RISK-DRIVEN MODEL-BASED TESTING

A Thesis

by

Abdulhadi Kırkıcı

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

Master of Science

in the
Department of Computer Science

Özyeğin University
May 2018

RISK-DRIVEN MODEL-BASED TESTING

Approved by:

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

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

Assoc. Prof. Mehmet Aktaş
Department of Computer Engineering
Yıldız Technical University

Date Approved: 2018

To my parents, Aliye - Cengiz KIRKICI, who have always loved me unconditionally and whose good examples have taught me to work hard for the things that I aspire to achieve.

This thesis work is also dedicated to my brother, Gökhan, who has been a constant source of support and encouragement during the challenges of graduate school and life. I am truly thankful for having them in my life.

ABSTRACT

Software is becoming larger and more complex in consumer electronics products. As a result, testing these products for reliability is becoming a major challenge. Traditional and manual testing activities are not effective and efficient in pinpointing faults. Consequently, manual testing activities are being replaced with automated techniques. Model-based testing is one of these techniques. It uses test models as input and automates test case generation. However, these models are very large for industry-scale systems. Hence, the number of generated test cases can be very large as well. However, it is not feasible to test every functionality of the system exhaustively due to extremely limited resources in the consumer electronics domain. Only those system usage scenarios that are associated with a high likelihood of failures should be tested. Therefore, we propose a risk-driven model-based testing approach in this thesis. Hereby, test models are augmented with information regarding failure risk. Markov chains are used for expressing these models, which are basically composed of states and transitions. Each state transition is annotated with a probability. Probability values are used for generating test cases that cover transitions with the highest probability values. The proposed approach updates transition probability values based on three types of analysis for risk estimation. First, usage profile is used for determining the mostly used features of the system. Second, static analysis is used for estimating fault potential at each state. Third, dynamic analysis is used for estimating error likelihood at each state. Test models are updated based on these analyses and estimations iteratively. The approach is evaluated with three industrial case studies for testing digital TVs, smart phones and washing machines. Results show that the approach increases test efficiency by revealing more faults in less testing time.

ÖZETÇE

Tüketici elektroniği ürünlerinde bulunan yazılımın hem boyutları, hem de karmaşıklığı artmaktadır. Bu eğilim, ürünlerin test edilmesi ve güvenilirliklerinden emin olunması için zorluk teşkil etmektedir. Geleneksel ve manuel test süreçleri, kritik hatalarının tespit edilmesinde yetersiz ve verimsiz kalmaktadır. Bu süreçleri iyileştirmek ve otomatik bir hale getirmek için çeşitli teknikler kullanılagelmiştir. Test modellerinden yararlanarak otomatik test adımlarının oluşturulmasını sağlayan model bazlı test, bu tekniklerden birisidir. Endüstriyel ölçekteki sistemlere ilişkin çok büyük olan test modellerinden, prensip olarak sonsuz sayıda test adımı üretilebilmek mümkündür. Ancak tüketici elektroniği alanında oldukça kısıtlı olan kaynaklar sebebi ile sistemin tüm fonksiyonlarını test etmek mümkün değildir. Bu sebepten dolayı, bu tezde risk odaklı model bazlı test yaklaşımı önerilmektedir. Bu yaklaşımda, test modelleri hata riskine ilişkin bilgilerle zenginleştirilmektedir. Test modelleri, durum geçişlerinin olasılık değerleri ile etiketlendiği Markov zincirleri şeklinde tanımlanmaktadır. Bu değerler, yüksek olasılıklı senaryoları kapsayacak şekilde test adımlarının otomatik üretim sürecini yönlendirmektedir. Yaklaşımımızda, durum geçişlerine ilişkin olasılık değerleri, üç farklı analiz tipi ile hesaplanan risk tahminlerine göre güncellenmektedir. İlk olarak, ürün kullanıcılarından toplanan kullanım profili verileri analiz edilmektedir. İkinci olarak, sistemin farklı durumlarındaki hata potansiyelinin tahmini için statik analiz kullanılmaktadır. Üçüncü olarak ise, hata tahmini için dinamik analiz kullanılmaktadır. Test modelleri, bu analizler ve tahminlere göre yinelemeli olarak güncellenmektedir. Yaklaşımımızı değerlendirmek için dijital televizyonlar, akıllı telefonlar ve çamaşır makineleri olmak üzere, üç farklı ürün üzerinde endüstriyel vaka çalışmaları yapılmıştır. Elde edilen sonuçlar, yaklaşımın kısa sürede daha fazla hata tespit edilmesini sağlayarak test verimliliğini arttırdığını göstermektedir.

ACKNOWLEDGMENTS

I would like to thank the committee members for their time and effort to evaluate my thesis. I would also like to thank my colleauges at Vestel Electronics. They helped me to conduct case studies that are reported in this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Systems are used to be mainly electromechanical in consumer electronics domain. However, they are currently manufactured as software-intensive systems. Software is becoming larger and more complex in consumer electronics products. Almost all the functionality is implemented in software, which is deployed on standardized hardware. Moreover, the number and variety of the provided functionalities are increasing. For instance, digital TV systems today (also named as smart TV systems) support networking with other home appliances, streaming content from the Internet and other new functionalities in addition to those previously provided by traditional TV systems [2]. As a result, a typical digital TV system comprise tens of millions of lines of code [3].

The size and complexity of this code base are continuously increasing. One the one hand, the reliability of software has become the main designator of overall system reliability. Therefore, software reliability should be high for targeting at a market with high competition. On the other hand, resources are extremely limited in consumer electronics domain due to short time-to-market. It is not feasible anymore to test systems exhaustively. Hence, software testing techniques must be efficient to be able to reveal and fix the most critical faults[1] with limited resources. These faults might not be critical in the sense that they have catastrophic consequences like those taking place in safety-critical systems. However, they are considered to be critical in consumer electronics domain if they lead to a user-perceived failure [5]. Such faults must be detected and removed for preventing user irritation, protecting the brand image and as such, survive through the tough competition.

[1]We adopt the terminology introduced by Avizienis et al. [4] in this thesis. We use the term *fault* (also named as bug or defect) as the cause of an error. We consider *error* as a system state that can lead to failure. The term *failure* is used for a user-observed event. This event can be an unexpected system behavior/output.

Automating test activities [6] is a typical approach preferred for increasing test efficiency. These activities can be related to test case generation, test execution or test oracle [7] implementation. Various techniques can be employed for automating one or more of these activities. One of these techniques is Model-Based Testing (MBT) [8, 9, 10], which is now a state-of-the-practice technique for automating test case generation. Hereby, test cases are generated by systematically exploring various executions paths on a test model, which defines the usage behavior of the system. This reduces the test case creation time. The input test model has to be created manually; however, there have been various techniques proposed to generate [11], extend [12] or refine [13] these models automatically. MBT also facilitates an increased and measured coverage defined based on the test model; however, it turns out to be infeasible to achieve an extensive coverage of the model in practice. For example, a typical test model regarding smart TV systems can comprise thousands of states and transitions [14]. Therefore, the test case generation process must be efficient to explore only those execution paths in the test model that are associated with high risk of failure.

In this thesis, we propose a risk-driven model-based testing approach, where test models are augmented with information regarding failure risk. The approach is automated with a tool called RIMA[2] (Risk Based Model Adapter), which iteratively updates test models to employ random-stochastic test case generation based on risk of failure. Markov chains are used for expressing these models, which are basically composed of states and transitions. Each state transition is annotated with a probability. Probability values are used for generating test cases that cover transitions with the highest probability values. RIMA updates transition probability values based on three types of analysis for risk estimation. The *RIPR model* [15] suggests that for a fault to be observed as a failure, the faulty location must be **R**eached, its execution must **I**nfect the program state, which **P**ropagates and gets **R**evealed. Hence, the first analysis is applied on usage profile for determining the mostly used features

[2]Available online at `https://github.com/csgebizli/RIMA`

of the system. Second, static analysis is used for estimating fault potential at each state. Third, dynamic analysis is used for estimating error likelihood at each state. RIMA updates test models based on these analyses and estimations iteratively. Test cases are generated and executed after each update.

We performed three industrial case studies to evaluate our approach for testing a Digital TV (DTV), a smart phone (SP) and a washing machine (WM). In all of these case studies, we were able to detect several new faults after each update of the test models performed by RIMA. Some of these faults were missed during the regular MBT process and they lead to critical failures such as crashes, system restarts or lack of response for user input events. We also compared the test efficiency achieved with our approach to that of random testing, where the same test model is used by MBT without any updates in successive test iterations. We measure test efficiency as the number of detected faults per unit of time. Hence, our metric can be improved in two ways; the same set of faults can be detected in less testing time, or additional/new faults can be discovered within the same test duration or less. Test efficiency of DTV tests is calculated as *1.09 faults/hrs* when our approach is used. It was measured as *0.56 faults/hrs*, when MBT is applied without the integration of RIMA. Results for the SP case were *0.38 faults/hrs* and *0.26 faults/hrs* obtained with and without RIMA, respectively. Likewise, results for the WM case were *0.052 faults/hrs* and *0.039 faults/hrs* obtained with and without RIMA, respectively. These results show that our approach is effective in detecting critical faults and it increases the efficiency of the MBT process.

We explain the organization of the remainder of the thesis before concluding this chapter. In the following chapter, we provide a brief background on MBT and risk-based testing. In Chapter 3, we introduce our risk-based MBT approach. In Chapter 4, we present the 3 industrial case studies conducted on DTV, SP and WM systems. We report and elaborate on the obtained results in this chapter as well. In Chapter 5, we summarize related work and position our study with respect to the literature. Finally, in Chapter 6, we conclude the thesis with a discussion of future work.

CHAPTER II

BACKGROUND

In this chapter, we provide a brief background regarding model-based testing and risk-driven testing. The chapter is divided into two sections. First, model-based testing is described in the following section. Then, we introduce risk-driven testing in the second section.

