Goal-Oriented Test Case Selection and Prioritization for Product Line Feature Models

Alireza Ensan[1], Ebrahim Bagheri[2,3], Mohsen Asadi[2,4], Dragan Gasevic[2], Yevgen Biletskiy[1]
[1]University of New Brunswick, [2]Athabasca University, [3]NRC Canada, [4]Simon Fraser University

Abstract

The software product line engineering paradigm is amongst the widely used means for capturing and handling the commonalities and variabilities of the many applications of a target domain. The large number of possible products and complex interactions between software product line features makes the effective testing of them a challenge. To conquer the time and space complexity involved with testing a product line, an intuitive approach is the reduction of the test space. In this paper, we propose an approach to reduce the product line test space. We introduce a goal-oriented approach for the selection of the most desirable features from the product line. Such an approach allows us to identify the features that are more important and need to be tested more comprehensively from the perspective of the domain stakeholders. The more important features and the configurations that contain them will be given priority over the less important configurations, hence providing a hybrid test case reduction and prioritization strategy for testing software product lines.

Keywords: Feature Models, Test Case Prioritization, Test Case Selection, Product Lines, Goal Models.

1. Introduction

Software Product Line (SPL) engineering is a paradigm that formally represents a set of software products that share many common characteristics. These common characteristics (aka features) are defined to model the capabilities and behaviour of a software system [1]. In fact, SPLs symbolize all possible software product configurations of a given domain; therefore, they are employed to model the feasible software product space of a given domain of interest. This software development paradigm has been one of the leading approaches for *re-use based software development* [2]; essentially, because the commonalities captured in the product line can be re-used in different applications and products of the family. Many industrial case studies such as the work of Nokia, and Boeing have shown that software product lines are effective in reducing time-to-market of new products and their development costs [3].

Software product lines are often graphically represented through a hierarchical tree structure known as feature models. Feature models are able to capture the commonalities as well as variabilities of all the products of a given domain. The nodes in a feature model are the features of the product line and the edges between the nodes characterize the variability of the features. Now, since feature models are graphical depictions of a product line and product lines are the representative of the feasible application space of a certain domain, it is possible to derive different applications by selecting some of the features of interest from the feature model. This process, which involves the selection of a set of desirable features from a feature model and entails the development of a final application, is referred to as the *configuration process*. Within the configuration process each valid selection of features will generate a *configuration* of the feature model. Given their hierarchical and tree-like structure, a feature model with n features, can potentially produce 2^n configurations. In effect, the number of configurations (products) increases exponentially when the number of features grows linearly.

This exponential growth in the number of possible products can be a two-edged sword in that although reusability and rapid application development are enhanced, an error or bug in one of the features can easily ripple through to a multitude of applications causing a cascading effect of failure in many products. To avoid such situations, a comprehensive testing strategy would need to be employed, which for a feature model with n feature, could potentially need to evaluate 2^n test cases (product line configurations/products) in the worst case. Clearly, testing this number of products is not realistic in terms of the required time and cost that it incurs. This is even more evident when the number of features in the feature model grows.

In the field of software testing, a testing method is called *idealistic*, if and when it employs the least number of test cases for identifying the available errors and faults [4]. Our objective in this paper is to reduce the number of required test cases for testing a product line while at the same time maintaining an acceptable error detection coverage. For this purpose, we propose a *test case prioritization scheme* that evaluates the importance and urgency of testing a feature model configuration (test case) based on the importance of the features (for the target stakeholders and users) that it includes. The rationale for our approach is based on the assumption that the features of a product line that are more important for the stakeholders have a higher likelihood to appear in the product configurations of the product line; therefore, such features possess a higher importance to be tested first.

In our previous work [5], we have already discussed that there is a close dependence between the stakeholders' preferences defined based on their goals and objectives and the final product line configurations/products that is developed, i.e., a feature model depicting the functionalities of a product line can be mapped onto a goal model that represents the goal and preferences of the stakeholders. We benefit from such correspondence and identify more important features of the product line as those that correspond with the more significant goals and objective of the stakeholders defined using goal models. This allows us to reason about the priority and significance of the product line configuration to be used as the potential test cases in the testing process.

