

# TOOL SUPPORT FOR MODEL BASED SOFTWARE PRODUCT LINE TESTING

A Thesis

by

Burcu Ergun

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  
January 2018

Copyright © 2018 by Burcu Ergun

# TOOL SUPPORT FOR MODEL BASED SOFTWARE PRODUCT LINE TESTING

Approved by:

---

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

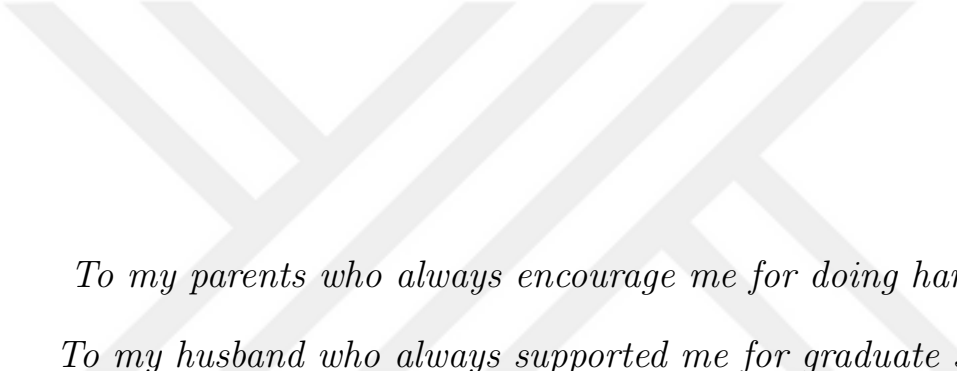
---

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

---

Assoc. Prof. Ali Fuat Alkaya  
Computer Science and Engineering  
Department  
*Marmara University*

Date Approved: January 4, 2018



*To my parents who always encourage me for doing hard works...*  
*To my husband who always supported me for graduate studies and*  
*master of science challenge...*

## ABSTRACT

We introduce a tool for automated adaptation of test models to be reused for a product family. Test models are specified in the form of hierarchical Markov chains. They represent possible usage behavior regarding the features of systems as part of the product family. A feature model documents the variability among these features. Optional and alternative features in this model are mapped to a set of states in test models. These features are selected or deselected for each product to be tested. Transition probabilities on the test model are updated by our tool according to these (de)selections. As a result, the test case generation process focuses only on the selected features. We conducted two controlled experiments, both in industrial settings, to evaluate the effectiveness of the tool. We used systems as part of digital TV and wireless access point(WAP) systems. For DTV systems 10 and for wireless access points 5 participants were involved in testing these systems, respectively. We measured the effort spent by each participant for the same set of tasks when our tool is used and when it is not. We observed that the tool reduces costs significantly. We also observed that the initial cost for adopting product line testing is amortized even for small product families with 13 DTV and 11 WAP products, respectively.

## ÖZETÇE

Bu tezde bir sistem ailesi için test modellerinin yeniden kullanılmasını kolaylaştıran bir araç tanıtılmaktadır. Test modelleri, bu sistemlerin özelliklerinin kullanım olasılıklarını belirlemek için hiyerarşik Markov zincirleri olarak tanımlanmaktadır. Bu özellikler arasındaki değişkenlik bir ürün özellik modeliyle belgelenmektedir. Ürün özellik modelinde isteğe bağlı ve alternatif özellikler test modellerinde bir dizi duruma eşlenmektedir. Bu durumlar için geçiş olasılıkları seçilen özelliklere göre değiştirilmekte, böylece oluşturulan test durumları yalnızca bu özelliklere odaklanmaktadır. Aracın etkinliğini değerlendirmek için endüstriyel ortamlarda iki kontrollü deney yapılmıştır. Deneyler için dijital TV (DTV) ve kablosuz erişim noktası (WAP) sistemleri kullanılmıştır. Bu sistemler için sırasıyla 10 ve 5 katılımcı, sistem testlerinde ve dolayısıyla deneylerde rol almışlardır. Her katılımcının harcadığı çaba, sistemlerin test modellerini güncellerken araç kullanıldığı ve kullanılmadığı durumlarda ölçülmüştür. Aracın maliyetleri önemli ölçüde düşürdüğünü ve sırasıyla 13 DTV ve 11 WAP ürününe sahip küçük ürün ailelerinde bile, araç kullanılarak ürün hattı mühendisliği uygulamasının adapte edilme maliyetinin karşılanarak, toplamda harcanan çabada düşüş olduğu gözlemlenmiştir.

## ACKNOWLEDGMENTS

Firstly, I would like to thank my advisor, Assoc. Prof. Dr. Hasan Sözer for all his help and guidance that he has given me over the past two years. Secondly, I would like to thank Dr. Ceren Şahin for providing me support during this period. I would also like to thank software developers and software test engineers at Airties Wireless Networks and Vestel Electronics for sharing their code base and experiences with me and such supporting my experiments.

# TABLE OF CONTENTS

<b>DEDICATION</b> . . . . .	<b>iii</b>
<b>ABSTRACT</b> . . . . .	<b>iv</b>
<b>ÖZETÇE</b> . . . . .	<b>v</b>
<b>ACKNOWLEDGMENTS</b> . . . . .	<b>vi</b>
<b>LIST OF TABLES</b> . . . . .	<b>ix</b>
<b>LIST OF FIGURES</b> . . . . .	<b>x</b>
<b>I INTRODUCTION</b> . . . . .	<b>1</b>
<b>II BACKGROUND</b> . . . . .	<b>4</b>
2.1 Model Based Testing . . . . .	4
2.2 Software Product Line Engineering . . . . .	6
<b>III RELATED WORK</b> . . . . .	<b>8</b>
<b>IV APPROACH</b> . . . . .	<b>11</b>
<b>V EXPERIMENTAL EVALUATION</b> . . . . .	<b>17</b>
5.1 Research Questions . . . . .	17
5.2 Experimental Setup . . . . .	18
<b>VI RESULTS AND DISCUSSION</b> . . . . .	<b>25</b>
6.1 Effort for Test Model Update without Tool Support . . . . .	27
6.2 Effort for Test Model Update with Tool Support . . . . .	27
6.3 Effort for Tool Setup . . . . .	28
6.4 Threats to Validity and Limitations . . . . .	30
<b>VII CONCLUSIONS AND FUTURE WORK</b> . . . . .	<b>31</b>
<b>APPENDIX A — DTV FEATURE AND TEST MODELS</b> . . . . .	<b>32</b>
<b>APPENDIX B — WAP TEST MODEL</b> . . . . .	<b>36</b>

<b>APPENDIX C — COLLECTED MEASUREMENTS DURING THE EXPERIMENTS . . . . .</b>	<b>38</b>
<b>REFERENCES . . . . .</b>	<b>41</b>





## LIST OF TABLES

1	The list of DTV products and the number of feature selections for the first experiment [37]. . . . .	19
2	The list of WAP products and the number of feature selections for the second experiment. . . . .	19
3	Participant information and the amount of provided training (hours) for the first company [37]. . . . .	21
4	Participant information and the amount of provided training (hours) for the second company. . . . .	21
5	The design of experiments. . . . .	22
6	Average effort required for model updates per DTV product with and without tool support [37]. . . . .	26
7	Average effort required for model updates per WAP product with and without tool support. . . . .	26
8	T-test results for the two experiments (paired two-samples for means). . . . .	26
9	Measurements collected during the first experiment. . . . .	39
10	Measurements collected during the second experiment. . . . .	40

## LIST OF FIGURES

1	A sample feature model. . . . .	7
2	The overall process for using tool support. . . . .	11
3	A part of the feature model created for the WAP product family and some of the feature selections regarding a product of this family. . . .	12
4	The updated test model of WAP. . . . .	15
5	Variation of effort for MBT of DTV systems with and without FORMAT. . . . .	27
6	Variation of effort for MBT of WAP systems with and without FORMAT. . . . .	28
7	A part of the feature model created for the DTV product family and some of the feature selections regarding a product of this family [1]. . . .	33
8	The top-level test model regarding the DTV product family [37]. . . . .	34
9	An updated test (sub)model regarding the DTV GUI states [1]. . . . .	35
10	The top-level test model regarding the WAP product family. . . . .	37

# CHAPTER I

## INTRODUCTION

We interact with many products in our daily lives such as Digital TV (DTV) systems and Wireless Access Point (WAP) systems. These systems are developed as embedded systems that are mainly controlled by software. They are produced with a high variety and increasing number of features. This leads to a continuously increasing size and complexity of software adopted in these products. For instance, current DTV systems include many features such as web browsing and on-demand streaming in addition to traditional TV functionalities [2]. As another example, current WAP systems employ a large code base for implementing a large variety of communication protocols as well as for providing several features regarding security, trouble shooting and system recovery. This trend necessitates to adopt efficient and effective testing techniques. The most critical faults must be detected with limited resources in a continuously increasing scope. Model-based testing (MBT) [3] has been employed as a solution for increasing test effectiveness and efficiency.

