Assessing Software Product Line Testing via Model-based Mutation: An Application to Similarity Testing

Christopher Henard*, Mike Papadakis*, Gilles Perrouin†,◇, Jacques Klein*, and Yves Le Traon*
*Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg, Luxembourg
†Precise Research Center In Software Engineering (PReCISE), University of Namur, Namur, Belgium

*{firstname.lastname}@uni.lu; †{firstname.lastname}@fundp.ac.be

***Abstract*—Needs for mass customization and economies of scale have pushed engineers to develop Software Product Lines (SPLs). SPLs are families of products sharing commonalities and exhibiting differences, built by reusing software assets abstractly represented by features. Feature models describe the constraints that link the features and allow the configuration of tailored software products. Common SPLs involve hundreds, even thousands of features, leading to billions of possible software products. As a result, testing a product line is challenging due to the enormous size of the possible products. Existing techniques focus on testing based on the product line's feature model by selecting a limited set of products to test. Being created manually or reverse-engineered, feature models are prone to errors impacting the generated test suites. In this paper, we examine ability of test suites to detect such errors. In particular, we propose two mutation operators to derive erroneous feature models (mutants) from an original feature model and assess the capability of the generated original test suite to kill the mutants. Experimentation on real feature models demonstrate that dissimilar tests suites have a higher mutant detection ability than similar ones, thus validating the relevance of similarity-driven product line testing.**

***Keywords*-Mutation, Testing, Feature Models, Software Product Lines, Similarity**

I. Introduction

Customer demands and market pressure forces software engineers to derive a wide range of different products at a low cost. Software Product Line [1], [2] (SPL) techniques and tools allow engineering such families of related products. Such techniques offer to build products by reusing and combining software assets in a systematic way. Some of these assets appear in all products (commonalities) while some do not (variabilities). To compactly represent an SPL, Feature Models (FMs) were introduced [3], each product being abstractly modeled as a combination of common and variable features linked to software assets. FMs allow visualization, reasoning [4] and configuration [5] of tailored software products. They are also ideal candidates for supporting model-based testing of SPLs [6].

Common SPLs involve hundreds or thousands of features, leading to complex FMs and billions of possible software products to configure. For instance, the Linux kernel FM has more than 6,000 features [7]. As a consequence, testing a SPL and the underlying FM is a inherently difficult activity [8]. Indeed, although testing all the products would be ideal, it is rarely feasible in practice since the size of the test suites have to be realistic enough to fit testing budget constraints. Test engineers are thus seeking for solutions to reduce the size of their test suites. In addition, checking manually the validity of the constraints present in the FM is not feasible.

Mutation analysis is a technique which aims at evaluating the quality of the testing process. Generally applied on programs, it involves the modification of the original software artifact into altered versions, called mutants. Each modified version contains a defect willingly introduced. The tests are then evaluated on these mutants to establish whether or not they are able to reveal the introduced problems. Model-based testing uses a model of the system to perform software testing. Its use is to guide the testing process. The underlying idea of this paper is to use the information provided by FMs to establish a mutation approach. In SPL context, mutants can be used to either produce or evaluate test cases. It leads to our first research question:

[RQ1] *How mutation analysis can be performed on model-based software product lines?*

Mutation analysis has been applied to various models, but not on FMs. The use of mutation in literature is twofold. First, it has been used to generate tests [9], [10]. Second, it has been used to evaluate other testing approaches [11], [12]. We focus on the second part. In our context, a test suite represents a set of software products and a mutant can be considered as a fault. In model-based testing, it has been found that dissimilar test suites have a higher fault detection power than similar ones [13]. This similarity heuristic can be used to reduce the size of the test suites by removing similar products. This approach is particularly useful since for SPL, the number of products to test is usually enormous, with potentially billions of possible products to test [8]. Moreover, the benefit of this heuristic has not been thoroughly assessed in the context of SPL testing. It leads to our second research question:

[RQ2] *Do dissimilar test suites have a higher mutant detection rate in the context of software product line and feature model testing?*

◇FNRS Postdoctoral Researcher.

