section, we provide an overview of the problems listed in the previous subsection and the research activities performed to address these problems. We illustrate the application of the industry-as-laboratory approach, where the emergence of research problems and identification of research topics are interleaved.

The process is depicted in Figure 27, which is an instance of the approach shown in Figure 26. There are two problems initially identified: *error-prone modeling* and *limited time*. Limited time is also related to the lack of *test automation*. *Utilization of ET for automated extension of models* is considered as a solution approach for error-prone modeling and this approach was introduced in Chapter 3. However, extension of models increases the testing time and as such amplifies the problem of limited time. Limited time prohibits the exhaustive coverage of the test model; hence, we investigated the *utilization of usage profile to focus test cases*. However,

fault detection effectiveness had to be improved by focusing test cases also on scenarios that are likely to expose failures. Failure likelihood had to be estimated for this purpose. The first attempt was the *utilization of static analysis for failure likelihood estimation*. The observation of negligible reduction in test duration and undetected *memory-related failures* led to the *utilization of dynamic analysis for memory-related failure likelihood estimation*. These problems and the solution approach were defined in Chapter 4. Test automation problem has two dimensions: *test oracle automation* and automation of test execution. *Test oracle automation* is not in the scope of the dissertation. Automation of test execution is addressed with *embedded test scripts in models*, which is also not in the scope of the dissertation. Embedded scripts are subject to variations for different products. This amplifies the *model variability* problem, which is addressed by explicitly *coupling of test models with feature models* and this approach was introduced in Chapter 5.

A.3 Discussion

The success of the industry-as-laboratory approach depends on its implementation, considering many alternatives of realization [100]. Variation points in the implementation of this approach were previously identified by [100] as follows:

- Kind of researchers, e.g., academic PhD student, industrial PhD student, academic Post-Docs, or industrial researchers; and their amount of academic and industrial experience.

- Kind of research, e.g., explorative or applied, and problem- or solution-driven.

- Kind of company involvement, from strongly pulling to resisting change, and from only a single manager to all employees.

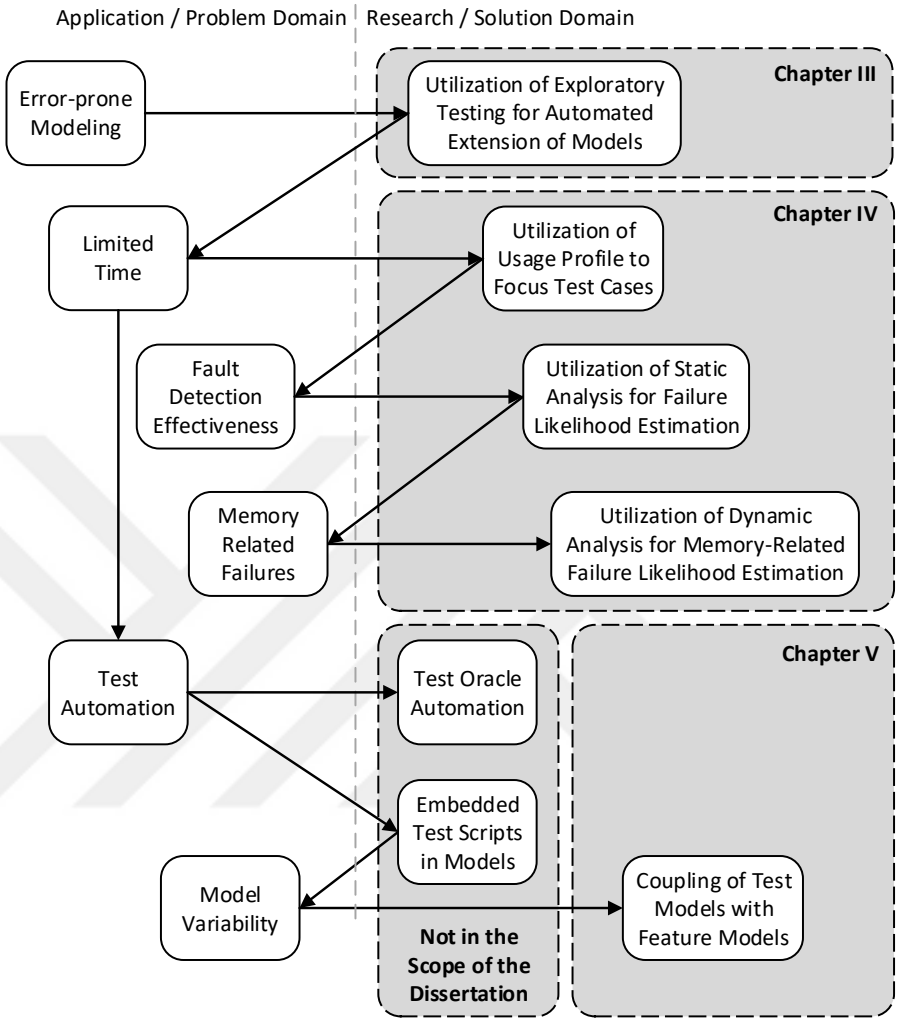


Figure 27: The application of the industry-as-laboratory approach (See Figure 26).

- The knowledge, skills, and abilities of the researchers.

In our implementation, an industrial Ph.D. student is involved as the main researcher, who is supported by her manager and an academic supervisor. The researcher has a M.Sc. degree focusing on MBT and she is a full time employee of the company with around 6 years of experience. There is an overlap between her job definitions as a software test engineer and the set of tasks she need to carry out as part of her Ph.D. studies. For example, she is responsible for maintaining and improving test

models created for MBT. She also manages a group of other test engineers and technicians for performing regular tests on products. So, performing experiments turns out to be part of daily activities. She spends 2 days per week for pure research work (not directly related to product development) on average. This work includes research meetings (at least 1 hour every week), literature study, research tool development, paper writing, presentations, and project proposal preparation.

The performed research is applied and problem-driven, which is based on problems directly observed during the first-hand experience on the adoption of MBT. We consider this as a major factor for the success of the approach. Due to obligations in the company, the researcher also have to be involved in problem-solving activities with a short-term focus. These activities should be separated from research activities that aim at generic results with a long-term focus. The main drawback of our implementation of the approach is the switching overhead caused by this conflict. On the other hand, it has also advantages. Research results can be very quickly validated since the researcher has the necessary background, expertise and full control of the industrial setting, i.e., the laboratory.