MBT is used for automatically generating test cases by systematically exploring test models that define the usage behaviour of the System Under Test (SUT). There are two main advantages of this approach. The first advantage is regarding test quality. The quality can be measured based on the coverage of the test model and this coverage can be improved by exploring the model in a systematic manner. The second advantage is regarding maintainability. Changes can be applied to the abstract model rather than going through each and every individual test case. As a drawback of MBT, one needs to develop and maintain test models, which can be a costly process that requires expertise. This cost is amplified in the scope of testing product families, which

can include a large number of products with systematic variations. These variations change the set of relevant features. Hence, scenarios to be tested for various products differ from each other. Software Product Line Engineering (SPLE) [4] was introduced as an approach for the systematic management of variability in product families. This approach aims at maximizing the amount of reuse for all the artifacts created during software development lifecycle, including those related to testing activities [5].

An application of SPLE approach together with MBT [1] involves the development of 3 separate models. First, a *reference test model* is specified in the form of hierarchical Markov chains and it represents all possible usage scenarios for a family of systems rather than a single system. Second, a feature model is created for documenting variations among products of the tested product family. Third, a specification defines a mapping between the other two models. Hereby, every alternative or optional feature is associated with a state in the reference test model. Probability values for performing a transition to these states are updated based on the available features of the SUT. As a result, the generated test cases focus only on these features. This approach makes it possible to reuse a test model for multiple systems. As a drawback, one needs to update the model manually for each SUT. In this thesis work, we developed a tool, namely **FORMAT** (Feature Oriented Model Adaptation Tool) to automate this SPLE approach for MBT. This tool automates the test model update process. The usage of the tool requires an initial (one-time) development of the 3 models required by the approach. Then, the reference test model can be reused for a family of systems by selecting features of the SUT via a graphical user interface.

We performed two controlled experiments, both in industrial settings, to evaluate the effort reduction achieved by **FORMAT**. The first of these experiments is conducted in a consumer electronics company, where 10 participants were involved in testing 10 DTV systems. The second experiment is conducted in a wireless home network system company, where 5 participants were involved in testing 5 WAP systems. Reference

test models that represent families of products had to be adapted for various member products based on their features. The amount of effort for each participant was measured both with and without the existence of tool support. Results show that FORMAT significantly reduces the overall effort and the cost of initial investment can be amortized after the reuse of the reference test model for a small number of products. We observed in the first experiment that this cost is amortized after the reuse of the reference test model for 13 DTV products. Similarly, we can conclude based on the results of the second experiment that the investment pays off after testing 11 WAP products.

The remainder of this thesis is organized as follows. In the following section, we provide background information on MBT and SPLE. In Section 3, we summarize the related work. In Section 4, we explain our approach and tool. In Section 5, we present the two controlled experiments for evaluating these in industrial settings. We present and discuss the obtained results in Section 6. Finally, we conclude the thesis in Section 7.

## CHAPTER II

### BACKGROUND

#### *2.1 Model Based Testing*

MBT adopts a systematic exploration of test models [6, 7] to automatically generate test cases [8, 9, 10]. It relies on a test model that defines the user-observable SUT behavior with respect to a set of inputs and actions of the user. This model is usually created manually by analyzing system requirements. A MBT tool takes such a model as input and generates test cases automatically by exploring possible behavioral scenarios. This tool can support various test case generation algorithms and it can be configured to achieve various coverage criteria.

Benefits of MBT was previously outlined as follows [12]: *i)* Early detection of inconsistencies in requirement specifications; *ii)* Improved communication (test models serve as documentation); *iii)* Automated generation of test cases; *vi)* Informed selection and prioritization of test suites; *v)* Improved maintainability of tests at the abstraction level of test models; *vi)* Increased test quality due to measured and optimized test coverage; *vii)* Lower costs.

Both the content and the structure of the test model can differ among MBT approaches and tools. It can be expressed with several types of formalisms such as Unified Modeling Language (UML) models [13], Finite State Automaton (FSA) [14], Event Sequence Graph (ESG) [15], state charts [16], Markov chains [17] and Labelled Transition System (LTS) [18]. MBT that is based on UML models mainly employ sequence and collaboration diagrams to specify test scenarios [13]. In addition, use case diagrams are associated with sequence or collaboration diagrams to organize these.

FSA is commonly used for representing state-based behaviors of SUT. It includes a set of inputs, a set of states, and a transition function that maps (input, state) pairs to next states. Test cases are specified as possible sequence of states on the model (i.e., an execution path).

ESG [19, 15] models have a more abstract representation relative to FSA models. In these models, inputs and states are represented together as *events*. Events correspond to user-observable actions. They are mapped to next events by a transition function.

State charts [16] are very similar to FSA models. As a difference, they can be hierarchically composed. In addition, states and transitions can be labeled with guards and actions.

LTS models are also based on FSA models and they are used for modeling reactive systems. As a difference from FSA models, transitions can be annotated with input and output labels [18]. These labels are used for synchronization [18].

A Markov chain conforms to another formalism that is based on FSA. As a difference, state transitions are labeled with probability values [17]. The system may change its state from the current state to another state, or remain in the same state, according to these values. In our approach, we employ this formalism to represent test models. We use state transition probabilities to eliminate some execution paths and as such focus the scope of the generated test cases.

In the following, we position the MBT approach adopted in this work with respect to a previously introduced taxonomy of MBT approaches [20]. Our model specification scope is input-output. The test model is untimed, stochastic, and discrete. The test case generation algorithm is stochastic. Test execution is performed offline. Concrete test steps are defined as test scripts. These scripts are associated with transitions of the test model.

## 2.2 Software Product Line Engineering

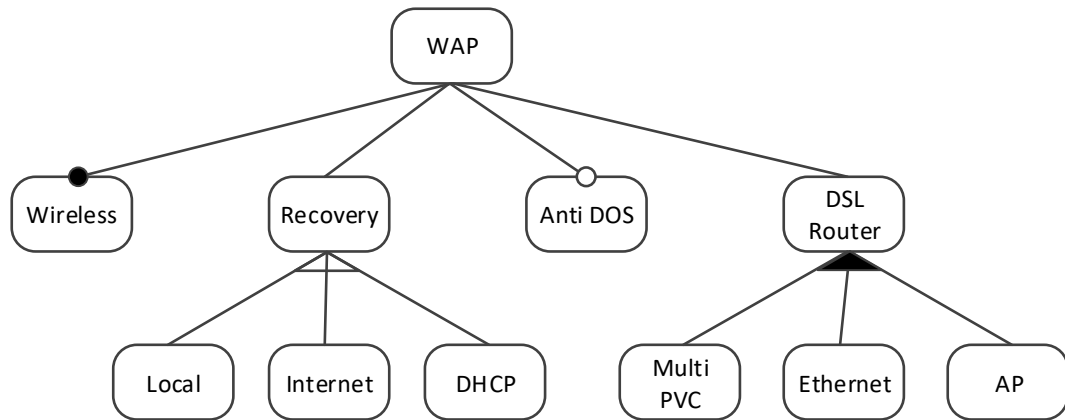
Software Product Line Engineering (SPLE) [4] is an approach for the systematic management of variability in product families. This approach aims at maximizing the amount of reuse for all the artifacts created during software development life cycle, including those related to testing activities [5]. The key principle of SPLE is the separate and explicit documentation as well as management of variability among the products of a product family. Variability can be documented with various types of models, where Orthogonal Variability Model (OVM) [4] and feature model [21, 22] are currently the most prominent types. In this work, we used feature models for this purpose.

Feature models [21] are defined in the form of tree structures. There is a standard visual notation for depicting the elements of the model. These elements basically represent features of products of a product family. These features can be mandatory or optional. There might also be inter-dependencies among the features. That is, including/excluding a feature for the product might necessitate the inclusion/exclusion of another related feature. Each combination of selected features defines a product of the product family.

A sample feature model is depicted in Figure 1. We can see that the model depicts features with parent-child relationships. These relations can be one of the 4 types [23]:

- **mandatory:** features are always selected if their parent is selected. (e.g., *Wireless* feature in Figure 1).
- **optional:** features may or may not be selected if their parent is selected. (e.g., *Anti DOS* feature in Figure 1).
- **inclusive-or:** features are grouped such that at least one of them is selected if their parent is selected. (e.g., *Multi PVC*, *Ethernet* and *AP* features in





**Figure 1:** A sample feature model.

Figure 1).

- **exclusive-or:** features are grouped such that exactly one of them is selected if their parent is selected.(e.g., *Local*, *Internet* and *DHCP* features in Figure 1).