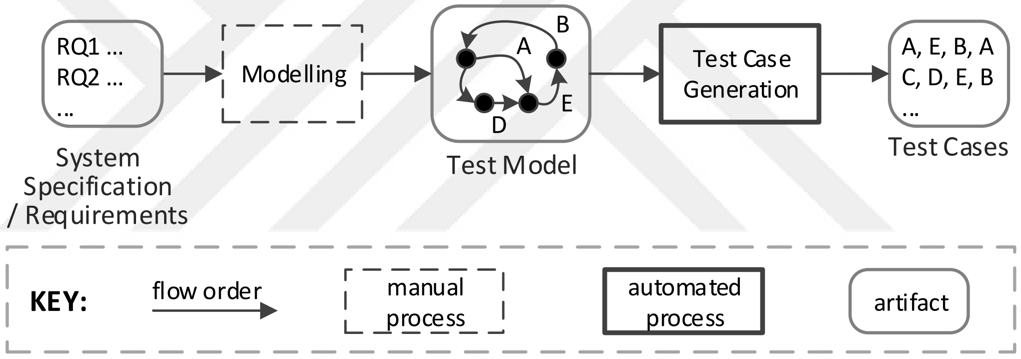

Figure 1: The overall model-based testing process.

2.1 Model-Based Testing

Model-based testing (MBT) [8, 10] is an automated test case generation technique that relies on test models. The overall MBT process is composed of two main steps as shown in Figure 1. First, a test model is created. This model represents the external behavior of the system under test. Hence, MBT can be considered as a black-box testing approach. Modeling activity is usually a manual process where informal system specifications and user requirements are interpreted to create a formal test model. Second, the created test model is used for generating test cases. The second activity is automated by a tool, which takes the formal test model as test input and explores this model systematically. The model can be specified in various formalisms including finite state automata [16], event sequence graphs [17], event flow graphs [18], UML state charts [19] or Markov chains [20, 21].

MBT has been proven to be valuable for increasing the testing effectiveness and efficiency [22]. It has also been applied in the industry for more than a decade. There are 3 main benefits of this approach [23]. First, it enables automated generation of test cases. Second, it reduces maintenance costs. Test models can be updated based on changes in requirements and test cases can be re-generated after these updates. Otherwise, one needs to update dozens of test cases. Third, test models serve as documentation and reveals specification mistakes early in the process. Internal inconsistencies in system specifications and user requirements can be detected while creating test models.

MBT has also limitations. Previously, 3 main limitations have been identified regarding the application of MBT in consumer electronics domain [23]. First, test models can be incomplete since model creation is a manual activity and the system specifications or user requirements are usually incomplete and imprecise. As a result, the generated test cases can fall short to detect all the critical faults. Second, test models can be subject to high variation for large product families. Therefore, systematic management of this variability is necessary to be able to reuse test models for all the products in the family. Third, test models can be very large such that it might not be possible to exhaustively cover them due to limited resources. Hence, test case generation should focus on execution paths that are associated with the highest risk for exposing failures to users. The first two problems have been addressed before [23]. In this thesis, we focus on the third problem, which was also addressed with a risk-driven model based testing approach [14, 23]. In that approach, test models are specified in the form of Markov chains. Therefore, they represent discrete-time stochastic systems [20] with states and transitions among them. The systems transitions to another state depending on the annotated transition probability values. MaTeLo[1] is used as

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

the MBT tool that generates test cases using these models. Hereby, transition probability values steer the test case generation process. These values are updated based on risk of failure estimated for the corresponding transition.

In this thesis, we build on top of the risk-driven model based testing approach [14, 23] by automating its activities and generalizing the risk estimation process. We observed that the risk of failure must be defined in different terms for various systems. For example, fluctuations in power consumption is a relevant risk indicator for washing machines. However, fluctuations in memory profile is a more relevant indicator for digital TV systems. In our unified approach, various types of risks can be defined and estimated based on either static or dynamic analysis of software artifacts.

2.2 Risk-Driven Testing

Risk-based testing aims at optimizing test efforts and mitigate risks by steering testing activities according to risk assessments [1]. Hereby, a risk is defined as a factor that may lead to negative consequences [24]; it is usually expressed with likelihood and impact attributes. These attributes are used for guiding decisions throughout a testing process [25] including phases such as test planning, design, implementation and evaluation. To be able to make use of limited resources in an efficient manner, risk-based testing focuses activities on scenarios that are more liable to critical situations [26].

Risk-based testing has become highly important and relevant since the size and complexity of software are continuously increasing whereas software testing has to be performed with limited resources. Therefore, there have been many risk-based testing approaches proposed [1], some of which were adopted in the industry [27]. These approaches have been classified with a taxonomy [1] as depicted in Figure 2. It considers the whole testing process and defines 3 top-level classes. These are risk drivers, risk assessment and risk-based test process. In the following, we position our approach with respect to this taxonomy.

In our study, we are concerned with functionality as the main risk driver. Safety and security are not major concerns in the consumer electronics domain in general. In our approach, we adopt MBT, where risk exposures are reflected to elements of test models. Therefore, risk item type can be considered as a functional artifact since we perform system level black-box testing. Risk factors are quantified from two perspectives: risk exposure and likelihood. We quantify risk exposure based on a user profile to determine if a user could make use of a feature of a system. We quantify fault and error likelihood in each feature based on static and dynamic analysis performed on the source code and the running system, respectively. Hence, our analyses are based on formal models and our risk estimation scale is quantitative. We do not consider impact rating as a risk assessment factor since it is hard to estimate this beforehand. However, we take criticality of failures into account, while evaluating our approach.

In our approach, we focus on test case generation rather than test execution and evaluations. Therefore, the corresponding categories under risk-based test process are not relevant in our case. Risk-based test planning can be considered to be relevant because our approach involves test prioritization and selection. Instead of exploring a test model exhaustively, we focus on parts of the model that are associated with high risk. Therefore, fault detection capability and quality of testing is increased, while the number of test cases is decreased. In terms of risk-based test design, we focus on functional testing as the test type and we adopt a model-based coverage criteria, which falls into the asset category. Finally, our risk-based test implementation involves test automation not only for test case generation, but also for executing the generated test cases. Test case generation is automated with an MBT tool, whereas test execution is automated with an in-house developed tool. The first tool is bought and the second tool is developed by the company, where we conducted our case studies.

In the following chapter, we introduce our approach. Then, we explain case studies and discuss the obtained results.

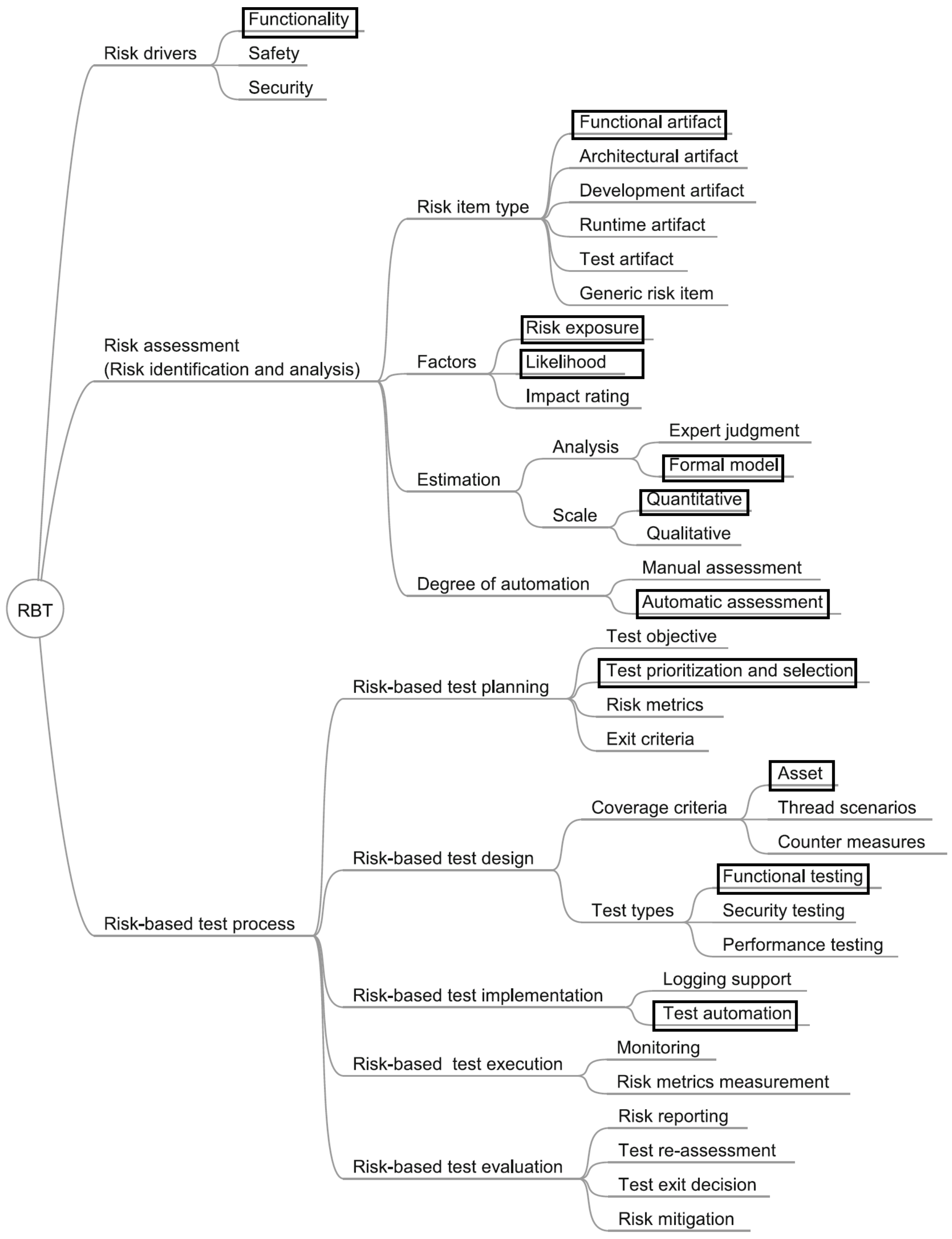

Figure 2: A taxonomy of risk-based testing [1] and the scope of this thesis.

CHAPTER III

RISK-DRIVEN MODEL-BASED TESTING APPROACH

In this chapter, we introduce our risk-driven model-based testing approach. An earlier version of this approach was proposed before [14, 23] to steer the model-based test case generation process towards those execution paths in the test model that are associated with the highest risk of failure as can be observed by the end users. That approach was later applied on various subject systems as part of a series of industrial case studies. We observed in these stuides that the risk of failure must be defined in different terms for various systems. Therefore, we refined the approach based on our observations. In this thesis, we propose a unified approach that can be applicable to various systems. Essentially, each application turns out to be an instance of the unified approach. Various types of risks can be defined and estimated in each instance application.

