A Comparison of Test Case Prioritization Criteria for Software Product Lines (v 1.0)

Ana B. Sánchez, Sergio Segura, Antonio Ruiz-Cortés
{anabsanchez,sergiosegura,aruiz}@us.es

Applied Software Engineering Research Group
University of Seville, Spain
July 2013

Technical Report ISA-2013-TR-02

This report was prepared by the

Applied Software Engineering Research Group (ISA)
Department of computer languages and systems
Av/ Reina Mercedes S/N, 41012 Seville, Spain
http://www.isa.us.es/

Support: This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project SETI (TIN2009-07366) and by the Andalusian Government under projects ISABEL (TIC-2533), THEOS (TIC-5906) and TAPAS (TIN2012-32273).

A Comparison of Test Case Prioritization Criteria for Software Product Lines

Ana B. Sánchez, Sergio Segura, and Antonio Ruiz-Cortés
Department of Computer Languages and Systems
University of Seville, Spain
{anabsanchez,sergiosegura,aruiz}@us.es

ABSTRACT

Software Product Line (SPL) testing is challenging due to the potentially huge number of derivable products. To alleviate this problem, numerous contributions have been proposed to reduce the number of products to be tested while still having a good coverage. However, not much attention has been paid to the order in which the products are tested. Test case prioritization techniques reorder test cases to meet a certain performance goal. For instance, testers may wish to order their test cases in order to detect faults as soon as possible which would translate in faster feedback and earlier fault correction. In this paper, we explore the applicability of test case prioritization techniques to SPL testing. We propose five different prioritization criteria based on common metrics of feature models and we compare their effectiveness in increasing the rate of early fault detection, i.e. a measure of how quickly faults are detected. The results show that different orderings of the same SPL suite may lead to significant differences in the rate of early fault detection. They also show that our approach may contribute to accelerate the detection of faults of SPL test suites based on combinatorial testing.

Keywords

Software product lines, feature models, automated analysis, test case prioritization.

1. INTRODUCTION

Software Product Line (SPL) engineering is about developing a set of related software products by reusing a common set of features instead of building each product from scratch. Products in an SPL are differentiated by their features, where a feature defines capabilities and behaviour of a software system [25]. Product lines are often represented through feature models. *Feature models* capture the information of all the possible products of the SPL in terms of features and relationships among them. Figure 1 shows a sample feature model representing an e-commerce SPL. The automated analysis of feature models deals with the computer-aided extraction of information from feature models. These analyses allow to study properties of the SPL such as consistency, variability degree, complexity, etc. In the last two decades, many operations, techniques and tools for the analysis of feature models have been presented [4].

Product line testing is about deriving a set of products and testing each product [23]. An *SPL test case* can be defined as a product of the product line to be tested, i.e. a set of features. The high number of feature combinations in SPLs may lead to thousands or even millions of different products, e.g. the e-shop model available in the SPLOT repository has 290 features and represents more than 1 billion of products [21]. This makes exhaustive testing of an SPL infeasible, that is, testing every single product is too expensive in general. In this context, there have been many attempts to reduce the space of testing through feature-based test case selection [7, 16, 22, 24]. *Test case selection approaches* choose a subset of test cases according to some coverage criteria. Most common test selection approaches are those based on combinatorial testing [16, 22, 23, 24]. In these approaches test cases are selected in a way that guarantee that all combinations of t features are tested. Other authors have proposed using search-based and grammar-based techniques to reduce the number of test cases while maintaining a high fault detection capability [2, 10].

Test selection techniques have taken a step forward to make SPL testing affordable. However, the number of test cases derived from selection could still be high and expensive to run. This may be especially costly during regression testing when tests must be repeatedly executed after any relevant change to the SPL. In this context, the order in which products are tested is commonly assumed to be irrelevant. As a result, it could be the case that the most promising test cases (e.g. those detecting more faults) are run in last place forcing the tester to wait for hours or even days before staring the correction of faults. In a worse scenario, testing resources could be exhausted before running the whole test suite remaining faults undetected.

Test case prioritization techniques schedule test cases for execution in an order that attempts to increase their effectiveness at meeting some performance goal [5, 18, 29, 30]. Many goals can be defined, for instance, testers may wish to order their test cases in order to achieve code coverage at the fastest rate possible, exercise components in expected frequency of use or increase the rate of fault detection of test cases. Given a goal, several ordering criteria may be proposed. For instance, in order to increase the rate of fault detection, testers could order test cases according to the number of faults detected by the test cases in previous executions of the suite, or according to the expected error-proneness of the components under test. Test case prioritization techniques have been extensively studied as a complement for test case selection techniques in demanding testing scenarios [9, 17, 27, 28, 36].

In this paper, we present a test case prioritization approach for SPLs. In particular, we explore the applicability of scheduling the execution order of SPL test cases as a way to reduce the effort of testing and to improve their effectiveness. To show the feasibility of our approach, we propose five different prioritization criteria intended to maximize the rate of early fault detection of the SPL suite, i.e. detect faults as fast as possible. Three of these criteria are based on the complexity of the products. Hence, more complex products are assumed to be more error-prone and therefore are given higher priority over less complex products, i.e. they are tested first. Another prioritization criterion is based on the degree of reusability of products features. In this case, products including the more reused features are given priority during tests. This enables the early detection of high-risk faults that affect to a high portion of the products. Finally, we propose another criterion based on the so-called dissimilarity among products, i.e. a measure of how different two products are. This criterion is based on the assumption that the more different two products are the higher is the feature coverage and the fault detection rate. The proposed prioritization criteria are based on common metrics of feature models extensively studied in the literature. This allowed us to leverage the knowledge and tools for the analysis of feature models making our approach fully automated. Also, this makes our prioritization criteria complementary to the numerous approaches for feature-based test case selection.

For the evaluation of our approach, we developed a prototype implementation of the five prioritization criteria using the SPLAR tool [20]. We selected a number of realistic and randomly generated feature models and generated both random and pairwise-based test suites. Then, we used our fault generator based on the work of Bagheri et al. [2, 10] to seed the features with faults. Finally, we reordered the suite according to the five criteria and we measured how fast the faults were detected by each ordering. The results show that different orderings of the same SPL suite may lead to significant differences in the rate of fault detection. More importantly, the proposed criteria accelerated the detection of faults of both random and pairwise-based SPL test suites in all cases. These results support the applicability and potential benefits of test case prioritization techniques in the context of SPL. We trust that our work will be the first of a number of contributions studying new prioritization goals and criteria as well as new comparisons and evaluations.

The rest of the paper is structured as follows: Section 2 presents some background about the analysis of feature model, combinatorial SPL testing and test case prioritization. In Section 3 we propose five prioritization criteria for SPLs. The evaluation of our approach is described in Section 4. Section 5 presents the threats to validity of our work. The related works are presented and discussed in Section 6. Finally, we summarize our conclusions and outline our future work in Section 7.

