International Journal of Engineering & Technology, 7 (4.15) (2018) 63-68

SPC

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper



Coverage-based Approach for Model-based Testing in Software Product Line

Rabatul Aduni Sulaiman[1]*, Dayang Norhayati A. Jawawi[1], Shahliza Abd Halim[2]

[1]*Software Engineering Department, Faculty of Computing, University Technology Malaysia, 81300 Skudai, Johor, Malaysia*
[2]*Software Engineering Department, Faculty of Computer Science and Information System, University Tun Hussein Onn Malaysia, 86400 Parit Raja, Johor, Malaysia*
**Corresponding author E-mail: raduni2@live.utm.my*

Abstract

Rapid Quality assurance is an important element in software testing in order to produce high quality products in Software Product Line (SPL). One of the testing techniques that can enhance product quality is Model-Based Testing (MBT). Due to MBT effectiveness in terms of reuse and potential to be adapted, this technique has become an efficient approach that is capable to handle SPL requirements. In this paper, the authors present an approach to manage variability and requirements by using Feature Model (FM) and MBT. This paper focuses on modelling the integration towards enhancing product quality and reducing testing effort. Further, the authors considered coverage criteria, including pairwise coverage, all-state coverage, and all-transition coverage, in order to improve the quality of products. For modelling purposes, the authors constructed a mapping model based on variability in FM and behaviour from statecharts. The proposed approach was validated using mobile phone SPL case study.

Keywords: *Mapping Model; Model-based Testing; Software Product Line; Software Testing.*

1. Introduction

At present, most users remain unsatisfied with product solutions' testing standards. This is attributed to an increased level of complexity in multiple kinds of features that a product has to handle. In addition, users are also concerned about the quality of product variants. The product needs to be managed well to ensure that product requirements and quality meet the standard provided.

Software Product Line (SPL) is one of the methods that is capable to handle multi-size products [1]. This method aims to minimize cost and time, enhance product quality, and improve product demand in the market [2]. New products can be derived via reusing and integrating software assets. Commonality and variability that exist in software assets distinguish products from one another. SPL variability describes the properties of each product. Nowadays, a model-driven technique is capable of supporting an engineering process; it brings model-based testing technique (MBT) as one of the efficient techniques which can be utilized [3]. The model can be used to show important information and to make information more understandable through formal description [4]. In SPL, model can support both manual and automatic test generation based on the defined model [5]. Much effort in software development involves developing model representing systems under test. However, in SPL scope, this kind of effort can be minimized by managing commonality and variability of the development model [6]. Quality assurance is an important element in SPL testing, which can help to assure that high-level product features can fulfil customer satisfactions and requirements [7]. MBT helps to enhance SPL quality through implementing modelling technique. Model-based test case generation also produces test cases with higher interaction coverage as compared to other techniques in the SPL [3]. In addition, the usage of MBT in SPL can help the testers to recognize defects in requirements in the early stage of the software development life cycle [8]. Consequently, the cost of development can be minimized. In addition, in MBT, behaviour model is commonly used to describe possible variation points. However, behavioural model still lacks in terms of the relationship between features and balanced information regarding the validity of the features selected. In order to solve this issue, the relation between features and behaviour model needs to be established to produce features with more semantics. Specifically, behaviour model serves MBT via selection of possible paths through graph constructed in the model.

In our previous work, we have proposed a product line modelling methodology that integrates Feature Model (FM) and behaviour model, known as statechart. We applied the proposed methodology to a mobile phone case study; a basic case study in the SPL. The proposed work on the case study revealed that modelling effort can be minimized. However, a complete description on how the features in FM were selected was not elaborated as the investigation had not progressed to that extent in the previous work. The work only described the functionalities of FM that were capable in managing variability as well as the needs of FM in MBT as a statechart's complement for test generation. Further, the previous work also did not describe in details the functionality and the definition of FM notations.

In this paper, we propose an extension of FM feature selection in order to select useful features from FM. The main idea of this work is to apply test coverage for model in MBT, which would conclude the previously proposed work. FM and statechart are two models applied in this work. We consider these two models for MBT in SPL since their functionalities can complement each other. Subsequently, the approach will be evaluated using mobile phone SPL case study.

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The paper is outlined as follow. Section 2 describes related works. Section 3 elaborates definition of FM used in this work. Section 4 provides the definition of statechart. Section 5 provides the proposed approach. Section 6 provides results obtained from the case study and discussion on the results. Lastly, Section 7 provides conclusions and future works for the paper.