In our study, we aim at optimizing the test process based on the risk of user-perceived failure. Such a failure occurs if the conditions hold [15]:

- there exist a fault in the system
- the faulty program statement is executed such that the fault is triggered, i.e., the associated system feature is utilized by an end user
- the triggered fault leads to an error
- the error propagates to a failure that is observed by the user

The likelihood (risk) of the first condition can be estimated with static analysis. The second condition depends on the usage profile of the system. The last two conditions are related to the runtime behavior and as such their likelihood can be estimated with dynamic analysis. Therefore, we employ the analysis of 3 types of profile in our approach: usage

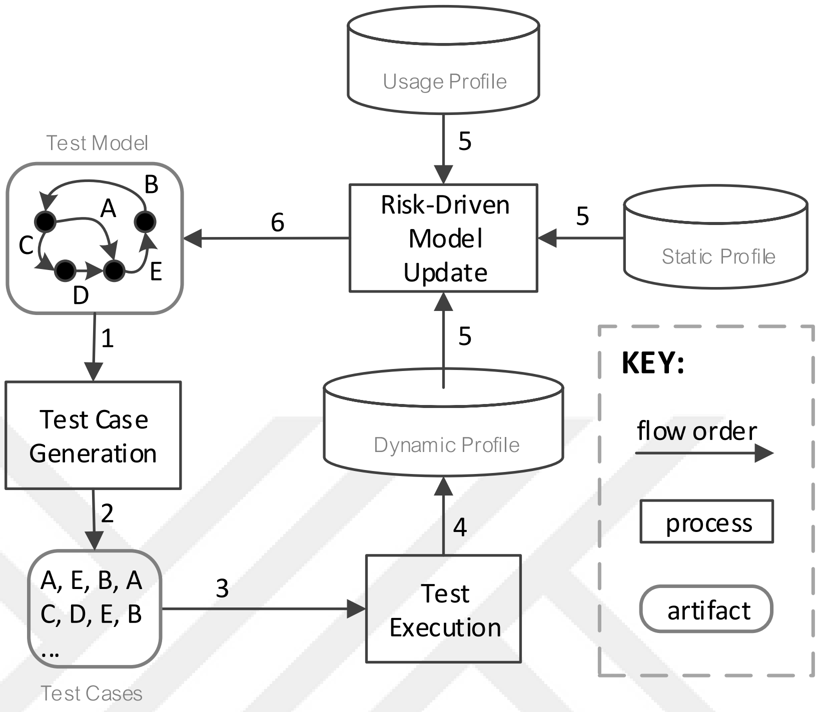

Figure 3: The unified risk-driven model-based testing approach.

profile, static profile and dynamic profile. The number of and the contents of these profiles depend on the risk definitions and the way that risks are estimated.

The unified risk-driven model-based testing approach is depicted in Figure 3. There are 6 steps depicted. In fact, the process is iterative. The number of iterations depend on the system under test and the considered types of risks. In our case studies, we implemented the approach in 3 iterations. That is, the cycle shown in Figure 3 with 6 steps is repeated 3 times. It is assumed that a test model regarding the system under test is available. In our approach this model is represented in the form of Markov chains so that risk estimations can be reflected to the model in terms of state transition probability values. First, the test model is provided to the MBT tool. This tool performs the test case generation process. This process focuses on covering the most probable paths on the model according to transition probabilities. A set of test cases is obtained as the output of this process. These test cases are executed on the system. The system is monitored during test execution so that a dynamic profile can be collected. The next step of the approach is to feed a collected profile to RIMA, a tool that we developed to perform the risk-driven model update process. The test model is updated as a result of this process. The updated test model is used in the next iteration. A different profile (of type usage, static or dynamic) and a different risk estimation procedure is utilized in each iteration.

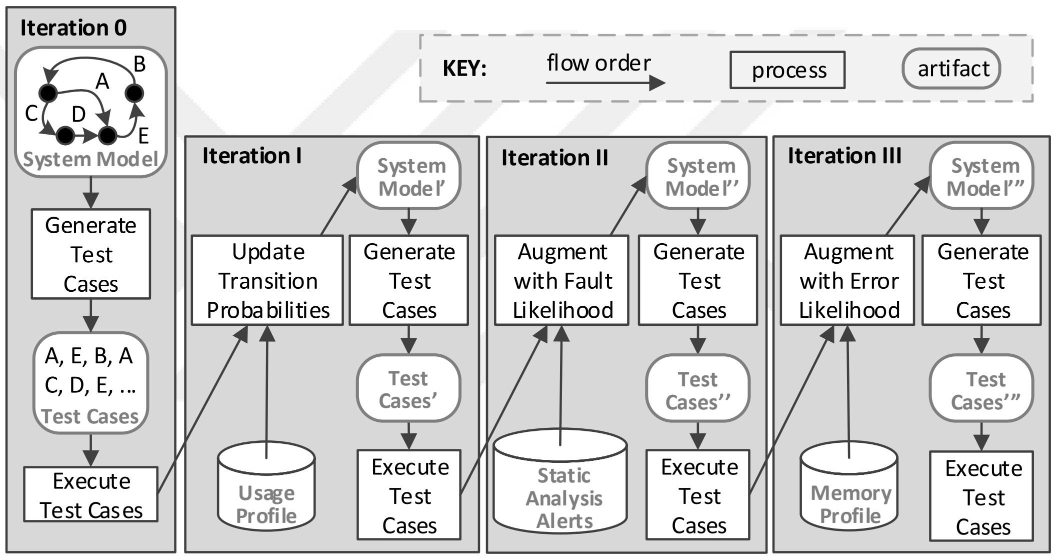

Figure 4: An instance of our approach adopted for DTV and SP systems.

Figure 4 shows an instance of our approach as applied for Digital TV (DTV) and Smart Phone (SP) systems. Hereby, the 3 iteration cycles are unrolled to depict the flow as a pipeline of processes and artifacts. We start with the existing test model created for the system under test. In *Iteration 0*, all the probability values for transitions exiting from a state are equal to each other. An MBT tool is used for generating test cases based on this model. These test cases are executed on the system. We collect data regarding the usage of memory while test cases are being executed. RIMA updates the test model based on various information sources in the succeeding 3 iterations. Each iteration involves a model update step, a test case generation step and finally the execution of test cases. The difference among them is the model update process performed by RIMA. Usage profile is used in *Iteration I* for assigning probability values to state transitions such that these values reflect the probability that an end user would make the system perform the same transitions. As a result, test cases that are generated and executed in *Iteration I* should focus on mostly visited execution paths in the field. A list of static analysis alerts are used in *Iteration II*. A static code analysis tool is executed on the system source code to obtain such a list. The list is used for estimating the relative risk of faults that can take place in various software modules. These modules actually implement various features of the system. Therefore, they are mapped to a set of states of the test model. *Iteration III* exploits the memory usage profile for risk estimation. This profile is collected during the test executions in the previous iterations. It is used for calculating the amount of memory leaks that take place during transitions among the system states. The calculated amount is interpreted as the relative risk of error for the corresponding state transition.

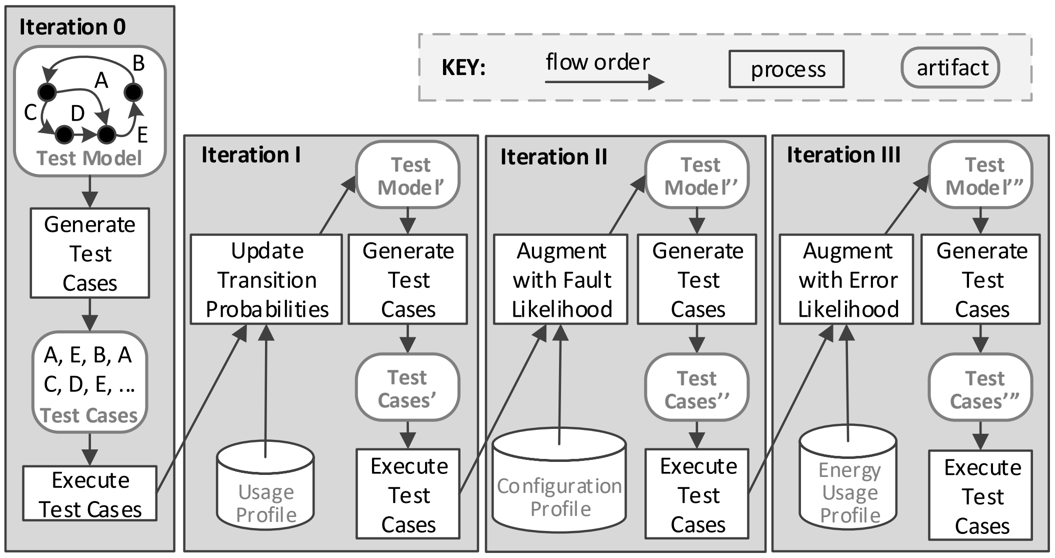

Figure 5: An instance of our approach adopted for WM software.

Figure 5 shows another application of the approach for Washing Machine (WM) software this time. The overall process is very similar to the one that is shown in Figure 4 as applied for DTV and SP systems. However, the risk estimation procedures are different in *Iteration II* and *Iteration III*. In Figure 4, we utilize static code analysis alerts to estimate the risk of fault. However, for this application, our discussion with the domain experts revealed that the code base used for WM systems is stable for the most part and it is not subject to many alerts. On the other hand, the configuration space of software is very large and the most of the faults are introduced due to wrong/conflicting settings of configuration options. Hence, we defined and calculated the risk of fault at a state as the ratio of configuration options associated with that state. Then, the model is updated according to this risk. In *Iteration III*, we perform an update based on dynamic analysis. In Figure 4, we utilize memory profile to estimate the risk of error. However, for this implementation, our discussion with the domain experts revealed that memory usage for WM systems is usually stable. On the other hand, fluctuations in power consumption are more relevant indicators of an error. Hence, we defined and calculated the risk of error at a state to be proportional to the amount of power consumption associated with that state. Then, the model is updated accordingly.

The following sections give detailed information on model updates perfomed in each iteration.