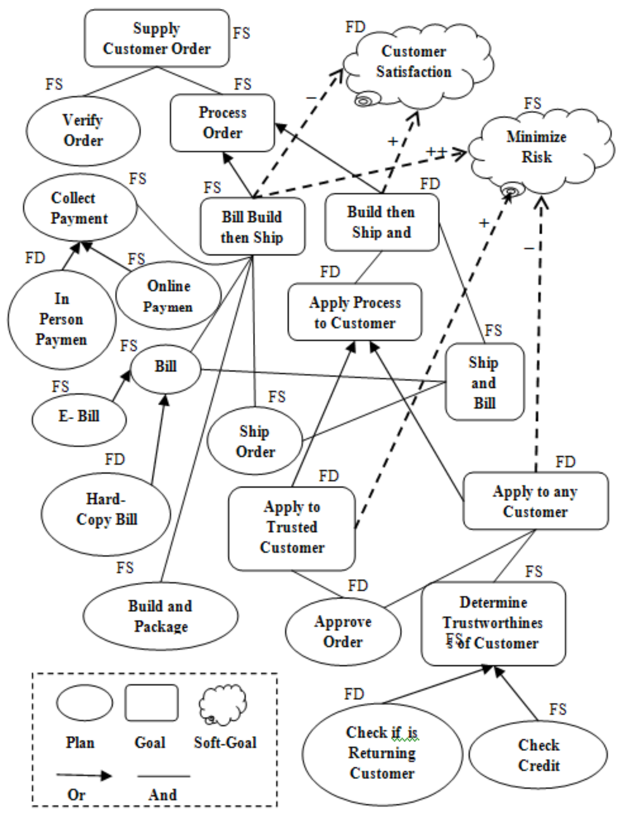

Figure 1. The goal model for the e-shop domain.

Assuming that an initial feature model has n features, our test case prioritization approach enables a test engineer to select a much smaller subset of the test space (2^n) through two main steps: 1) in the first step, the priorities of the stakeholders are taken into consideration to determine the top m most important features of the feature model. This reduces the number of available features to m, where $m < n$, and therefore, the possible configurations would be at most 2^m, where $2^m \ll 2^n$; 2) in the second step, we prioritize the possible 2^m configurations as test cases based on the degree of stakeholder goal satisfaction that they provide based on a goal priority partial order defined over the goal model. Given the developed priority over the 2^m test cases and based on the degree of desired error coverage, it is possible to select the top p test cases, where $p \ll 2^m$. With this approach, we decrease the size of the feature model based on our observations from the stakeholders' preferences and select certain number of test cases with regard to those expectations and goals for testing the system and reaching an acceptable degree of error coverage.

Our work is essentially designed for test case reduction and prioritization based on the correspondence between the feature model elements and the domain stakeholders' goals. We employ a mapping mechanism to connect feature models to standard goal models (that are used for formally representing the preferences and objectives of the domain stakeholders) and do reasoning based on this connection to find the most important test cases.

This paper is organized as follows: first, we will discuss goal models and reasoning based on them in Section 2. Then, we review the concept of feature models in Section 3. After that, we will introduce our approach to effectively select and prioritize feature model test cases in Section 4. Section 5, will go through and analyze a case study for test case prioritization based on our approach. Section 6 will review the related work. Finally, in Section 7, we will conclude the paper with concluding remarks and discussion of our future work.

2. Goal Models

Goal-oriented requirement engineering is a way to analyze, specify, and model requirements specifications based on stakeholders' objectives [7,8]. Expressive graph-based models have been widely employed to represent goal models. Nodes of such graphs represent stakeholder goals and the edges represent the relations between the goals. Goals can be decomposed into more detailed goals by AND, OR, or XOR relations. Moreover, goals are often categorized in three types, called goals (hard goals), plans (tasks), and soft goals. The first two types refer to the functional specifications of the domain while the third type is related to the non-functional quality aspects of the domain. Furthermore, hard goals are subjective expectations while plans (tasks) are objective or observable goals. For each goal four possible level of satisfaction can be defined including FS (fully satisfied), PS (partially satisfied), FD (fully denied), and PD (partially denied). To clarify better, Figure 1 shows a part of a goal model for the e-shop domain from [5].

