MODEL-BASED TESTING OF SOFTWARE REUSE

CENG3001, SUMMER PRACTICE 1

REMZI BULUTLU
remzibulutlu@posta.mu.edu.tr

Friday 7th May, 2021

Abstract

Systems are becoming more complex day by day, and they are needed to be well traced in terms of correspondence between requirements, models, and test cases. The topic of this report is testing of reusable software. The objective of reuse is creating new systems using existing ones or their components. Average saving potential of software reuse will be guessed up to 40% of total development costs. There are some techniques that can help to trace effectively like feature modeling and model-based testing. In this report, we present feature modeling and model-based testing. Explanations are supported by examples of online shop and mobile phone models.

1 Introduction

Many products we use today have to be multi-functional in order to meet our different needs in the same product. But at the same time, these combined properties cause increase in the complexity of the products. Trying to preserve the ability to monitor and control interactions between models, code, and test cases which are used for products that have increased complexity (the system which is tested) can be challenging, costly and time consuming. To present our solution to those challenging cases, it will be better to start with the keywords and what they mean. In computer science and software engineering, reusability is the use of existing assets in some form within the software product development process; these assets are products and by-products of the software development life cycle and include code, software components, test suites, designs and documentation[1]. Software reuse is the use of existing software, or software knowledge, to build new software[2]. Software product lines are sets of products with similar features, but differences in appearance or price and software reuse are the use of existing software or software knowledge. Feature modeling is a compact representation of all the products of the software product line in terms of feature. Model-based testing is an application of model-based design for designing and also executing artifacts to perform software testing or system testing. State machine diagram is a type of behavioral diagram in the Unified Modeling

Language (UML) that shows transitions between various objects. Event Sequence Graph (ESG) is a simple powerful formalism for capturing the behavior of a variety of interactive systems that include real-time, embedded systems, and graphical user interfaces. A collection of ESGs is proposed as a model of an interactive system. This collection is used for the generation of tests to check for the correctness of system behavior in the presence of expected and unexpected input event sequences. Creating a feature model (or feature diagram) allows us to see all the features and functions of a product and allow us to keep in control as the complexity of the product increases. In this report we show feature model of a mobile phone that we created. In addition, model-based testing helps us to understand better the system tests of a product and find problems more easily. In this report we show online shop state machine diagram that we created.