3.1 Risk Estimation based on Usage Profile

The first risk estimation and model update process is performed based on usage profile in *Iteration I*. Beforehand, in *Iteration 0*, probability values for transitions outgoing from a state are equal and they add up to 1. Basically, if there are n such outgoing transitions, then each of these transitions would have $1/n$ probability. A test model represents the usage behavior of the system. If some of the features of the system are never used, then users would not be exposed to any failure even if there exist many faults in these features.

Likewise, the risk of failure is directly proportional to the probability that the corresponding state will be visited in the test model. Risk estimation based on usage profile aims at capturing this information. Data collection procedures for obtaining a usage profile are explained in the following chapter in detail. In this section, we discuss the risk calculation and model update procedure based on the collected profile.

Algorithm 1 Model update procedure based on usage profile.

```
1:  Inputs:
2:  M(S,T) is the test model,
3:  $V[1...|S|]$ is the number of visits for each state $s \in S$
4:  procedure USAGE PROFILE BASED TEST MODEL UPDATE
5:      foreach state $s \in S$ do
6:          total-visits $\leftarrow$ 0
7:          foreach transition $t \in T$ do
8:              if t.src= s then
9:                  total-visits $\leftarrow$ total-visits + $V[t.dest]$
10:         end foreach
11:         foreach transition $t \in T$ do
12:             if t.src= s then
13:                 t.p $\leftarrow$ $V[t.dest]$ / total visits
14:         end foreach
15:     end foreach
```

The update procedure is listed in Algorithm 1. It takes two inputs. The first one is the test model M with a set of states S and a set of transitions T. Each transition $t \in T$ is defined as a 3-tuple *(src, dest, p)*, where *src* defines the source state, *dest* defines the target state, and p defines the probability associated with the transition t. The second input, V is a list of size $|S|$. This list is derived from the collected usage profile and it contains the visit counts regarding each state $s \in S$. The procedure iterates over all the states in S (Lines 5-15). In the first part of this iteration (Lines 6-10), it calculates the total number of visits for states that can be directly reached from each state $s \in S$. The variable *total-visits* saves the result, which is initialized as 0 (Line 6). Then, all the transitions are checked if they are originating from s (Lines 7-8). The number of visits for the target state of each such transition is added to the total number of visits for s (Line 9). In the second part of the iteration (Lines 11-14), the procedure updates transition probability values. All the transitions are checked if they are originating from s (Lines 11-12). The probability value for each such transition is updated as the number of visits for its target state ($V[t.dest]$) divided by the total number of visits calculated for its source state s (Line 13).

3.2 Risk Estimation based on Static Profile

The second risk estimation and model update process is performed based on static profile in *Iteration II*. The data collection procedures and the resulting contents for this profile may differ among systems under test. In our case studies, we utilized two types of static profiles. The first type of profile, which includes a list of static code analysis alerts, is used for estimating fault/error likelihood for DTV and SP systems. Hereby, we associate modules in the software code base to features of the system. These features are represented as states in the test model. Hence, each state is associated with a subset of the reported alerts. We calculate the ratio of this subset for every state. Note that each state of the test model represents a feature of the system, which is implemented by a set of source code modules. We consider this ratio as the relative risk of failure for the state. A failure might be observed if the reported alerts are not false positives and if they are triggered by a visit to the state.

The second type of profile, which includes the number of configuration options, is used for estimating fault/error likelihood for WM systems. Hereby, for each state, we calculate the ratio of the number of configuration options. We consider this ratio as the relative risk of failure for the state. The assumption is that a failure is more likely for those states being subject to a high number of configuration options. Regardless of the utilized profile type, a fault/error likelihood is calculated as the risk factor. RIMA updates the test model according to this calculation as follows.

The update procedure is listed in Algorithm 2. It takes two inputs. The first one is the test model M as defined and used in Algorithm 1. The second input, R is a list of size $|S|$, which keeps the amount of risk estimated for each state $s \in S$. In this iteration, the amount of risk for each state s is determined based on the static profile. This profile includes the

Algorithm 2 Model update procedure based on fault/error likelihood.

```
1: Input:
2: M(S,T) is the test model,
3: $R[1...|S|]$ is the amount of risk estimated for each state $s \in S$
4: procedure ERROR RISK ESTIMATION BASED TEST MODEL UPDATE
5:     e = new state
6:     $S \leftarrow S + e$
7:     $total\text{-}risk \leftarrow 0$
8:     foreach state $s \in S$ do
9:         $total\text{-}risk \leftarrow total\text{-}risk + R[s]$
10:    end foreach
11:    foreach state $s \in S$ do
12:        te = new transition
13:        te.src = s
14:        te.dest = e
15:        te.p = R[s] / total-risk
16:        foreach transition $t \in T$ do
17:            if $t.src = s$ then
18:                $t.p \leftarrow t.p \times (1 - te.p)$
19:        end foreach
20:        $T \leftarrow T + te$
21:    end foreach
```

number of static code analysis alerts that are associated with *s* for DTV and SP systems in our case studies. On the other hand, it includes the number of configuration options associated with *s* for the WM system. The procedure adds a new state (error state, *e*) to the model (Lines 5-6). Then, it calculates the total amount of risk by summing up all the values in *R* (Lines 7-10). The procedure iterates over each state $s \in S$ (Lines 11-21) after these initialization steps. It introduces a new transition, te from *s* to *e* (Lines 12-14). The probability value for this transition, $te.p$ is equal to the risk associated with *s* divided by the total risk (Line 15). All the existing transitions are checked if they are originating from *s* (Lines 16-17). The probability value for each such transition is multiplied by $(1 - te.p)$ (Line 18). Finally, the new transition te is added to the model (Line 20).

3.3 Risk Estimation based on Dynamic Profile

The third and the last risk estimation and model update process is performed based on dynamic profile in *Iteration III*. The data collection procedures and the resulting contents for this profile may differ among systems under test. In our case studies, we utilized two types of dynamic profiles. The first type of profile is the memory profile, which is used for estimating fault/error likelihood for DTV and SP systems. The required data is recorded, while test cases are being executed as part of previous iterations. A tool was developed [28] for this purpose. This tool measures memory leakage for each feature in the system, and as such, for each state of the test model. Then, a risk value is calculated for each state. This calculation reflects the relative amount of memory leakage associated with the state. The test model is updated with the RIMA tool using the same update procedure listed in Algorithm 2.

The second type of profile, which includes the amount of power consumption, is used for estimating fault/error likelihood for WM systems. We used an in-house developed tool to measure the amount of power consumption. Then, for each state, we calculate a risk, which is proportional to the amount of power consumed during the visit of that state. RIMA and the employed update procedure (See Algorithm 2) is agnostic to the type of profile and the way that a risk is estimated.

In the following section, we introduce 3 case studies to evaluate the approach.

CHAPTER IV

INDUSTRIAL CASE STUDIES

We conducted 3 case studies to evaluate our approach. In this chapter, we present these studies. We used 3 subject systems, DTV, SP and WM, all of which are being developed and maintained by Vestel[1]. Vestel is a large-scale consumer electronics company in Turkey. It is also one of the largest in Europe.

A risk-driven testing approach is especially relevant in the consumer electronics domain, where resources are extremely limited and the market is very competitive. The brand image is directly affected by user-perceived failures [29]. Hence, although it is not feasible to target at preventing any failure as it would be the case for safety-critical systems, the testing process must be optimized to be able to prevent those failures that can be exposed to end users. It was reported by the domain experts in Vestel that approximately %35 of the overall product development is attributed to testing efforts. Hence, optimizing the testing process has a promising potential for reducing the overall costs, while capturing critical failures that can be observed by end users. In this chapter, our goal is to evaluate our risk-driven MBT approach towards this goal.

In the following section, we introduce and motivate our research questions. Then, we introduce DTV, SP and WM subject systems that are used in case studies. This is followed by detailed explanations regarding the employed data collection and analysis procedures. Then, results are presented and discussed. Finally, we discuss validity threats for our case studies.

[1]http://www.vestel.com.tr

4.1 Research Questions

In this section, we define our research questions that are aligned with the goals of our approach and constraints imposed by the research context. Our overall goal is to increase the efficiency of the MBT process. This goal can be decomposed into two sub-goals. First, our approach can improve the MBT process such that new and critical faults can be detected. These are the faults that are missed with the regular MBT process and that lead to failures observed by end users. Second, the number of test steps and as such the overall testing time can be reduced. That is, the same set faults can be detected in less time, with less resources. We update test models based on 3 types of profiles. Therefore, our research questions are defined to reveal if/how each of these updates contribute to our goals and sub-goals. We defined 4 research questions as follows:

> ***RQ1:*** How effective it is to update test models based on usage profile to detect new faults and/or reduce the the number of test steps and testing time?
>
> ***RQ2:*** How effective it is to update test models based on static analysis to detect new faults and/or reduce the the number of test steps and testing time?
>
> ***RQ3:*** How effective it is to update test models based on dynamic analysis to detect new faults and/or reduce the the number of test steps and testing time?
>
> ***RQ4:*** Does RIMA improve test efficiency with respect to random testing?

The first 3 research questions above are defined to evaluate the result of the MBT process when test models are updated based on 3 types of profile data. These updates are also aligned with the successive iterations of our approach. Results are evaluated from two perspectives: *i)* the number of new faults detected as exposed to end users, and *ii)* the number of test steps and the testing time.

The last research question is defined to compare our approach with respect to random testing as the baseline approach. We observed in the state-of-the-practice that test models are mainly used for documenting the overall usage behavior. Transition probability values are mostly fixed and they are not subject to any updates before successive test iterations. We have seen updates made by practitioners based on the collected usage profile. However, this adjustment only covers the first iteration of our approach and its was being performed manually. Hence, we defined *RQ4* to investigate how the test efficiency of MBT is without RIMA. We measured test efficiency as the number of distinct faults found divided by the total test duration.

In the following, we introduce the subject systems used in our case studies. Then, we explain our data collection and analysis procedures. These explanations are followed by the discussion of results and threats to validity.

4.2 Subject Systems

We used 3 subject systems in our case studies: DTV, SP and WM. In this section, we briefly explain their features and introduce top-level test models that are created for these systems. These models were previously developed by the software test group in the company for MBT. We used them directly as input for our case studies.