2. Related Works

In this section, we describe related works on MBT that have utilized FM and UML model. The ability of software testing to enhance product quality has led to a high demand for testing across many domains. In SPL testing domain, FM plays a critical role as it is used to manage variability. As FM represents features through symbols, this leads to generating test cases that are insufficient to yield a valid result. The integration of models can help to fulfil inadequacy of each model. However, there exist challenges to integrate the models due to their different characteristics. In order to integrate the models, several techniques have been proposed in the literature.

In [7] proposed two approaches comprising top-down and bottom-up approaches. These two approaches integrate FM with statechart. Subsequently, the researchers compared the efficiency of product variants covered and the system behaviour covered [7]. Works done by [3, 9] also used FM and statechart for test generation. However, they considered coverage criteria in the model. In our work, a pairwise coverage is implemented in FM, whereas statechart implements all-states, all-transition, and all-transition-pairs. In contrast to our work, researchers such as [3, 9] focused on defining and evaluating coverage criteria for models in detail, whereas we only set the aim of coverage criteria without investigating the condition in great detail, followed by comparing the results. In our proposed approach, pairwise coverage, all-states, and all-transition coverage are utilized for automation purpose. Work by [10] utilized functionality of statechart as test model. However, they only utilized a single test model known as 150% test model consisting of multiple states and conditions. In comparison to our approach, several test models are provided depending on the features from FM. In [10] utilized IBM modelling Rhapsody for modelling purposes and ATG tools to trim the test model. In comparison to this, we utilize Eclipse plugin Papyrus for modelling and manually select our test suite for test generation. Furthermore, our objective is also distinguishable as compared to the existing work which tried to analyse the efficiency of coverage criteria. In our work, we aim to obtain a set of test cases while at the same time trying to reduce testing effort.

Our previous work had only implemented the approach based on FM and test model. In this work, we apply coverage criteria to reduce feature selection possibilities with an addition of model-based coverage.

3. Definition of Feature Model (FM)

The use of model can reduce complexity and increase users' understanding of software. Here, we present FM to describe aforementioned SPL features. According to American English Webster's Dictionary, feature is defined as an important element of a characteristic. In Software Engineering community, feature is defined as a characteristic that is distinguishable from software item such as performance and functionality [10]. The feature has been used frequently to express variable and regular elements in the SPL [11]. The FM itself consists of elements known as features. Features are used to show the "logical set of product requirements". The requirements and information from users and developers can be managed well by using FM since it can be used to separate important information that users prefer [12].

Figure 1 depicts a feature model for mobile phone SPL case study, which will be used in this paper. The FM is depicted utilizing Pure::Variants tool, in order to show variability that exists in mobile phone industry [13]. An FM diagram is a tree structure consisting of multiple features. Feature variability is represented as arcs and grouping of features. There are four main feature groups comprising mandatory, optional, alternative, and or [14]. Up until now, the main standard of constructing an FM is non-existing, but the graphical notation from FODA method is commonly used in the SPL [15]. In Pure::Variants, individual features are represented in an acyclic graph form. The graph consists of nodes and edges. The nodes represent features that exist, whereas edges represent basic relations.