A.4 Related Work and Our Contributions

There exists a large body of work on MBT techniques [62] and tools [44]. There also exist several case studies [13, 65] where the effectiveness of MBT is evaluated with a special focus on embedded, safety-critical systems. Some of the proposed MBT techniques are extensively evaluated also for other application domains with large case studies [69]. However, these case studies are mainly based on open-source software systems.

Unfortunately, many academical methods never reach industrial application [1]. Industry-as-laboratory was introduced as a research approach to address this problem. This approach was previously applied in the Trader project [101] by bringing a

set of academic and industrial partners together to increase the dependability of consumer electronic devices. In that project, DTV systems were used as case studies as well. These studies took place in their industrial context to promote the applicability and scalability of solutions under relevant practical constraints [101]. The project involved research efforts in several different areas like run-time verification [102], error recovery [103], fault diagnosis [104], software architecture analysis [105, 106] and source code analysis [107]. However, MBT was not in the focus of the project.

There also exist other applications of the industry-as-laboratory approach reported in the literature [100, 108, 109, 110, 111, 112]. However, they focus on different application domains and different research areas in software engineering and systems engineering.

APPENDIX B

CASE STUDY ON EXPLORATORY TESTING

ET [45, 46, 113] adopts a continuous learning and adaptation process for testing. Hereby, the tester iteratively learns about the product and its faults, plans the testing activities, designs, and executes the tests [14]. Unlike traditional test case based testing, ET is not based on a set of predesigned test cases. Instead, tests are dynamically designed, executed, and modified by testers. Hence, test design, execution, and learning are all concurrent activities.

ET does not utilize formal descriptions or detailed methodologies. Testing activities are performed manually and testers do not strictly follow any procedures during these activities. As a result of these facts, one might consider ET as an ad-hoc approach. Nevertheless, it is known to be one of the mostly applied and one of the most effective approaches [50, 51, 114, 115] in terms of revealing failures. This recognition is also aligned with our own observations in the industry [25, 116]. We have seen that test models that are refined based on ET activities were more effective in terms of finding failures.

ET aims at exploiting human effort efficiently by utilizing intuition, experience, and knowledge. This experience and knowledge can be related to the application domain, system (i.e., specific to the tested product) or background (e.g., general software engineering and testing knowledge) [82]. In particular, ET activities are supposed to be performed by testers who have both technical knowledge and accumulated unwritten knowledge on where failures most likely exist [14]. Therefore, it is believed that ET is largely dependent on the skills and experience of the tester [115]. However, to the best of our knowledge, the impact of education and experience level on the

effectiveness of ET has not been formally evaluated before.

B.1 Industrial Case Study and Discussion

We report a case study that is performed in an industrial context. 19 practitioners, who have different education and experience levels, were involved in applying ET for testing a DTV system. We measured the number of detected failures and categorized these failures based on their severity. We also measured the time spent by each subject for performing the tests. Then, we compared different groups of subjects. These groups are formed based on experience and education levels. Comparisons are performed with respect to two criteria: *i)* the number of detected failures that are critical, and *ii)* efficiency measured as the number of failures detected per unit of testing time.

Results show that the efficiency of ET is significantly affected by both the educational background and experience levels. Experience level has a significant impact on the number of detected critical failures as well. However, we can not observe an impact of education on the number of critical failures detected.

Our case study is performed for testing DTV systems developed by Vestel. DTV systems are highly cost sensitive and they are subject to short development time periods. The market is highly competitive and end users are less tolerant to failures [117]. Hence, effective testing methods are essential. ET is known to be one of the methods that has been applied for years [25, 116]. It has been applied by different employees in the company over time.

In this study we aimed at evaluating the impact of the educational backgrounds and experience levels of these employees on the effectiveness of ET. Hereby, we differentiate experience regarding the domain or the system under test from the experience in testing activities in general. In terms of educational background, we differentiate

between those who have higher education (college or university) on a relevant subject (Computer Science/Engineering or Software Engineering) and those who do not have (high school graduates or graduates of two-year educational programs). Testers who have higher education and do not have higher education participated in software testing training which was given internally in the company for 2 days.

B.2 Research Questions

We evaluate the effectiveness of ET from two aspects. First, we consider test efficiency based on the effort and the number of detected failures. Second, we consider how critical the detected failures are. Therefore, we defined the following research questions.

> ***RQ1:*** How domain and testing experiences are affecting the test efficiency in terms of number of failures detected per unit of time?
>
> ***RQ2:*** How domain and testing experiences are affecting the number of critical failures detected?
>
> ***RQ3:*** How educational background is affecting the test efficiency in terms of number of failures detected per unit of time?
>
> ***RQ4:*** How educational background is affecting the number of critical failures detected?

Accordingly, we defined the following 4 hypothesis:

- H_o^1 The level of domain experience and testing experience do not have any effect on the ET efficiency.
- H_o^2 The level of domain experience and testing experience do not have any effect on the the criticality of detected failures during ET.

- H_o^3 The level of education does not have any effect on the ET efficiency.
- H_o^4 The level of education does not have any effect on the the criticality of detected failures during ET.

. In the following, we explain the experimental setup we used for addressing the research questions.

B.3 Experimental Setup

There is a dedicated software testing group in the company who is performing tests of various consumer electronics products such as DTVs, refrigerators, washing machines, dishwashers, air conditioners, cookers, and smart phones. Most of the tests are automated but there are also manual tests being performed and we focused on such tests in this study. The testing group is composed of either test engineers who studied at a college/university or test technicians who completed two-year educational programs. We refer to both test engineers and test technicians as practitioners in the rest of the paper.

In total, 19 practitioners were involved in the case study as subjects. These practitioners were instructed to apply ET for a particular feature of a real DTV system. This feature was explained to all the subjects for 15 minutes before the case study. Some of the subjects have already had domain knowledge, i.e., they have been previously working on testing DTVs. Some other subjects lacked this knowledge, i.e., they were involved in the testing of products other than DTVs.

The list of all the subjects are provided in Table 19. For each subject, 3 properties are listed in the 2^{nd}, 3^{rd} and 4^{th} columns, respectively. These properties also define the factors we consider in the case study:

- *Domain Experience*: measured in ratio scale in terms of the number of years.
- *Testing Experience*: measured in ratio scale in terms of the number of years.

- *Higher Education*: measured in nominal scale, at two levels: exists, not exists.

In Table 19, it can be seen that, some practitioners have more domain experiences than testing experiences. These practitioners are testers who do not have higher education and they had worked for various products other than DTVs.