Although MBT helps in managing variability and improving maintainability, it cannot scale for large scale product families. These families are characterized by a large number of products with high number of variations. These variations tend to cross-cut test models [24, 25]. Hence, the adopted MBT approach and test models enable systematic management of variability. Ideally, test models should be reusable across a family of products and the cost of this reuse must be amortized. That is, the effort spent for adapting the test models and using them for various products should not exceed the cost of using a separate test model for each product. In this work, we introduced tool support for facilitating such a reuse of test models in MBT.

## CHAPTER III

### RELATED WORK

An analysis of the literature reveals that studies related to product line testing [5] mainly focus on system testing. There also exist studies among these, which are concerned with the adoption of MBT for software product line testing [26]. A common property among all of these studies is that they explicitly specify variability. Another common property is regarding the specification of test models. These models reflect the usage behavior for a family of products rather than a single product. They are modified to end up with a test model that is specific to a product. These properties are also valid for our approach. However, there are variations with respect to formalisms used for modelling and the level of automation.

There exist a subset of studies that combine MBT and SPLE make use of UML models. CADeT [28] and ScenTED [27] can be provided as example representatives of this subset. In particular, these approaches utilize UML sequence diagrams to define test cases. All the variations are specified as part of these diagrams. In our approach, we do not utilize UML diagrams. We develop a test model in the form of Markov chains. Variability is not documented as part of this diagram but it is documented separately instead. We utilize a feature model for this purpose. Our approach necessitate a third specification to relate elements of this feature model with those of the test model. As an advantage of this separate specification, variability information is not scattered throughout the test model and the model is not tangled with two different information: usage behavior and variability. As a disadvantage, an additional specification has to be created manually; however, this is only a one-time effort if the other models are not subject to subtle changes.

There are also differences with respect to the goals of existing studies. For example, unlike CADeT [28], our goal is not to obtain coverage measures, which is already provided by the employed MBT tool. Our main goal is to reuse the same model for a variety of products by systematically updating it with minimal effort.

We can find MPLM [29] in the literature, when we focus on approaches that integrate SPLE and MBT by using Markov chains as the formalism to express test models. In fact, this tool has been introduced as an extension of MaTeLo, the MBT tool that we employed in the implementation of our approach. As a difference from our approach, MPLM uses OVM [4] to specify variability information, instead of feature models. As the second difference, MPLM removes states and transitions from the test model to obtain a test model that is specific to a product. These modifications are applied based on the selected variants in the OVM model. In our approach, we only modify probability values associated with state transitions. We do not modify the overall structure of the model by adding or removing elements from it.

When we focus on the main difference between our approach and MPLM, we have 3 reasons for employing feature models instead of OVM for documenting variability. First, it is more intuitive to map elements of a feature model to elements of a test model. This is because, elements of the model directly represent features of the system as we apply system-level testing. The system is modeled completely from the user perspective. This is not the case behavior with OVM models, which aim at defining both external and internal variability. Hereby, external variability represents user-observable variability [4] whereas internal variability is related to platform and implementation level variations in products. So, the scope of an OVM model is unnecessarily broad for representing features. The second reason is that both feature models and our test models have a hierarchical structure. So, not only their content, but also their structure lead to a more compatible and straight-forward mapping. The third reason is that feature models are more prominent at the time of writing

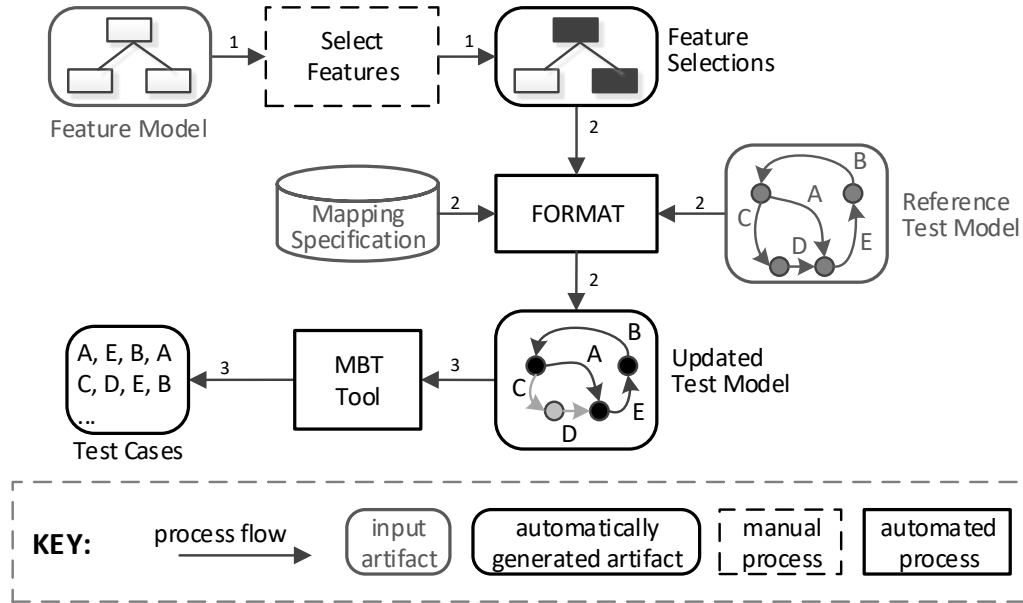
this thesis. There are more resources available for feature diagrams both in terms of theoretical work [22] and tool support [31].

We can also find approaches [32, 33] in the literature, when we focus on approaches that integrate SPLE and MBT by using feature models for documenting variability. However, state machines are used for specifying test models in this case. Moreover, these approaches are alike MPLM in the sense that they remove states and transitions from the test model to obtain a test model that is specific to a product. In our approach, we do not modify the overall structure of the model but modify probability values associated with state transitions instead.