2. PRELIMINARIES

2.1 Automated analysis of feature models

SPLs are often graphically represented using feature models. A feature model is a tree structure that capture the information of all the possible products of an SPL in terms of features and relationships among them. Figure 1 shows a simplified feature model representing an e-commerce SPL taken from the SPLOT repository [21]. The model depicts how features are used to specify the commonalities and variabilities of the on-line shopping systems that belong to the SPL.

The analysis of feature models consists on examining their properties. This is performed in terms of analysis operations. Among others, these operations allow finding out whether a feature model is void (i.e. it represents no products) whether it contains errors (e.g. dead features) or what is the number of possible feature combinations in an SPL. Catalogues with up to 30 different analysis operations on feature models have been reported in the literature [4]. Some tools supporting the analysis of feature models are AHEAD Tool Suite [1], FaMa Framework [35], SPLAR [20], and pure::variants [26]. Next, we introduce some of the operations that will be mentioned throughout this paper:

All products: This operation takes a feature model as input and returns all the products represented by the model. For the model in Figure 1, this operation would return the following list of products:

```
P1 = {E-Shop,Catalogue,Payment,Bank Transfer,Security,High}
P2 = {E-Shop,Catalogue,Payment,Bank Transfer,Security,
Standard}
P3 = {E-Shop,Catalogue,Payment,Credit Card,Security,High}
P4 = {E-Shop,Catalogue,Payment,Bank Transfer,Credit Card,
Security,High}
P5 = {E-Shop,Catalogue,Payment,Bank Transfer,Security,High,
Search}
P6 = {E-Shop,Catalogue,Payment,Bank Transfer,Security,
Standard,Search}
P7 = {E-Shop,Catalogue,Payment,Bank Transfer,Security,
Standard,Search,Public report}
P8 = {E-Shop,Catalogue,Payment,Credit Card,Security,High,
Search}
P9 = {E-Shop,Catalogue,Payment,Credit Card,Bank Transfer,
Security,High,Search}
```

Commonality: This operation takes a feature model and a feature as inputs and returns the commonality of the feature in the SPL represented by the model. Commonality is a metric that indicates the reuse ratio of a feature in an SPL, this is, the percentage of products that include the feature. This operation is calculated as follows:

$$Comm(f, fm) = \frac{filter(fm, f)}{\#products(fm)} \tag{1}$$

$\#products(fm)$ returns the number of products of an input feature model, fm, and $filter(fm, f)$ returns the number of products in fm that contain the feature f. The result of this operation is in the domain [0,1].

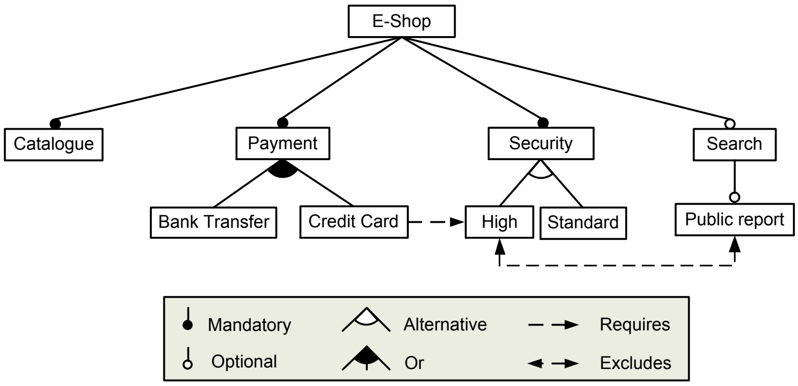


Figure 1: A sample feature model.

As an example, consider the model in Figure 1 and the feature *Credit Card*. The commonality of this feature is calculated as follows:

$Comm(f, fm) = \frac{filter(fm, Credit\,Card)}{\#products(fm)} = \frac{4}{9} = 0.45$

The *Credit Card* feature is therefore included in 45% of the products. A more generic definition of this operation is presented in [4].

Cross-Tree-Constraints Ratio (CTCR): This operation takes a feature model as input and returns the ratio of the number of features in the cross-tree constraints (repeated features counted once) to the total number of features in the model [3, 4, 19]. This metric is usually expressed as a percentage value. This operation is calculated as follows:

$$CTCR(fm) = \frac{\#constraintsfeatures(fm)}{\#features(fm)} \quad (2)$$

$\#constraintsfeatures(fm)$ is the number of features involved in the cross-tree constraints and $\#features(fm)$ is the total number of features of the model fm. The result of this operation is in the domain [0,1]. For instance, the CTCR of the model in Figure 1 is $3/10 = 0.3$ (30%).

Coefficient of Connectivity-Density (CoC): In graph theory, this metric represents how well the graph elements are connected. Bagheri et al. [3] defined the CoC of a feature model as the ratio of the number of edges (any connection between two features, including constraints) over the number of features in a feature model. This is calculated as follows:

$$CoC(fm) = \frac{\#edges(fm)}{\#features(fm)} \quad (3)$$

$\#edges(fm)$ denotes the number of parent-child connections plus the number of cross-tree constraints of an input model *fm* and $\#features(fm)$ is the number of total features in the model fm. For instance, the model in Figure 1 has 11 edges (9 parent-child connections plus 2 constraints) and 10 features, i.e. $CoC(fm) = 11/10 = 1.1$.

Cyclomatic Complexity (CC): The cyclomatic complexity of a feature model can be described as the number of distinct cycles that can be found in the model [3]. Since a feature model is a tree, cycles can only be created by cross-tree constraints. Hence, the cyclomatic complexity of a feature model is equal to the number of cross-tree constraints of the model. In Figure 1, $cc(fm) = 2$.

Variability Coverage (VC): The variability coverage of a feature model is the number of variation points of the model [10]. A variation point is any feature that provide different variants to create a product. Thus, the variation points of a feature model are the optional features plus all non-leaf features with one or more non-mandatory subfeatures. In Figure 1, $vc(fm) = 5$.

2.2 Combinatorial SPL testing

Testing an SPL is a challenging activity compared to testing single systems. Although testing each SPL product individually would be ideal, it is too expensive in practice. In fact, the number of possible products derived from a feature model usually increases exponentially when the number of features grows, leading to thousand or even millions of different products. In this context, there have been many attempts to reduce the space of testing through test selection [7, 16, 22, 24]. The goal of *test selection approaches* is to reduce the set of feature combinations to a reasonable but representative set of products achieving a high coverage of feature interactions [7]. Most common test selection approaches are those based on combinatorial testing [16, 22, 23, 24]. In these approaches test cases are selected in a way that guarantees that all combinations of t features are tested, this is called t-wise testing [24]. One of the best-known variants of combinatorial testing is the 2-wise (or pairwise) testing approach [16]. This proposal generates all possible combinations of pairs of features based on the observation that most of faults originate from a single feature or by the interaction of two features [23]. As an example, Table 1 depicts the set of products obtained when applying 2-wise selection to the model in Figure 1. The rows of the table represent features and the columns products. An “X” means that the feature of the row is included in the product of the column, and a gap means that the feature is not included. The number of products to be tested is reduced from 9 to 6. Oster et al.