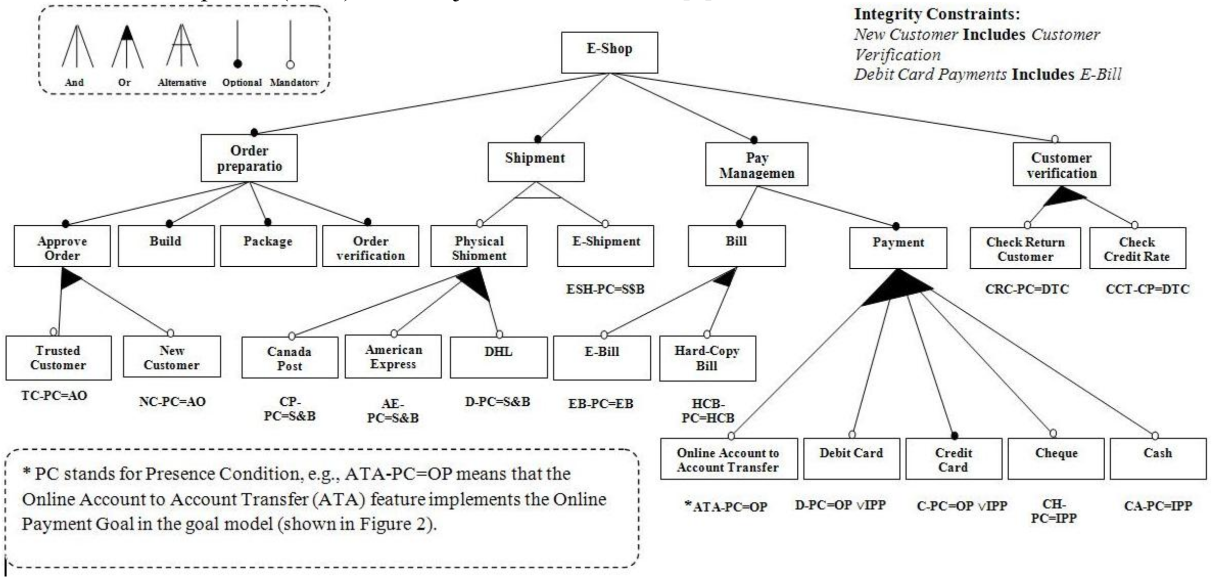

Figure 2. The feature model for the e-shop domain.

Moreover, it is possible to define lateral relations between goals in a goal model [9]. Consider the relation $G_1 \overset{R}{\rightarrow} G_2$, which can also be shown as: $R: G_1 \rightarrow G_2$. Here, G_1 is called the subject goal and G_2 is the object goal. R is the relation between the subject and the object. It is also possible that many subjects go to one object by R as $R: (G_1, G_2, \dots, G_n) \rightarrow G_m$. These relations are assigned qualitative labels such as $"+"$, $"-","++"$, $"--","+s","++s","+D","++D"$, $"-s","--s"$, $"-D","--D"$[9]. For example, $G_1 \overset{+}{\rightarrow} G_2$ means that if goal G_1 is satisfied then there is some evidence that goal G_2 is partially satisfied, or $G_1 \overset{-}{\rightarrow} G_2$ states that if G_1 is satisfied (denied) then there is some evidence that G_2 is denied (satisfied). $"+"$, $"-","++"$, and $"--"$ are symmetric relations while others are asymmetric. For example, $G_1 \overset{-s}{\rightarrow} G_2$ states that if G_1 is satisfied then there is some evidence that G_2 is denied; however, if G_1 is denied, nothing can be said about the satisfaction of G_2. Furthermore, $G_1 \overset{-D}{\rightarrow} G_2$ means that if $G_1$is denied, then there is some evidence that G_2 is satisfied, but if G_1 is satisfied, nothing can be said about the denial of G_2. It is worth mentioning that the difference between $"+"$ and $"++"$ ($"-","--"$) is just about the amount of (partial or full) positive (negative) effect of the subject goals on to the object goals. Reasoning over the goal graph is possible based on the decomposition links that exist between the goals of the model. For example, in an AND decomposition such as $G_1 = G_2 \wedge G_3$, if G_1 is satisfied, both G_2 and G_3 must be satisfied. In addition for an OR decomposition such as $G_1 = G_2 \vee G_3$, if G_1 is satisfied, at least one of G_2 or G_3 must be satisfied. Moreover, a goal in goal model We can benefit from these lateral relation and goals values definitions to do qualitative reasoning over goal models [9].

To implement reasoning, the satisfaction levels of some goals are assigned by stakeholders. Then, a reasoning approach like backward chaining (top-down) or forward chaining (bottom-up) is employed to propagate these values to all of the goals in the goal model. We have employed backward chaining in this paper in which stakeholder assigned values to high level soft- and hard- goals and other goals' satisfaction levels are decided. For more details on this, interested readers are encouraged to see [5].