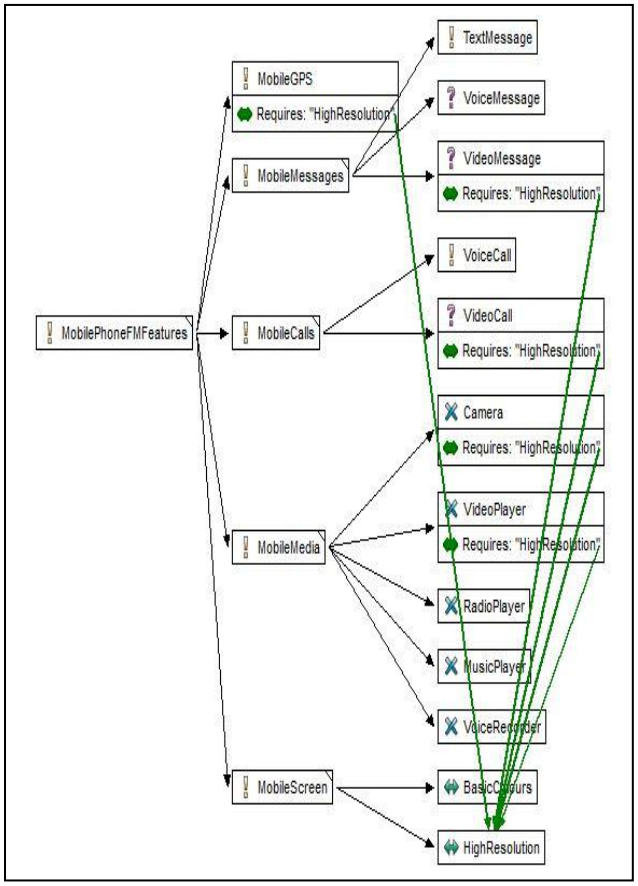


Fig 1: Mobile phone SPL FM

In our constructed FM, it consists of four relations, including mandatory, optional, alternative, and or. Five parent nodes of mandatory relations will always be included within the products listed. The list of parent nodes comprises of MobilePhone (MP), MobileMessage (MM), MobileCalls (MC), MobileGPS (MG), MobileMedia (MMe) and MobileScreen (MS). Other mandatory relations (i.e., TextMessage (TM) and VoiceCall (VC) will always be selected if the parent node is included in the FM. Optional features comprising VoiceMessage (VCM), VideoMessage (VDM) and VideoCall (VDC) can either be selected or not selected in the FM. Meanwhile OR group will select at least one feature if the parent feature is included. Examples of OR features include Camera (CM), VideoPlayer (VP), and MusicPlayer (MY). Alternative feature refers to one of the features that must be selected.

The list of alternative features comprises of BasicColours (BC), HighResolution (HR), RadioPlayer (RP) and VoiceRecorder (VRD). Requires relations in Figure 1 show alternative feature selections. The require relations indicate constraints that exist in the case study. These relations illustrate the implementation of optional feature such as VDM requires HR features to ensure that this feature will be selected. The details of feature configuration will be provided as follows. Concerning the mobile phone case study, we have listed all valid product configurations from overall

mobile phone products. This SPL consists of 18 features for mobile phone (Fmp) as per valid configuration as follows:

$$F_{mp} = \{MP, MM, MC, MG, MS, MMe, TM, VCM, VDM, VC, VDC, BC, HR, VP, CM, MY, RP, VRD\} \quad (1)$$

In Definition 1, FM is defined following definitions expressed in [9, 16]. Formal definition of FM is used as a fundamental calculation of valid feature selection and to map out feature to test model artefacts.

Definition 1 (FM):
FM is defined as follows:

$$FM = (F, W, C) \quad (2)$$

FM is defined as a set of features, *F*, comprising of whole nodes collection, W, with the consideration of constraint, C, (if it exists). In order to have a basic set of *FM*, FM_F is defined to denote a set of features from feature name, *F*, ($FM \in FM_F$). The whole nodes collection is separated, $w \in W$, into every node, which refers to one feature, $f \in F$, where the related node is denoted as $F \rightarrow W$.
The types of feature variability are described by using four types of relations as follows. For each relation, variability is defined in such a fashion that allows it to be distinguishable from other variability's definition.

Variability definitions:

$W_{man} \subseteq F \times F - if\,(f, f') \in D_{man}$ = *Mandatory* variability is represented f' as from feature f. (3)

$W_{opt} \subseteq F \times F - if\,(f, f') \in D_{opt}$ = *Optional* variability is represented as f' from feature f. (4)

$W_{alt} \subseteq F \times F - if\,(f, f') \in D_{alt}$ = *Alternative* variability is represented as $F' \subseteq F$ from the group of features, f. (5)

$W_{or} \subseteq F \times F - if\,(f, f') \in D_{or}$ = *Or* variability is represented as $F'\{\Phi\} \subseteq F$. (6)

Definition 2 (Overall FM):
Overall FM is defined as follows:

$$W = W_{man} \cup W_{opt} \cup W_{alt} \cup W_{or} \quad (7)$$

Overall FM is defined as the combination of all four relations comprising, *mandatory*, *optional*, *alternative*, and *or*. Definition 2 covers the whole FM tree structure. Existing constraint in features is not included in this definition.
Constraint definition is provided in Definition 3. Features combination creates relation restriction, which we formally indicate as constraint, *C*.

Definition 3 (FM constraints):
FM constraints are defined as follows:

$C_{req} \subset F \times F - if\,(f, f') \in F$ = Feature f *requires* feature f' in each product. (8)

$C_{exd} \subset F \times F - if\,(f, f') \in F$ = Feature f *excludes* feature f' in each product. (9)

In this paper, the implementation of FM is carried once pairwise coverage among the features of mobile phone case study has been located. The details of pairwise implementation are described in the following section.