[22] achieved to reduce the number of products of Electronic Shopping model [21] from $2.26 \cdot 10^{49}$ to 62 products using pairwise coverage.

Features/Products	P1	P2	P3	P4	P5	P6
Bank Trasnsfer		X	X	X	X	
Payment	X	X	X	X	X	X
Security	X	X	X	X	X	X
High	X			X	X	X
Catalogue	X	X	X	X	X	X
Public Report		X				
Credit Card	X				X	X
E-Shop	X	X	X	X	X	X
Standard		X	X			
Search		X		X	X	X

Table 1: 2-wise coverage results for SPL in Figure 1

2.3 Test case prioritization

Running all the test cases in an existing test suite can suppose a large amount of effort or even become infeasible due to deadlines and cost constraints. Rothermel et al. [30] reported about an industrial application of 20,000 lines of code whose test suite required seven weeks to be run. For these reasons, various techniques for reducing the cost of testing have been proposed including test case prioritization techniques. *Test case prioritization techniques* [5, 18, 29, 30] schedule test cases for execution in an order that attempts to increase their effectiveness at meeting some performance goal [30]. There are many possible goals of prioritization [30]. For example, testers may wish to order their test cases in order to reduce the cost of testing (e.g. measuring the testing execution time) or increase the rate of critical fault detection of test cases. Furthermore, given a prioritization goal, various prioritization criteria may be applied to a test suite with the aim of meeting that goal. For instance, in an attempt to increase the rate of fault detection, we could prioritize test cases in terms of the complexity of the system giving priority to the test cases that exercise the most complex components, e.g. those with the higher cyclomatic complexity. Alternatively, we could order test cases according to their coverage running first those test cases which exercise a larger portion of the code.

3. TEST CASE PRIORITIZATION CRITERIA FOR SOFTWARE PRODUCT LINES

In this section, we define and compare five test case prioritization criteria to maximize the rate of early fault detection of an SPL test suite. This goal aims to achieve a sequence of test cases to be run in a way that faults are detected as soon as possible. This enables faster feedback about the system under test and lets developers begin correcting faults earlier. Hence, it could provide faster evidence that quality objectives were not met and the assurance that those test cases with greatest fault detection ability will have been executed if testing is halted [31].

Figure 2 depicts a rough overview of the general SPL testing process and how our prioritization approach fits on it.

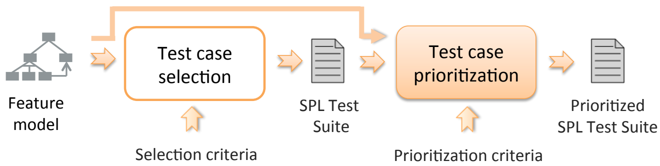


Figure 2: Overview of the SPL testing process

First, the variability model (usually a feature model) is inspected and the set of products to be tested is selected. The selection could be done either manually (e.g. selecting the product portfolio of the company) or automatically (e.g. using t-wise). Once the suite is selected, specific test cases should be designed for each product under test. Then, the set of products to be tested could be prioritized according to multiple criteria determining the execution order of the test cases. Some of the criteria may need analyzing the feature model or using feedback from previous test executions during regression testing. A point to remark is that prioritization does not require creating new test cases, just reordering the existing ones. As a result, prioritization could be reexecued as many times as needed with different criteria. For instance, during the initial stages of development products could be reordered to maximize feature coverage and during regression testing they could be reordered to detect critical faults as soon as possible, e.g. those causing failures in a higher number of products. The prioritization criteria proposed are presented in the next sections.

3.1 CTCR prioritization criterion

This criterion is based on the Cross-Tree-Constraints Ratio (CTCR) defined in Section 2.1. The CTCR metric has been used to calculate the complexity of feature models and it is correlated with the possibility and ease of change in a model when modifications are necessary [3]. This metric inspired us to define the CTCR prioritization criterion as a way to identify the more complex products in terms of the degree of involvement in the constraints of their features. We hypothesize that this criterion can reduce testing effort while retaining a good fault detection rate by testing earlier the more complex products in terms of constraints.

Given a product p and a feature model fm, we define the CTCR criterion as follows:

$$CTCR(p, fm) = \frac{\#constraintsfeatures(p, fm)}{\#features(p)} \tag{4}$$

$\#constraintsfeatures(p, fm)$ denotes the number of distinct features in p involved in constraints and $\#features(p)$ is the number of total features in product p. This formula returns a value that indicates the complexity of product p in terms of features involved in constraints.

As an example, the CTCR prioritization value of the products P4 and P6 presented in Section 2.1 is calculated as follows:

$CTCR(P4, fm) = 2/7 = 0.28$

P4={E-Shop,Catalogue,Payment,Bank Transfer,Credit Card, Security,High}

$CTCR(P6, fm) = 0$
P6={E-Shop,Catalogue,Payment,Bank Transfer,Security, Standard,Search}

In P4, the *Credit Card* and *High* Security features share an *include* constraint and also *High* Security feature has an exclude constraint with *Public report*. However, the features in P6 do not involve any constraints. Thus, product P4 will be tested earlier than product P6 according to the CTCR values i.e. $CTCR(P4, fm) > CTCR(P6, fm)$.

3.2 CoC prioritization criterion

The Coefficient of Connectivity-Density (CoC) metric, presented in Section 2.1, was proposed to calculate the complexity of a feature model in terms of the number of edges and constraints of the model [3]. We propose to adapt this metric for SPL products and use it as a test case prioritization criterion. The goal is to measure the complexity of products in terms of their CoC and give higher priority to those products with higher complexity. Given a product p and a feature model fm, we define the CoC of a product as shown below:

$$CoC(p, fm) = \frac{\#edges(p, fm)}{\#features(p)} \tag{5}$$

$\#edges(p, fm)$ denotes the number of edges (parent-child connections plus cross-tree constraints) among the features in the product p. This formula returns a value that indicates the complexity of p based on the coefficient of connectivity-density.

As an example, the CoC value of the products P7 and P9 derived from the model in Figure 1 is calculated as follows:

$CoC(P7, fm) = 8/8 = 1$
P7 = {E-Shop,Catalogue,Payment,Bank Transfer,Security, Standard,Search,Public report}

$CoC(P9, fm) = 9/8 = 1.13$
P9 ={E-Shop,Catalogue,Payment,Credit Card,Bank Transfer, Security,High,Search}

In P7, the *E-Shop* feature is connected with edges to four features (*Catalogue*, *Payment*, *Security* and *Search*). Also, *Payment* is connected to *Bank Transfer* feature, *Security* to *Standard* Security feature, *Search* to *Public report* and *Public report* has an exclude constraint with *High* Security. Note that the exclude constraint is considered because it is being fulfilled by this product since it includes *Public report* feature and not *High* Security feature. Thus, P9 has higher priority than P7 and therefore it would be tested first.