3. Feature Models

Features are important distinguishing aspects, qualities, or characteristics of a family of systems [6]. They are widely used for depicting the shared structure and behavior of a set of similar systems. To form a product family, all the features of a set of similar/related systems are composed into a feature model. A feature model represents possible configuration space of all the products of a system product family in terms of its features. Feature models can be represented both formally and graphically; however, the graphical notation depicted through a tree-like structure is more favored due to its visual appeal and easier understanding.

In a feature model, features are hierarchically organized and can be typically classified as: 1) *Mandatory*: a feature must be included in the description of its parent feature; 2) *Optional*: a feature may or may not be included in its parent description given the situation; 3) *Alternative feature group*: one and only one of features from the feature group can be included in the parent description; 4) *Or feature group*: one or more features from a feature group can be included in the description of the parent feature. In some case, the tree structure of feature models falls short at fully representing the complete set of mutual interdependencies of features; thus, additional constraints are often added to feature models and are referred to as *Integrity Constraints*. The two most widely used integrity constraints are: *Includes* – the presence of a given feature (set of features) requires the inclusion of another feature (set of features); and *Excludes* – the presence of a given (set of) feature(s) requires the elimination of another (set of) feature. Moreover, cardinality based feature models (an extension of feature models) define *feature cardinality* and *feature group cardinality*. The former shows the number of instances of a feature in the final products, and the latter shows the minimum and maximum number of sub-features within the grouped feature that can be chosen for the final product.

Figure 2 depicts a part of the well-known feature model for the e-shop domain that we will focus on later in this paper. In order to create final applications from the feature model, configurations need to be developed that select appropriate features from the model. For instance in the e-shop example, 27 features exist, and 2,480 different configurations (corresponding to 2,480 different final applications) can be developed from the feature model configuration process.

Now that feature models and goal models are defined, it is possible to define a set of Boolean variables to link feature model elements onto goal model elements. These Boolean variables are called Presence Conditions (PC) [5]. By mapping features to goals, their PC variables are formulated according to the goal to which they are mapped by considering a Boolean variable corresponding to each goal. Figure 2 depicts the PC of each feature based on Boolean expression of its related goals (related goals to each feature are shown beneath the features in abbreviated form). For instance, ATA-PC=OP in Figure 2 means that the "Online Account To Account Transfer" feature in the feature model implements and realizes the "Online Payment" goal in the goal model.

4. Proposed Approach

As discussed earlier, in order to comprehensively test a feature model with n features, 2^n feature model configurations (test cases) should at most be tested. If the process of testing each configuration requires a time equal to t, the comprehensive test of a feature model would be of $O(2^n \times \mathrm{t})$, which is not a practical complexity. To get better efficiency in testing, it is desirable to find an acceptable trade-off point between error coverage and the number employed test cases. In order to achieve this objective, we propose a two-step approach for reducing the number of test cases. Our approach is based on the assumption that the features that are more important from the perspective of the domain stakeholders have a higher probability of being employed in the developed final applications from the product line. For this reason, such features should have a higher priority in the testing process. In the first step of our approach (*feature selection*), the feature search space of a feature model is reduced by selecting the features that are correlated with the most important goals and objectives of the stakeholders. Then in the second step (*test case prioritization*), the possible configurations of this reduced feature model are prioritized. This prioritization will allow the test engineers to select and evaluate the top p test cases based on their available resources. It is important to note that the goal models used in this paper represent the preferences and objectives of the stakeholders with regards to the target domain and do not refer to a specific target application of that domain.

4.1. Step 1: Feature Selection

The objective of this first step is to reduce the size of a feature model from *n* features to a smaller number of *m* features. We achieve this by exploiting the relationship between feature models and stakeholders' goal models. We have already developed this relationship between goal models and feature models in our previous work [5]. This relationship provides us with the means to select the features that are related to the most important goals and objectives of the stakeholders as represented in the goal model. Such an approach ensures that the feature selection process is based on the domain stakeholders' goals and preferences [5]. For instance, suppose that the domain stakeholders have two goals: achieving higher sales and lowering maintenance costs, such that achieving higher sales is more important for them than lowering maintenance costs. Now assume that the payment feature of our feature model can be realized by either credit card payment (which contributes to the achieving higher sales goal) or cash only payment (that contributes to the lowering maintenance costs goal). In our approach, the credit card payment feature will have a higher importance to be selected since it is connected to the more important goal of the domain stakeholders. Using this feature selection approach, we are able to reduce the feature model size. Now, the central issue is the use of label propagation in goal models (introduced earlier in this paper) for identifying the most important features in the feature model. The details of this process is introduced and discussed in our earlier work. Interested readers are encouraged to see [5].