To answer RQ1, we introduce a mutation analysis for SPLs based on FMs. Thus, we produce different erroneous variants of the original FM by introducing possible defects. Then, we evaluate test suites generated from the original FM towards the modified FMs. To answer RQ2, we use a similarity heuristic [14] to compare two products and to evaluate the similarity degree of a given test suite. An experiment conducted on both similar and dissimilar test suites towards FMs of different size demonstrate the higher ability of dissimilar test suites to detect the defect embodied in the modified FMs. Further, the validity of a similarity-driven prioritization technique [14] is also evaluated.

In brief, the contributions of the present paper are:

- A mutation analysis approach for SPLs based on FMs,
- An experimentation performed on real FMs from small to large scale ones, which (a) confirm the hypothesis that dissimilar test suites have a higher mutant detection rate than similar ones and (b) assess a similarity-driven prioritization technique.

The remainder of this paper is organized as follows: Section II and III respectively present the challenges of SPL testing and introduce the concepts underlying the proposed approach. Section IV details the mutation testing and similarity approaches. Section V reports on the conducted experiments. Finally, Section VI discusses related work and Section VII concludes the paper.

II. Challenges of Software Product Line Testing

Testing a SPL is challenging due to the combinatorial explosion of the number of products to consider [8]. Indeed, even relatively small FMs might allow configuring billions of possible products. For instance, the FM of a video player [15] of less than 200 features allows deriving around 4.5×10^{13} different variants of this player. In that context, testing all the possible products is infeasible since in a real world industrial environment, the resources are limited. It thus becomes necessary to reduce the number of products to test to a reasonable value while trying to maximize the level of confidence in the products that are tested.

Mutation analysis has been applied effectively on different kind of models, e.g. [16], [17], [18]. However, in the context of SPLs, it has not yet been introduced. FMs are the standard models used for describing and testing a SPL [3]. Hence, we apply mutation on the constraints embodied in a FM. Doing so enables targeting at possible errors contained in inherent constraints of a product line as modeled by FMs. In other words, mutation analysis simulates possible defects in the logic constraints of the FM representing the SPL.

Scalability forms an open issue for testing large SPLs [8]. Recently, similarity was shown to be a simple, scalable and effective approach, capable of both reducing the number of products to test and to prioritize them [14]. The idea introduced in this paper is to use mutation testing as a way to assess the similarity method [13], [14]. Since mutants represent possible defects of the model, they can be used to evaluate the quality of the similarity-driven selected tests. The benefit of this practice is that it becomes possible to measure the appropriateness of the generated products at the model level. Additionally, the usefulness of the similarity-driven prioritization can also be assessed.

III. Background

A. Feature Models (FMs)

Feature Models (FMs) represent the features and constraints between the features of the product line. A feature represent an abstraction of a software asset, like a functionality. FMs allow constructing tailored software products by selecting the features to be present in the final products. A FM is generally represented by a Feature Diagram (FD), which graphically represents the hierarchy and the constraints linking the features or by a propositional formula, which includes all the constraints linking the features and translates the graphical representation of the diagram into propositional logic.

Figure 1 illustrates an example of a FD with 10 features [4]. It represents the different features and constraints that hold among them. The translation of a FM to logic [5] allows ensuring the configuration semantic is preserved. Indeed, the formula of a FM defines the valid configurations[1], i.e. configurations which fulfill the constraints of the FM. In the following, we will refer to configurations as products.

Definition: A product is said to be *valid* if it satisfies the boolean formula of the FM, and *invalid* if the product does not satisfy the formula of the FM.

The formula of a FM can be expressed in Conjunctive Normal Form (CNF), i.e. with a conjunction of n clauses $C_1, ..., C_n$, where a clause is a disjunction of m literals:

$$FM = \bigwedge_{i=1}^{n} C_i.$$

[1]A configuration represents the selected and unselected features of the product.