3.3 VC&CC prioritization criterion

In [10], the authors presented a genetic algorithm for the generation of SPL products with an acceptable tradeoff between fault coverage and feature coverage. As part of their algorithm, they proposed a fitness function to measure the ability of a product to exercise features and reveal faults, i.e. the higher the value of the function, the better the product. This function is presented below:

$$VC\&CC(p, fm) = \sqrt{vc(p, fm)^2 + cc(p, fm)^2} \tag{6}$$

$vc(p, fm)$ calculates the variability coverage of a product p of the model fm (i.e. number of bounded variation points of the product) and $cc(p, fm)$ represents the cyclomatic complexity of p, i.e., the number of constraints enforced in the product.

Since this function has been successfully applied to SPL test case selection, we propose to explore its applicability for test case prioritization. According to this criterion, those products with higher values for the function are assumed to be more effective in revealing faults and will be tested first.

As an example, the VC&CC value of the products P3 and P6 derived from the model in Figure 1 is calculated as follows:

$VC\&CC(P3, fm) = \sqrt{3^2 + 2^2} = \sqrt{9 + 4} = 3.6$
P3={E-Shop,Catalogue,Payment,Credit Card,Security,High}

$VC\&CC(P6, fm) = \sqrt{4^2 + 0^2} = \sqrt{16 + 0} = 4$
P6={E-Shop,Catalogue,Payment,Bank Transfer,Security, Standard,Search}

In product P3, *E-Shop*, *Payment* and *Security* features are variation points. Also, P3 presents a require constraint with *Credit Card* and *High* Security features and an exclude constraint between *High* Security and *Public report*. According to this criterion, product P6 would be tested earlier than product P3, $VC\&CC(P6) > VC\&CC(P3)$.

3.4 Commonality prioritization criterion

We define a commonality-based prioritization criterion that calculates the degree of reusability of products features. That is, the features that have higher commonality and the products that contain them will be given priority to be tested. This enables the early detection of faults in highly reused features that affect to a high portion of the products providing faster feedback and letting software engineers begin correcting critical faults earlier.

Given a product p and a feature model fm, we define the Commonality criterion as follow.

$$Common(p, fm) = \sum_{i=1}^{\#features(p)} (Comm(fi))/\#features(p) \tag{7}$$

fi denotes a feature of product p. The range of this measure is [0,1]. Roughly speaking, the priority of a product is calculated by summing up the commonality of its features. The sum is then normalized according to the number of features of the product.

As an example, the Commonality value of the products P1 and P2 derived from the model in Figure 1 is calculated as follows:

$Comm(P1, fm) = (Comm(EShop) + Comm(Catalogue)$
$+Comm(Payment) + Comm(BankTransfer)+$
$Comm(Security) + Comm(High))/6 =$
$((9 + 9 + 9 + 7 + 9 + 6)/9)/6 = 5.44/6 = 0.91$
P1={E-Shop,Catalogue,Payment,Bank Transfer,Security, High}

$Comm(P2, fm) = ((9 + 9 + 9 + 7 + 9 + 3)/9)/6 = 0.85$
P2={E-Shop,Catalogue,Payment,Bank Transfer,Security, Standard}

Based on the results, product P1 would appear before than product P2 in the prioritized list of products and would be tested first.

3.5 Dissimilarity prioritization criterion

A (dis)similarity measure is used for comparing similarity (diversity) between a pair of test cases. Hemmati et al. and Henard et al. [11, 12] investigated ways to select an affordable subset with maximum fault detection rate by maximizing diversity among test cases using dissimilarity measure. The results obtained in that paper suggested that two dissimilar test cases have a higher fault detection rate than similar ones since the former ones are more likely to cover more features than the latter.

In this context, we propose to prioritize the test cases based on this dissimilarity metric, testing the most different test cases first, assuring a higher feature coverage and a higher fault detection rate. In order to measure the diversity between two products, we use the Jaccard distance which compare similarity of sample sets [34]. Jaccard distance is defined as the size of the intersection divided by the size of the union of the sample sets. In our context, each set represents a product containing a set of features. Thus, we choose first the two more dissimilar products (that is, the products with the highest distance between them) and we add them to a list. Then, we continue adding the products with the highest distance between them until all products have been added to the list. The resulting list of products represents the order of products to be tested.

Given two products p_a and p_b, we define the Dissimilarity formula as follows:

$$Dissimilarity(p_a, p_b) = 1 - \frac{|p_a \bigcap p_b|}{|p_a \bigcup p_b|} \qquad (8)$$

p_a and p_b represent different set of features (i.e. products). The resulting distance varies between 0 and 1, where 0 denotes that the products p_a and p_b are the same and a value close to 1 indicates that p_a and p_b share no features (excluding mandatory features).

The dissimilarity of the products P1, P7 and P8 in Figure 1 is calculated as follows:

$Dissimilarity(P1, P7) = 1 - 5/9 = 0.44$
$Dissimilarity(P7, P8) = 1 - 5/10 = 0.5$
P1={E-Shop,Catalogue,Payment,Bank Transfer,Security, High}
P7={E-Shop,Catalogue,Payment,Bank Transfer,Security, Standard,Search,Public report}
P8={E-Shop,Catalogue,Payment,Credit Card,Security, High,Search}

For example, regarding to the distance between P1 and P7, they have 5 features in common (i.e. *E-Shop, Catalogue, Payment, Bank Transfer and Security*) out of 9 total features (i.e. the previous five plus *High, Standard, Search, Public report*). P7 and P8 present greater distance between them than P7 and P1. Thus, products P7 and P8 would be tested earlier than P1.

4. EVALUATION

In this section, we present two experiments to answer the following research questions:

RQ1: Is the order in which SPL products are tested relevant?
RQ2: Are the prioritization criteria presented in Section 2 effective at improving the rate of early fault detection of SPL test suites?
RQ3: Can our prioritization approach improve the rate of early fault detection of current test selection techniques based on combinatorial testing?

We begin by describing our experimental settings and then we explain the experimental results.

4.1 Experimental settings

In order to assess our approach, we developed a prototype implementation for each prioritization criterion. Our prototype takes an SPL test suite and a feature model as inputs and generates an ordered set of test cases according to the prioritization criterion selected. We used the SPLAR tool [20] for the analysis of feature models to implement our prototype. All the performed experiments were implemented using Java 1.6. We ran our tests on a Linux CentOS release 6.3 machine equipped with an Intel Xeon X5560@2.8Ghz microprocessor and 4 GB of RAM memory.

4.1.1 Models

For our experiments we selected 7 feature models of various sizes from the SPLOT repository [21]. Also, we generated 8 random models with up to 500 features using the BeTTy online feature model generator [32]. Table 2 lists the characteristics of the models. For each model, the name, the number of features, the number of products and the CTCR are presented.