Pairwise coverage can be achieved by applying a pairwise testing approach [17]. This approach is a well-known combinatorial testing technique. It is used to find software faults based on the interaction between features [18-19]. There are several algorithms which have been proposed to conduct pairwise testing approach, including IPOG [20] and IPOG-D [21]. In this paper, CITLAB framework is utilized to generate a set of valid features [22]. Furthermore, this framework helps to satisfy 100% pairwise coverage for mobile phone SPL based on FM. The results of the pairwise coverage implementation produce subsets of all potential products. The result can also be obtained through deriving valid pair of features. Valid pair of features is defined for understanding. Definition 4 lists out pair of features that have been categorized.

Definition 4 (FM pairwise coverage):
FM pairwise coverage is defined as follows:

$F \subset \{f_a, f_b\}$ = Valid combination of feature pairs $\{f_a, f_b\}$ from the list set of feature, F. (10)

$(C_{req}, C_{alt}) \subseteq F \times F - if\,(f, f') \in F$ Valid pairwise covering all pairwise interaction including requires and alternative constraints. (11)

Pairwise coverage conducted in CITLAB framework can explain combinatorial generation by using FM [23]. The structure of CITLAB integrated with the Eclipse framework helps to convert FM from pure::variants into CITLAB in a straightforward fashion. Currently, CITLAB does not provide any kinds of importer plugin that can support FM from pure::variants by design. Due to this, the current work converts model manually using CITLAB in order to generate valid feature pairs. As depicted in Figure 2, a code sample is provided to illustrate manual model conversion written in CITLAB framework. As CITLAB concepts support Boolean variables, all features involved, including their constraints are subsequently listed down. Related constraints are also stated in each of the variable. Upon executing IPOG in CITLAB, a list of valid pairs of features are obtained. The results of the features are provided in the result section.
From the list of valid results, only several feature pairs are utilized to produce new FM, as a test generation using this kind of generator has produced 12 sets of valid features. Due to limited space to list down all the valid features, only several feature pairs have been randomly selected from the results generated by CITLAB. Statechart modelling was proposed by [24]. This model is widely used in Software Engineering which considers the behaviour concept derived from Finite State Machine (FSM).

```
Parameters:
//MobileMessages
 Boolean TextMessage;
 Boolean VoiceMessage;
 Boolean VideoMessage;
 //MobileCalls
 Boolean VoiceCall;
 Boolean VideoCall;
 //MobileGPS
 Boolean MobileGPS;
 //MobileScreen
 Boolean BasicColours;
 Boolean HighResolution;
 //MobileMedia
 Boolean VideoPlayer;
 Boolean Camera;
 Boolean MusicPlayer;
 Boolean RadioPlayer;
 Boolean VoiceRecorder;
end
Constraints:

 # (TextMessage or VoiceMessage or VideoMessage) #
 # (VoiceCall or VideoCall) #
 # (BasicColours or HighResolution) #
 # (VideoPlayer or Camera or MusicPlayer or RadioPlayer or VoiceRecorder) #
 # (HighResolution => VideoMessage and VideoCall and Camera and VideoPlayer) #
 # (BasicColours => MobileGPS) #
end
```

Fig 2: Pairwise code for features selection

Functionalities of statechart including capability to simulate and ability to automate software derivation brands it as an effective model to be used in comparison to other models [10]. There are two main components in statechart comprised of state and transition. State refers to concurrent submachine whereas transition refers to the relationship between two states.

4. Definition of Statechart

In order for a statechart to be more understandable and standardized with FM, statechart condition used in this work is defined. Definition 5 expressed the proposed statechart.

Definition 5 (Statechart):
{*S*, *s*, *t* }is defined as a statechart equipped with a set of states, *S*, which consists of each state, *s*, and each state *s* is related by using transition, *t*.
In this paper, statechart is used for simulation and generation. Simulation refers to the flow of the system while generation refers to creation of test suites. Basic variants are derived based on the behaviour of the system. Figure 3 depicts the statechart based on the behaviour of a mobile phone. The behaviour illustrated in the case study relates to product variant that shows the process of user interaction. In the example, a statechart is chosen for one process of mobile phone; mobile phone messages. This product variant allows the sending of a message to receiver. Mobile users can choose the types of message to be sent by selecting either, (OpenMessageTab), (OpenVideoMessageTab) or (OpenVoiceMessageTab). In TextMessage state, (SendMessage) will send the user's message to the server. Server state validates it either as SuccessfullySent or FailToSend. Similar validation is applied on VideoMessage and VoiceMessage states.