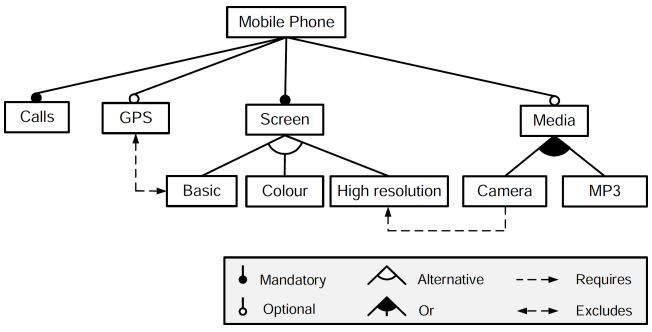

Figure 1. A Simple Feature Diagram of a Mobile Phone Product Line [4].

As a result, a clause represent a constraint that has to be satisfied by a given product. A literal represent either a selected or an unselected feature. Thus, the general form of a clause C_i is:

$$C_i = \bigvee_{j=1}^{m} f_j, \text{ where } f_j \text{ is a feature or its negation.}$$

For instance, the boolean formula in CNF of the mobile phone product line depicted by Figure 1 is:

$$\begin{aligned} FM = {} & (mobile\ phone) \wedge (\neg calls \vee mobile\ phone) \\ & \wedge (\neg mobile\ phone \vee calls) \wedge (\neg gps \vee mobile\ phone) \\ & \wedge (\neg screen \vee mobile\ phone) \\ & \wedge (\neg mobile\ phone \vee screen) \\ & \wedge (\neg media \vee mobile\ phone) \wedge (\neg basic \vee screen) \\ & \wedge (\neg color \vee screen) \wedge (\neg high\ resolution \vee screen) \\ & \wedge (\neg screen \vee basic \vee color \vee high\ resolution) \\ & \wedge (\neg basic \vee \neg color) \wedge (\neg basic \vee \neg high\ resolution) \\ & \wedge (\neg color \vee \neg high\ resolution) \\ & \wedge (\neg camera \vee media) \wedge (\neg mp3 \vee media) \\ & \wedge (\neg media \vee camera \vee mp3) \wedge (\neg camera \vee \neg mp3) \\ & \wedge (\neg gps \vee basic) \wedge (\neg basic \vee gps) \\ & \wedge (\neg camera \vee high\ resolution). \end{aligned}$$

It contains 21 clauses represented here between brackets. A valid assignment of the variables of this formula represents a valid product that can be derived from the FM. For instance, the product

$$\begin{aligned} P = {} & mobile\ phone, calls, \neg gps, screen, media, \\ & \neg basic, color, \neg high\ resolution, \neg camera, mp3 \end{aligned}$$

is valid since it satisfies the formula of the FM. This product has 6 features selected and 4 features not selected.

B. Mutation Testing and Analyis

Mutation analysis forms a powerful technique with various applications like software testing [19], [16] and debugging [20]. It is applied by creating altered (mutant) versions of the various programs artifacts like source code, specification models, etc. [19], [16]. The main idea behind this approach is to evaluate the power of test cases to reveal behavior differences between the original (unaltered) and the mutated (altered) artifact versions. The mutated versions represent possible defects of the artifact under test and they are produced based on a set of well defined rules called mutant operators [16]. Mutant operators are defined on "syntactic descriptions to make syntactic changes to the syntax or objects developed from the syntax" [16]. The process of introducing mutants is called mutation analysis. The ability of the utilized test cases to reveal the introduced mutants is examined in order to use this approach for testing purposes (mutation testing). If a mutant can be detected by a test, the mutant is called killed. Otherwise, it is called live. Therefore, measuring the ratio of the killed mutants to the totally introduced ones results in a quality measure of the testing process. This measure is called mutation score and demonstrates the ability of the tests to detect errors.