In summary, the input for this step is the correspondence between the domain stakeholders' goal model and the feature model to be tested. The output of this step is a sufficiently reduced-size feature model with m features (0<m<<n determined by the test engineer). The advantage of performing this step is that features, which are less important from the perspective of the domain stakeholders and have a lower likelihood to be included in the actual final products of the product line are removed from the feature model and hence the test space is considerably reduced.

4.2. Step 2: Test Case Prioritization

In the first step, we were able to reduce the size of the test space by removing the less important features from the feature model. The result is that instead of testing 2^n test cases, it would only be needed to test 2^m test cases, which is much smaller $2^m \ll 2^n$ given the exponential nature of the complexity and also m<<n. However, although the feature selection step significantly reduces the size of the test space, it is still *quadratic* in nature. It is desirable to develop a test suite with linear size, i.e., in the order of the feature model size (n). For this purpose, we develop a test case prioritization method for prioritizing the 2^m test cases. Our proposed prioritization technique is based on the domain stakeholders' goals and objectives that were also used in Step 1.

Let's assume $\phi(\alpha)$ is a function that takes α test cases and arranges them in a certain rank order. Essentially, this function returns a list of ordered test cases. In addition, suppose that a function such as $f(g): G \rightarrow A[W]$that takes as input the goals of the stakeholders and provides a priority relationship between them, where G is the goal model that contains the goals to be evaluated and $A[W]$ be a matrix containing the priority relations between the goals. We have already proposed such a function before, called S-AHP [10]. S-AHP is ideal to represent $f(g)$ as it is able to develop $A[W]$ based on the opinions of the domain stakeholders. In short, we have:

If $g = [G_1, G_2, \dots G_m]$

$f(g) = [w_{ij}]_{m\times m} \quad ; \quad 0 < i,j \leq m$

Where w_{ij} refers to the relative weight of Goal i with regards to Goal *j*. For instance, $w_{23} = 2$ shows that Goal 2 is twice as important as Goal 3 from the domain stakeholders' point of view. Given $f(g)$, it is possible to create a *partial order* between the goals, e.g., Goal 2>Goal 3. This partial order relationship between the domain stakeholders' goals can be used to define the importance of features that are available in a corresponding feature model. The importance of each goal is determined by its rank in the partial order, i.e., the highest ranked goal will have the highest importance (for *m* goals the highest importance will be *m*) and the lowest ranked goal will have an importance of 1.

Subsequently, we define the importance of a feature *f*, $weight(f)$, as the rank of its corresponding goal in the goal model (features connected to more than one goal can be weighed based on their most important goal). For instance, suppose Feature 1 and Feature 2 are connected to Goal 3 and Goal 2, respectively. The importance of these features is defined based on the rank of the goals that they are connected to and would hence be $weight(Feature\ 1) = 2$ and $weight(Feature\ 2) = 1$. Now, we are able to show how $\phi(\alpha)$ operates, in the following:

The rationale for the rank ordering function, $\phi(\alpha)$, is quite similar to the idea behind Step 1, i.e., it is assumed that features that are related to the domain stakeholders' goals with higher importance possess a higher likelihood to appear in the final software products and hence need to have higher priority while performing tests. We further extend this notion as: test cases (feature model configurations) that have a collection of more important features in them have a higher priority to be tested compared with test cases that have less important features. Let us now formalize this as follows:

Let tc_{fm}^i be the i^{th} test case (feature model configuration) for feature model *fm*. We define the significance of tc_{fm}^i, $SIG(tc_{fm}^i)$, as the sum of the importance of its constituent features ($f_j \in tc_{fm}^i$):

$$SIG(tc_{fm}^i) = \sum_{j=1}^{|tc_{fm}^i|} weight(f_j).$$

where $|tc_{fm}^i|$denotes the size of tc_{fm}^i.

With the above definitions, more significant test cases would have a higher priority in terms of being tested first. It is clear that $\phi(\alpha)$ takes m test cases as input and rank orders them based on their significance. An important point that needs to be mentioned is that although $\phi(\alpha)$ provides a prioritization over the test cases, it does not reduce the number of test cases. For this purpose, we define the *satisfaction threshold* ζ, which represents the expected error coverage from the test cases.