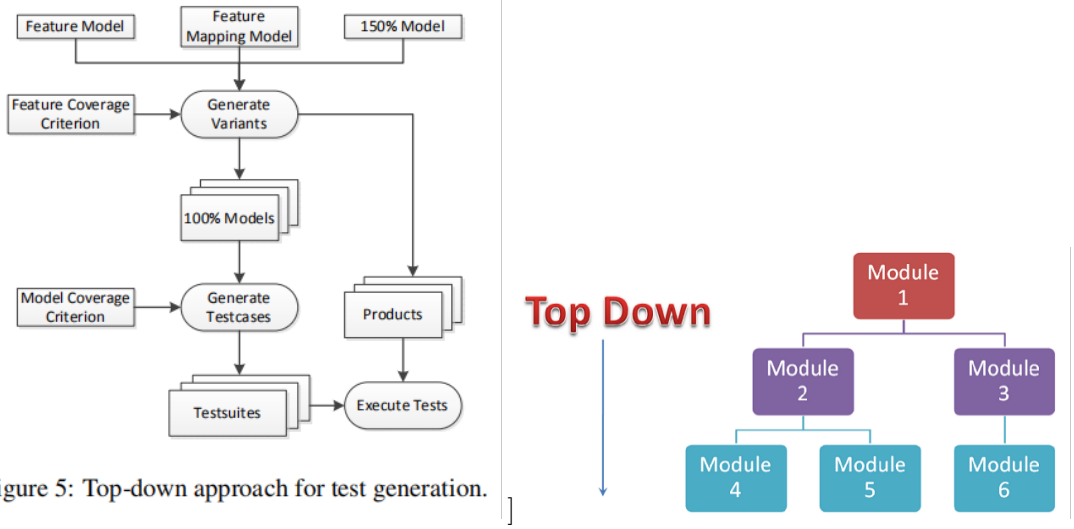
2 Related Work

Models can also be used for testing. The corresponding technique is called model-based testing (MBT) and there are many different approaches to it.[3] Because of the growing complexity of systems, it is necessary to reduce the effort for testing without reducing the test quality. As Weißleder and Lackner stated[4], model-based test design automation is a powerful approach to reach this goal and it can be used to automatically derive test cases from models. Also to ensure that the system under test behaves well in unexpected situations caused by undesirable events; Belli, Budnik, and White introduced an event-based modeling and testing approach based on event sequence graphs (ESG).[5]

3 Proposed Approach

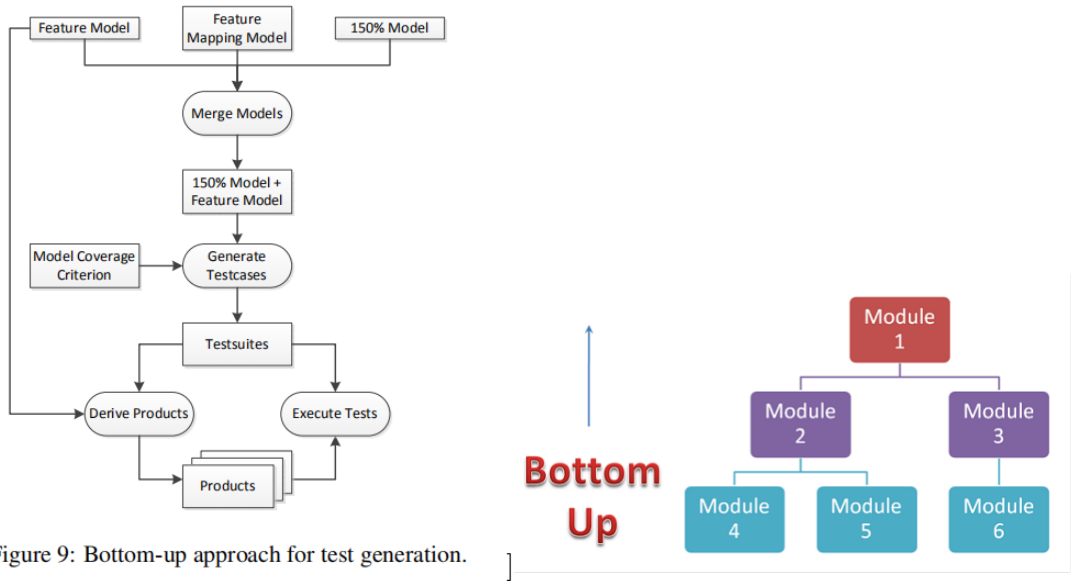
Models can be used to represent the desired behavior of a system under test. We present model-based testing that helps us to understand the system tests of a product and find problems better. There are two approaches in model-based testing; top-down and bottom-up approaches.

Top-down approach: In the top-down approach, we first derive a set of product variants from the feature model, derive the set of corresponding 100% models, and apply standard model-based testing to each 100% model. This approach as presented for state machines can also be applied to feature models. They can also be used to automatically derive a representative set of product variants. With this approach, it is possible to obtain an early prototype. Moreover, critical Modules are tested on priority; major design flaws could be found and fixed first[6].



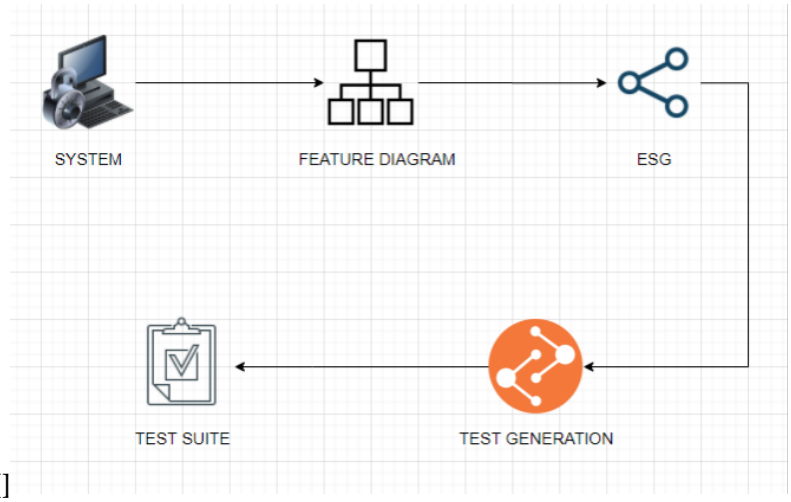
[] Figure 5: Top-down approach for test generation.]