Practitioner ID (PId)	Domain Experience (# of years)	Testing Experience (# of years)	Higher Education (Yes/No)
1	11	11	No
2	8	8	No
3	16	14	No
4	7	5	No
5	7	7	No
6	11	10	No
7	8	10	No
8	1	1	Yes
9	6	6	Yes
10	1	8	Yes
11	1	1	Yes
12	1	1	Yes
13	1	1	Yes
14	1	1	Yes
15	4	4	Yes
16	4	4	Yes
17	12	12	Yes
18	0	3	Yes
19	0	2	Yes

Table 19: The whole list of subjects.

We asked all the participants to perform ET on the same system. They were just observed without interference throughout this process. The system under test was also not altered (no bug fixes) throughout the study. We measured/calculated the following variables:

- *Test duration*: measured for each subject in ratio scale in terms of the number of days.

- *Number of failures detected*: measured for each subject in absolute scale for each of the failure categories listed as *Critical*, *Major*, *Minor*, and *Trivial*.
- *Efficiency*: calculated for each subject as the ratio of the total number of failures detected and test duration.

In the following, we present and discuss the results.

Practitioner ID (PId)	Test Duration (# of days)	# of Failures Detected			
		Critical	Major	Minor	Trivial
1	10	2	2	5	1
2	10	2	2	6	2
3	9	2	2	5	0
4	9	2	2	5	0
5	10	2	2	4	0
6	10	2	2	5	0
7	10	2	2	5	0
8	15	2	2	5	2
9	6	2	2	6	0
10	10	1	2	2	0
11	15	1	2	5	1
12	14	1	2	5	0
13	15	1	2	5	1
14	12	0	2	5	1
15	8	2	2	6	0
16	9	2	2	6	0
17	6	2	2	6	0
18	16	1	2	4	2
19	16	1	2	4	2

Table 20: The list of overall results.

B.4 Results and Discussion

The overall results are summarized in Table 20. The first column lists the ID of each subject just like in Table 19. The second column lists the test duration. For instance, we can observe that $PId-18$ and $PId-19$ (Practitioner ID 18 and 19) completed the test in 16 days, where $PId-9$ and $PId-17$ completed the test in 6 days. The last column lists the number of failures detected. This list is provided separately for the 4 different failure categories; *Critical*, *Major*, *Minor*, and *Trivial*. For instance,

we can see that $PId-2$ detected the maximum total number of failures (12); however $PId-10$ could only find 5 failures.

We compared different groups of subjects with respect to efficiency and the number of detected failures that are of type *Critical*. These groups are formed based on experience and education levels. Education level is already provided in nominal scale and as such it is trivial to separate the two groups. To be able to separate subjects with respect to experience, we set a threshold for the number of years of experience as 2. Subjects, who have at least 2 years of (both domain and testing) experience are considered *experienced*, while the others are considered *inexperienced*.

In the following, we list the results for different groupings to answer the 4 research questions.

Impact of domain and testing experience on test efficiency To be able to evaluate the impact of experience, we have separated the results into two; *i)* those for subjects who do not have higher education, and *ii)* those for subjects who have higher education. These results are listed in Table 21 and 22, respectively.

Practitioner ID (PId)	Domain Experience	Testing Experience	#Critical Failures	Efficiency
1	11	11	2	1.00
2	8	8	2	1.20
3	16	14	2	1.00
4	7	5	2	1.00
5	7	7	2	0.80
6	11	10	2	0.90
7	8	10	2	0.90

Table 21: Results for subjects who do not have higher education.

When we look at the subjects that do not have higher education (Table 21), we see that they all have high experience levels. Therefore, we could not evaluate the impact of experience for this group. However, experience levels vary for the subjects who have higher education (See Table 22). We can also observe that the efficiency is

Practitioner ID (PId)	Domain Experience	Testing Experience	#Critical Failures	Efficiency
8	1	1	2	0.73
9	6	6	2	1.67
10	1	8	1	0.50
11	1	1	1	0.60
12	1	1	1	0.57
13	1	1	1	0.60
14	1	1	0	0.67
15	4	4	2	1.25
16	4	4	2	1.11
17	12	12	2	1.67
18	0	3	1	0.56
19	0	2	1	0.56

Table 22: Results for subjects who have higher education.

higher for those subjects that have experience. We performed a t-test to validate this observation. We formed two groups, Group A and Group B from the list of subjects in Table 22. Group A consists of experienced subjects (who have 2 or more years of experience), whereas Group B consists of inexperienced subjects (who have less than 2 years of experience). We compared the efficiency for these groups.

Results[1] are listed in Table 23.

We can see that P(T¡=t) one-tail and P(T¡=t) two-tail values are very low. They are well below the commonly accepted threshold (0.05) [118], which means that the difference is significant. That also means that the null hypothesis H_o^1 can be rejected and the impact of experience on ET efficiency can be confirmed. We also evaluated the impact of experience on the criticality of the detected failures. This is discussed in the following.

Impact of domain and testing experience on the criticality of the detected failures We can observe in Table 22 that experienced subjects detected more failures of type *Critical*. We performed a separate t-test to evaluate the significance of

[1]We used Microsoft Excel (2010) to obtain the results.

	Group A	Group B
Mean	1.423	0.599
Variance	0.081	0.005
Observations	4	8
Hypothesized Mean Difference	0	
df	3	
t Stat	5.668	
P(T $\leq$ t) one-tail	0.005	
t Critical one-tail	2.353	
P(T $\leq$ t) two-tail	0.01	
t Critical two-tail	3.182	

Table 23: T-test results regarding the comparison of experienced (Group A) and inexperienced (Group B) groups by means of test efficiency.

this difference. We performed the comparison between the same groups, Group A and Group B as formed in the previous test.

The results are listed in Table 24. Again, the P values turn out to be well below 0.05, which points out the significance of the difference. Hence, we conclude that the null hypothesis H_o^2 can be rejected. The level of experience has a significant impact on the criticality of failures detected during ET.

In fact, we can observe from Table 21 that all the subjects detected all the 2 critical failures. Recall that all of these subjects have high experience though they do not have higher education. Their efficiency (0.97) is also higher than those listed in Table 22 (0.87) on average. We evaluate the impact of education in more detail in the following.

Impact of higher education on test efficiency To be able to evaluate the impact of higher education, we have separated the results into two; *i)* those for subjects who have 2 or more years of domain and testing experience, and *ii)* those who have less than 2 years of experience. These results are listed in Table 25 and 26, respectively.