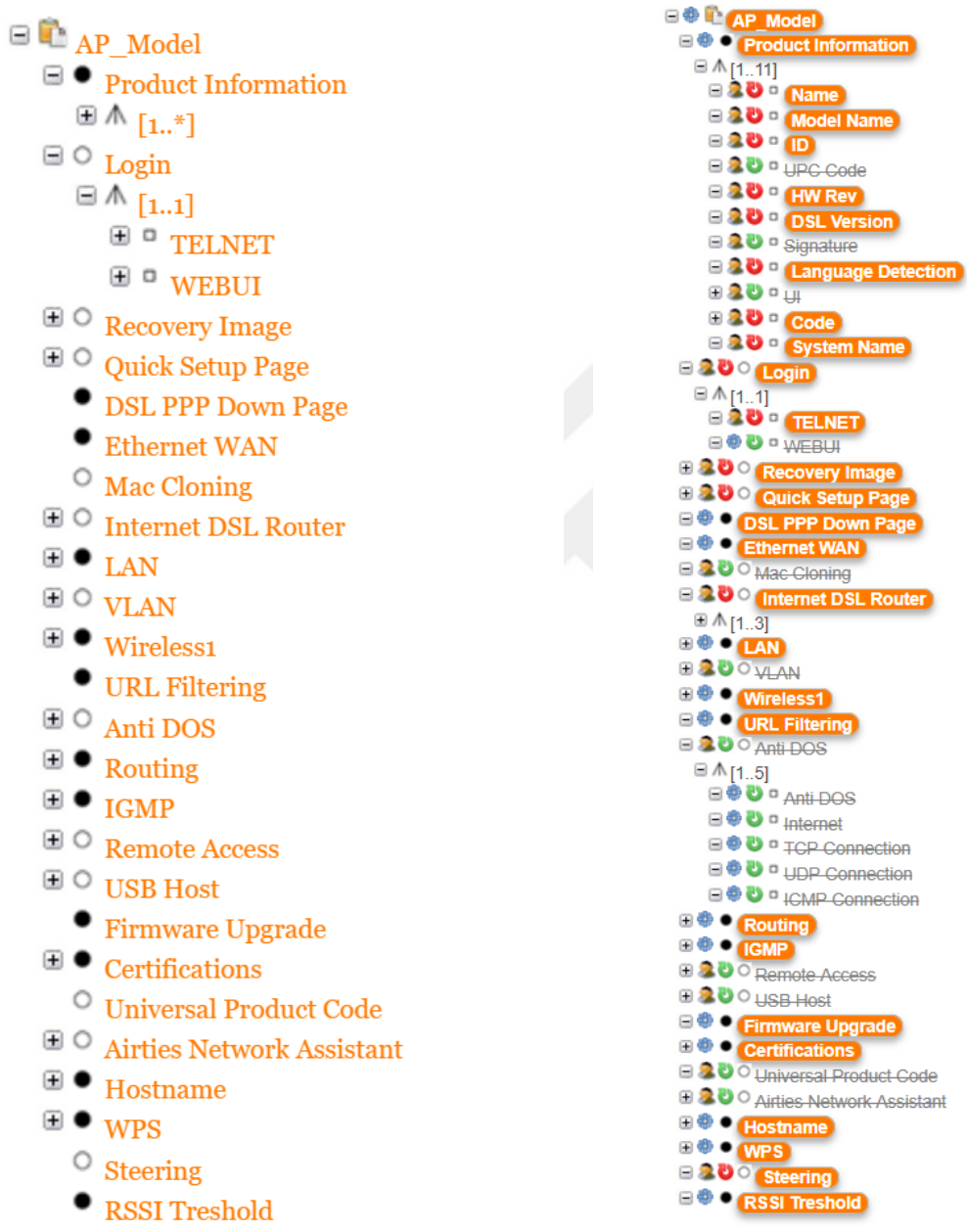
# CHAPTER IV

## APPROACH

Our approach, modify probability values associated with state transitions instead. We do not modify the overall structure of the model. Figure 2 depicts the overall process for using tool support to adopt SPLE in the context of MBT [1]. The set of input artifacts are shown with gray color in the figure. One of these artifacts is the *reference test model*, which specifies the system usage behavior for a family of products. The second one is the *feature model*, which captures the variability information regarding these products. The last one is the *mapping specification*, which is basically a store keeping mapping of elements of the feature model to those of the test model.



**Figure 2:** The overall process for using tool support.



(a) A part of the feature model of the WAP product family. (b) Feature selections for a sample WAP product.

**Figure 3:** A part of the feature model created for the WAP product family and some of the feature selections regarding a product of this family.

We used an online, free tool, namely SPLOT tool<sup>1</sup> to create feature diagrams. This tool also provides a user interface where one can (de)select features on the diagram and specify various product configurations. Hence, although the first step of the approach is performed manually, there exist tool support for this part of the process as well. As an example, we can see a part of the feature diagram created for the WAP product family and an example product configuration regarding this family in Figure 3(a) and Figure 3(b), respectively. In this example configuration, we can see that the optional feature *AntiDOS* is not selected for instance. This choice automatically leads to the deselection of all of its descendent features as well. The generated test cases for this product configuration should not cover such deselected features.

Each product configuration that is created via the SPLOT user interface can be exported in the form of an XML file. This file is used as an input for the second step. We implemented the tool, **FORMAT**<sup>2</sup> for automating this step of the approach. This tool takes two more inputs in addition to the product configuration that includes feature selections. One of these is the reference test model. The other one is the mapping specification which maps selected features to states of the reference test model. In principle, this specification can be automatically generated, if both the names of features and names of states in the test model follow a naming convention.

**FORMAT** updates the transition probability values in the test model based on feature selections in the product configuration. Algorithm 1 [37] outlines the update procedure implemented by the tool. Hereby, for each state  $s$ , all the transitions are processed, where  $s$  is the source state of the transition (Line 4). Probability values for transitions to states that represent unselected features are set as 0 (Line 7). Prior to this update, the original value is summed up ( $p$ ) for all such transitions where the

---

<sup>1</sup><http://www.splot-research.org/>

<sup>2</sup>The source code of **FORMAT** is available at <https://github.com/brcoztn/AgileSWRepo>

---

**Algorithm 1** Model Update Procedure [37].

---

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

---

source state is  $s$  (Line 6). Then, probability values of all the transitions outgoing from this state are multiplied by  $1/(1 - p)$  (Lines 10-14). This step of the algorithm ensures that two properties hold. First, the relative weight of the transitions remains the same excluding those that are set as 0. Second, the sum of all the probability values for transitions outgoing from the same state is kept as 1.

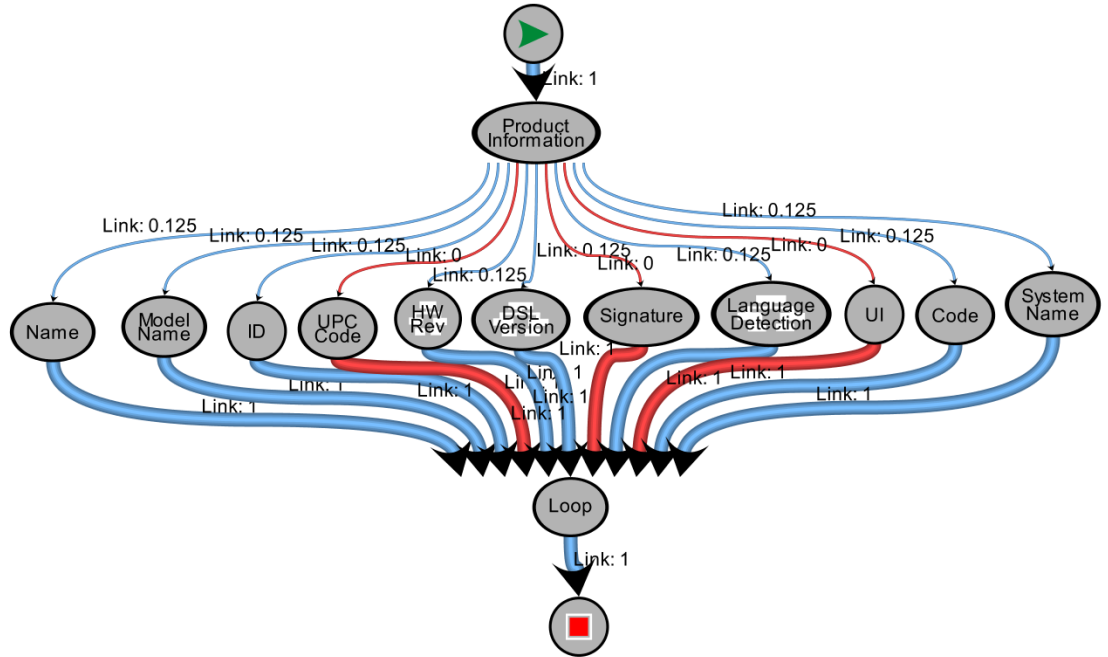
For example, let's consider a state with 3 outgoing transitions,  $t_1$ ,  $t_2$  and  $t_3$ , with probability values 0.3, 0.6 and 0.1, respectively. Assume that the probability value for  $t_2$  is set to be 0 because the feature represented by its target state is deselected. Then, the probability values for  $t_1$  and  $t_3$  are multiplied by  $1/(1 - 0.6)$ , which is  $10/4$ . As a result,  $t_1$  and  $t_3$  are associated with probability values 0.75 and 0.25, respectively. The overall sum of the values add up to 1 and the relative priority between  $t_1$  and  $t_3$  remains the same.

Figure 4 shows an example test model<sup>3</sup> for the WAP product family that is updated with **FORMAT**. Hereby, probability values for 3 transitions are set as 0. These transitions are all originating from the *Product Information* state. They are targeting

---

<sup>3</sup>Please note that this figure depicts the top-level model only. 3 of the states in this model actually represent other test models. States represented in these models might hierarchically include further sub-models.





**Figure 4:** The updated test model of WAP.

at *UPCCode*, *UI* and *Signature* states. All the other transitions that are originating from the *Product Information* state are also updated. We can see that they have equal weights and they add up to 1. The probability values regarding the 3 transitions are set as 0 due to the corresponding features that are deselected as shown in Figure 3(b).

The third and the last step of the approach involves test case generation and test execution activities performed based on the updated model. These activities are performed with two external tools. MaTeLo<sup>4</sup> is used for test case generation with the so-called “Most Probable” test case generation algorithm [34]. This algorithm explores the paths on the test model until the lengths of these paths reach to a threshold point. It selects alternative transitions based on their probability values during this process. As a result of the update performed on the WAP test model, for instance, *UPC Code*, *Signature* and *UI* states will never be visited in the explored

<sup>4</sup><http://www.all4tec.net>

paths. Test execution is automated with VESTA, which is an in-house developed tool.

In the following chapter, we present two controlled experiments that are conducted in industrial settings to evaluate the effort reduction achieved by the use of **FORMAT**.



# CHAPTER V

## EXPERIMENTAL EVALUATION

We conducted two controlled experiments, both in industrial settings. These experiments provide us to evaluate our tool’s efficiency in terms of effort reduction. Our aim is comparing manual effort of test model update and automatic test model update with FORMAT tool. The traditional approach adopted by companies is updating reference test model manually for each product configuration with selecting and de-selecting features. FORMAT tool enable automated updates of reference test models based on the configured product model as input. But firstly we need prepare the product feature diagram and feature mapping which is used for associating product features and reference model states. This is additional effort needs to be paid for updating test model automatically. Hence, we also wanted to measure how/when one can amortize this additional effort.