Bottom-up approach: The idea of the bottom-up approach is contrary to the top-down approach. Here, we create test cases based on the 150% state machine and match the resulting sequences to single product variants, afterwards[7]. With this approach, no time is wasted waiting for all modules to be developed and fault localization is easier[8].



[] Figure 9: Bottom-up approach for test generation.]

In our study we present 2 study cases, online shop website and a basic mobile phone. At this point, it will be good to explain what meaning positive and negative testings are. Positive testing is the type of testing works on the system by providing the valid data as input. It is performed only for expected conditions. If this test works successfully, that means system doesn't have errors in terms of validity. Negative Testing is the type of testing works on the system by providing invalid data as input. It is performed for unexpected conditions. If this testing works successfully, that means system has errors in terms of invalidity. We first create feature models of our cases and then we create state machine diagrams(event sequence graphs) by using their feature models. Finally we can generate tests of them by using their event sequence graphs and finally we have the test suite.



[]

4 Case Studies

4.1 Online Shop Website

In this study, we first observed an existing feature model and its existing state machine diagram as shown below. In this existing feature model, the topmost feature contains the name of the product line(Online Shop). Four features are connected to it: The features Catalog, Payment, and Security are connected to the top-most feature by arcs with fulfilled circles at their end, which describe that these three features are mandatory, i.e., exist in every product variant. The Search feature is optional, which is depicted by using an arc that ends with an empty circle. This hierarchy of features is continued. For instance, the feature Payment contains the three subfeatures Bank Account, ECoins, and Credit Card, from which at least one has to be selected for each product variant. The subfeatures High and Low of the feature Security are alternative, which means that exactly one of them has to be chosen for each product variant. Furthermore, there is a textual condition that states that credit cards can only be selected if the provided security level is high[9].

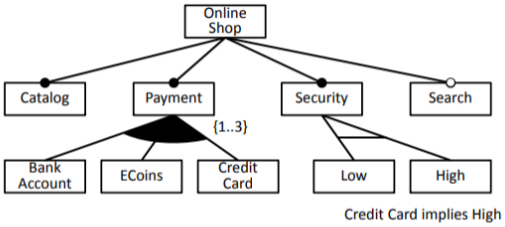


Figure 1: Feature model for online shops.

[]