When we look at the inexperienced subjects (Table 26), we see that they all have

	Group A	Group B
Mean	2	1
Variance	0	0.285
Observations	4	8
Hypothesized Mean Difference	0	
df	7	
t Stat	5.291	
P(T ≤ t) one-tail	0.0005	
t Critical one-tail	1.894	
P(T ≤ t) two-tail	0.0011	
t Critical two-tail	2.364	

Table 24: T-test results regarding the comparison of experienced (Group A) and inexperienced (Group B) groups by means of number of detected critical failures.

higher education. Therefore, we could not evaluate the impact of education for this group. However, education level varies for the subjects who are experienced (See Table 25). On average, the efficiency of those who have higher education (1.42) is more than the efficiency of others (0.97). We also performed a t-test to evaluate the significance of this difference. We formed two groups, Group C and Group D from the list of subjects in Table 25. Group C consists of subjects with higher education, whereas Group D consists of subjects without higher education. We compared the efficiency for these groups. Results are listed in Table 27.

We can see that P(T¡=t) one-tail and P(T¡=t) two-tail values are 0.019 and 0.039, respectively. These values are below 0.05, suggesting that education has a significant impact on efficiency among the experienced subjects. Hence, the null hypothesis H_o^3 can also be rejected although, P values are closer to the threshold in this case. However, we can not conclude the same for the criticality of the detected failures as discussed in the following.

Impact of higher education on the criticality of detected failures We can see in Table 25 that all the subjects detected 2 critical failures. Hence, we can not

Practitioner ID (PId)	Higher Education	# of Critical Failures	Efficiency
1	No	2	1.00
2	No	2	1.20
3	No	2	1.00
4	No	2	1.00
5	No	2	0.80
6	No	2	0.90
7	No	2	0.90
9	Yes	2	1.67
15	Yes	2	1.25
16	Yes	2	1.11
17	Yes	2	1.67

Table 25: Results for subjects who have at least 2 years of experience.

Practitioner ID (PId)	Higher Education	# of Critical Failures	Efficiency
8	Yes	2	0.73
10	Yes	1	0.50
11	Yes	1	0.60
12	Yes	1	0.57
13	Yes	1	0.60
14	Yes	0	0.67
18	Yes	1	0.56
19	Yes	1	0.56

Table 26: Results for subjects who have less than 2 years of experience.

observe any impact of higher education in that respect. There is no difference at all. So, we conclude that higher education does not have an impact on the criticality of detected failures. As such, we have to accept the null hypothesis H_o^4.

In fact, we can also observe from Table 21 that all the subjects who do not have higher education could find all the critical failures. On the other hand, subjects who do have higher education could not (See Table 22). However, we can not compare these directly since the level of experience is different between the groups.

	Group C	Group D
Mean	0.971	1.423
Variance	0.015	0.081
Observations	7	4
Hypothesized Mean Difference	0	
df	4	
t Stat	-2.998	
P(T $\leq$ t) one-tail	0.019	
t Critical one-tail	2.131	
P(T $\leq$ t) two-tail	0.039	
t Critical two-tail	2.776	

Table 27: T-test results regarding the comparison of groups who have higher education (Group C) and who do not have higher education (Group D) by means of efficiency.

In the following, we discuss validity threats for our case study.

B.5 Threats to Validity and Limitations

Our study is subject to an external validity threat [91] since it is based on a single case study. Internal validity threats are mitigated by using a real system and involving real participants from the industry to our study. Conclusion and construct validity threats are mitigated by observing the activities of participants without interfering with them. The number of participants (19) can also lead to conclusion and construct validity threats. However, we performed statistical tests to evaluate the significance of the results.

We also performed Anova analysis [119] to test the significance of differences among different groups of participants. 4 different groups can be considered based on the 2 factors we evaluate in this study. These groups are shown in Table 28.

Hereby, we are missing one of the groups as highlighted in the table since we do not have any subjects in that category. We performed one-way (single factor) Anova analysis on the remaining 3 groups in terms of test efficiency. Table 29 lists sum, average and variance values for each group, whereas Table 30 lists the analysis

Factors	Higher Education	No Higher Education
Experience	Members of both Group A and C	Members of both Group A and D
No Experience	Members of both Group B and C	Members of both Group B and D

Table 28: 4 different groups of subjects based on 2 factors: experience and education level.

results.

	Group A & D	Group A & C	Group B & C
Count	7	4	8
Sum	6.8	5.69	4.79
Average	0.97	1.42	0.59
Variance	0.015	0.081	0.005

Table 29: Descriptive statistics regarding the test efficiency of groups listed in Table 28.

We can see that the F value is much greater than F-critical value. We can also see that the P-value is much smaller than 0.05. These results also indicate significant difference among the groups of subjects.

	Between Group	Within Group	Total
SS	1.852	0.376	2.228
df	2	16	18
MS	0.926	0.023	0.59
F	39.4		
P-value	0.0000007		
F crit	3.63		

Table 30: Anova analysis results regarding the comparison of groups listed in Table 28 in terms of test efficiency.

The significance of the results can also be observed with the box plot depicted in Figure 28. Hereby, we compare the distributions of test efficiency values for the 3

groups on which we applied Anova analysis. We can see that error bars (i.e., whiskers) are short and variance within each group is small. This also increases our confidence regarding the significance of the results.

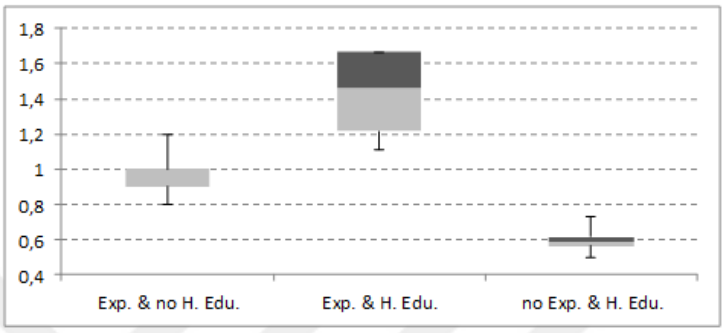


Figure 28: Box plots regarding the test efficiency of the 3 groups listed in Table 28; Exp. & no H. Edu (Group A & D), Exp. & H. Edu. (Group A & C), no Exp. & H. Edu. (Group B & C)

B.6 Related Work and Our Contributions

Although ET is commonly applied in practice, scientific research and empirical studies on ET are scarce, especially when we consider those that are conducted in an industrial context [79, 115]. In the following, we summarize existing publications on related experimental studies.