This chapter is organized to provide the following information in order: *i)* Defining to our research questions for our approach; *ii)* Explanation of experimental setup for the both companies for different product domains; *iii)* Presentation and interpretation of results; *iv)* Discussion on experiment design validity and known limitations.

### ***5.1 Research Questions***

Our evaluation goals explained above led us to the following research questions:

***RQ1:*** What is the effort for updating test models manually?

***RQ2:*** What is the effort for updating test models with FORMAT?

***RQ3:*** What is the effort for creating the input models for FORMAT?

*RQ1* is formulated to learn the cost of the effort of the traditional approach for updating reference test model for each product. *RQ2* lets us understand the cost of effort with for updating reference test model for each product using **FORMAT**. *RQ3* is necessary to measure the cost of additional effort for creating a feature diagram and mapping specification. Hence, we defined our null hypothesis like the following to test the significance of effort reduction achieved with **FORMAT**:

- $H_o$  The use of **FORMAT** tool does not have any impact on the effort necessary to adapt test models for various products of a product family.

Our hypothesis significance level is 0.01 for rejecting the hypothesis [35].

## **5.2 Experimental Setup**

In this section, we explain the setup of our experiment that is designed to find answers for the 3 research questions and test  $H_o$ .

### **5.2.1 Factors**

In our experiments, our unique factor is **tool support**. Hereby, we are interested in the support provided by **FORMAT** as the tool for updating test models. So we can measure our factor at two levels in nominal scale: with tool support, without tool support.

### **5.2.2 Independent Variables**

In each of our experiments, we have only one independent variable and this is **product family**. We **randomly** selected 10 DTV products for the first experiment. On the other hand 5 products from the WAP product family is selected for the second experiment. The 3 input models are prepared for both DTV and WAP product families; a reference test model, a feature model and a mapping specification. These models are provided to all the participants. We can see selected and deselected

features for DTV products in Table 1 with a total number of 96 features. We can see selected and deselected features for WAP products in Table 2 with a total number of 144 features. The effort required for updating test models is related to the number of features deselected in the feature model.

**Table 1:** The list of DTV products and the number of feature selections for the first experiment [37].

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

**Table 2:** The list of WAP products and the number of feature selections for the second experiment.

Product ID	# of Selected Features	# of Deselected Features
P1	75	69
P2	55	89
P3	42	102
P4	117	27
P5	96	48

### 5.2.3 Dependent Variables

We measure the **Effort** with ratio scale in seconds, as the single dependent variable in both experiments.

#### 5.2.4 Participant Selection

Both companies have their own dedicated software testing groups. The test group in the first company tests various consumer electronics products including DTV systems. In the second company, WAP products are being tested. Both companies have similar employee profiles. Their test groups consist of both test engineers who graduated from an university and test technicians who graduated from two years educational program. For the first experiment **10 random participants** are selected from first companies test group and **5 random participants** are selected from second companies test group to run the second experiment. We can see the participant profile information at Table 3 and Table 4 for the first and the second experiments. We can see the participants as enumerated as  $S1$ ,  $S2$ , etc. in the first columns of these tables and job titles of each participants shown in second columns. As shown in Table 3 , eight participants are test engineers and 2 participants are test technicians. And also shown in Table 4 , three participants chosen for experiment 2 are test engineers and 2 of them are test technicians. We can see also experience of each participants from the third column.

#### 5.2.5 Experiment Design

As it is shown in Table 5, we have only one factor which is *tool (FORMAT) support* and two methods which are *with tool support* and *without tool support*. Both methods are tried by each participants. For the first experiment, firstly, each participants updated the model manually and then they repeated the process by using FORMAT. During manual updating the test models, each participant had to work on the test model but during using FORMAT they had to work only on the feature diagram. Because of this, we do not expect to affect the effort with the sequence of the methods. To validate this assumption in the second experiment, two participants (S2, S4) used FORMAT firstly and then they tried the next method by updating the test model

**Table 3:** Participant information and the amount of provided training (hours) for the first company [37].

Participant ID	Job Title	Experience (months)	FORMAT	MaTeLo	SPLIT
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

**Table 4:** Participant information and the amount of provided training (hours) for the second company.

Participant ID	Job Title	Experience (months)	FORMAT	MaTeLo	SPLIT
S1	Test Engineer	61	2	4	1
S2	Test Engineer	40	2	4	1
S3	Test Engineer	154	2	4	1
S4	Test Technician	120	2	4	1
S5	Test Technician	50	2	4	1

manually.

**Table 5:** The design of experiments.

	Factor: Tool Support	
	Level: with tool support	Level: without tool support
Experiment 1	10 participants	10 participants
Experiment 2	5 participants	5 participants

### 5.2.6 Preparation

Reference test model, product feature diagram and mapping specifications are all prepared by a senior test engineer who has a test experience more than five years and has extensive product domain knowledge in the company. This is the case for both companies and experiments although different engineers took this role.

We give these experiment material to participants before experiment. In the first company, they have already a reference test model for DTV product family. They are using reference model for test DTV products and they update reference model manually for different product configuration. Creating a feature diagram for DTV product family has taken 5 hours. And also creating a mapping that maps product features to reference model states has taken 3 hours.

In the second company, there was no available reference model being used. So, a reference model is created from scratch to be able to conduct the experiment. Creating feature diagram has taken 6 hours and creating a mapping which maps product features to reference test model states has taken 4 hours. There must be an experienced test engineer in the company, who has experience on MBT in particular; otherwise, the application of the approach would not be feasible. It took 15 hours to create this reference test model.

The participants have had a training about tools before the experiment, which are



FORMAT, MaTeLo and SPLOT. We recorded the training hours for each participant. We can see the hours of training in Table 3 and Table 4. Some of the participants from the first company have already had experience with MaTeLo. So, training durations vary according to the participant. (Table 3). In the second company, none of the participants had prior experience on MBT and SPLE. It was the first time they use MaTeLo, SPLOT and FORMAT.

### 5.2.7 Execution

For the first experiment we provided the reference test model, product feature diagram and mapping to every participant. In addition, we provided the product feature list, which has a ten products listed in Table 1. Then we assign the tasks. Participant's first task is updating reference test model manually with using MaTeLo for each product each of which has various features available. Thus, participant must find the reference model states which are deselected in the product feature model. Then, they need to update the probability values of transitions for these states as 0. Participant's second task is selecting and de-selecting product features from product feature diagram as for product configuration. Then the exported configuration is used for updating the reference test model with FORMAT. This task is defined in the Figure 2.

The second experiment was basically a replication of the first one with 5 participants and 5 products (Table 2) as part of a different product family. The set of tasks was the same. As a difference with respect to the first experiment, the order of the task assignments were varied as explained in Section 5.2.5. This change of experiment design was performed to check the validity of the assumption that the order of treatments does not have a significant impact on the results.

The tasks involved in our experiment have three main steps. We measured the effort for each of these steps as listed in the following:

***F***: Deselecting product features by using SPLOT.

***M***: Manually updating the reference test model with MaTeLo.

***T***: Automatically updating the reference test model by using the FORMAT tool.

For the first task, updating reference test model manually is defined by  $M$ . The second task, which is updating the test model automatically by using the FORMAT tool requires two steps; *i*) deselection of features and exporting the obtained configuration and *ii*) using this configuration as input for FORMAT to update the test model. Hence, the overall cost is defined by  $F + T$ . We explain and interpret the experiment results to address our research questions in the following chapter.

## CHAPTER VI

### RESULTS AND DISCUSSION

We evaluate the results based on the average effort required per product. The overall list of all the collected measurements are listed in Appendix C. We can calculate average effort with tool support and without tool support by average values of  $(F+T)$  and  $M$ , respectively. We can see these values in Table 6 and Table 7. As we can see from both experiments, when participants use **FORMAT**, the required effort reduces significantly.

The conducted t-tests<sup>1</sup> prove the significance of results as listed in Table 8. The set of participants was altogether assigned to two tasks. Hence, we used paired t-tests.  $P(T \leq t)$  one-tail and  $P(T \leq t)$  two-tail values turned out to be  $2.39 \times 10^{-8}$  and  $4.79 \times 10^{-8}$ , respectively for the first experiment. These values are 0.0005 and 0.001, respectively for the second experiment. Relatively smaller values for the second experiment can be explained by the involvement of a smaller group of subjects. However, P values are well below the threshold (0.01) for both experiments. We can reject the null hypothesis based on these results. That is, the use of **FORMAT** does indeed have a significant impact on the effort necessary to adapt test models for various products of a product family. We can see from Figure 5 and Figure 6 that the variation of effort for a chosen method is low but it is also shown that the required effort for two different method is significant.

In the next section, we try to find answers for the research questions with our findings. We can compare the initial investment with the effort gain by using the **FORMAT** tool.

---

<sup>1</sup>We used Microsoft Excel (2016) to obtain the results.