In the context of this paper, mutants are produced by applying a set of mutant operators to the original FM. The test evaluation is performed by checking whether the tests satisfy the boolean formula of the modified FMs, i.e. whether the formula is evaluated to *true*. Since the examined tests are produced based on the original FM, they always satisfy their respective boolean formulas. Consequently, a mutant is said to be killed if its formula is not satisfied, i.e. if the formula is evaluated to *false*.

C. Similarity

Similarity is an heuristic which is used here to compare valid software products (i.e. the test cases) and to evaluate the similarity degree of a given test suite. Previous work on model-based testing, such as [13], [21] have shown that dissimilar test suites bestow a higher fault detection power than similar ones. The experiment's results presented in this paper (see Section V) show that dissimilar test suites kill more mutants than similar ones. Similarity involves the definition of a distance metric d between any two products P_i and P_i, where $1 \leq i, j \leq n$. This metric is used to evaluate the degree of similarity between two given products: the higher the resulting distance is, more different the two products are.

IV. Approach

A. Mutation Analysis for Software Product Lines Based on Feature Models

In this paper, we introduce a mutation testing approach for SPLs based on FMs. The approach works as follows. From a FM represented as a boolean formula, we produce several erroneous versions of this model by applying mutant operators on the clauses of the formula. These erroneous versions of the original FM are the mutants. Then, using a SAT solver [22], we generate products from the original FM and we check their validity towards the mutants. This evaluation is performed by checking whether the generated products satisfy or not the boolean formula of the mutants. This process allows evaluating the quality of the test suite through the computation of the mutation score. The approach is depicted by Figure 2.

We propose two mutation operators which perform at the clause level of the boolean formula of the FM. These two operators are summarized in Table I. The first operator takes a clause C_i and randomly change a literal of this clause into its negation. As a result, this operator alters an existing

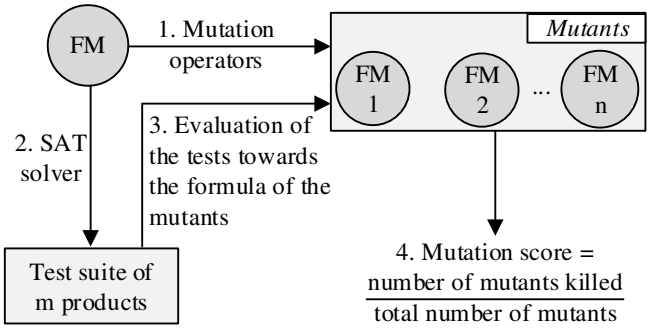

Figure 2. Mutation Analysis Approach.

clause of the FM formula. The second operator aims at creating two clauses from a given one by replacing one of the disjunction operator in this clause by a conjunction operator. Thus, this second operator creates two clauses from an existing one, increasing the total number of clauses of the boolean formula by one.

B. Test Suite Generation

In our context, a test represents one valid product that can be configured from a FM. As a result, a test suite is composed of the products to test. Usually, the number of products that can be generated from a FM explodes with the number of features, leading quickly to billions of possible products to test, even for FM with less than 200 features (see Table II).

We use a SAT solver [22] to generate products randomly from the space of all the valid products. The random generation of products is performed via the method described in [14]. Products randomly generated were found to be dissimilar due to the large size of the search space. Besides, to generate similar products, one product is randomly selected from the search space of all the valid products. Then, adjacent products to the randomly selected one are retrieved. These are products sharing many selected or unselected features in common. Section IV-C1 gives more details about similar and dissimilar products.

C. Evaluation of the Quality of the Test Suite

Here, we try to link the mutation score of the examined test suites with the quality of the test suite in terms of dissimilarity between the products. To this end, we first present a similarity-based distance between two products and a prioritization technique which makes use of this distance to order the products [14].