There have been experimental studies on testing techniques in general [120]. An evaluation of these techniques [115] suggests that skills and experience of the tester turn out to be an important factor for testing effectiveness even in test case based testing. Similarly, the importance of both testing knowledge and domain knowledge was confirmed by industrial case studies before [121]. In another study [122], the impact of experience was evaluated for the effectiveness of test case design. Hereby, results show that neither the experienced nor the inexperienced testers performed better with respect to each other in all aspects. The two groups had both relative strengths and weaknesses with respect to different aspects.

An experimental study [123] for comparing ET and test case based testing revealed that these two approaches do not differ in terms of failure detection effectiveness. However, they have both pros and cons with respect to efficiency, the number of false positives and management overhead.

A controlled experiment [124] was conducted to compare the failure detection effectiveness of MBT and model-based ET. Results show that the overall test effectiveness was improved though the two approaches detect different types of failures. We have also conducted case studies before for evaluating the effectiveness of MBT, when it is supported by ET [25]. Results show that the test effectiveness can be significantly improved when the test models are refined based on execution traces collected during ET activities.

An empirical study [80] that focuses particularly on the effectiveness of ET studied the impact of personality traits as a factor. Results show that testers having extrovert personality might be more likely to be good at ET. In this study, we evaluate the impact of education and experience level on the failure detection effectiveness of ET.

Bibliography

[1] C. Potts, "Software-engineering research revisited," *IEEE Software*, vol. 10, no. 5, pp. 19–28, 1993.

[2] G. Sivaraman, P. Csar, and P. Vuorimaa, "System software for digital television applications," in *IEEE International Conference on Multimedia and Expo*, pp. 784–787, 2001.

[3] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 11–33, 2004.

[4] I. de Visser, *Analyzing user perceived failure severity in consumer electronics products : incorporating the user perspective into the development process*. Ph.D. thesis, Eindhoven University of Technology, 2008.

[5] R. Mathijssen, *TRADER: Reliability of High-volume Consumer Products*. Eindhoven, The Netherlands: Embedded Systems Institute, 2009.

[6] L. Apfelbaum and J. Doyle, "Model-based testing," in *Software Quality Week Conference*, pp. 296–300, 1997.

[7] E. Dustin, J. Rashka, and J. Paul, *Automated Software Testing: Introduction, Management, and Performance*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[8] J. Boberg, "Early fault detection with model-based testing," in *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*, pp. 9–20, 2008.

[9] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Software Testing Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012.

[10] A.Pretschner, *Proceedings of the International Symposium of Formal Methods Europe*, ch. Model-Based Testing in Practice, pp. 537–541. Springer Berlin Heidelberg, 2005.

[11] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[12] B. Nguyen, B. Robbins, I. Banerjee, and A. Memon, "GUITAR: an innovative tool for automated testing of gui-driven software," *Automated Software Engineering*, vol. 21, no. 1, pp. 65–105, 2014.

[13] M. Lindvall, D. Ganesan, R. Ardal, and R. Wiegand, "Metamorphic model-based testing applied on NASA DAT – an experience report," in *Proceedings of the 37th International Conference on Software Engineering*, pp. 129–138, 2015.

[14] J. A. Whittaker, *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design*. Addison-Wesley Professional, 1st ed., 2009.

[15] V. Pekovi, N. Tesli, I. Resetar, and T. Tekcan, "Test management and test execution system for automated verification of digital television systems," in *IEEE International Symposium on Consumer Electronics (ISCE 2010)*, pp. 1–6, 2010.

[16] D. Marijan, V. Zlokolica, N. Teslic, V. Pekovic, and T. Tekcan, "Automatic functional tv set failure detection system," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 1, pp. 125–133, 2010.

[17] M. F. Kirac, B. Aktemur, and H. Sozer, "Visor: A fast image processing pipeline with scaling and translation invariance for test oracle automation of visual output systems," *Journal of Systems and Software*, 2017.

[18] J. Whittaker and M. Thomason, "A markov chain model for statistical software testing," *IEEE Transactions on Software Engineering*, vol. 20, no. 10, pp. 812–824, 1994.

[19] H. Lackner, M. Thomas, F. Wartenberg, and S. Weißleder, "Model-based test design of product lines: Raising test design to the product line level," in *Proceedings of the 7th IEEE International Conference on Software Testing, Verification and Validation*, pp. 51–60, 2014.

[20] H. Samih and R. Bogusch, "MPLM - matelo product line manager: [relating variability modelling and model-based testing]," in *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, pp. 138–142, 2014.

[21] E. Engstrm and P. Runeson, "Software product line testing a systematic mapping study," *Information and Software Technology*, vol. 53, no. 1, pp. 2 – 13, 2011.

[22] M. Felderer and I. Schieferdecker, "A taxonomy of risk-based testing," *International Journal of Software Tools and Technology Transfer*, vol. 16, no. 5, pp. 559–568, 2014.

[23] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2008.

[24] C. S. Gebizli and H. Sozer, "Improving models for model-based testing based on exploratory testing," in *Proceedings of the 6th IEEE Workshop on Software Test Automation*, pp. 656–661, 2014. (COMPSAC Companion).

[25] C. Gebizli and H. Sozer, "Automated refinement of models for model-based testing using exploratory testing," *Software Quality Journal*, 2016. published online, DOI: 10.1007/s11219-016-9338-2.

[26] C. S. Gebizli, D. Metin, and H. Sozer, "Combining model-based and risk-based testing for effective test case generation," in *Proceedings of the 9th Workshop on Testing: Academic and Industrial Conference - Practice and Research Techniques*, pp. 1–4, 2015. (ICST Companion).

[27] C. S. Gebizli, H. Sozer, and A. Ercan, "Successive refinement of models for model-based testing to increase system test effectiveness," in *Proceedings of the 10th Workshop on Testing: Academic and Industrial Conference - Practice and Research Techniques*, pp. 263–268, 2016. (ICST Companion).

[28] C. S. Gebizli and H. Sözer, "Model-based software product line testing by coupling feature models with hierarchical markov chain usage models," in *2016 IEEE International Conference on Software Quality, Reliability and Security, QRS 2016, Companion, Vienna, Austria, August 1-3, 2016*, pp. 278–283, 2016.

[29] C. S. Gebizli and H. Sozer, "Impact of education and experience level on the effectiveness of exploratory testing: An industrial case study," in *Proceedings of the 12th Workshop on Testing: Academic and Industrial Conference Practice and Research Techniques*, (Tokyo, Japan), pp. 23–28, 2017.

[30] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Softw. Test. Verif. Reliab.*, vol. 22, no. 5, pp. 297–312, 2012.