4.1.2 Fault generator

In order to measure the effectiveness of our proposal, we evaluated the ability of our test case prioritization criteria to detect faults in the SPL under test. For this purpose, we implemented a fault generator for feature models. This generator is based on the fault simulator presented by Bagueri et al. which has been used in several works to evaluate the fault detection rate [2, 10]. Our fault generator produces faults in n-tuples of features where n can be 1, 2, 3 or 4 features. We considered these kind of faults because there are studies that show that they appear in real tools quite frequently [6, 15]. All these types of faults appear in the same proportion. For this, randomly features are selected to be seeded with faults. Thus, our generator receives as input a feature model and returns as output a list of faulty features.

Name	Features	Products	CTCR	Faults
Web portal	43	2120800	25%	4
Video player	71	$4,5 \cdot 10^{13}$	0%	4
Car selection	72	$3 \cdot 10^{8}$	31%	4
Model transf.	88	$1 \cdot 10^{12}$	0%	8
Fm test	168	$1,9 \cdot 10^{24}$	28%	16
Printers	172	$1,14 \cdot 10^{27}$	0%	16
Electronic shop	290	$4,52 \cdot 10^{49}$	11%	28
Random1	300	$7,65 \cdot 10^{39}$	8%	28
Random2	300	$1,65 \cdot 10^{32}$	5%	28
Random3	350	$7,41 \cdot 10^{37}$	10%	32
Random4	400	$3,06 \cdot 10^{44}$	10%	40
Random5	450	$3,80 \cdot 10^{54}$	0%	44
Random6	450	$1,03 \cdot 10^{48}$	5%	44
Random7	500	$4,97 \cdot 10^{36}$	5%	48
Random8	500	$2,21 \cdot 10^{58}$	5%	48

Table 2: Feature models used in our experiments

For simplicity, we considered a size of this list approximate to the 10% of the number of model features. The number of faults introduced on each SPL is detailed in the last column of Table 2. Faults were marked as detected by a test case if it included the features containing the fault.

4.1.3 Evaluation metric

In order to evaluate how quickly faults are detected during testing we used the *Average Percentage of Faults Detected (APFD)* metric [28, 30, 31, 33]. The APFD metric measures the weighted average of the percentage of faults detected during the execution of the test suite. To formally illustrate APFD, let T be a test suite which contains n test cases, and let F be a set of m faults revealed by T. Let *TFi* be the position of the first test case in ordering *T'* of T which reveals the fault i. According to [8], the APFD metric for the test suite *T'* could be given by the equation:

$$APFD = 1 - \frac{TF_1+TF_2+...+TF_n}{n \times m} + \frac{1}{2n}$$

APFD value ranges from 0 to 1. A prioritized test suite with higher APFD value has faster fault detection rates than those with lower APFD values.

For example, consider a test suite of 4 test cases, T1 through T4, and 5 faults detected by those test cases, as shown by the table in Figure 3. Consider two orderings of these test cases, ordering O1: T1,T2,T3,T4, and ordering O2: T3,T2,T4,T1. According to the previous APFD equation, ordering O1 produces an APFD of 58% ($1 - \frac{1+1+2+3+4}{4\times5} + \frac{1}{2\times4} = 0.58$) and ordering O2 an APFD of 78% ($1 - \frac{1+1+1+1+3}{4\times5} + \frac{1}{2\times4} = 0.78$), being O2 much faster detecting faults than O1.

4.2 Experiment 1. Prioritizing SPL test suites

In order to answer *RQ1* and *RQ2*, we checked the impact on the rate of early fault detection of the prioritization criteria defined in Section 3. The experimental setup and the results are next reported.

Tests/Faults	F1	F2	F3	F4	F5
T1	X	X			
T2	X		X		
T3	X	X	X	X	
T4					X

Table 3: Test suite and faults exposed

Experimental setup. For each model presented in Table 2, we performed several steps. First, we used our fault generator to simulate faults in the SPL obtaining as a result a list of n-tuples faulty features. Then, we randomly generated a test suite using SPLAR. The suite was composed of between 100 and 500 products depending on the size of the model. This step simulates the manual test case selection of an SPL engineer who could choose the products to be tested following multiple criteria: cost, release plan, users requests, marketing strategy, etc. For each fault in the model, we made sure that there was at least one product in the suite detecting it, i.e. the suite detected 100% of the faults. Once generated, the suite was ordered according to the prioritization criteria defined in Section 3 resulting in six total test suites, one random suite and five prioritized suites. Finally, we measured how fast the faults were detected by each suite calculating their APFD. Faults were marked as detected by a test case if it included the features containing the fault. The APFD of the random suite was calculated as the average of 10 random orderings to avoid the effects of chance.

Experimental results. Table 4 depicts the size of the test suites and the APFD values obtained by the random and the five prioritized suites for each model. The best value on each row is highlighted in boldface. Also, main values are shown in the final row. As illustrated, there are significant differences on the APFD average values ranging from the 74.9% of the Commonality-ordered suite to the 96.5% reached by the VC&CC-ordered suite. These differences are even more noticeable in individual cases. For the model “Random3”, for instance, the difference between the random and VC&CC-ordered suite is 33.3 points, i.e. from 63.9% to 97.2%. The best average results were obtained by the VC&CC criterion with 96.5%, followed by CoC (90.6%), Dissimilarity (87.4%), CTC (84.5%), random criterion (77.4%) and Commonality (74.9%). Interestingly, the APFD values obtained by the random suites in the models of lower size were remarkably high, e.g. $APFD(Videoplayer) = 92.0\%$. We found that this was due to the low number of faults seeded and to the size of the models that made the faults easily detectable using just a few test cases. In the eight largest models, however, the difference between the APFD of the random suite (63.9%-77.5%) and the ones of the prioritized suites (91.5%-98.2%) was noticeable. This suggests that our prioritization approach is especially helpful in large test spaces. Finally, we may remark that all the proposed prioritization criteria except Commonality improved the average results obtained by the random ordering. In fact, for all models at least several of our prioritized suites improved the random APFD values.

Figure 3 shows the percentage of detected faults versus the fraction of the test suite used for the model "Random3". Roughly speaking, the graphs show how the APFD value evolves as the test suite is exercised. It is noteworthy that the VC&CC-ordered suite, for instance, detected all the faults (32) by using just 15% of the suite, i.e. 75 test cases out of 500 with the highest priority. Another example is the Dissimilarity-ordered suite that detected all the faults by using only the 45% of the suite. The random suite, however, required to use 95% of the test cases to detect exactly the same faults. This behaviour was also observed in the rest of the models under study. In real scenarios, with a higher number of faults and time-consuming executions, this acceleration in the detection of faults could imply important saving in terms of debugging efforts.

The results obtained answer positively to *RQ1* and *RQ2*. Regarding *RQ1*, the results show that the order in which test cases are run is definitely relevant and can have a clear impact on the rate of early fault detection of an SPL test suite. Regarding *RQ2*, the results suggest that the presented prioritization criteria, especially the VC&CC, CoC, Dissimilarity and CTC criteria, could be effective at improving the rate of early fault detection of SPL test suites.