Given the satisfaction threshold, two strategies can be undertaken by the test engineer: 1) iteratively test the feature model by taking test cases from the prioritization developed by $\phi(\alpha)$ until the required error coverage is reached; 2) an empirical study is performed based on previous testing experience to find a correlation between the desirable ζ and error coverage. This way it is possible to estimate how many of the top test cases (top p test cases) would be sufficient to reach a desired ζ and error coverage. The selection of the top p test cases based on our proposed approach means that a linear size complexity for the size of the required tests has been achieved ($p \ll 2^n$). The empirical investigation of the correlation between ζ, error coverage and p is one of the main directions of our future work.

In summary, the input of this step is a reduced-size feature model. The output of the step is a set of prioritized test cases for the reduced feature model that can be used to select the top p test cases in order to test the feature model and reach a defined satisfaction threshold. Given this two-step approach, which consists of feature selection and test case prioritization, we are able to initially reduce the size of the test cases that need to be considered in the evaluation process. Then, these remaining test cases are prioritized according to their importance for the goals and objectives of the stakeholders. The feature selection and test case prioritization steps are both defined over our previous theoretical modeling for interconnecting software stakeholders' goal models and product line feature model that also provides tooling support for the process [5].

5. Illustrative Example

In this section, we apply our approach to the widely used benchmark e-shop feature model. A portion of the e-shop feature model that we will be using to demonstrate our approach is illustrated in Figure 2. Given the fact that our approach is dependent on a goal model for the purposes of feature selection and test case prioritization, the matching goal model for the e-store feature model, which is widely used in the literature, is shown in Figure 1. As seen in Figure 2, the e-store feature model consists of 27 features. A simple calculation shows that it is possible to create 2^{27} possible produts (feature model configuration) from this feature model. However, some general constraints (mandatory, and, or, alternative or), and integrity constraints limit the feasible configuration space. Considering all of the general and integrity constraints, the total possible numbers of feature configurations for this feature model is 2,480. This is equivalent to 2,480 test cases for testing the feature model. It is clear that for a small feature model with 27 features, 2,480 test cases is a rather high number. Our task here is to test the least number of possible test cases while at the same time ensure that high error coverage is maintained.

In order to apply the test-case prioritization process, we should have defined the mappings between the feature model and goal model (This is easily done using our tool support explained in [5]). To map the feature model to the goal model, for each task in the goal model, the features that are required to implement that task are identified and linked to the goal. For example, ***Credit Card***, ***Debit Card***, and ***On-Line Account to Account Transformation*** features in the feature model are connected to the ***on-line Payment*** task in the goal model. Moreover, ***Credit Card***, ***Debit Card***, ***Cheque***, and ***Cash*** features are also connected to ***In Person Payment*** in the goal model. This process is continued until all of the features in the feature model are connected with their appropriate corresponding goals in the goal model. By mapping features to goals, their Presence Conditions (PC) variables are formulated according to the goal to which they are mapped by considering a Boolean variable corresponding to each goal. Figure 2 depicts the PC of each feature based on Boolean expression of its related goals (related goals to each feature are shown beneath the features in abbreviated form).

Having mapped the e-shop feature model to its corresponding goal model, we perform the two proposed steps to prioritize the test cases. In the first step, feature selection based on the importance of the stakeholders' objectives defined in the goal model should be performed. For this purpose, we determine an expectation for the highest level goal. For example, suppose that "FS" (Fully Satisfied) is our expectation of the ***Supply Customer Order*** goal. Backward reasoning determines the state of all goals in the goal model based on this expectation. Figure 1 illustrates the satisfaction level of each goal in the graph with the label "FS" (Fully Satisfied) and "FD" (Fully Denied). For example, the ***Supply Customer Order*** goal is fully satisfied. It can also be decomposed into ***Verify Order*** and ***Process Order*** through an AND relationship. Therefore, the assigned labels for both of those goals are "FS". Another instance in the goal graph is the ***Process Order*** goal, which is evaluated to "FS". In addition, the ***Process Order*** goal can be decomposed using an OR-relation into ***Bill, Build, then Ship*** and ***Build, then Ship and Bill***. If the ***Process Order*** goal is fully satisfied, at least one of its sub goals must also be satisfied. We consider "FS" for to ***Bill, Build, then Ship*** and "FD" for the ***Build, then Ship and Bill***. We continue this approach until all goals in the graph have been evaluated and been assigned appropriate labels. These labels for our example are shown in Figure 1.