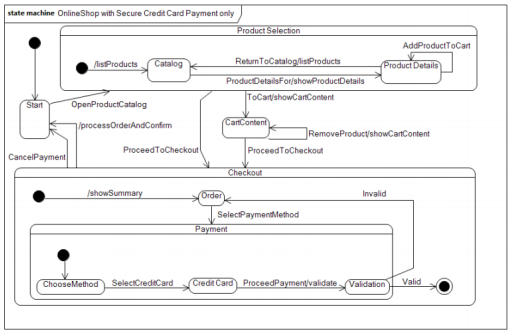
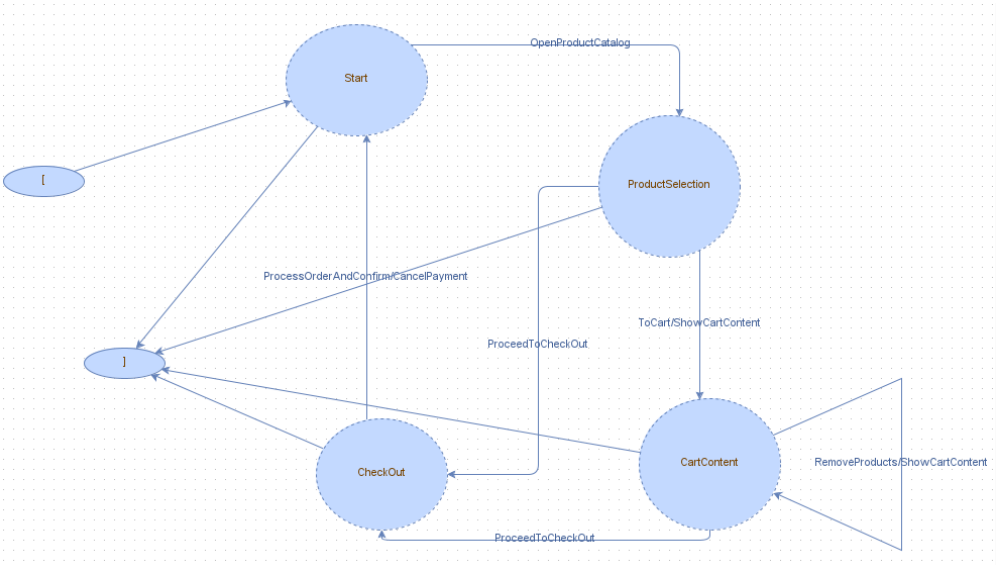


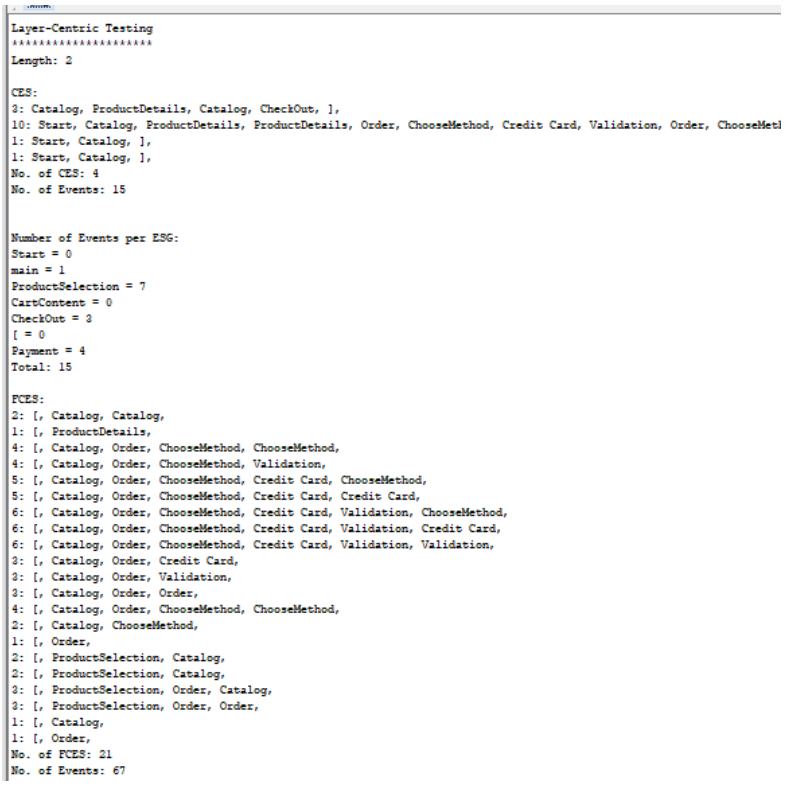
Figure 2: Online shop state machine diagram for one product variant.

]

Afterwards, in order to generate and test the event sequence graph of this study , we created new state machine diagram on ADT ToolSuite[10] by using the properties of this existing diagram as shown below.