[31] Q. A. Malik, A. Jskelinen, H. Virtanen, M. Katara, F. Abbors, D. Truscan, and J. Lilius, "Model-based testing using system vs. test models - what is the difference?," in *Proceedings of the 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, pp. 291–299, 2010.

[32] S. Weißleder and H. Lackner, "System models vs. test models -distinguishing the undistinguishable?," in *Informatik 2010: Service Science - Neue Perspektiven für die Informatik, Beiträge der 40. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Band 2, 27.09. - 1.10.2010, Leipzig*, pp. 321–326, 2010.

[33] L. Briand and Y. Labiche, "A uml-based approach to system testing," *Software and Systems Modeling*, vol. 1, no. 1, pp. 10–42, 2002.

[34] T. Chow, "Testing software design modeled by finite-state machines," *IEEE Transactions on Software Engineering 4*, vol. 4, no. 3, pp. 178–187, 1978.

[35] F. Belli, A. T. Endo, M. Linschulte, and A. Simao, "A holistic approach to model-based testing of web service compositions," *Software: Practice and Experience*, vol. 44, no. 2, pp. 201–234, 2014.

[36] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987.

[37] J. Tretmans, *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18,*

2011. Advanced Lectures, ch. Model-Based Testing and Some Steps towards Test-Based Modelling, pp. 297–326. Springer Berlin Heidelberg, 2011.

[38] S. Naito and M. Tsunoyama, "Fault detection for sequential machines by transitions tours," *IEEE Fault Tolerant Computer Symp.*, pp. 238–243, 1981.

[39] G. Gonenc, "A method for the design of fault detection experiments," *IEEE Transactions on Computers*, vol. C-19, no. 6, pp. 551–0558, 1970.

[40] F. Belli, "Finite state testing and analysis of graphical user interfaces," in *Proceedings of 12th International Symposium on Software Reliability Engineering, ISSRE2001*, pp. 34–43, 2001.

[41] I. Schieferdecker, "Model-based testing," *IEEE Software*, vol. 29, no. 1, pp. 14–18, 2012.

[42] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[43] S. Weißleder, *Test Models and Coverage Criteria for Automatic Model-Based Test Generation with UML State Machines*. Ph.D. thesis, Humboldt-Universitt zu Berlin, 2010.

[44] 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 *Proceedings of the International Conference on Software Engineering*, pp. 285–294, 1999.

[45] G. J. Myers and C. Sandler, *The Art of Software Testing*. John Wiley & Sons, 2004.

[46] W. C. Hetzel and B. Hetzel, *The Complete Guide to Software Testing*. New York, NY, USA: John Wiley & Sons, Inc., 2nd ed., 1991.

[47] J.Bach, "Exploratory testing explained," tech. rep., 2003.

[48] C. Kaner, "Exploratory testing," in *Quality Assurance Institute Worldwide Annual Software Testing Conference*, 2006.

[49] A. Tinkham and C. Kaner, "Exploring exploratory testing," in *Proceedings of the Software Testing and Analysis and Review East Conference*, 2003.

[50] J. Itkonen, M. V. Mantyla, and C. Lassenius, "Defect detection efficiency: Test case based vs. exploratory testing," in *First International Symposium on Empirical Software Engineering and Measurement*, pp. 61–70, IEEE Computer Society, 2007.

[51] J. Itkonen, *Empirical Studies on Exploratory Software Testing*. Ph.D. thesis, Aalto University, 2011.

[52] M. Felderer and R. Ramler, "Integrating risk-based testing in industrial test processes," *Software Quality Journal*, vol. 22, no. 3, pp. 543–575, 2014.

[53] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[54] L. C. Briand, Y. Labiche, and S. He, "Automating regression test selection based on uml designs," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 16–30, 2009.

[55] K. Pohl, G. Bockle, and F. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.

[56] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," tech. rep., Carnegie-Mellon University Software Engineering Institute, 1990.

[57] D. Batory, "Feature models, grammars, and propositional formulas," in *Proceedings of the 9th International Conference on Software Product Lines*, pp. 7–20, 2005.

[58] K. Czarnecki and A. Wasowski, "Feature diagrams and logics: There and back again," in *11th International Software Product Line Conference (SPLC 2007)*, pp. 23–34, 2007.

[59] S. Oster, A. Wbbeke, G. Engels, and A. Schrr, "A survey of model-based software product lines testing," in *Model-Based Testing for Embedded Systems* (J. Zander, I. Schieferdecker, and P. Mosterman, eds.), pp. 339–383, CRC Press, 2012.

[60] H. L. Guen, R. Marie, and T. Thelin, "Reliability estimation for statistical usage testing using markov chains," in *Proceedings of the 15th International Symposium on Software Reliability Engineering*, pp. 54–65, 2004.

[61] C. Joye, "Matelo test case generation algorithms: Explanation on available algorithms for test case generation," 2014. http://www.all4tec.net/MaTeLo-How-To/understanding-of-test-cases-generation-algorithms.html.

[62] A. C. D. Neto, R.Subramanyan, M.Vieira, and G. H. Travassos, "A survey on model-based testing approaches: A systematic review," in *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies*, pp. 31–36, 2007.

[63] H. Robinson, "Finite state model-based testing on a shoestring," in *Proceedings of the Software Testing and Analysis and Review West Conference*, 1999.

[64] A. Chander, D. Dhurjati, S. Koushik, and Y. Dachuan, "Optimal test input sequence generation for finite state models and pushdown systems," in *Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation*, pp. 140–149, 2011.

[65] J. Keranen and T. Raty, "Model-based testing of embedded systems in hardware in the loop environment," *IET Software*, vol. 6, no. 4, pp. 364–376, 2011.

[66] L. Mariani, M. Pezzè, and D. Zuddas, "Recent advances in automatic black-box testing," in *Advances in Computers* (A. Memon, ed.), vol. 99, pp. 157 – 193, Elsevier, 2015.

[67] L. Mariani, M. Pezz, O. Riganelli, and M. Santoro, "Automatic testing of gui-based applications," *Software Testing, Verification and Reliability*, vol. 24, no. 5, pp. 341–366, 2014.

[68] K. Meinke and M. A. Sindhu, "Lbtest: A learning-based testing tool for reactive systems," in *Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg, March 18-22, 2013*, pp. 447–454, 2013.

[69] B. Nguyen and A. Memon, "An observe-model-exercise* paradigm to test event-driven systems with undetermined input spaces," *IEEE Transactions on Software Engineering*, vol. 40, no. 3, pp. 216–234, 2014.