Figure 6, Figure 7 and Figure 8 depict the test models that are used for DTV, SP and WM, respectively. The models are defined in the forms of a Markov chain with the MaTeLo[2] tool. They constitute a hierarchical structure, in which states can further comprise sub-models. In our case study, we used RIMA for updating the the top-level test models only. States of this model can be easily mapped to a particular system feature or a set of software modules.

We see the initial test model for DTV in Figure 6. This figure represents DTV features such as Media Browser (Audio, Picture, Video), Portal, Youtube, EPG, HDMI-SCART (Source Switch), Record (PVR), HBBTV, Channel List and Teletext.

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

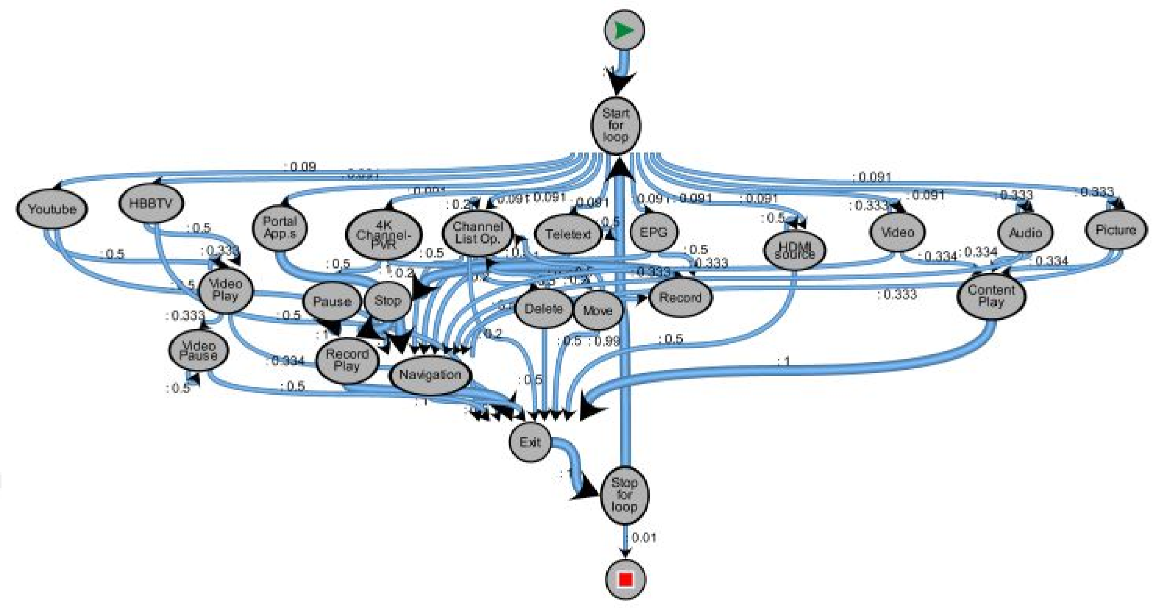

Figure 6: The first version of the test model developed for DTV.

The Figure 7 demonstrates main features in the SP system. In our case study, we focused on those applications that are developed by Vestel only. We see these features are VCloud, VCam, Photos, Alarm, VMarket, Contacts, Phone Call, Themes, and SMS-MMS

The Figure 8 is the first test model developed for the WM system. In our case study, we focused on those applications that are developed by Vestel only. The list of main features for WM include Cotton, Easycare, Wool30, Rinse, Spin, Handwash, Sportwear, Mix30, Duvet, Blouses, Daily60, Rapid15, and Babycare.

We explain the data collection and analysis procedures in the following.

4.3 Data Collection

In addition to test models, we collected additional data regarding the subject systems to be able to perform the 3 model refinement steps of the approach. In particular, we collected a usage profile, a static profile (based on static code analysis) and a dynamic profile (by measuring memory usage and power consumption at runtime) for each subject system. In the following, we explain how the data collection is performed.

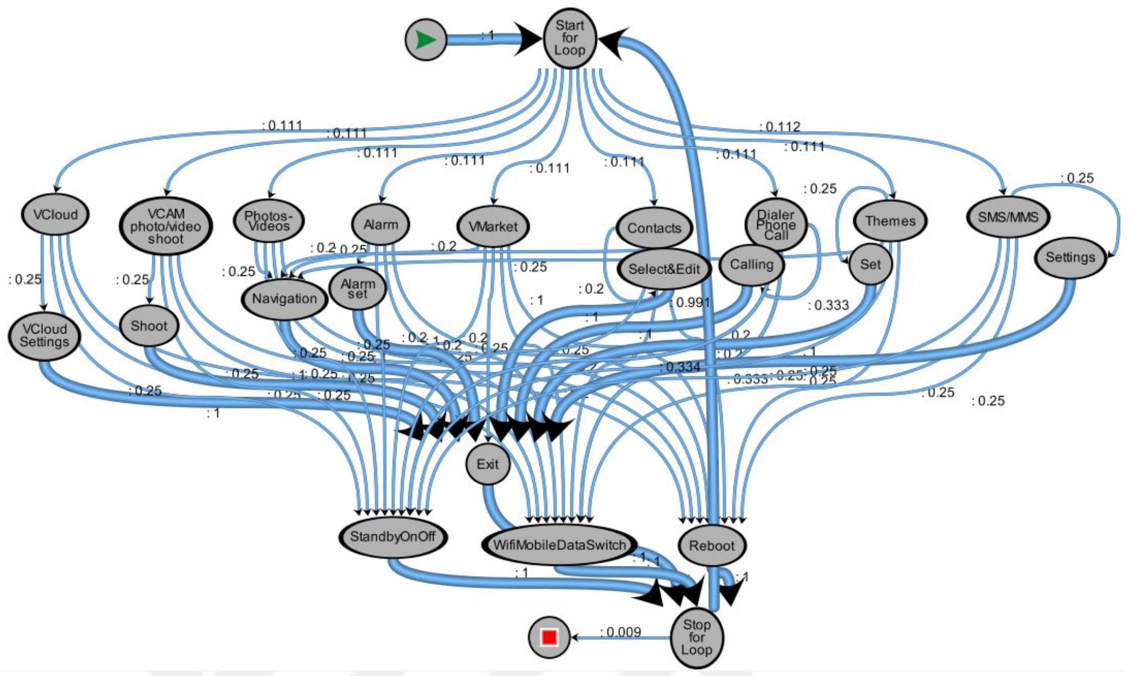

Figure 7: The first version of the test model developed for SP.

- ***Usage Profile***

The first model update is performed based on usage profile. Usage profile is collected as part of the regular alpha-test procedure of the company that we did not interfere. Hereby, field testers are employed before a project is introduced to the production line for manufacturing and then to the market. These testers are actually the employees of the company as well; however, they are selected from various departments such as Sales, Quality Assurance, Manufacturing and Financing. The selection process starts with an announcement made to all the employees in the company, informing them about the product, and data to be collected. Then, applications are collected for 1 week. Normally, 30 to 50 people among the applicants are selected as field testers. The selection is performed to maximize the diversity among the testers in terms of the following properties: *i)* age, *ii)* department, *iii)* peripheral devices being used at home, and *iv)* type/availability of network connection at home. The selected field testers are asked to use the given products for 15 to 30 days. The testing time depends on the product and its features. An application continuously runs

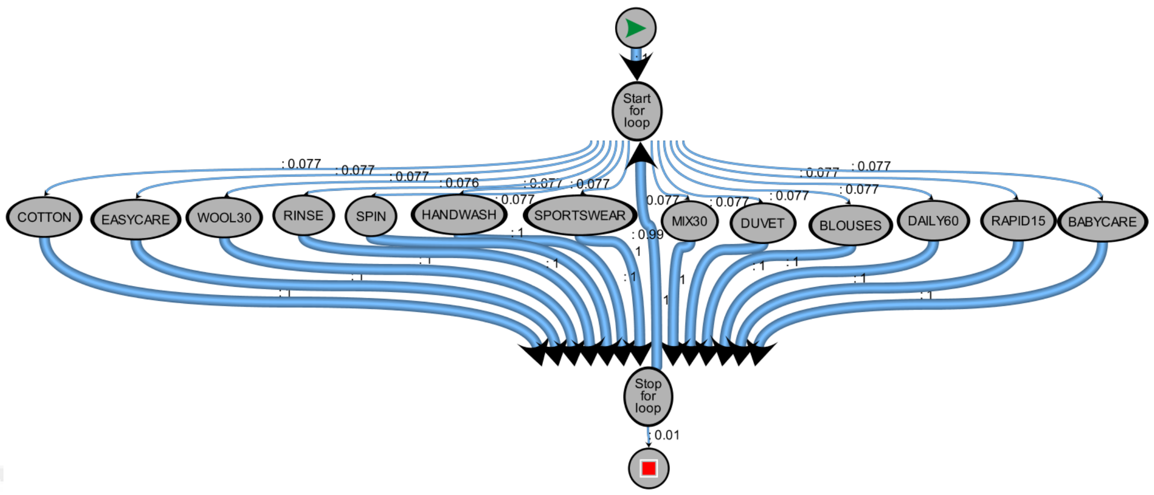

Figure 8: The first version of the test model developed for WM.

in the background and records usage profile regarding the system during this period. The collected information is analyzed by the test group. The main goal of this process is to find usage specific failures caused by diverse (networked) environments for products that require interactions with a variety of peripheral devices. We directly used the usage profile data that is collected for the DTV, SP and WM subject systems.

- ***Static Profile***

We used the list of static code analysis alerts as the static profile for DTV and SP. We used Klocwork[3] as the static code analysis tool to collect alerts regarding the system source code. The only reason for this choice is that the tool was already being used by the company. We directly used the list of alerts reported by the tool for the source code of DTV and SP systems.

We used the number of configuration options per feature, as the static profile for WM. We collected this profile manually by analyzing system specification documents. We iteratively refined the collected profile by discussing it with domain experts. In fact, this

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

profile is by-and-large obtained as a result of efforts spent for developing a test model for the subject system.

- ***Dynamic Profile***

We used the memory profile as the dynamic profile for DTV and SP. We used a custom developed tool [30] to collect memory profile. On the other hand, we used the power consumption profile as the dynamic profile for WM. We calculated the power consumption of each washing program such as Sportwear, Babycare etc. by using a measurement device called the PCE (Personal consumption expenditure) instrument. We recorded these measurements manually.