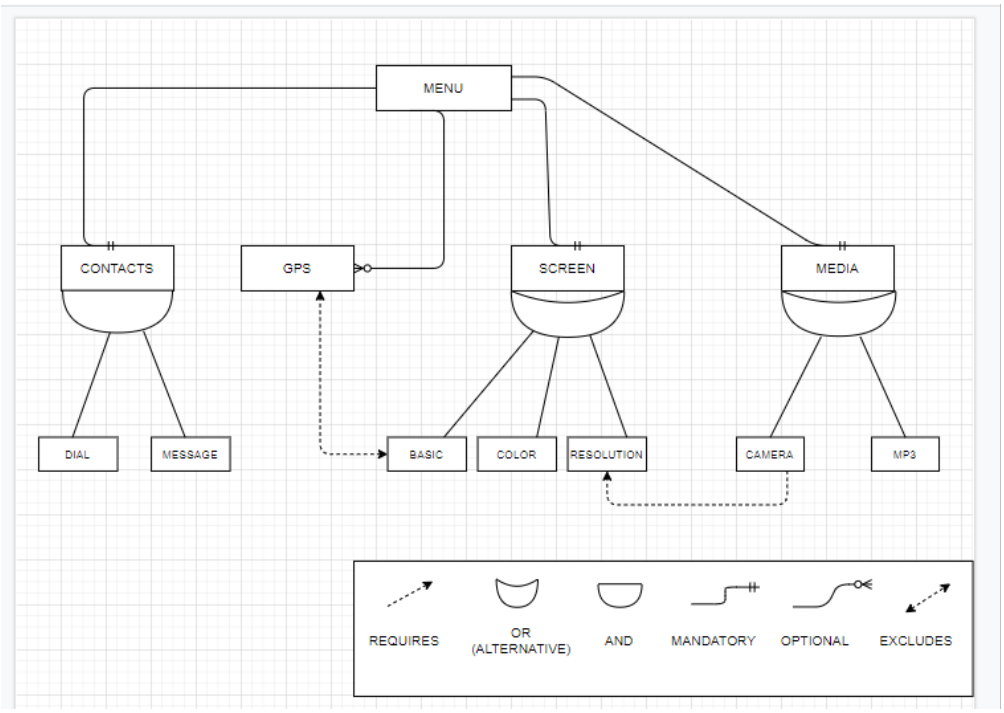
After we generated the test on this ToolSuite we were given 67 events as shown below



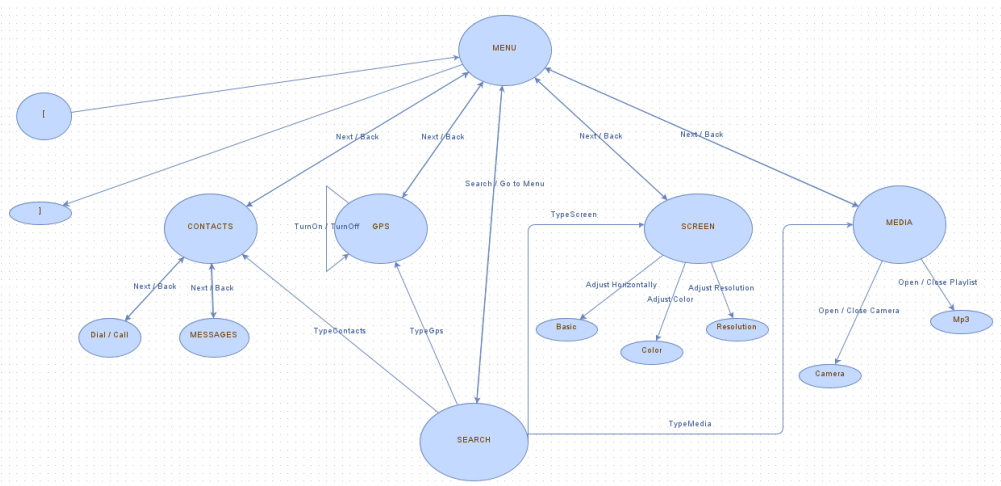
```
Layer-Centric Testing
**********
Length: 2

CES:
3: Catalog, ProductDetails, Catalog, CheckOut, ],
10: Start, Catalog, ProductDetails, ProductDetails, Order, ChooseMethod, Credit Card, Validation, Order, ChooseMetl
1: Start, Catalog, ],
1: Start, Catalog, ],
No. of CES: 4
No. of Events: 15


Number of Events per ESG:
Start = 0
main = 1
ProductSelection = 7
CartContent = 0
CheckOut = 3
[ = 0
Payment = 4
Total: 15

FCES:
2: [, Catalog, Catalog,
1: [, ProductDetails,
4: [, Catalog, Order, ChooseMethod, ChooseMethod,
4: [, Catalog, Order, ChooseMethod, Validation,
5: [, Catalog, Order, ChooseMethod, Credit Card, ChooseMethod,
5: [, Catalog, Order, ChooseMethod, Credit Card, Credit Card,
6: [, Catalog, Order, ChooseMethod, Credit Card, Validation, ChooseMethod,
6: [, Catalog, Order, ChooseMethod, Credit Card, Validation, Credit Card,
6: [, Catalog, Order, ChooseMethod, Credit Card, Validation, Validation,
3: [, Catalog, Order, Credit Card,
3: [, Catalog, Order, Validation,
3: [, Catalog, Order, Order,
4: [, Catalog, Order, ChooseMethod, ChooseMethod,
2: [, Catalog, ChooseMethod,
1: [, Order,
2: [, ProductSelection, Catalog,
2: [, ProductSelection, Catalog,
3: [, ProductSelection, Order, Catalog,
3: [, ProductSelection, Order, Order,
1: [, Catalog,
1: [, Order,
No. of FCES: 21
No. of Events: 67
```