[70] Z. Zuo, *Refinement Techniques in Mining Software Behavior*. Ph.D. thesis, National University of Singapore, 2015.

[71] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic generation of software behavioral models," in *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pp. 501–510, ACM, 2008.

[72] V. Dallmeier, N. Knopp, C. Mallon, G. Fraser, S.Hack, and A. Zeller, "Automatically generating test cases for specification mining," *Software Engineering, IEEE Transactions on*, vol. 38, no. 2, pp. 243–257, 2012.

[73] V. Entin, M. Winder, B. Zhang, and S. Christmann, "Combining model-based and capture-replay testing techniques of graphical user interfaces: An industrial approach," in *Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation Workshops*, pp. 572–577, 2011.

[74] X. Yuan and A. Memon, "Generating event sequence-based test cases using GUI runtime state feedback," *IEEE Transactions on Software Engineering*, vol. 36, pp. 81–95, Jan. 2010.

[75] R. Ferguson and B. Korel, "The chaining approach for software test data generation," *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 1, pp. 63–86, 1996.

[76] C. Michael, G. McGraw, and M. Schatz, "Generating software test data by evolution," *IEEE Transactions on Software Engineering*, vol. 27, no. 12, pp. 1085–1110, 2001.

[77] T. Xie and D. Notkin, "Tool-assisted unit-test generation and selection based on operational abstractions," *Automated Software Engineering*, vol. 13, no. 3, pp. 345–371, 2006.

[78] C. Pacheco, S. Lahiri, M. Ernst, and T. Ball, "Feedbackdirected random test generation," in *Proceedings of the 29th International Conference on Software Engineering*, pp. 396–405, 2006.

[79] J. Itkonen and K. Rautiainen, "Exploratory testing: a multiple case study," in *Proceedings of International Symposium on Empirical Software Engineering*, pp. 84–93, 2005.

[80] L. Shoaib, A. Nadeem, and A. Akbar, "An empirical evaluation of the influence of human personality on exploratory software testing," *Multitopic Conference and IEEE 13th International*, pp. 1–6, 2009.

[81] A. Tinkham and C. Kaner, "Learning styles and exploratory testing," in *Proceedings of the Pacific Northwest Software Quality Conference*, 2003.

[82] J. Itkonen, M. V. Mantyla, and C. Lassenius, "The role of the testers knowledge in exploratory software testing," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 707–724, 2013.

[83] H. Robinson, "Intelligent test automation a model-based method for generating tests from a description of an applications behavior," *Software Testing and Quality Engineering Magazine*, pp. 24–32, 2000.

[84] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[85] C. S. Gebizli, D. Metin, and H. Sözer, "Combining model-based and risk-based testing for effective test case generation," in *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation*, pp. 1–4, 2015.

[86] F. Bohr, "Model based statistical testing and durations," in *Proceedings of the 17th IEEE International Conference on Engineering of Computer Based Systems*, pp. 344–351, 2010.

[87] F. Bohr, "Model based statistical testing of embedded systems," in *Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation*, pp. 18–25, 2011.

[88] C. Gebizli and H. Sozer, "Improving models for model-based testing based on exploratory testing," in *Proceedings of the 8th IEEE International Computer Software and Applications Conference Workshops*, pp. 656–661, 2014.

[89] J. Gibbons, *Nonparametric Statistics: An Introduction*. No. 90. no. in Nonparametric Statistics: An Introduction, SAGE Publications, 1993.

[90] L. Surhone, M. Timpledon, and S. Marseken, *Wilcoxon Signed-Rank Test*. VDM Publishing, 2010.

[91] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering*. Springer-Verlag, 2012.

[92] A. Reuys, E. Kamsties, K. Pohl, and S. Reis, "Model-based system testing of software product families," in *Proceedings of the 17th International Conference on Advanced Information Systems Engineering*, pp. 519–534, 2005.

[93] E. Olimpiew and H. Gomaa, "Model-based test design for software product lines," in *Proceedings of the 12th International Conference on Software Product Lines*, pp. 173–178, 2008.

[94] H. Samih and R. Bogusch, "MPLM - MaTeLo product line manager: [relating variability modelling and model-based testing]," in *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, pp. 138–142, 2014.

[95] M. E. Dammagh and O. D. Troyer, "Feature modeling tools: Evaluation and lessons learned," in *Advances in Conceptual Modeling. Recent Developments and New Directions* (O. De Troyer et al., ed.), vol. 6999 of *Lecture Notes in Computer Science*, pp. 120–129, Springer Berlin Heidelberg, 2011.

[96] S. Oster, I. Zorcic, F. Markert, and M. Lochau, "MoSo-PoLiTe: tool support for pairwise and model-based software product line testing," in *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, pp. 79–82, 2011.

[97] H. Lackner, M. Thomas, F. Wartenberg, and S. Weissleder, "Model-based test design of product lines: Raising test design to the product line level," in *Proceedings of the 7th IEEE International Conference on Software Testing, Verification and Validation*, pp. 51–60, 2014.

[98] B. Balle, *Learning Finite-State Machines - Statistical and Algorithmic Aspects*. Ph.D. thesis, Universitat Polit'ecnica de Catalunya, 2013.

[99] E. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[100] P. van de Laar, "Observations from the industry-as-laboratory research project darwin," in *Proceedings of the 8th Conference on Systems Engineering Research*, pp. 658–667, 2010.

[101] E. Brinksma and J. Hooman, "Dependability for high-tech systems: an industry-as-laboratory approach," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1226–1231, 2008.

[102] H. Sozer, C. Hofmann, B. Tekinerdogan, and M. Aksit, "Runtime verification of component-based embedded software," in *Proceedings of the 26th International Symposium on Computer and Information Sciences*, pp. 471–477, 2011.

[103] H. Sozer, B. Tekinerdogan, and M. Aksit, "FLORA: A framework for decomposing software architecture to introduce local recovery," *Software: Practice and Experience*, vol. 39, no. 10, pp. 869–889, 2009.

[104] P. Zoeteweij, R. Abreu, R. Golsteijn, and A. van Gemund, "Diagnosis of embedded software using program spectra," in *Proceedings of the 14th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems*, pp. 213–220, 2007.

[105] B. Tekinerdogan, H. Sozer, and M. Aksit, "Software architecture reliability analysis using failure scenarios," *Journal of Systems and Software*, vol. 81, no. 4, pp. 558–575, 2008.