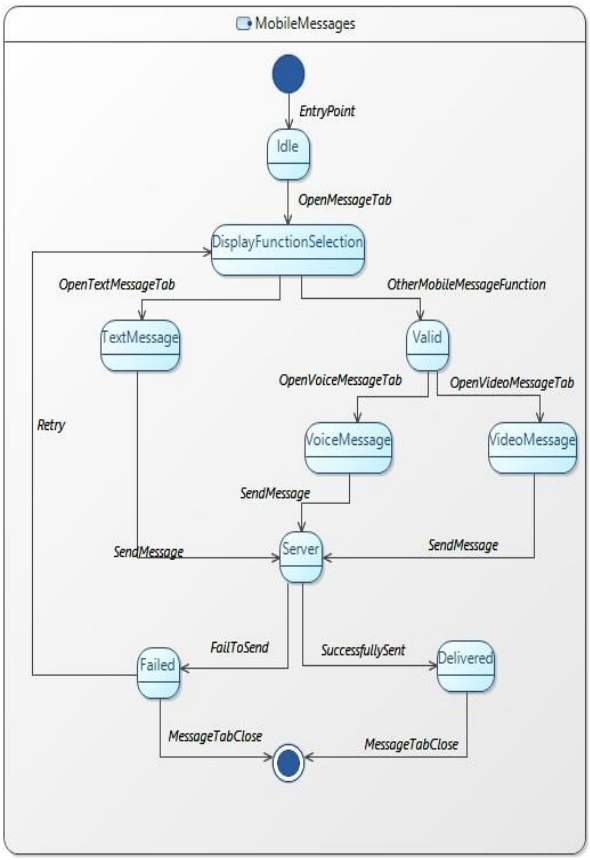


Fig 1: Statechart for sending message by using mobile phone

In MBT, the main function of a test model is to get the behaviour of the system under test. It is used to obtain a set of test cases based on coverage criteria defined either manually or automatically depending on the test case generator. In this work, valid feature configurations have their own test model. This can be defined as in Definition 6.

Definition 6 (Feature configuration and test model):
$FC \rightarrow tmforfc_a \in FC$ is defined as the feature configuration mapped to test model for all types of valid product features.
In software testing, it is difficult to determine the right time to stop testing. The implementation of coverage criteria can help to determine a suitable cut-off time for testing. Coverage criteria can enforce requirements from test model [25]. In this work, only two coverage criteria for MBT are considered to comprise all-states and all-transition [4]. These criteria have been chosen as they are two most basic coverage criteria for MBT [26].

5. Approach

An overall approach is depicted in Figure 4. Several inputs are required in the approach. Input such as product variation can be illustrated by using FM. FM is used to describe features dependency and illustrate the relationships among product features. In order to conduct MBT, one of the models that can be used to automate test case generation is by using a behaviour model. In this work, statechart is used to consider product behaviour and generate test cases. Further, statechart is used to describe the behaviour of product features. The combination of functionalities of these two models is chosen as the functionalities would be lacking if there is only one model used. In this approach, FM is used to present features whereas statechart is used to describe the behaviour of feature for test case generation. This illustrates that these two models complement each other in a model-based testing for SPL. In order to connect the relationship of the models used, the mapping needs to be performed so that all features can be handled. State machine revises the behaviour of the features depending on the features of FM. This dependency is balanced by using statechart elements, which are used to show the behaviour of FM. The mapping between these two models is based on multiple FMs with multiple statecharts. Due to limited space, only one example is illustrated as shown in Figure 4. Product requirements are represented as features in FM. As depicted in Figure 4, product variants are derived from FM. In order to derive the products, a coverage criterion is applied to the FM.

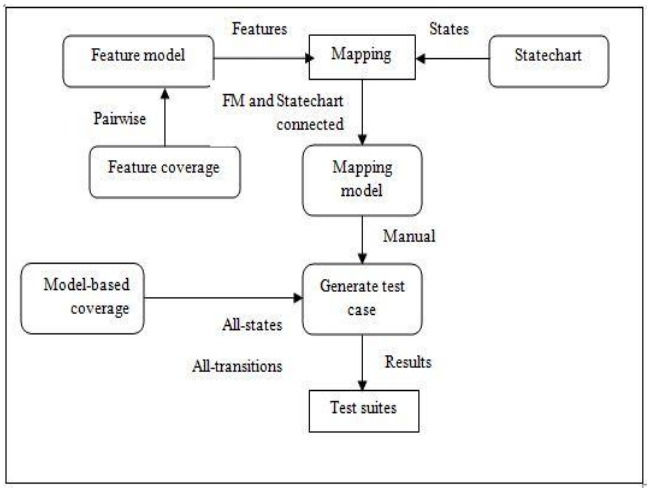


Fig 2: Model-based proposed approach

It helps the tester to select important features from FM before applying MBT. By having the implementation of coverage criteria for FM, a set of possible product variants can be selected. This can avoid unsuitable features from being selected as the removal of unnecessary product variants will only involve more effort. The detail regarding coverage criteria is explained in coverage criteria