Dynamic profile is collected during every test execution. Test execution is performed with a tool that is developed in-house by the company. The executed test cases are generated by the MaTeLo tool. There are several algorithms that proposed by MaTeLo for this purpose. We employed two of these, namely the *Minimum Arc Coverage (MAC)* and the *User Oriented (UO)* algorithm [31].

MAC is used for generating test cases in the initial iteration (*Iteration 0*). In this iteration, test models are not yet updated. Hence, all the probability values for outgoing transitions of a state are equal to each other. MAC aims at edge coverage on such a model. It systematically explores the model and stops generating test cases when all the state transitions of it are visited.

UO is used in succeeding iterations, after each model update. As a result of these updates, the probability values for outgoing transitions of a state differ from one another. UO explores the test model and chooses a transition at each state based on these values in a non-deterministic manner. UO has two parameters; the limit on the number of test steps per test case, and the limit for the total number of test cases. In our studies, these parameters were set as 5000 and 1, respectively.

4.4 Analysis

- *Experimental Setup for DTV*

In our DTV case study, we have used usage profile to collect sample which consists of 30 products. This sample have been sent to 30 different field testers. After 30 days past from sending of the sample, the log files are collected and they are also analyzed. Then, for each module, probability values are calculated by RIMA based on the raito of usage per feature. These values can be seen in the first column of Table 1.

Feature	% of usage	% of warnings	% of leakage
Portal	14.6	32.2	21.8
Youtube	17.3	32.2	47.7
HBBTV	3.8	10.8	4.7
Video(MB)	13.4	3.6	11.9
Audio(MB)	3	1.7	2.2
Picture(MB)	0.7	1.7	2.1
PVR	7.6	5.4	5
Channel List	13.4	5.4	1.3
EPG	15.3	3.6	1.7
Teletext	9.6	1.7	0.9
HDMI-SCART	0.7	1.7	0.5

Table 1: The collected data regarding the Digital TV system.

In the second column of Table 1, the ratio of the warning messages (as generated by the static code analysis tool) for each module are listed. The model is updated by RIMA accordingly, as it can be seen in Figure 9.

In the third column of Table 1, the ratio of memory leackage caused by each module are listed. These values are obtained with runtime measurements performed by a custom-developed tool. The model is updated by RIMA accordingly, as shown in Figure 10.

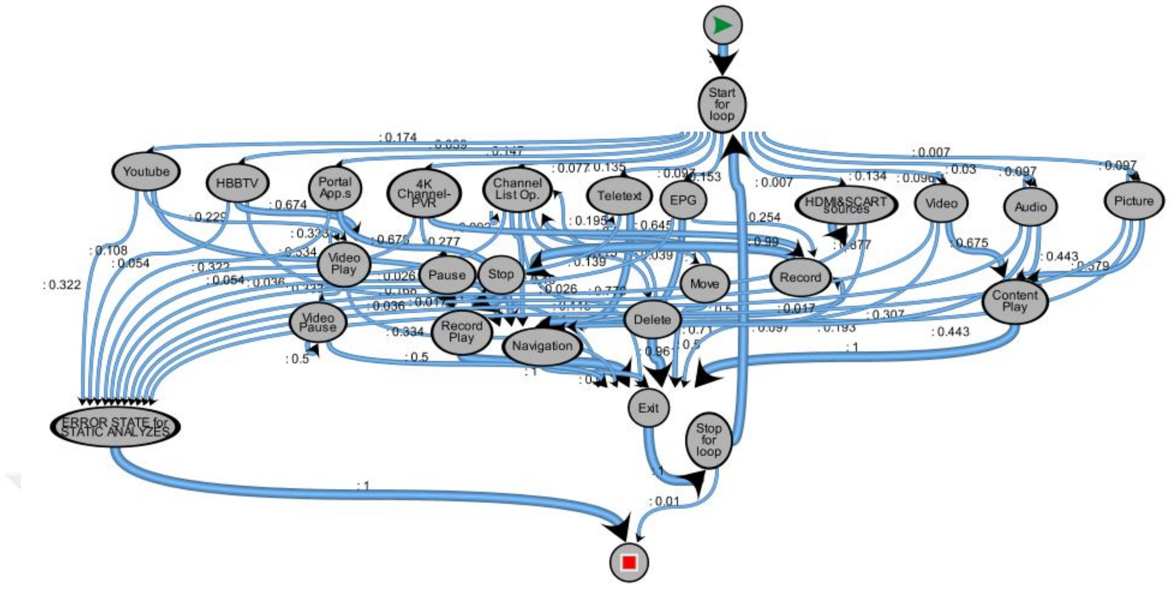

Figure 9: The test model updated based on static profile after the second iteration (DTV).

- ***Experimental Setup for SP***

As a sample of 50 products have been sent to 50 different field testers in order to collect usage profile for SP. After 30 days past from sending of the sample, we have analyzed usage of all of the users. After that, for each module, the probability values are calculated by RIMA according to the ratio of usage, as listed in the first column of Table 2.

We have collected the alerts (warnings) reported by the static code analysis tool. In the second column of the Table 2, ratio of these alerts are listed for each module. These are used for calculating probability values taking part in the test model. Accordingly, the model was updated by RIMA as it can be seen in Figure 11.

Finally, memory profiles are used in the last iteration. The required data is collected by a custom-developed tool during the test runs performed in previous iterations. The ratio of memory leakage associated with each module is listed in the third column of Table 2. These are used for calculating probability values taking part in the test model. The resulting model is shown in Figure 12.

- ***Experimental Setup for WM***

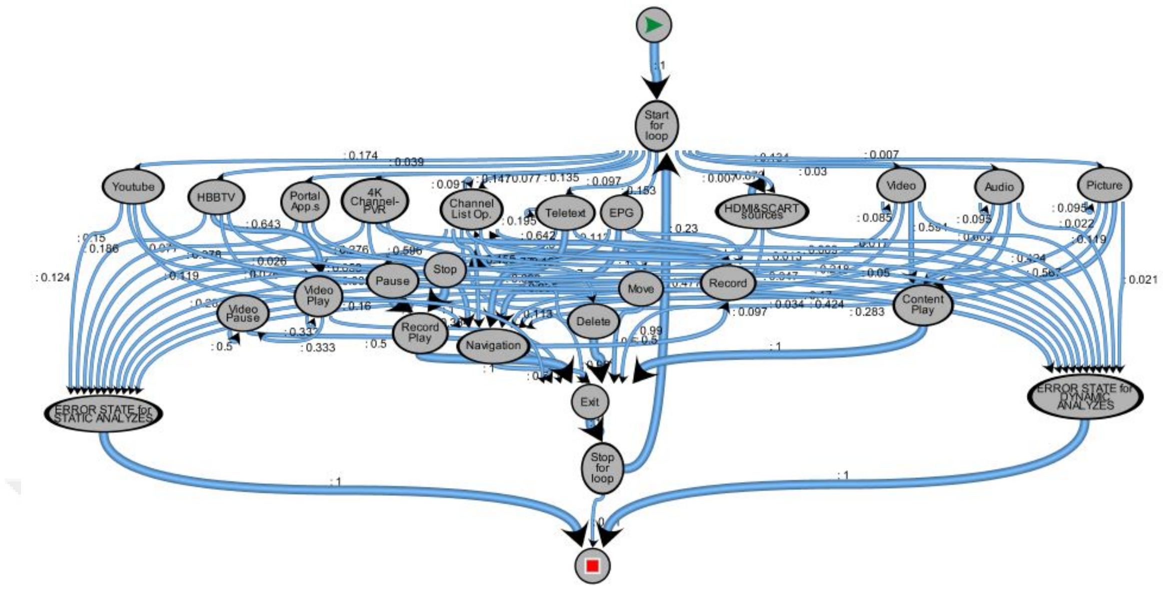

Figure 10: The test model updated based on dynamic profile after the third iteration (DTV).

The 8 washing machines were sent to 8 different people who are users of the machines in order to collect usage profile for 100 days. With the helps of the Internet, these WMs send log file to the company to be analyzed. The first column of Table 3 shows the ratio of usage, which is taken as the basis for calcuating probability values regarding each module.

We have used program configuration analysis for the second iteration. The ratio of the total number of configuration options are listed in the second column of Table 3 for each module. It is assumed that the more the number of configuration options is, the more it is likely to have a fault in the corresponding module. Hence, the test model is updated according to these ratios. The Figure 13 demonstrates the model updated by RIMA.

For the third iteration, we used power consumption measurements. The third column of Table 3 lists the power consumption ratio per feature, which is reflected to the probability value on the correspondign transition. The model (Figure 14) was updated by using these values with the help of RIMA.

Feature	% of usage	% of warnings	% of leakage
VCloud	0.9	9.5	0.5
VCam	22.2	11.6	3.8
Photos-Videos	14.8	11.6	57.1
Alarm	13.0	7.4	5
VMarket	5.6	9.5	14.8
Contacts	10.2	8.4	1.2
Dialer Phone Call	18.5	16.8	2.4
Themes	11.1	7.4	1.6
SMS-MMS	3.7	17.9	13.6

Table 2: The collected data regarding the Smart Phone system.

4.5 Results

4.5.1 Results for DTV

The summary of the results for DTV, is listed in Table 4. By using the original model, 7 faults and 2 more new faults were caught. With the help of the RIMA tool, 5 more new faults could be found in the other iterations. All of the detected faults were critical[4] for usage of the DTV. It took 11 hours (4+4+1.5+1.5) to execute all the test cases and 12 faults (7+2+1+2) were found in total.

Random testing results for the DTV case study are listed in Table 5. We kept the *Iteration 0* results for the initial test execution, which is the same with our approach. 7 faults had been detected (2 new) with using the original model. Then, we iteratively generated test cases without changing any probability values in the model. We generated test cases 3 more times and executed these cases. We observed that it took 16 hours (4+4+4+4) to run all the test cases. However, we could find 9 distinct faults (7+0+2+0) in total.

[4]We deemed these faults to be relevant and important because they lead to failures such as crash or lack of response, which can only be recovered by restarting the whole system. As such, they were categorized by the collaborators as critical faults in the issue management system of the company.