4.3 Experiment 2. Prioritization + combinatorial testing

In order to answer *RQ3*, we checked whether our prioritization criteria could be used to increase the rate of fault detection of test suites based on combinatorial selection. The experimental setup and results are next reported.

Experimental setup. The experimental procedure was similar to the one used in Experiment 1. Feature models were seeded with the same faults used in our previous experiment. Then, for each model, we generated a 2-wise test suite using the SPLCAT tool presented by Johansen et al. [13]. As a result, we obtained a list of products covering all the possible pairs of features on each model. Then, we prioritized the list of products according to our five prioritization criteria and we calculated the APFD of the resulting six suites, 2-wise and five prioritized suites. It is noteworthy that SPLCAT uses an implicit prioritization criteria placing first in the list those products which covers the most uncovered pairs of features. This tends to place those products with more features at the top of the list getting a fast feature coverage. This approach therefore is likely to increase the rate of fault detection and thus it is considered as an extra prioritization approach in our comparison.

Experimental results. The results of this experiment are presented in Table 5. For each model, the size of the pairwise test suite, the number of faults detected out of the total number of seeded faults and the APFD values of each ordering are presented. Note that the generated pairwise suites did not detect all the faults seeded on each model. As illustrated, the average APFD values ranged from 55.0% to 90.7%. As expected, the APFD average value of the pairwise suite (85.0%) was higher than the one of the random suite (77.4%) in Experiment 1. This was due to implicit prioritization criteria used by the SPLCAT tool which places at the top of the list those products containing a higher number of uncovered pairs of features, usually the largest products. As in the previous experiment, the best APFD average results were obtained by the VC&CC criterion with 90.7%, followed by CoC (88.0%) and Dissimilarity (86.9%). The pairwise suite got the fourth best APFD average value (85.0%). The CTC and Commonality prioritization criteria did not get to improve the results of the original suite. In terms of individual values, CoC got to improve the pairwise APFD values in 13 out of 15 models, VC&CC in 12 out of 15 models and Dissimilarity in 11 out of 15. As in our previous experiment, there was not a single model in which the pairwise suite obtained a higher APFD value than the rest of prioritized suites.

Figure 4 shows the percentage of detected faults versus the fraction of test suite used for the feature model "Random3". In this example, it is remarkable that VC&CC-ordered suite detected all the faults (i.e. the 28 faults that revealed the pairwise approach) with just 20% of the suite (i.e. 27 test cases out of 135). Furthermore, the CoC-ordered and the pairwise suites detected the same faults with only the 50% of the suite. However, the CoC-ordered suite achieved to detect more faults earlier, i.e. with just the 25% of the suite, CoC detected the 85% of the faults, whereas, the pairwise suite only detected the 75% of faults. A similar behaviour was also observed in the rest of the models under study.

In response to *RQ3*, our results show that our prioritization criteria can be helpful to increase the rate of early fault detection of the current combinatorial testing techniques.

5. THREATS TO VALIDITY

Despite our best efforts these experiments suffer from some threats to validity. In order to avoid any bias in the implementation and make our work reproducible, we used a number of validated and publicly available tools. In particular, we used the tool SPLAR [20] for the analysis of feature models, BeTTy [32] for the generation of random feature models, SPLCAT [13] for pairwise test selection and we implemented a fault generator based on the fault simulator presented in [10]. Due to the lack of real SPLs with available test cases, we evaluated our approach by simulating faults in a number of SPLs represented by published and randomly generated models of different sizes. This may be a threat to our conclusions. However, we may remark that the evaluation of testing approaches using feature models is extensively used in the literature [12, 14, 13].

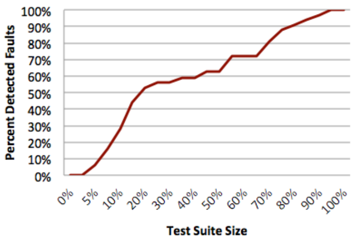
The use of a fault generator also implies several threats. The type, number and distribution of generated faults could not be the one found in real code. We may emphasize, however, that our generator is based on the fault simulator presented by Bagheri et al. [3] which has been validated in the evaluation of several SPL test case selection approaches [2, 10]. Furthermore, we remark that the characteristics and distribution of faults have a limited impact in our work since we are not interested in how many faults are detected but how fast they are revealed by different orderings of the same suite.

6. RELATED WORK

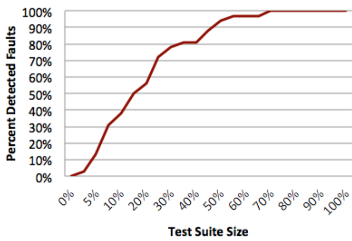
The challenges of software product lines testing have been extensively discussed [25]. Some of these challenges such as complex interaction between features and the large number

FM	Suite size	APFD					
		Random	CoC	CTC	Comm	VC&CC	Diss
Web portal	100	81.5	95.8	94.3	60.0	**99.0**	92.3
Car selection	100	83.6	96.5	94.8	92.5	**97.8**	90.5
Video player	100	92.0	**98.8**	93.0	76.0	**98.8**	97.3
Model transf.	100	83.3	94.8	75.5	79.9	**95.4**	91.5
Fm test	300	85.2	87.3	83.4	84.3	**94.3**	93.6
Printers	300	92.9	94.1	**98.4**	93.9	96.3	96.5
Electronic shop	300	90.2	93.4	92.1	87.6	96.3	**97.0**
Random1	300	64.7	92.6	84.2	60.1	**97.9**	89.1
Random2	300	73.1	87.9	78.6	77.5	**98.2**	80.4
Random3	500	63.9	79.6	70.7	80.7	**97.2**	85.2
Random4	500	69.7	93.9	92.2	53.8	**97.8**	65.8
Random5	500	77.5	**91.6**	78.4	73.7	91.5	89.1
Random6	500	69.5	83.5	73.6	43.8	**95.1**	88.8
Random7	500	66.3	90.4	81.6	72.6	**95.7**	87.0
Random8	500	67.3	79.3	76.3	86.3	**95.9**	67.0
Average		**77.4**	**90.6**	**84.5**	**74.9**	**96.5**	**87.4**

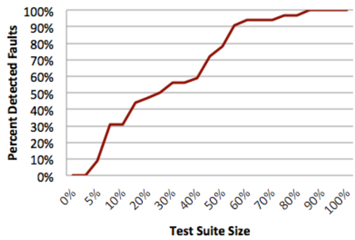
Table 4: APFD for random and prioritized suites