subsection. Once the product variants have been selected, various FMs will be constructed to show all possible features. At the same time, we have manually drawn a statechart consisting of case study's variation point. However, the variation point of statechart drawn still lacks information (i.e. relationship among features with information regarding the validity of feature selections). This indicates that the connection between features of FM with the statechart is greatly required.

6. Results and Discussion

Our main contribution in this paper is to relate feature-based and requirement-based concept in MBT. The paper was proposed with the intent to minimize the testing effort by using model-based testing with consideration of coverage criteria. This approach relies on the integration between FM and statechart as the models used for test generation. Furthermore, the proposed approach is supported by pairwise and model-based coverage criteria which had assisted in reducing the selection of unrelated test cases before and after test generation. A preliminary study was performed on a small case study; a mobile phone SPL.

The approach was evaluated based on pairwise coverage criteria, all-states, and all-transition. Valid generated pairwise features are listed in Table 1. Based on Table 1, there are 12 valid features. In this paper, only one valid combination of pairwise features was selected and demonstrated for testing. Further, as for behaviour model, the phone message example was selected to demonstrate the flow of testing. Test 7 (Refer Table 1) was selected as all features configurations are marked as TRUE as compared to other test instances.

Table 1: List of valid features configurations

Label	Variation Type	Parent Unique Name	Parent Type
MobilePhone	ps:mandatory		
MobileMessage	ps:mandatory	MobilePhone	ps:feature
MobileCall	ps:mandatory	MobilePhone	ps:feature
TextMessage	ps:mandatory	MobileMessage	ps:feature
VoiceMessage	ps:optional	MobileMessage	ps:feature
VideoCall	ps:optional	MobileCall	ps:feature
VideoMessage	ps:optional	MobileMessage	ps:feature
VoiceCall	ps:mandatory	MobileCall	ps:feature

For mapping of features to statechart, manual mapping of the features with the statechart based on CITLAB result was carried out. Based on the mapping, configurations of statechart that were not related to valid features configurations were removed. Table 2 contains the list of valid features upon the removal of invalid configurations. The list of valid configurations are based on the results of implementation of pairwise coverage.

Table 2: Selected features from valid configurations

Text Message	Voice Message	Video Message	Voice Call	Video Call	Mobile GPS
TRUE	true	false	false	true	true
TRUE	false	true	true	false	true
FALSE	true	true	false	true	false
FALSE	false	true	true	true	false
TRUE	false	false	false	true	true
FALSE	true	false	true	false	true
TRUE	true	true	true	true	false

Theorem 1 SPL test suite SPL_{ts}:

$SPL_{ts} = (tc, tc_a) tc \subset T,\ fc_a \subseteq fc_a$ where a test case from feature configuration is selected with consideration of coverage criteria based on product set.

Based on all-transition coverage criteria used in statechart, a manual generation of test suite was conducted producing several test cases as shown in Table 3. The test suite generation had considered all defined coverage criteria, including whole transition and states in the statechart. As a result, nine test suite sequences with 51 event calls were produced. All test sequences are based on pairwise coverage as per set in CITLAB.

Table 3: List of test sequences

No.	Test Suite Sequences
1	{OpenMessageTab,OpenTestMessageTab, SendMessage, FailToSend,MessageTabClose}
2	{OpenMessageTab,OpenTestMessageTab, SendMessage, SuccessfullySent,MessageTabClose}
3	{OpenMessageTab,OpenTestMessageTab, SendMessage, FailToSend,Retry}
4	{OpenMessageTab,OpenMobileMessageFunction, OpenVoiceMessageTab,SendMessage, FailToSend,MessageTabClose}
5	{OpenMessageTab,OpenMobileMessageFunction, OpenVoiceMessageTab,SendMessage, FailToSend,Retry}
6	{OpenMessageTab,OpenMobileMessageFunction, OpenVoiceMessageTab,SendMessage, SuccessfullySent,MessageTabClose}
7	{OpenMessageTab,OpenMobileMessageFunction, OpenVideoMessageTab,SendMessage, FailToSend, MessageTabClose }
8	{OpenMessageTab,OpenMobileMessageFunction, OpenVideoMessageTab,SendMessage, FailToSend, Retry}
9	{OpenMessageTab,OpenMobileMessageFunction, OpenVideoMessageTab,SendMessage, SuccessfullySent, MessageTabClose }