1) A Similarity-based Distance: We consider products represented as a set of selected and unselected features. In this context, one product is represented as a set of n features of a FM as $P = \{\pm f_1, ..., \pm f_n\}$, where $+f_i$ indicates a feature which is selected by this product, and $-f_i$ an unselected one. The distance d between two products is given by [14]:

$$d : \begin{array}{ccc} P \times P & \longrightarrow & [0, 1.0] \\ (P_i, P_j) & \longmapsto & 1 - \dfrac{\#P_i \cap P_j}{\#P_i \cup P_j}, \text{ where } P_i, P_j \in P. \end{array}$$

The resulting distance varies between 0 and 1. More particularly, a distance equal to 1 indicates that the two considered products are completely different. A distance equals to 0 denotes that the two products are the same (redundant). It is noted that an unselected feature is also an element of the set representing a product.

2) A Similarity-driven Prioritization: Here, we use the above-mentioned distance d to prioritize a test suite S of m products [14]. The objective is to order the products such as the first products selected are the most distant the one from the others. Formally, the prioritization is defined as [23]:

Given: a set of products, S, the set of all the permutations of S, P_S and a function f from P_S to the real numbers, $f : P_S \longrightarrow \mathbb{R}_+$.

Problem: finding $S' \in P_S$ such as $(\forall S'' \in P_S \mid S'' \neq S')[f(S') \geq f(S'')]$. In this context, f is the dissimilarity achieved by S. The experimental study (see Section V) demonstrates the correlation between the dissimilarity between products and the mutation score.

The approach used in this paper is presented in Algorithm 1 [14]. Informally, this prioritization approach selects at each step the product which is the most distant to all the products already selected during the previous steps. To this end, the two products belonging to S and sharing the highest distance are first added to L (lines 4 to 7). These two products are then removed from S (line 8). The next step consists in adding to L and removing from S the product sharing the maximum distance to all the products already added to L (lines 9 to 14): for each product of S, we sum the individual distances with the other products of L, thus giving a value for the set. Then the maximum is obtained by comparing these set values (line 11). This process is repeated until S is empty.

Table I
MUTATION OPERATORS FOR FEATURE MODELS

Input	Applies on	Result
A clause: $C_i = f_1 \vee ... \vee f_k \vee ... \vee f_m$	a literal of the clause: $f_k, k \in [1, m]$	A modified clause: $C'_i = f_1 \vee ... \vee \neg f_k \vee ... \vee f_m$
A clause: $C_i = f_1 \vee ... \vee f_k \vee ... \vee f_m$	a disjunction operator in the clause	2 clauses: $C'_i = f_1 \vee ... \vee f_k$ and $C''_i = f_{k+1} \vee ... \vee f_m$

Algorithm 1 Similarity-driven Prioritization(S)

input: $S = \{P_1, ..., P_m\}$ ▷ Unordered set of m products
output: L ▷ Prioritized list of m products
$L \leftarrow []$
Select P_i, P_j from S **where** $max\,(d(P_i, P_j))$
▷ Take the first ones in case of equality
$L.add(P_i)$
$L.add(P_j)$
$S \leftarrow S \setminus \{P_i, P_j\}$
while $\#S > 0$ **do**
  $s \leftarrow size(L)$
  Select $P_i \in S$ **where** $max\left(\sum_{j=1}^{s} d(P_i, L.get(j))\right)$
  ▷ Take the first one in case of equality
  $L.add(P_i)$
  $S \leftarrow S \setminus \{P_i\}$
end while
return L

V. Experiments

In this section, the mutation testing approach of FMs and the evaluation of the quality of the generated test suites are assessed. The experimental study employs 12 real FMs from two common repositories [15], [24]. These FMs are recorded in Table II. It presents, for each FM, the number of features it contains and the total number of products that can be configured from the model.

A. Evaluation of The Mutation Score Depending on the Type of Tests

The first experiment aims at evaluating the impact of the quality of the test suite on the mutation score. In other words, the objective is to evaluate whether dissimilar test suites kill more mutants than similar ones.