**Table 6:** Average effort required for model updates per DTV product with and without tool support [37].

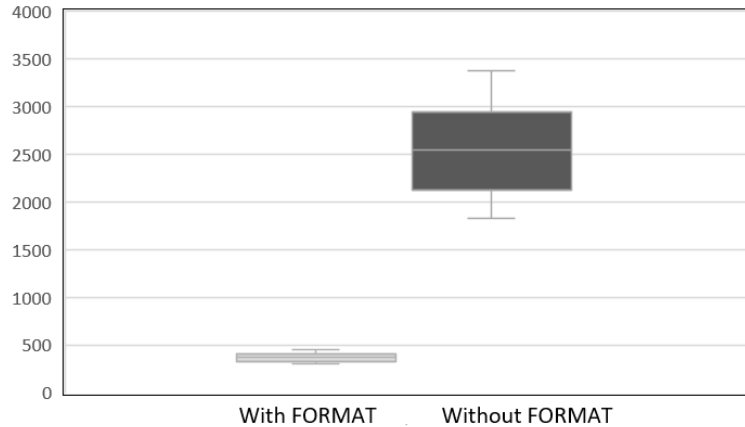
Participant ID	with tool support	without tool support
S1	306.3	1824
S2	339.0	2446
S3	350.9	2422
S4	372.2	2707
S5	403.3	2488
S6	402.7	3048
S7	318.8	2032
S8	411.0	2732
S9	455.7	3382
S10	440.4	3019
Average:	380.03	2610

**Table 7:** Average effort required for model updates per WAP product with and without tool support.

Participant ID	with tool support	without tool support
S1	304	2520
S2	487	4620
S3	363	3180
S4	425	4140
S5	662	4980
Average:	448.2	3888

**Table 8:** T-test results for the two experiments (paired two-samples for means).

	Experiment 1	Experiment 2
Mean	380.03	448.2
Variance	2573.73	18951.7
Observations	10	5
Pearson Correlation	0.92	0.91
Hypothesized Mean Difference	0	0
df	9	4
t Stat	-16.55	-8.58
P(T <sub>i</sub> =t) one-tail	$2.39 \times 10^{-8}$	0.0005
t Critical one-tail	1.83	2.13
P(T <sub>i</sub> =t) two-tail	$4.79 \times 10^{-8}$	0.001
t Critical two-tail	2.26	2.78



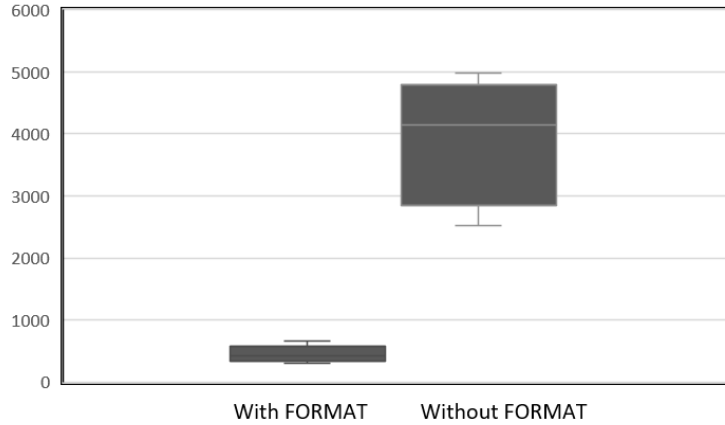
**Figure 5:** Variation of effort for MBT of DTV systems with and without FORMAT.

### ***6.1 Effort for Test Model Update without Tool Support***

We saw that results regarding the measured effort is inherently dependent on the types of participants and products involved in the experiments. In Figure 5, the box plot shows us 10 participants sampling distribution. Effort is saved in order of 1650 as minimum value and 3680 as maximum. Additionally, effort for 5 participants are showed in Figure 6 box-plot showed the sample distribution. Effort is saved in order of 2100 as minimum value and 5810 as maximum. Each participant's average effort cost is listed in the third columns of Table 6 and Table 7.

### ***6.2 Effort for Test Model Update with Tool Support***

We observed the automatic tasks are not subject to high variation of effort. The average effort is shown in the second columns of Table 6 and Table 7 for each participant per product. Creation of product configurations on SPLOT is the part that has to be performed manually. Hence, this part is responsible for the majority of the effort. For the first experiment, the minimum and maximum cost effort are recorded as 223 and 525 seconds. These values are 275 and 698 seconds for the second experiment.



**Figure 6:** Variation of effort for MBT of WAP systems with and without FORMAT.

### ***6.3 Effort for Tool Setup***

For both experiments the product feature diagram and mapping were created at first. Product feature diagram and mappings are created for one time only. We can use them for all the products in the same the product family. They remain unchanged if the system usage behavior (as such, the test model) does not change. Hence, participants of the experiments were not involved in creating product feature diagram and the corresponding mappings. These are prepared by a senior test engineer based on product requirement specifications. DTV product feature diagram preparation has taken 5 hours for the first experiment. And also mapping the features in the feature diagram to reference test model states have taken 3 hours. Thus, for preparation of product feature diagram and mappings, we calculated the overall cost of the initial effort as 8 hours (480 minutes).

We repeated the same tasks for the second experiment with the WAP product family. The preparation of the product feature diagram and the mapping has taken, 6 and 4 hours, respectively. So, the initial investment required 10 hours (600 minutes) in total for the WAP product family. In fact, MBT had not been applied for WAP products prior to our experiment. So, a test engineer prepared a reference test model for the WAP product family from scratch. This process took 15 hours; however, we

did not include this duration as part of the initial investment. This is because, our goal is to evaluate the cost of introducing SPLE approach to MBT with FORMAT. We assume that MBT is already in place and a test model is already in use before the application of our approach. Of course, this test model should be applicable to a family of products, possibly being adapted manually.

The overall effort is decreased from 2610 seconds to 380.03 which is the phase test model in DTV products. (See Table 6). This means to 2229.97 seconds we have gained for per product, it is also equal to 37 minutes. Since  $480/37 = 12.915 < 13$ , we can amortize cost of effort for 13 different products for DTV products. There exist 90 products in total in the DTV product family. So, the initial cost can be redemptory if we use feature diagram for at most 15% of the DTV product family.

Similarly, the cost of effort required for WAP products is decreased from 3888 seconds to 448.2 seconds on average (See Table 7). This means that we have gained 3439.8 seconds per product, which is equal to 57 minutes. Since  $600/57 = 10.53 < 11$ , we can amortize cost of effort for 11 different products for WAP products. There exist 140 products in total in the WAP product family. So, the initial cost can be redemptory if we use feature diagram for at most 7% of the WAP product family. If we include the effort of creating the test model (15 hours) as part of the of initial investment, the overall cost becomes 25 hours, which is equal to 1500 minutes. In this case,  $1500/57 = 26.32 < 27$ . So, we can amortize cost of effort for 27 different products, which is only 30% for the WAP products.

According to the these results, we have reach a decision that our SPLE approach supported by FORMAT provides significant effort reduction for adaptation of test models for MBT.

## ***6.4 Threats to Validity and Limitations***

To mitigate external validity threats [36], we performed two experiments in the settings of two different companies with two different product families. These experiments might be further replicated; however, the cost of conducting these experiments with real engineers and technicians is highly costly. We use real systems in our experiments so we mitigated the internal validity threats. We also involved real engineers and technicians from the industry as participants. Conclusion validity threats are decreasing by monitoring the activities of participants without intervening with them. Statistical hypothesis tests are used for evaluating the experiments results. The major assumption behind our approach that makes the tool support beneficial is regarding the stability of the product family. We assume that the created feature diagram and mappings can be reused for the same product family, which changes rarely.



## CHAPTER VII

### CONCLUSIONS AND FUTURE WORK

We developed a tool for supporting model based product line testing. The usage of the tool requires an initial (one-time) development of the 3 models required by our approach. First, a reference test model is specified to represent all possible usage scenarios for a family of systems rather than a single system. Second, a feature model is created for documenting variations among products of this family. Third, a specification defines a mapping between the other two models. Although the development of these tools incur a cost, the testing effort per product gets significantly reduced, which reduces the overall cost of testing a product family. This cost reduction is achieved by reusing the reference test model by automatically adapting it for different products.