[106] H. Sozer, B. Tekinerdogan, and M. Aksit, "Optimizing decomposition of software architecture for local recovery," *Software Quality Journal*, vol. 21, no. 2, pp. 203–240, 2013.

[107] C. Boogerd and L. Moonen, "Prioritizing software inspection results using static profiling," in *Proceedings of the 6th International Workshop on Source Code Analysis and Manipulation*, pp. 149–160, 2006.

[108] M. Eriksson, *Engineering Families of SoftwareIntensive Systems using Features, Goals and Scenarios*. Ph.D. thesis, Umea University, 2007.

[109] L. Damm, L. Lundberg, and C. Wohlin, "A model for software rework reduction through a combination of anomaly metrics," *Journal of Systems and Software*, vol. 81, no. 11, pp. 1968–1982, 2008.

[110] G. Muller, "Industry-as-laboratory applied in practice: The boderc project," 2012. http://www.gaudisite.nl/IndustryAsLaboratoryAppliedPaper.pdf. (accessed in 2015).

[111] T. Arias, *Execution architecture views for evolving software-intensive systems*. Ph.D. thesis, University of Groningen, 2011.

[112] M. Aksit, B. Tekinerdogan, H. Sozer, H. Safi, and M. Ayas, "The DESARC method: An effective approach for university-industry cooperation," in *Proceedings of the International Conference on Advances in Computing, Control and Networking*, pp. 51–53, 2015.

[113] J. Itkonen, M. V. Mntyl, and C. Lassenius, "Test better by exploring: Harnessing human skills and knowledge," *IEEE Software*, vol. 33, no. 4, pp. 90–96, 2016.

[114] A. Naseer and M. Zulfiqar, "Investigating exploratory testing in industrial practice : A case study," Master's thesis, Blekinge Institute of Technology, 2010.

[115] W. Afzal, A. Ghazi, J. Itkonen, R. Torkar, A. Andrews, and K. Bhatti, "An experiment on the effectiveness and efficiency of exploratory testing," *Empirical Software Engineering*, vol. 20, no. 3, pp. 844–878, 2015.

[116] H. Sozer and C. Gebizli, "Model-based testing of Digital TVs: An industry-as-laboratory approach," *Software Quality Journal*, pp. 1–18, 2016. published online, DOI: 10.1007/s11219-016-9321-y.

[117] I. de Visser, *Analyzing User Perceived Failure Severity in Consumer Electronics Products Incorporating the User Perspective into the Development Process*. PhD thesis, Eindhoven University of Technology, The Netherlands, 2008.

[118] S. Boslaugh and P. Watters, *Statistics in a Nutshell: A Desktop Quick Reference*. In a Nutshell (O'Reilly), O'Reilly Media, 2008.

[119] B. Winer, D. Brown, and K. Michels, *Statistical Principles in Experimental Design*. McGraw-Hill series in psychology, McGraw-Hill, 1991.

[120] N. Juristo, A. Moreno, and S. Vegas, "Reviewing 25 years of testing technique experiments," *Empirical Software Engineering*, vol. 9, no. 1-2, pp. 7–44, 2004.

[121] A. Beer and R. Ramler, "The role of experience in software testing practice," in *Proceedings of the 34th Euromicro Conference Software Engineering and Advanced Applications*, pp. 258–265, 2008.

[122] P. Poon, T. H. Tse, S. Tang, and F. Kuo, "Contributions of tester experience and a checklist guideline to the identification of categories and choices for software testing," *Software Quality Journal*, vol. 19, no. 1, pp. 141–163, 2011.

[123] J. Itkonen and M. Mäntylä, "Are test cases needed? replicated comparison between exploratory and test-case-based software testing," *Empirical Software Engineering*, vol. 19, no. 2, pp. 303–342, 2014.

[124] C. Schaefer and H. Do, "Model-based exploratory testing: A controlled experiment," in *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation Workshops*, pp. 284–293, 2014.

VITA

Ceren Şahin Gebizli received B.Sc. in Computer Engineering from Dokuz Eylul University, 2006-2010, M.Sc. in Computer Engineering from Ozyegin University, 2012-2014. Also, she is working as a Test Architect at Vestel Electronics R&D since 2010. She is responsible for analysis of requirements, designing tests with model-based testing and test automation of Digital TVs.

Acronyms

ARME Automated Refinement of Models for Model-Based Testing Using Exploratory Testing. 7, 8, 19–21, 27–29, 32, 36, 39–45, 90

DLNA Digital Live Networking Alliance. 17

DMR Digital Media Renderer. 17

DTV Digital TV. 1–5, 8, 14, 15, 23, 29, 31, 33, 40, 50, 51, 54–56, 58, 62, 64, 65, 67, 70, 74, 88, 89, 91, 94–98, 102, 104, 106, 107

DVB-TCI Digital Video Broadcasting - Terrestrial Channel Installation. 31, 32, 35–37, 39–41, 43, 44

EFG Event Flow Graph. 48

EPG Electronic Program Guide. 55

ESG Event Sequence Graph. 11, 12, 21

ET Exploratory Testing. 3–10, 14, 15, 19, 20, 23, 24, 30, 32, 35, 36, 39–49, 88, 89, 95, 98, 103–107, 110, 111, 116, 117

FORMAT Feature Oriented Model Adaptation Tool. 8, 9, 67, 68, 71, 73, 74, 76–79, 81–83, 85, 91

FSA Finite State Automata. 11, 12, 48

GUI Graphical User Interface. 48, 49

HBBTV Hybrid Broadcast Broadband TV. 17, 42, 55

LTS Labeled Transition Systems. 11, 12

MB Media Browser. 31–37, 41, 42, 44, 55, 96

MBT Model-based Testing. 2–8, 10–17, 20–22, 31, 32, 36, 40, 43, 44, 46–50, 54, 56–58, 62, 63, 65, 67, 68, 78, 85–89, 94–97, 100–102, 117

OVM Orthogonal Variability Model. 16, 87

PVR Personel Video Recorder. 55

RBT Risk-based Testing. 7–10, 15, 16, 66

RIMA Risk-based Model Adapter. 7, 50–54, 58–61, 63, 91

RTT Random Torture Test. 31–33, 35, 37, 38, 42–44

SP Smart Phone. 54–56, 60–65, 89

SPLE Software Product Line Engineering. 5, 7–10, 16, 17, 67, 68, 85–87, 89, 98

SUT System Under Test. 10, 20, 49

UML Unified Modeling Language. 11, 86

VesTA Vestel Test Automation. 3, 31, 72, 96, 97