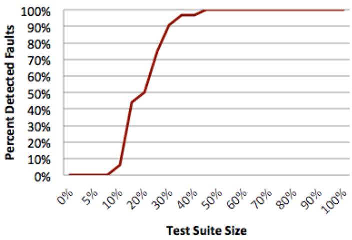
(a) Random APFD



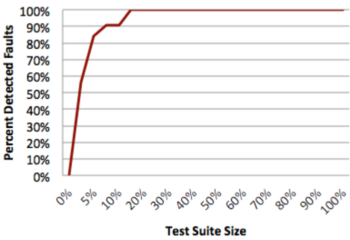
(b) CoC APFD



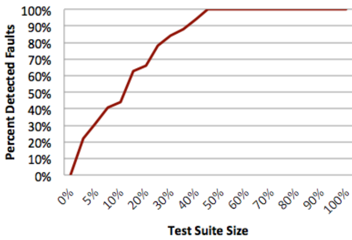
(c) CTC APFD



(d) Commonality APFD



(e) VC&CC APFD



(f) Dissimilarity APFD

Figure 3: APFD metrics for 350F10CTC0-4 model

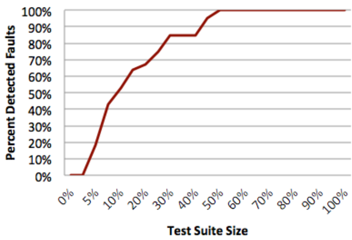
of possible configurations to be tested have been a research topic in recent years [7, 12, 16, 23].

Concerning the reduction of the number of test cases, Sebastain Oster et al. [22] provided a description of a methodology to apply combinatorial testing to a feature model of an SPL combining graph transformation, combinatorial testing and forward checking. In [16], the authors implemented several combinatorial testing techniques adapted to the SPL context. Additionally, in [2] proposed eight coverage criteria using a grammar-based technique to reduce the number of test cases to be tested, and in [10] presented a search-based approach using Genetic Algorithms to generate reduced test suites too. In this paper, we focus our attention to the combinatorial approaches. In particular, we make use of the tool proposed by Johansen et al. [13] that calculates the 2-wise covering. Even though this is an efficient pairwise tool, we get to improve its results using our prioritization approach complementary. In our work, we propose an approach in which we consider not only the specific set of test cases but also the order in which these tests will be tested. As a special case of tests selection, we mention the work performed by Henard et al. [12] that proposed t-wise covering and prioritization to generate products based on similarity heuristics. However, our work is focused on, once the test cases have been selected to be tested, reordering them according to different criteria in order to obtain an early fault detection.

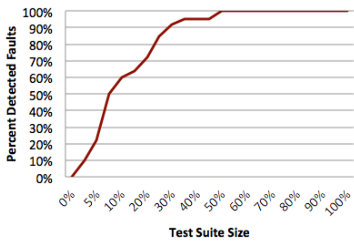
Respecting the test case prioritization, Rothermel et al. [30]

FM	Suite size	Detected faults	APFD					
			2wise	**CoC**	**CTC**	**Comm**	**VC&CC**	**Diss**
Web portal	19	3(4)	90.3	93.9	90.4	46.5	**97.4**	**97.4**
Car selection	24	4(4)	81.2	**90.6**	**90.6**	51.0	80.2	71.9
Video player	18	4(4)	76.4	**93.1**	76.4	22.2	83.3	83.3
Model transf.	28	7(8)	78.8	**88.0**	78.8	49.2	85.5	80.9
FM test	43	15(16)	85.0	68.8	55.3	53.6	**93.9**	75.4
Printers	129	13(16)	96.2	**97.1**	96.2	58.0	89.5	96.8
Electronic shop	24	24(28)	81.7	82.6	79.7	55.6	78.8	**85.2**
Random1	124	23(28)	81.1	83.4	83.6	61.7	**96.6**	92.2
Random2	105	25(28)	86.2	91.6	90.1	52.3	**94.8**	90.0
Random3	135	28(32)	84.7	87.4	84.7	68.8	**97.1**	89.9
Random4	178	34(40)	86.9	88.8	87.8	62.9	**95.7**	90.7
Random5	126	38(44)	86.9	**94.9**	86.9	61.3	94.6	80.6
Random6	157	39(44)	85.1	86.1	82.0	48.7	**92.4**	88.3
Random7	253	37(48)	84.9	84.2	79.2	63.1	85.8	**91.8**
Random8	216	40(48)	89.2	90.4	86.1	70.8	**95.1**	88.3
Average			**85.0**	**88.0**	**83.2**	**55.0**	**90.7**	**86.9**

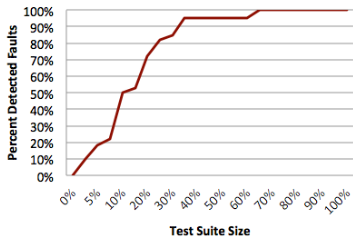
Table 5: APDF for 2-wise and prioritized suites



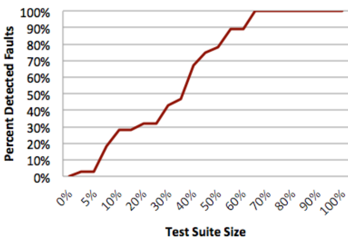
(a) 2-wise APFD



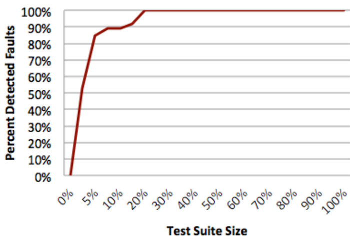
(b) CoC APFD



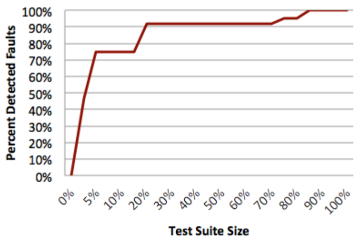
(c) CTC APFD



(d) Commonality APFD



(e) VC&CC APFD



(f) Dissimilarity APFD

Figure 4: APFD metrics for 350F10CTC0-4 model

proposed several test prioritization techniques for regression testing by using test execution information with the aim of obtain cost-benefits tradeoffs. Zhang et al. [37] used the total and additional prioritization strategies to prioritize based on total numbers of elements covered per test, and numbers of additional (not-yet-covered) elements covered per test with the aim of increasing the rate of fault detection. The work presented in [9] proposed an approach to reduce the SPL test space using an goal-oriented method to select and prioritize the most desirable features from feature models. Part of our work is also focused on reflecting the more relevant features of a product line, however, we use different prioritization criteria as the complexity of the features, the degree of reusability or the dissimilarity among the products features in order to accelerate the detection of faults. Also, these criteria are based on standard metrics for the analysis of feature models and therefore are fully automatic.