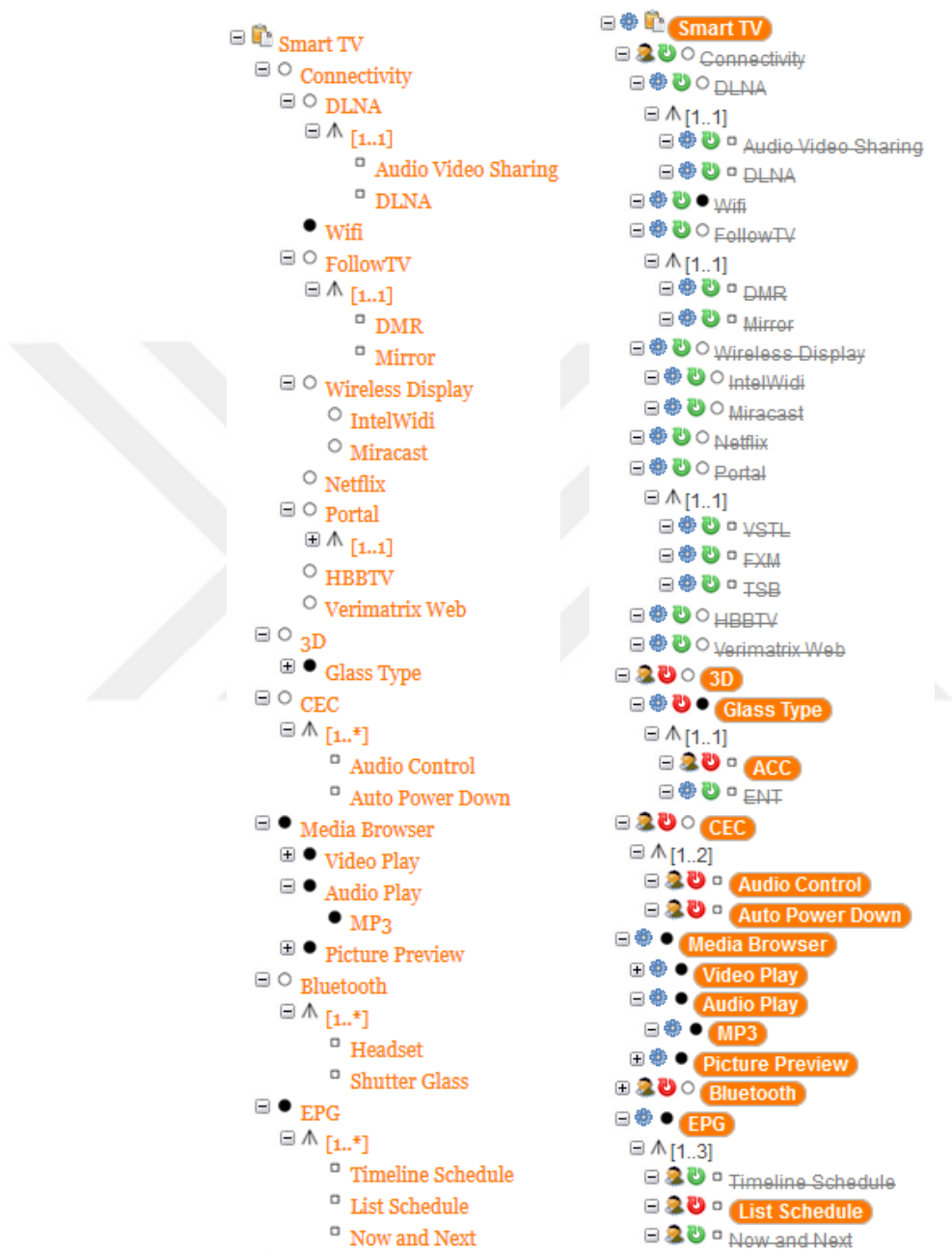
We conducted two controlled experiments in industrial settings to evaluate the effort reduction achieved with the use of our tool and approach. These experiments confirmed with significant measurements that the overall cost is reduced. We observed that the cost of initial investment can be amortized after the reuse of the reference test model for a small number of products. The number of products for amortizing costs turned out to be 13 and 11, as a result of the first and second experiments, respectively.

As future work, more experiments can be performed in various contexts to be able to further generalize the results. However, we should note that the cost of conducting such experiments with real engineers/technicians is high.

# APPENDIX A

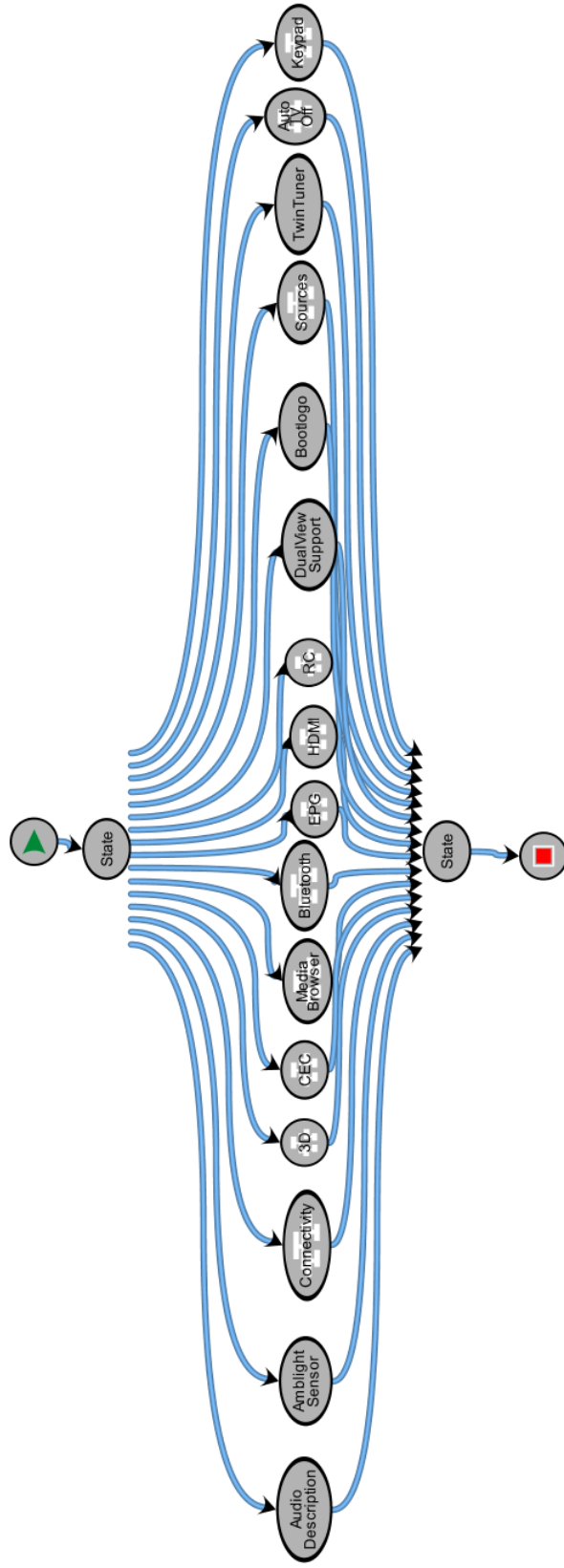
## DTV FEATURE AND TEST MODELS



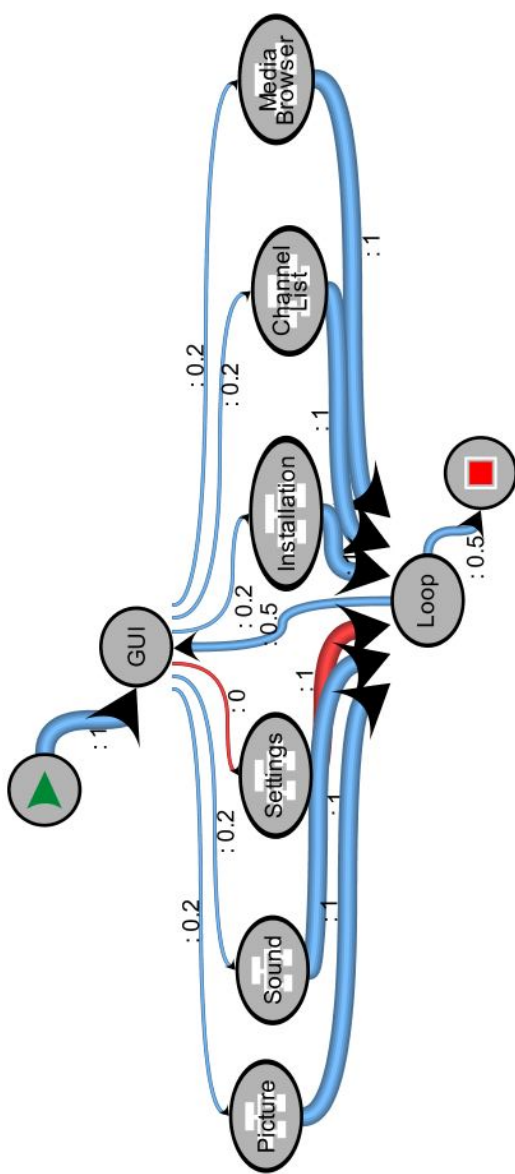


(a) A part of the feature model (b) Feature selections for a sample DTV product.