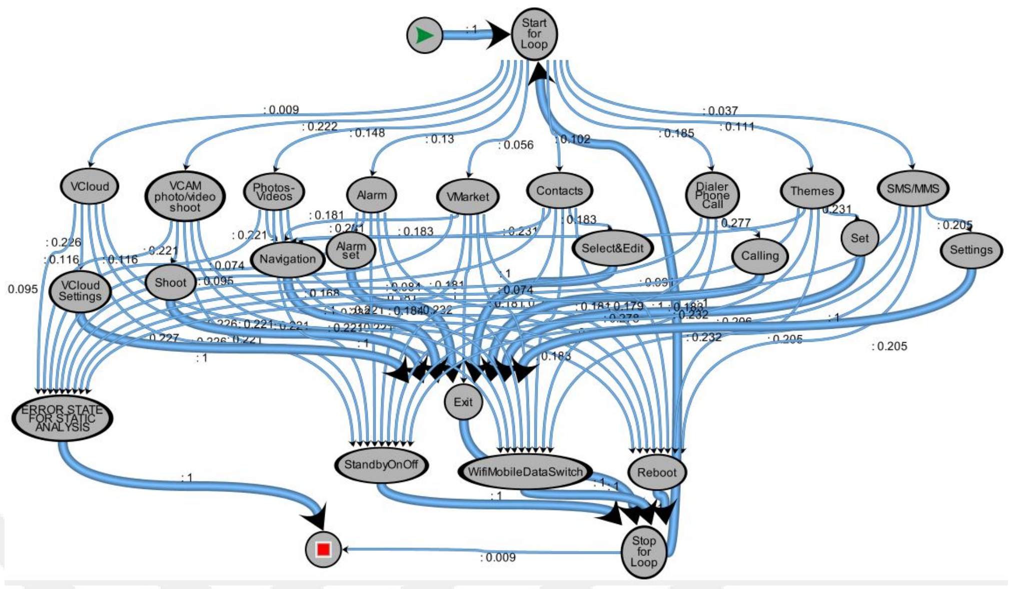

Figure 11: The test model updated based on static profile after the second iteration (SP).

4.5.2 Results for SP

Table 6 summarizes the SP case study results. 3 faults were pinpointed and only one new fault were found by using the original model, before updates. After the model was updated with RIMA, 5 more new faults were caught. These faults are related to performance problems in the system. For example, they include missing a message in the message box, a wrong call information in the contact list etc. It took 21 hours (6.5+5.5+5+4) to run all the test cases and 8 faults (3+3+1+1) were found in total.

Random testing results for the SP case study are in Table 7. We see that it took 26.5 hours (6.5+6.5+7+6.5) to run all the test cases. However, we could find 7 faults (3+2+2+0) in total.

4.5.3 Results for WM

The WM case study results are listed in Table 8. When we used the original model, 8 faults were detected. This model was transformed with RIMA. 4 new faults could be found in the the succeeding iterations. All of the detected faults caused failures related to functionally problem. For instance, they include door lock, spin, drain and shaking problems. It took

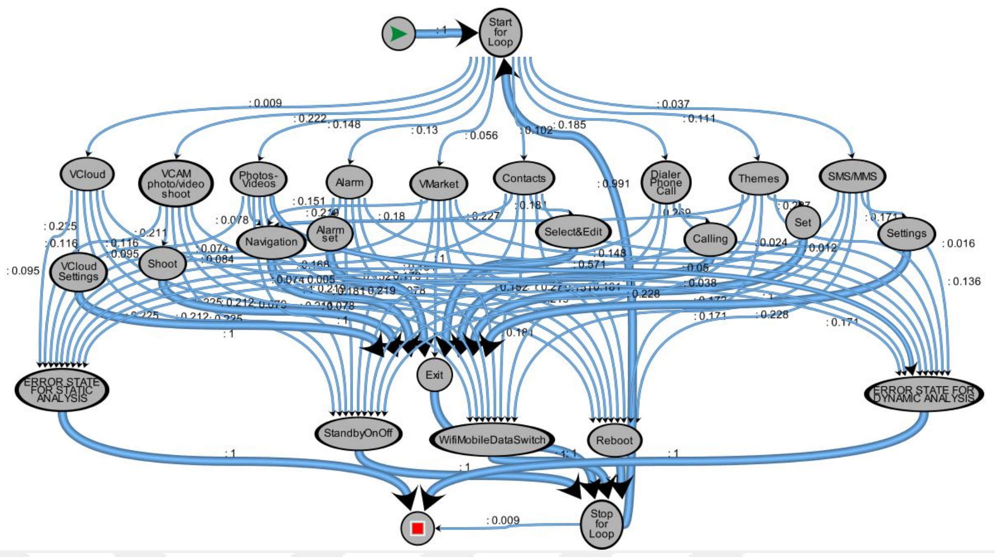

Figure 12: The test model updated based on dynamic profile after the third iteration (SP).

227 hours (84+82+39+22) to run all the test cases and 12 faults (8+1+2+1) were found in total.

Random testing results for the WM case study are in Table 9. We see that it took 301 hours (84+70+78+69) to run all the test cases. However, we could find 12 faults (8+3+1+0) in total.

4.5.4 Interpretation of Results

In the following, we provide an answer to each research question by interpreting the obtained results.

Effectiveness of RIMA **updates based on usage profile (RQ1)** In Table 4, Table 6 and Table 8 the second rows regarding *Iteration 1* show the results after test models are updated based on usage profile. We can see for the DTV system that the number of test steps was decreased by 4.49%. Test execution time remained the same. We could detect 2 new faults. Otherwise, the number of test steps was decreased by 14.9% for the SP system. Test execution time was decreased by 15.38%. 3 new faults were detected in this system. On the other hand, the number of test steps was decreased by 1.61% for the WM system. Test

Feature	% of usage	% of Configurations	% of Power Consumption
COTTON	14.2	13.8	18.2
EASYCARE	7.5	8	22.5
WOOL30	2.4	6.5	7.2
RINSE	1.1	4.3	0.7
SPIN	1.3	0.6	0.7
HANDWASH	5.1	6.7	1.3
SPORTWEAR	4.3	7.5	3.4
MIX30	17.5	11	3
DUVET	1	11	6
BLOUSES	4.6	10.1	9.6
DAILY60	18.5	5.6	7
RAPID15	17	2.8	0.7
BABYCARE	5.5	12.1	19.7

Table 3: The collected data regarding the Washing Machine system.

execution time was decreased by 2.38%. We could detect a new fault in this system as well. Overall, we can conclude that test suites became more effective in terms of the number of detected new faults per executed test step.

Effectiveness of RIMA **updates based on static code analysis (RQ2)** In Table 4, Table 6 and Table 8, the third rows regarding *Iteration 2* show the results after test models are updated based on static code analysis alerts. We could detect 1 new fault in the DTV system, where the number of test steps was decreased by 83.19% and test execution time was decreased by 62.5%. Otherwise, the number of test steps was decreased by 15.85% for the SP system. Test execution time was decreased by 9.1%. We could detect a new fault as well. On the other hand, the number of test steps was decreased by 52.87% for the WM system. Test execution time was decreased by 52.44% and 2 new faults were defined. Therefore, we conclude that test suites became more effective after this update.

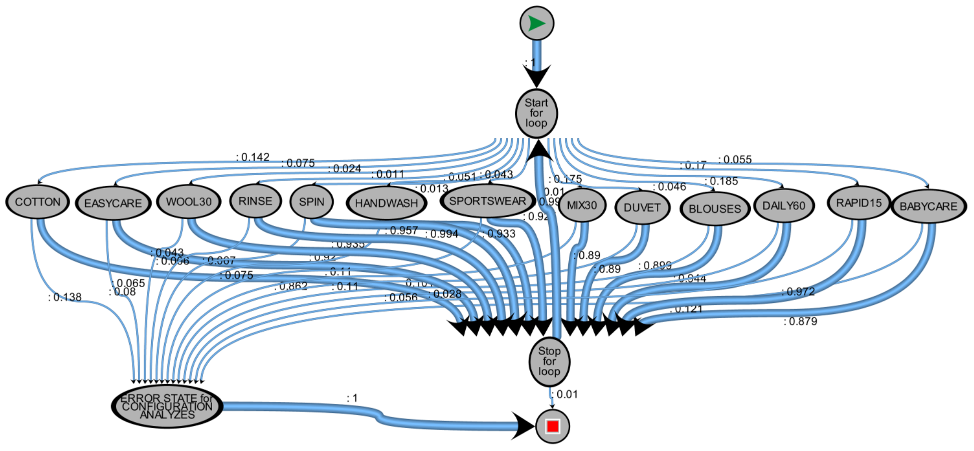

Figure 13: The test model updated based on static profile after the second iteration (WM).

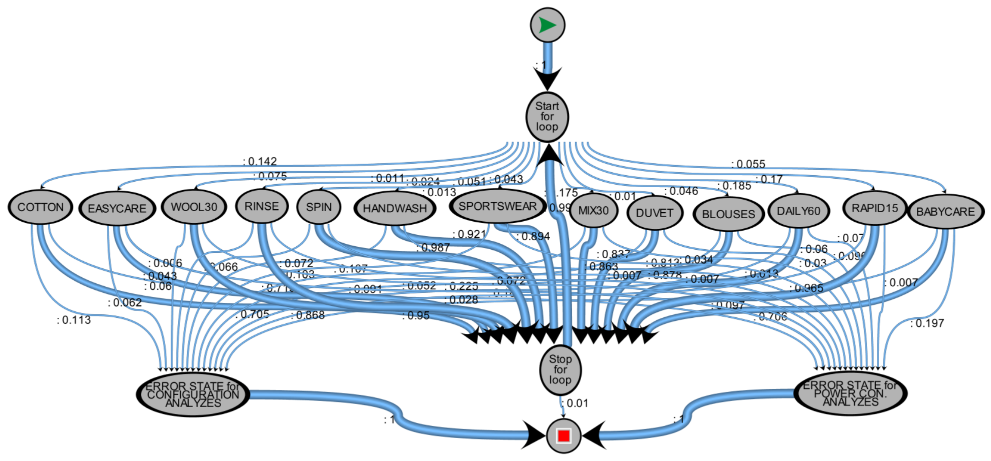

Figure 14: The test model updated based on dynamic profile after the third iteration (WM).