Based on Definition 5 and 6, we have generated SPL test suite for coverage criterion. The applications of SPL mobile phone case study were introduced in Section 2. The case study was evaluated according to pairwise coverage and model-based coverage. The coverage was determined from the combination of two models comprising of FM and statechart. Two main advantages of this approach include:

1. Features configuration can be considered by using statechart.
2. Incorrect selection of requirements based on products can be avoided.

In terms of product selection, mapping can help to check the validity of the requirements before conducting MBT. However, in terms of efficiency, much work need to be done to refine the approach as FM and statechart models need to be tested rigorously prior to conducting testing. Time constraint remains an area that still could be improved as we had tried to minimize the time by automating the process of drawing and mapping the model. Furthermore, in this work, we chose to apply coverage criteria to the FM. Through the proposed work, valid product features were able to be obtained while unnecessary features were managed to be removed prior to mapping with statechart. Based on the valid features obtained, there were 12 features that could be validated with the statechart. We considered to validate each constructed model carefully to make sure that the result is as detailed as possible in describing each requirement.

For mapping strategy, an automated mapping had been considered. However, as the work is still in its early stage, much work still needs to be done prior to achieving full automation of the entire manual process of mapping. The proposed approach had considered two MBT coverage criteria. The coverage had helped the process of test generation by checking on the transition of statechart at least once. Despite this, in this work, all-states and all-transition coverage produced similar results, in which they both yielded six test cases. Despite this, the results could be affected due to cases of the implemented statechart. Owing to this, we could not conclude that this result would be the same if coverage criteria were implemented as it depends on the behaviour of the model. Different behaviour model would produce different results depending on the types of case study implemented.

7. Conclusion

In this paper, we had described a state-of-the-art approach that was used to link FM with model-based artefacts. We had also discussed a few observations and limitations of our proposed model-based approach. We explained on potential improvements of this topic which had combined feature-based and requirement-based concepts in MBT for SPL. This paper had produced test generation which integrated Pure::variants with CITLAB and other tools. Some steps in our proposed approach still require further refinement whereby the results might vary in the future once the research has progressed further. Our main aim in this paper was to minimize testing effort. The reuse concept in model remains an interesting niche that MBT has to offer in testing as compared to other techniques. The implementation of pairwise for FM also leads to valid feature selection which can potentially replace manual checking of possible valid feature configurations. From the usability viewpoint, this approach is intuitive since there is an integration of feature-based and requirement-based concepts under one approach. Our concern in this work is regarding the size of features implemented for test generation. Once a manual test generation step could be fully automated, it would be possible to evaluate and compare the proposed approach on features of a larger size. In the future, features of a larger size would be considered, including considering all statechart features in mapping. Additionally, coverage criteria metric would be defined to evaluate the coverage utilized in the proposed approach. In addition, it remains our focus to try to minimize the complexity of the proposed approach so as to minimize future user's effort in utilizing the approach. Furthermore, we will also try to automate test generation from selected states and transition with the integration of existing tools such as ParTeG.

Acknowledgement

This research was supported by the Ministry of Higher Education Malaysia (MOHE) and Universiti Tun Hussein Onn Malaysia (UTHM). We would like to express our profound gratitude to lab members of Software Engineering Department, Universiti Teknologi Malaysia (UTM) for their continuous support and constructive feedbacks towards the completion of this paper.