**Figure 7:** A part of the feature model created for the DTV product family and some of the feature selections regarding a product of this family [1].



**Figure 8:** The top-level test model regarding the DTV product family [37].



**Figure 9:** An updated test (sub)model regarding the DTV GUI states [1].

# APPENDIX B

## WAP TEST MODEL



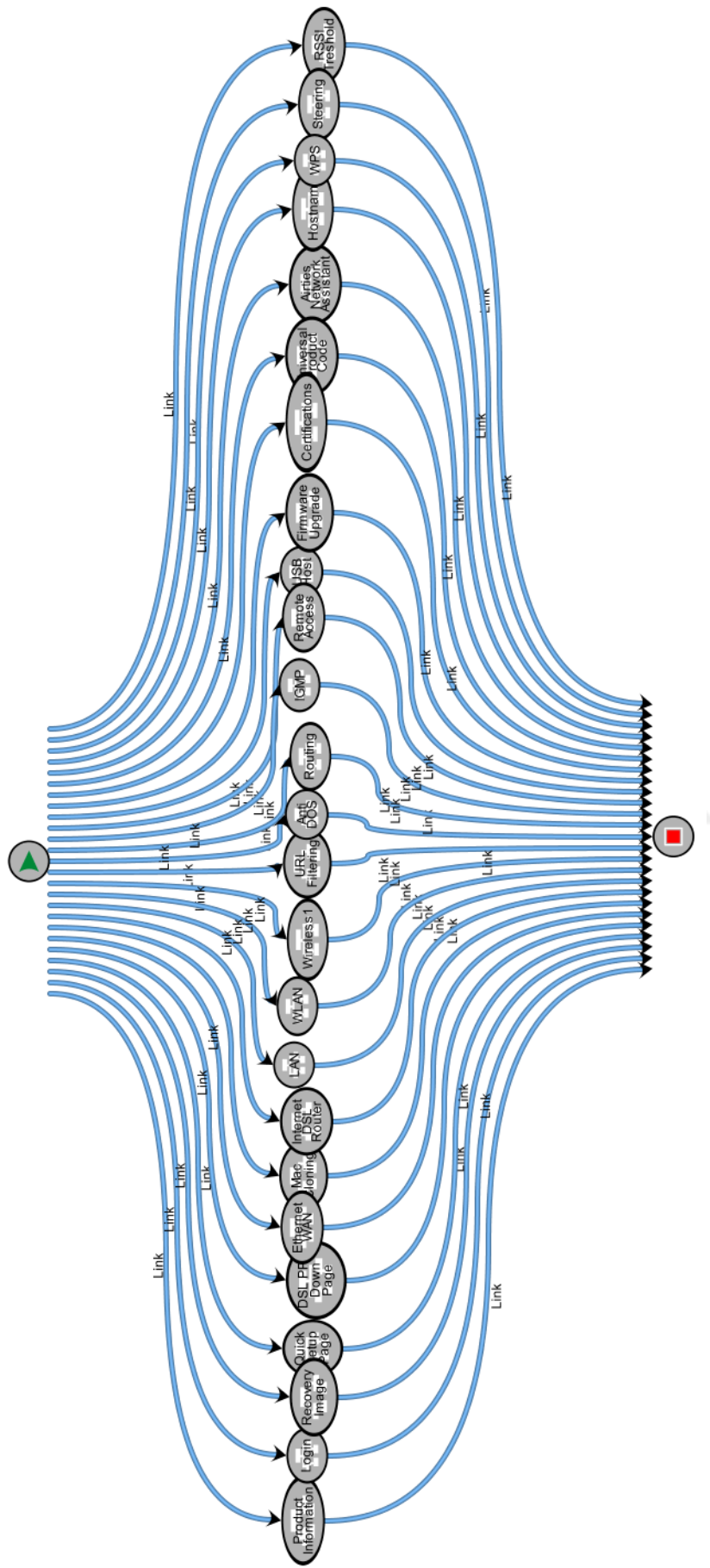


Figure 10: The top-level test model regarding the WAP product family.

# APPENDIX C

## COLLECTED MEASUREMENTS DURING THE EXPERIMENTS





**Table 9:** Measurements collected during the first experiment with DTV systems as the experimental objects [37]; Subject ID (**S#**), effort for Manual updates (**M**), Feature selection effort (**F**) and effort for automated updates performed with the Tool (**T**), all in seconds.

Subject	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	12
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

Subject	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

**Table 10:** Measurements collected during the second experiment with WAP systems as the experimental objects; Subject ID (S#), effort for Manual updates (M), Feature selection effort (F) and effort for automated updates performed with the Tool (T), all in seconds.

Subject	S1			S2			S3			S4			S5		
	F	M	T	F	M	T	F	M	T	F	M	T	F	M	T
P1	275	2100	10	394	3650	12	324	3110	13	307	3870	13	638	4750	14
P2	281	2570	11	530	3770	13	376	2860	15	440	3550	14	576	4320	16
P3	276	2910	13	412	4740	14	313	3690	11	458	3740	12	698	4970	16
P4	348	2160	12	513	5200	12	339	2670	14	335	4610	12	687	5810	15
P5	279	2860	15	519	5740	16	397	3570	13	521	4930	13	629	5050	21
Total	1459	12600	61	2368	23100	67	1749	15900	66	2061	20700	64	3228	24900	82

## References

- [1] C. S. Gebizli and H. Sozer, “Model-based software product line testing by coupling feature models with hierarchical markov chain usage models,” in *Proceedings of the 6th IEEE International Workshop on Model-Based Verification and Validation*, (Vienna, Austria), pp. 278–283, 2016.
- [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] J. Boberg, “Early fault detection with model-based testing,” in *Proc. of the 7th ACM SIGPLAN workshop on ERLANG*, pp. 9–20, 2008.
- [4] K. Pohl, G. Bockle, and F. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] A.Pretschner, *Proceedings of the International Symposium of Formal Methods Europe*, ch. Model-Based Testing in Practice, pp. 537–541. Springer Berlin Heidelberg, 2005.
- [10] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.
- [11] E. Dustin, J. Rashka, and J. Paul, *Automated Software Testing: Introduction, Management, and Performance*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.
- [12] I. Schieferdecker, “Model-based testing,” *IEEE Software*, vol. 29, no. 1, pp. 14–18, 2012.

- [13] L. Briand and Y. Labiche, “A uml-based approach to system testing,” *Software and Systems Modeling*, vol. 1, no. 1, pp. 10–42, 2002.
- [14] T. Chow, “Testing software design modeled by finite-state machines,” *IEEE Transactions on Software Engineering* 4, vol. 4, no. 3, pp. 178–187, 1978.
- [15] 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.
- [16] D. Harel, “Statecharts: A visual formalism for complex systems,” *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] D. Batory, “Feature models, grammars, and propositional formulas,” in *Proc. of the 9th International Conference on Software Product Lines*, pp. 7–20, 2005.
- [23] 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.
- [24] 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.

- [25] 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.
- [26] S. Oster, A. Wbbecke, 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.
- [27] A. Reuys, E. Kamsties, K. Pohl, and S. Reis, “Model-based system testing of software product families,” in *Proc. of the 17th International Conference on Advanced Information Systems Engineering*, pp. 519–534, 2005.
- [28] E. Olimpiew and H. Gomaa, “Model-based test design for software product lines,” in *Proc. of the 12th Int. Conference on Software Product Lines*, pp. 173–178, 2008.
- [29] H. Samih and R. Bogusch, “MPLM - MaTeLo product line manager: [relating variability modelling and model-based testing],” in *Proc. of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, pp. 138–142, 2014.
- [30] H. Samih, H. L. Guen, R. Bogusch, M. Acher, and B. Baudry, “Deriving usage model variants for model-based testing: An industrial case study,” in *Proc. of the 19th International Conference on Engineering of Complex Computer Systems*, pp. 77–80, 2014.
- [31] 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.
- [32] S. Oster, I. Zorcic, F. Markert, and M. Lochau, “MoSo-PoLiTe: tool support for pairwise and model-based software product line testing,” in *Proc. of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, pp. 79–82, 2011.
- [33] 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 *Proc. of the 7th IEEE International Conference on Software Testing, Verification and Validation*, pp. 51–60, 2014.
- [34] 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>.
- [35] T. Dyba, V. Kampenes, and D. Sjoberg, “A systematic review of statistical power in software engineering experiments,” *Information and Software Technology*, vol. 48, no. 8, pp. 745 – 755, 2006.

- [36] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering*. Springer-Verlag, 2012.
- [37] C. S. Gebizli, *Automated Refinement of Models for Model-Based Testing*. PhD thesis, Ozyegin University, 2017.