4.2 Mobile Phone

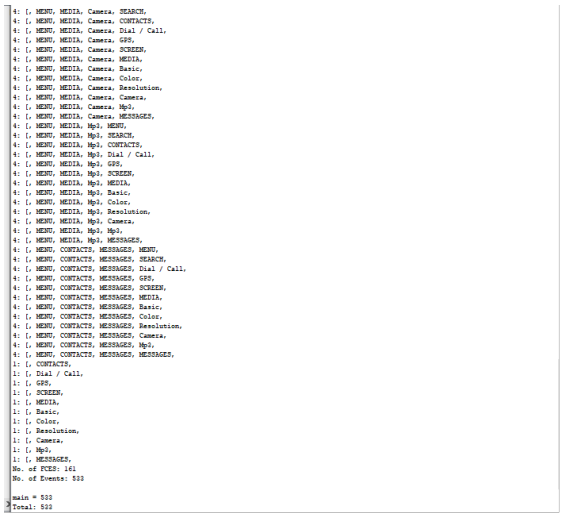
In this study, we first created a feature model of a basic mobile phone. What we expect from a mobile phone as mandatory features are basically menu that shows us a map of mobile phone functions, dialing, messaging, having basic screen properties, and media player. As an example, an optional property we can expect from a basic mobile phone is GPS feature that requires basic screen properties. As shown below, we combined those features in a feature model:



Then we created a state machine diagram of it by using this feature model. The state machine diagram is shown below.