References

[1] Clements, P. & Northrop, L. (2003). Software Product Lines. Addison-Wesley, 1–105.
[2] Wang, S. Ali, S., Yue, T. & Liaaen, M. (2013). Using Feature Model to Support Model-Based Testing of Product Lines: An Industrial Case Study. Proceedings of the 13th International Conference on Quality Software, pp. 75-84.
[3] Cichos, H., Oster, S., Lochau, M. & Schuerr, A. (2011). Model-Based Coverage-Driven Test Suite Generation for Software Product Lines. Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science, 6981, 425–439.
[4] Utting, M., Pretschner, A. & Legeard, B. (2012). A Taxonomy of Model-Based Testing Approaches. Software Testing, Verification and Reliability, 24(5), 297-312.
[5] Siavashi, F. & Truscan, D. (2015). Environment Modeling in Model-Based Testing: Concepts, Prospects and Research Challenges: A Systematic Literature Review. Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–6.
[6] Wang, S., Gotlieb, A., Ali, S. & Liaaen, M. (2013). Automated Test Case Selection Using Feature Model: An Industrial Case Study. Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science, 237–253.
[7] Weißleder, S. & Lackner, H. (2013). Top-Down and Bottom-Up Approach for Model-Based Testing of Product Lines. Electron. Proc. Theor. Comput. Sci., 111, 82–94.
[8] Olimpiew, E. M. & Gomaa, H. (2009). Reusable Model-Based Testing, Formal Foundations of Reuse and Domain Engineering. Lecture Notes in Computer Science, 5791, 76–85.
[9] Lochau, M., Oster, S., Goltz, U. & Schürr, A. (2012). Model-Based Pairwise Testing For Feature Interaction Coverage in Software Product Line Engineering. Software Quality Journal, 20 (3–4), 567–604.
[10] Oster, S. (2012). Feature Model-based Software Product Line Testing. PhD thesis, Technische Universitat.
[11] Haslinger, E. N., Lopez-Herrejon, R.E. & Egyed, A. (2013). Using Feature Model Knowledge to Speed Up the Generation of Covering Arrays. Proceedings of the Seventh International Workshop on Variability Modelling of Software-Intensive Systems, pp. 1–6.
[12] Olimpiew, E. M. (2008). Model-Based Testing for Software Product Lines. Doctoral Dissertation, George Mason University Fairfax.
[13] Beuche, D. (2003). Variant Management With Pure:: Variants. Technical report, Pure-Systems GmbH.
[14] Lian, X. Zhang, L., Jiang, J. & Goss, W. (2018). An Approach for Optimized Feature Selection in Large-Scale Software Product Lines. Journal of Systems and Software, 137, 636-651.
[15] Beuche, D. & Dalgarno, M. (2007). Software Product Line Engineering with Feature Models. Overload Journal, 78, 5-8.
[16] Machado, I. (2014). Fault Model-Based Variability Testing. PhD thesis, Universidade Salvador.
[17] Lopez-Herrejon, R. Ferrer, J., Haslinger, E. N., Chicano, F., Egyed, A., & Alba, E. (2014). Comparing Pairwise Testing in Software Product Lines: A Framework Assessment. Proceedings of the International Conference Software Product Line, pp. 1-15.
[18] Parejo, J. A., Sánchez, A. B., Segura, S., Ruiz-Cortés, A., Lopez-Herrejon, R. E., & Egyed, A. (2016). Multi-Objective Test Case Prioritization in Highly Configurable Systems: A Case Study. Journal of Systems and Software, 122, 287-310.
[19] Perrouin, G., Oster, S., Sen, S., Klein, J., Baudry, B. & le Traon, Y. (2012). Pairwise Testing for Software Product Lines: Comparison of Two Approaches. Software Quality Journal, 20(3–4), 605–643.
[20] Lei, Y., Kacker, R., Kuhn, D. R., Okun, V., & Lawrence, J. (2007). IPOG: A General Strategy for T-Way Software Testing. Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, pp. 549–556.
[21] Brucker, A. D. & Julliand, J. (2008). IPOG/IPOG-D: Efficient Test Generation for Multi-Way Combinatorial Testing. Software Testing, Verification and. Reliability, 18(3), 125-148.
[22] Calvagna, A, Gargantini, A. & Vavassori, P. (2013). Combinatorial Testing for Feature Models Using CITLAB. Proceedings of the IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, pp. 338-347.
[23] Calvagna, A., Gargantini, A. & Vavassori, P. (2013). Combinatorial Interaction Testing with CITLAB. Proceedings of the IEEE Sixth International Conference on Software Testing, Verification and Validation, pp. 376-382.
[24] Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming, 8(3), 231–274.
[25] Rodrigues, E. D. M. (2013). Plets: A Product Line of Model-Based Testing Tools. PhD thesis, Pontifical Catholic University of Rio Grande do Sul.
[26] Doungsa-ard, C., Dahal, K. P. Hossain, M. A., & Suwannasart, T. (2007). An Automatic Test Data Generation from UML State Diagram using Genetic Algorithm. Proceedings of the Second International Conference on Software Engineering Advances, pp. 47-52.