The next step in our approach is to perform feature selection based on the correspondence that has been developed between the feature model and the goal model. We employ the pre-configuration algorithm which is able to determine the candidate features to be removed from the feature model (features that are mapped to goals labeled with FD) and hence reduce the size of the feature model [5]. Using this algorithm we eliminate the removal candidate features if their removal does not lead to any constraints violation. The ***Cheque***, ***Cash***, ***Hard Copy***, ***Check Return Customer***, ***Trusted Customer*** and ***New Customer*** features are candidate to be removed from the feature model because they are related to the less important goals and objectives of the stakeholders. Except for the deletion of the ***New Customer*** feature that leads to an integrity constraint violation, other candidates can be removed from the model. This will reduce the total number of possible feature configurations from 2,480 to just 16. Given that this number is equivalent to the number of test cases for the feature model is considered to be significant test case reduction.

According to our proposed approach, the next step is to prioritize these 16 test cases that are generated based on the new feasible feature configuration space. In order to compute the significance of each test case, we need a matrix of weights that shows the partial relationships between the goals and objectives of the stakeholders. In our previous work we have already shown how such a matrix can be gathered from the stakeholders using the S-AHP technique [10]. A portion of this matrix for the e-shop goal model is as follows:

$$A = \begin{bmatrix} 1 & 1 & .5 & .25 \\ 1 & 1 & .5 & .25 \\ 2 & 2 & 1 & .5 \\ 4 & 4 & 2 & 1 \end{bmatrix}$$

Where the rows and columns denote the ***Collect Payment***, ***Bill***, ***Approve Order***, and ***Determine Trustworthiness*** goals, respectively. For instance A[3,1]=2 shows that the ***Approve Order*** goal is twice as important as the ***Collect Payment*** goal for the stakeholders. This weight matrix allows us to create a partial order between the goals of the goal model based on their importance. It is easy to see that for matrix A shown above, the order between the goal models is: $\boldsymbol{Determine\ Trustworthiness} >$ $\boldsymbol{Approve\ Order} > \boldsymbol{Bill} = \boldsymbol{Collect\ Payment}$ where $\boldsymbol{Bill}$ and $\boldsymbol{Collect\ Payment}$ have the same order. Given this ordering, it is possible to calculate the weight of each related feature to those goals, which is equivalent to the rank of the goal associated with each feature. For example, $weight\ (\boldsymbol{Check\ Credit\ Rate})$ is equal to 4 because this feature is associated with the $\boldsymbol{Determine\ Trustworthiness}$ goal.

Now, given that we have the significance of each feature, we need to compute the significance of each test case (feature model configuration) based on the significance of the features that build it. For this, we compute the sum of the significance of each feature in that test case. For example, the test case (feature model configuration) consisting of the following features *{check credit rate, credit card, DHL, New Customer}* has the highest rank: $SIG(tc_{fm}) = 13$; therefore, it can be considered as the most important test case in terms of the goals and objectives of the domain stakeholders, so it should be tested earlier in the testing process. This process will provide a prioritization over the available 16 test cases. Test engineers can use this provided prioritization to decide about the test cases that they want to use during their testing sessions.

6. Related Work

The challenges of testing commonality and variability of software product lines have been extensively discussed in [11] and [12], which typically revolve around the high number of test cases that need to be tested for achieving high error coverage. Some attempts have been made to overcome the challenges of SPLs testing such as complex interaction between features and the large number of possible configurations that should be tested. Similar to our work, some researchers have concentrated on the use of specification-based techniques to draw test cases from the software product line specifications [13,14]. In our work, we have used goal models as a representative of the domain stakeholders' objectives and preferences. The goal models in our work are similar in their role to the product line specifications in the specification-based techniques.

Some SPLs testing approaches have focused on the reduction of the test space [15,16]. Some others have used analytic methods to take sample test cases from the test space [17]. On the other hand, the authors of [17] base their work on the assumption that although the number of product line configurations is exponential, many of these configurations are irreverent for the purpose of testing. This means that the absence or presence of some of the feature in the feature model does not have considerable effects on the validity and performance of the final feature model configuration. Thus, many test cases can be removed and hence the size of the test suite could be reduced. This approach for test space reduction has both similarities and differences from our work. It is similar to our work in that it removes some of the test cases in order to create a more focused set of test cases. It is different in that the basis for relevancy of features and significance of test cases in our work is based on the domain stakeholders' objectives and preferences which is not considered in their work. Other work which reduce the test space also exist that mainly focus on the use of user requirements to identify the most important set of features that need to be tested [18]. In such approaches, it is the end users' need and objectives that drives the test case selection process. This category of works is most relevant to our approach in this paper.