After we generated the test on this ToolSuite we were given 533 events as shown below.



```
4: [, MENU, MEDIA, Camera, SEARCH,
4: [, MENU, MEDIA, Camera, CONTACTS,
4: [, MENU, MEDIA, Camera, Dial / Call,
4: [, MENU, MEDIA, Camera, GPS,
4: [, MENU, MEDIA, Camera, SCREEN,
4: [, MENU, MEDIA, Camera, MEDIA,
4: [, MENU, MEDIA, Camera, Basic,
4: [, MENU, MEDIA, Camera, Color,
4: [, MENU, MEDIA, Camera, Resolution,
4: [, MENU, MEDIA, Camera, Camera,
4: [, MENU, MEDIA, Camera, Mp3,
4: [, MENU, MEDIA, Camera, MESSAGES,
4: [, MENU, MEDIA, Mp3, MENU,
4: [, MENU, MEDIA, Mp3, SEARCH,
4: [, MENU, MEDIA, Mp3, CONTACTS,
4: [, MENU, MEDIA, Mp3, Dial / Call,
4: [, MENU, MEDIA, Mp3, GPS,
4: [, MENU, MEDIA, Mp3, SCREEN,
4: [, MENU, MEDIA, Mp3, MEDIA,
4: [, MENU, MEDIA, Mp3, Basic,
4: [, MENU, MEDIA, Mp3, Color,
4: [, MENU, MEDIA, Mp3, Resolution,
4: [, MENU, MEDIA, Mp3, Camera,
4: [, MENU, MEDIA, Mp3, Mp3,
4: [, MENU, MEDIA, Mp3, MESSAGES,
4: [, MENU, CONTACTS, MESSAGES, MENU,
4: [, MENU, CONTACTS, MESSAGES, SEARCH,
4: [, MENU, CONTACTS, MESSAGES, Dial / Call,
4: [, MENU, CONTACTS, MESSAGES, GPS,
4: [, MENU, CONTACTS, MESSAGES, SCREEN,
4: [, MENU, CONTACTS, MESSAGES, MEDIA,
4: [, MENU, CONTACTS, MESSAGES, Basic,
4: [, MENU, CONTACTS, MESSAGES, Color,
4: [, MENU, CONTACTS, MESSAGES, Resolution,
4: [, MENU, CONTACTS, MESSAGES, Camera,
4: [, MENU, CONTACTS, MESSAGES, Mp3,
4: [, MENU, CONTACTS, MESSAGES, MESSAGES,
1: [, CONTACTS,
1: [, Dial / Call,
1: [, GPS,
1: [, SCREEN,
1: [, MEDIA,
1: [, Basic,
1: [, Color,
1: [, Resolution,
1: [, Camera,
1: [, Mp3,
1: [, MESSAGES,
No. of FCES: 161
No. of Events: 533

main = 533
Total: 533
```

5 Discussion

One of the main problems can be encountered is being German of the ADT Toolsuite language instead of being English. Moreover, sometimes drawing and designing can be challenging to build feature models. Because of model-based testing seems a new testing method of test automation, testers will need to learn this new concept of modeling over traditional testing methods.

6 Conclusion

In our study, we presented different approaches to the automatic test design for software product lines and software reuse. We explained advantages of those approaches, and we described feature modeling and model-based testing. We designed and depicted of our case studies. We generated test of them by using their event-sequence graphs and finally we had the test suite for each case study. Model-based testing generates tests automatically from feature models and this testing provides us describing test environments and test strategies, generating test cases, test execution and test design quality, possibility to trace models, code and test cases which are used for the tested system. To conclude, in order to overcome challenges what we stated in section 2, we can use model-based testing.

7 References

[1] Lombard Hill Group (October 22, 2014). "What is Software Reuse" https://web.archive.org/web/20141023013122/http://lombardhill.com/what*r**euse.htm*

[2] W. B. Frakes and Kyo Kang, "Software reuse research: status and future," in IEEE Transactions on Software Engineering, vol. 31, no. 7, pp. 529-536, July 2005, doi: 10.1109/TSE.2005.85.

[3] Mark Utting, Alexander Pretschner Bruno Legeard (2012): A Taxonomy of Model-Based Testing Approaches. Softw. Test. Verif. Reliab. 22(5), pp. 297–312, doi:10.1002/stvr.456.

[4] Weißleder, S., Lackner, H. (2013). Top-down and bottom-up approach for model-based testing of product lines. arXiv preprint arXiv:1303.1011./page:83.

[5] Belli, F., Budnik, Ch. J., White, L., Event-based Modeling, Analysis and Testing of User Interactions: Approach and Case Study, Software Testing, Verification and Reliability, Vol. 16, No. 1, pp. 3-32, John Wiley Sons, Ltd, 2006

[6] Top-down integration: https://www.guru99.com/integration-testing.html9

[7] Weißleder, S., Lackner, H. (2013). Top-down and bottom-up approach for model-based testing of product lines. arXiv preprint arXiv:1303.1011./page:90.

[8] Bottom-up integration: https://www.guru99.com/integration-testing.html8

[9] Weißleder, S., Lackner, H. (2013). Top-down and bottom-up approach for model-based testing of product lines. arXiv preprint arXiv:1303.1011./page: 83-84

[10] ADT ToolSuite TestSuite Desginer, http://download.ivknet.de/