Another works about prioritization of configurable systems in general are those presented in [27, 28, 33]. In [28], Qu et al. examined several Combinatorial Iteration Testing (CIT) prioritization techniques and compared them with a re-generation/prioritization approach. The last approach is a combined generation and prioritization technique, rather than pure prioritization, since it does not re-order tests, but re-generates them each time. Srikanth et al. [33] studied the prioritization of configurable software systems driven not only by fault detection but also by the cost of configuration and setup time. In our work, we also present an approach that can combine combinatorial testing and different prioritization criteria to detect faster faults. However, we focuses on SPLs and, in particular, we adapt our implementation to feature models since those are widely used for SPLs community. This allows this work can be useful for a large amount of researchers.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a test case prioritization approach for SPLs. In particular, we proposed five prioritization criteria to schedule test execution in an order that attempt to accelerate the detection of faults providing faster feedback and reducing debugging efforts. These prioritization criteria are based on standard techniques and metrics for the analysis of feature models and therefore are fully automated. The evaluation results show that there are significant differences in the rate of early fault detection provided by different prioritization criteria. Also, the results show that some of the criteria proposed may contribute to accelerate the detection of faults of both random and pairwise-based SPL test suites. This suggests that our work could be a nice complement for current techniques for test case selection. To the best of our knowledge, our work is the first considering not only *which* SPL products should be tested but *how* they should be tested. The main conclusion of this work is that the order in which SPL test cases are run does matter.

Many challenges remain for our future work. First and foremost, we plan to further validate our approach on the source code of a real ecosystems such as FaMa and Eclipse. Also, we plan to work on new prioritization criteria exploiting the analysis of non-functional properties, e.g. order tests according to their cost. Test case prioritization technique has shown to be especially helpful during regression testing. We also intend to work on that direction by defining prioritization criteria based on the feedback from previous tests.

Material

Our test case prioritization tool together with the feature models and the seeded faults used in our evaluation are available at `www.isa.us.es/~isaweb/anabsanchez/material.zip`

8. ACKNOWLEDGMENTS

This work was partially supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (grants TIN2009-07366 (SETI), TIN2012-32273 (TAPAS), TIC-5906 (THEOS)).

9. REFERENCES

[1] AHEAD Tool Suite. `http://www.cs.utexas.edu/users/schwartz/ATS.html`, accessed April 2013.

[2] E. Bagheri, F. Ensan, and D. Gasevic. Grammar-based test generation for software product line feature models. In *Conference of the Centre for Advanced Studies on Collaborative Research*, 2012.

[3] E. Bagheri and D. Gasevic. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Control*, 2011.

[4] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analyses of feature models 20 years later: A literature review. *Information Systems*, 2010.

[5] C. Catal and D. Mishra. Test case prioritization: a systematic mapping study. *Software Qual*, 2012.

[6] D. R. W. D. Richard Kuhn and A. M. Gallo. Software fault interactions and implications for software testing. *Transactions on Software Engineering*, 30, 2004.

[7] I. do Carmo Machado, J. D. McGregor, and E. S. de Almeida. Strategies for testing products in software product lines. *SIGSOFT Software Engineering*, 2012.

[8] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky. Selecting a cost-effective test case prioritization technique. *Software Quality Journal*, 2004.

[9] A. Ensan, E. Bagheri, M. Asadi, D. Gasevic, and Y. Biletskiy. Goal-oriented test case selection and prioritization for product line feature models. In *International Conference on Information Technology: New Generations*, 2011.

[10] F. Ensan, E. Bagheri, and D. Gasevic. Evolutionary search-based test generation for software product line feature models. In *International Conference on Advanced Information Systems Engineering (CAiSE'12)*, 2012.

[11] H. Hemmati and L. Briand. An industrial investigation of similarity measures for model-based test case selection. In *ISSRE*, 2010.

[12] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. L. Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test suites for large software product lines. Technical report, 2012.

[13] M. F. Johansen, O. Haugen, and F. Fleurey. Properties of realistic feature models make combinatorial testing of product lines feasible. In *MODELS*, 2011.

[14] M. F. Johansen, O. Haugen, and F. Fleurey. An algorithm for generating t-wise covering arrays from large feature models. In *SPLC*, 2012.

[15] D. R. Kuhn and M. J. Reilly. An investigation of the applicability of design of experiments to software testing. In *27th NASA/IEEE Software Engineering Workshop*, 2002.

[16] B. P. Lamancha and M. P. Usaola. Testing product generation in software product lines using pairwise for feature coverage. In *International conference on Testing Software and Systems*, 2010.

[17] D. Leon and A. Podgurski. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. In *International Symposium on Software Reliability Engineering*, 2003.

[18] Z. Li, M. Harman, and R. M. Hierons. Search algorithms for regression test case prioritization. *IEEE Transactions on Software Engineering*, 33, 2007.

[19] M. M., W. A., and C. K. Sat-based analysis of feature models is easy. In *Proceedings of the Sofware Product Line Conference*, 2009.

[20] M. Mendonca. *Efficient Reasoning Techniques for Large Scale Feature Models*. PhD thesis, University of Waterloo, 2009.

[21] M. Mendonca, M. Branco, and D. Cowan. S.p.l.o.t. - software product lines online tools. In *OOPSLA*, 2009.

[22] S. Oster, F. Markert, and P. Ritter. Automated incremental pairwise testing of software product lines. In *SPLC*, 2010.

[23] G. Perrouin, S. Oster, S. Sen, J. Klein, B. Budry, and Y. le Traon. Pairwise testing for software product lines: comparison of two approaches. *Springer*, 2011.

[24] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. le Traon. Automated and scalable t-wise test case generation strategies for software product lines. In *International Conference on Software Testing, Verification and Validation*, 2010.

[25] B. G. V. D. L. F. Pohl, k. Software product line engineering: Foundations, principles, and techniques. *Springer*, 2005.

[26] pure::variants. `http://www.pure-systems.com/`, accessed April 2013.

[27] X. Qu, M. B. Cohen, and G. Rothermel. Configuration-aware regression testing: An empirical study of sampling and prioritization. In *International Symposium in Software Testing and Analysis*, 2008.

[28] X. Qu, M. B. Cohen, and K. M. Woolf. Combinatorial interaction regression testing: A study of test case generation and prioritization. 2007.

[29] G. Rothermel, R. Untch, C. Chu, and M. Harrold. Test case prioritization: An empirical study. In *Proc. Int. Conf. Software Maintenance*, 1999.

[30] G. Rothermel, R. Untch, C. Chu, and M. Harrold. Prioritizing test cases for regression testing. *IEEE Trans. Software Eng*, 27:929–948, 2001.

[31] A. G. M. Sebastian Elbaum and G. Rothermel. Test case prioritization: A family of empirical studies. *Transactions on Software Engineering*, 2002.

[32] S. Segura, J. Galindo, D. Benavides, J. Parejo, and A. Ruiz-Cortés. Betty: Benchmarking and testing on the automated analysis of feature models. In *International Workshop on Variability Modelling of Software-intensive Systems*, 2012.

[33] H. Srikanth, M. B. Cohen, and X. Qu. Reducing field failures in system configurable software: Cost-based prioritization. 2009.

[34] P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2006.

[35] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez. Fama framework. In *International Software Product Line Conference*, 2008.

[36] S. Yoo and M. Harman. Regression testing minimisation, selection and prioritisation : A survey. In *Software Testing, Verification and Reliability*, 2007.

[37] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei. Bridging the gap between the total and additional test-case prioritization strategies. In *ICSE*, 2013.