In a different line of work, Perrouin et al. employ the concept of T-wise test generation [19]. Their approach attempts to address the large combinatorial number of required tests for a software product line by only generating test sets that cover all possible T feature interactions. To achieve this, the authors devise strategies to disintegrate T-wise combinations into smaller manipulable subsets, which are then used in a developed toolset over Alloy for generating the tests. A comprehensive survey of software product line testing methods are presented in [20], interested readers are encouraged to see that paper.

7. Concluding Remarks

In this paper, we have proposed a new approach for reducing and prioritizing the test cases for evaluating a software product line feature model. In this proposed approach, our target is to achieve higher error coverage by testing fewer test cases. To attain this goal, we reduce the test space and prioritize the remaining test cases in two steps. We have proposed a feature selection method based on goal models which is able to model the expectations of the domain stakeholders. Reasoning in these goal models can be used to select the most important features of the feature model. Alongside this test case reduction strategy, we have suggested a prioritization method for product line test cases which is also based on the preferences of the domain stakeholders' goals and objectives. We have discussed the details of our approach through the use of the well known e-shop illustrative example. It has been shown in this case study that our method is able to significantly reduce test space and effectively prioritize the remaining test cases. Our current ongoing work involves the empirical evaluation of our proposed approach for testing different software product line feature models.

8. References

[1] Pohl, K., Bockle, G., Van Der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, 2005.

[2] Richard W. Selby, "Enabling Reuse-Based Software Development of Large-Scale Systems," IEEE TSE, 495-510, 2005.

[3] David M. Weiss, Paul C. Clements, Kyo Kang, Charles Krueger, "Software Product Line Hall of Fame," Software Product Line Conference, p. 237, 2006

[4] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge University Press, 2008.

[5] M. Asadi, E. Bagheri, D. Gasevic, M. Hatala, "Goal-Oriented Requirements and Feature Modeling for Software Product Line Engineering", The 26th ACM Symposium on Applied Computing (SAC 2011), 2011.

[6] Kang, K., Lee, J., Donohoe, P., Feature-oriented product line engineering. IEEE software 19 (2002) 58-65.

[7] Van Lamsweerde, A. et al, "Goal-oriented requirements engineering: A guided tour". *RE* 2001.

[8] Borba, C. and Silva. C, "A Comparison of Goal-Oriented Approaches to Model SPLs Variability". *Proc. ER Workshops* (2009), 244-253.

[9] P. Giorgini, E. Nicchiarelli, J. Mylopoulos, and R. Sebastiani, "Reasoning with Goal Models", In *Proc.* ER'02, Finland, October 2002. Springer.

[10] E. Bagheri, M. Asadi, D. Gasevic, and S. Soltani, "Stratified Analytic Hierarchy Process: Prioritization and Selection of Software Features", The International Software Product Line, 2010.

[11] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles ,and Techniques*, Springer, August (2005).

[12] M. Jaring, R.L Krikhaar, J. Bosch. Modeling Variability and Testability Interaction in Software Product Line Engineering. *ICCBSS*. 120-129. (2008)

[13] Uzuncaova, E., Garcia, D., Khurshid, S., Batory, D.S.:A specification-based approach to testing software product lines. In: ESEC/SIGSOFT FSE. (2007), 525-528.

[14] T. Kakola, J.C., Duenas (Ed.), *Software Product Lines: Research Issues in Engineering and Management*, Springer-Verlag, 2006.

[15] Jackson, D., Alloy: a lightweight object modeling notation, *ACM Trans. Softw. Eng. Methodol.* 11(2):256-290, 2002.

[16] M. B. Cohen, M. B. Dwyer, and J. Shi. Interaction testing of highly-configurable systems in the presence of constraints. ISSTA, 2007.

[17] Kim, C., Batory, D., Khurshid, S., Reducing Combinatorics in Testing Product Lines, UTexas Technical Report, 2010.

[18] Scheidemann, K.: Optimizing the selection of representative configurations in verification of evolving product lines of distributed embedded systems, SPLC, 2006.

[19] Perrouin, G., Sen, S., Klein, J., Baudry, B., Le Traon,Y.: Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines, ICST, 459-468, 2010.

[20] Emelie Engstrom, Per Runeson, Software Product Line Testing - A Systematic Mapping Study, Information and Software Technology, In Press, DOI: 10.1016/j.infsof.2010.05.011.