Effectiveness of RIMA **updates based on dynamic analysis (RQ3)** In Table 4, Table 6 and Table 8, the last rows (*Iteration 3*) show the results after test models are updated based on memory profile. We can see for the DTV system that the number of test steps was decreased by 15.83% although test execution time remained the same. In the mean time, we could detect 2 new faults. We could detect a new fault in the SP system as well. However, the number of test steps was decreased by 15.5% and test execution time was decreased by 20%. On the other hand, a new fault was detected. We could identify a new fault in the WM system as well. However, the number of test steps was decreased by 41.74% in this system. Test execution time was decreased by 43.59%. Hence, we conclude that test suites

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

Table 4: Test results regarding the Digital TV sytem (with RIMA).

Iter. #	# of Test Steps	Time (hr)	# of Faults	# of New Faults
0	847	4	7	2
1	854	4	7	0
2	901	4	9	2
3	867	4	8	0

Table 5: Test results regarding the Digital TV system (without RIMA).

became more effective after this update as well.

Test efficiency with and without RIMA **(RQ4)** Finally, we compare the test efficiency achieved with RIMA to that of random testing, where the same test model is used by MBT without any updates in successive test iterations. We measure test efficiency as the number of detected faults per unit of time. So, an improvement can be achieved in two ways; the same set of faults can be detected in less testing time, or additional/new faults can be discovered within the same test duration or less.

Test efficiency of DTV tests is calculated as *1.09 faults/hrs* with RIMA based on the results listed in Table 4. It would be *0.56 faults/hrs* (See Table 5), when RIMA was not used. Similarly, the achieved test efficiency in SP tests can be calculated as *0.38 faults/hrs* (See Table 6) and *0.26 faults/hrs* (See Table 7) with and without RIMA, respectively. Also similarly, the obtained test efficiency in WM tests can be determined as *0.052 faults/hrs* (See Table 8) and *0.039 faults/hrs* (See Table 9) with and without RIMA, separately. We conclude based on these results that RIMA does improve test efficiency with respect to

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

Table 6: Test results regarding the Smart Phone system (with RIMA).

Iter. #	# of Test Steps	Time (hr)	# of Faults	# of New Faults
0	1416	6.5	3	1
1	1456	6.5	5	2
2	1502	7	7	2
3	1489	6.5	7	0

Table 7: Test results regarding the Smart Phone system (without RIMA).

random testing, where the model is not subject to any updates.

4.6 Threats to Validity

In this section, we discuss possible validity threats [32] regarding our case studies.

As a threat to *construct validity*, one might question the validity and completeness of test models that are used as input in our case studies. These models were previously developed by the software test group in the company based on functional requirements documents. These documents are not always complete and precise in the consumer electronics domain. In principle, the more the developed test models are incomplete, the less would be the number of detected faults, regardless of the use of RIMA. Another threat is regarding our evaluation of the detected faults in terms of their severity. Hereby, we attributed them as critical faults based on the categorization scheme of the company. This evaluation does not really reflect the perception of the end users.

Threats to *internal validity* are concerned with various factors that might have influenced

Iter. #	# of Test Steps	Time (hr)	# of Faults	# of New Faults
0	248	84	8	-
1	244	82	5	1
2	115	39	6	2
3	67	22	3	1

Table 8: Test results regarding the Washing Machine system (with RIMA).

Iter. #	# of Test Steps	Time (hr)	# of Faults	# of New Faults
0	248	84	8	-
1	250	70	5	3
2	240	78	9	1
3	236	69	4	0

Table 9: Test results regarding the Washing Machine system (without RIMA).

the obtained results. To mitigate this threat, we compared these results with those obtained with random testing. Hereby, we only removed the application of RIMA from the process and kept everything else the same, including the test models, tools, employed test case generation algorithms and their parameter settings. The selection of the algorithms and their parameter settings were directly adopted from the existing practices of the company.

To mitigate *external validity* threats, we performed two case studies focusing on three products, Digital Television, Smart Phone systems, and Washing Machine. However, both of these systems are developed by the same company. Additional case studies in different contexts can be performed to address concerns regarding the generalisability of the approach. In principle, our approach is applicable to black-box testing of any non-real-time reactive system. It is especially relevant for the consumer electronics domain where the resources are scarce. In terms of tool support, the current implementation only supports the update of test models specified in Markov chain formalism. One of the authors was an employee of Vestel and had access to resources/support in the company to perform case studies. This was the reason for conducting case studies in this company and using the Markov chain formalism, which was already adopted by the company.

We use a set of external tools, which might be considered as a threat to the *reliability* of our studies. Our approach employs RIMA for updating test models automatically. However, it relies on a set of external tools and data for risk estimation and analysis. For instance, we rely on alerts provided by a static code analysis tool to estimate the risk of fault occurrence. If the output of this tool is not accurate, then our estimations would also be wrong.

CHAPTER V

RELATED WORK

There have been various MBT techniques proposed [9]. These techniques employ a variety of formalisms to document a test model like finite state machines [33, 34]. In our approach, test models are specified in the form of Markov chains [21].

MBT is now a state-of-the-practice technique actively being used in industry. There have been several case studies [35, 36] reported for evaluating MBT in an industrial setting. In our work, we evaluate the effectiveness of our approach in an industrial setting as well. In particular, we focus on the consumer electronic domain.

Our approach basically involves iterative refinements of a test model used for MBT. There have been many other approaches [18, 12, 37, 14] that are also based on iterative model refinements. These studies differentiate from our approach by their goal of extending test models. They augment existing models with additional states and transitions to incorporate new system behavior. This behavior is mainly discovered by monitoring the system execution at run-time. In our approach, we do not aim at extending existing test models. We assume that the model is complete enough and it is so large that it is not possible to achieve an extensive coverage of the model during test case generation. We only add fault/error states and update transition probabilities to steer the test case generation process on various execution paths on the model. We employ multiple types of analysis for this purpose based on usage profile, static analysis as well as dynamic analysis.

A risk-based MBT approach [28] was previously proposed based on an inspiration from the principles of risk-based testing [1]. The risk estimation in that approach was only based on dynamic analysis focusing on memory leaks. That approach was later extended with the use of usage profile and static analysis for risk estimation [14]. Hereby, dynamic analysis was again focusing on memory leaks. Usage profile was used for calculating the probability that a particular feature of the system is used and associated faults gets triggered. Static code analysis alerts were used to estimate the likelihood of faults regarding each feature. In this thesis, we generalize that approach and introduce a unified risk-driven MBT approach, where the types of risks and their estimations can be defined based on the system under test and its context. We conduct 3 case studies with various subject systems for which we instantiate the approach according to each system.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

In this thesis, we presented a risk-driven model testing approach. Our approach refines a test model iteratively for increasing the efficiency of model-based testing. The approach is automated with a tool, called RIMA, which takes several types of profile data as input. These include the usage profile, a profile collected using static analysis and another profile obtained via dynamic analysis. RIMA adapts test models for steering test case generation such that various sections of a test model is covered after each refinement iteration to effectively detect critical faults. These faults are the ones that have a higher chance to get exposed to end users as failures. We refined a previously proposed risk-driven model-based testing approach to make it generically applicable to various systems. In our unified approach, various types of risks can be defined and estimated based on either static or dynamic analysis of software artifacts. Hence, the approach can be instantiated differently for each subject system.

We evaluated our approach with 3 industrial case studies. In the first case study, we applied our approach in the context of model-based testing of a Digital TV system. We obtained promsing results where new faults were detected in less time. In the second case study, we focus on smart phones as subject systems and likewise, we were able to pinpoint new faults after each refinement iteration. We also measured the number of detected faults and test duration when random-stochastic test case generation is used without model updates of RIMA. We observed in both case studies that the number of detected faults was decreased. Moreover, the test duration was increased for both case studies. We also instantiated the approach for washing machine software, where risk estimations were performed differently. Yet, we obtained promising results also in this case study, where the test duration was reduced without any decrease in the number of detected faults. These results suggest that RIMA increases the efficiency of model based testing.

As future work, RIMA can be extended to take additional fault/error types into account and employ additional analysis to improve fault and error likelihood estimations. The number of case studies can also be increased for evaluating the general applicability and effectiveness of the approach in various application domains.

References

[1] 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.

[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] U. Yuksel and H. Sözer, “Automated classification of static code analysis alerts: A case study,” in *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, pp. 532–535, 2013.

[4] 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.

[5] 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.

[6] D. Rafi, K. Moses, K. Petersen, and M. Mäntylä, “Benefits and limitations of automated software testing: Systematic literature review and practitioner survey,” in *Proceedings of the 7th International Workshop on Automation of Software Test*, pp. 36–42, 2012.

[7] 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.

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

[9] 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.

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

[11] A. Mesbah, A. van Deursen, and D. Roest, “Invariant-based automatic testing of modern web applications,” *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 35–53, 2012.

[12] 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.

[13] C. Gebizli and H. Sozer, “Automated refinement of models for model-based testing using exploratory testing,” *Software Quality Journal*, vol. 25, no. 3, pp. 979–1005, 2016.

[14] C. Gebizli, H. Sözer, and A. Ercan, “Successive refinement of models for model-based testing to increase system test effectiveness,” in *Proceedings of the 11th Workshop on Testing: Academia-Industry Collaboration, Practice and Research Techniques*, pp. 263–268, 2016.

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

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

[17] 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.

[18] 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.

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

[20] G. H. Walton and J. H. Poore, “Generating transition probabilities to support model-based software testing.,” *Software: Practice and Experienc*, vol. 30, no. 10, pp. 1095–1106, 2000.

[21] 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.

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

[23] C. S. Gebizli, *Automated refinement of models for model-based testing*. Ph.D. thesis, Ozyegin University, 2017.

[24] “Standard glossary of terms used in software testing,” tech. rep., ISTQB, 2012.

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

[26] M. Wendland, M. Kranz, and I. Schieferdecker, “A systematic approach to risk-based testing using risk-annotated requirements models,” in *Proceedings of the 7th International Conference on Software Engineering Advances*, pp. 636–642, 2012.

[27] M. Felderer, C. Haisjackl, R. Breu, T. Margaria, and B. Steffen, "A risk assessment framework for software testing," in *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*, pp. 292–308, 2014.

[28] 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.

[29] 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.

[30] 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).

[31] 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.

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

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.