1) Setup: We generated 100 mutants for each of the 12 FMs used in this case study. The chance to produce a mutant with one of the two mutation operators was set to 0.5. We generated three type of test suites: test suites containing only dissimilar products, test suites containing half similar and dissimilar products, and test suites containing only similar products. Different size of tests suites were generated for each of these types: test suites of 2, 10 and 50 products. We evaluated the test suites towards the 100 mutants to compute the mutation score. The generation of the test suites and the evaluation of the mutation score has been repeated 100 times.

2) Results: The results are recorded in Table III. It presents, for each FM, the average, minimum and maximum mutation score achieved for the different size and types of test suites. Following this table, one may observe that the mutation score for the test suites of dissimilar products is higher than the tests suites, containing both similar and dissimilar products, and the latter is higher than similar test suites. In some cases, like for the Linux kernel 2.6.28.6 FM with test suites of 50 products, the mutation score achieved by dissimilar test suites is more than three time bigger than the mutation score reached by similar test suites.

To evaluate whether these differences are statistically significant, we followed the guidelines suggested by Arcuri and Briand in [25] by performing a Mann-Whitney U Test. It is a non-parametric statistical hypothesis test for assessing whether one of two samples of independent observations tends to have larger values than the other. We obtain from this test a probability called p-value which represents the probability that the two samples are equal. It is conventional in statistics to consider that the difference is not significant if the p-value is higher than the 5% level.

For each size of test suites and for each of the 100 executions, we took the mutation score achieved by the dissimilar and similar test suites for each feature model. We thus have on the one hand the 12 mutation scores for the similar test suites, and on the other hand the 12 mutation scores of the dissimilar test suites. It leads to 100 p-values corresponding to the number of executions performed. The results are presented in Figure 3. It represents via a box plot the distribution of the 100 p-values resulting of the Mann-Whitney U test between the mutation score achieved by similar and dissimilar test suites for the 100 executions. From this figure, it can be observed that the difference is statistically significant for test suites of 10 and 50 products since all the p-values are lower to the significance level of 5%.

Table II
12 Various Size Feature Models.

	Counter Strike Simple FM	DS Sample	Electronic Drum	Smart Home v2.2	Video Player	Model Transformation	Coche Ecologico	Printers	Electronic Shopping	eCos 3.0 i386pc	FreeBSD kernel 8.0.0	Linux kernel 2.6.28.6
Number of Features	24	41	52	60	71	88	94	172	290	1,244	1,396	6,888
Number of Valid Products (≈)	18,176	6,912	331,776	3.87E9	4.5E13	1.65E13	2.32E7	1.14E27	4.52E49	NA	NA	NA

Table III
MUTATION SCORE ACHIEVED WITH DIFFERENT TYPES OF TEST SUITES (%).

	Number of products	2		10			50		
	Type of products	Dissim.	Sim.	Dissim.	Half Sim./Dissim.	Sim.	Dissim.	Half Sim./Dissim.	Sim.
Counter Strike Simple FM	avg	69.50	53.02	82.29	80.09	64.40	84.83	84.44	76.86
	min	54	31	77	75	45	83	83	63
	max	77	71	85	84	75	85	85	84
DS Sample	avg	50.64	35.91	69.26	64.71	52.72	88.67	86	77.07
	min	46	22	60	56	39	81	76	66
	max	58	50	81	75	63	90	90	90
Electronic Drum	avg	63.59	45.92	81.31	78.99	60.04	83	83	80.05
	min	55	33	75	73	45	83	83	76
	max	69	58	83	83	69	83	83	83
Smart Home v2.2	avg	78.18	58.83	93.13	91.71	66.49	94	93.86	77.62
	min	62	34	89	85	46	94	93	63
	max	90	76	94	94	79	94	94	86
Video Player	avg	69.16	53.64	82.50	80.70	58.36	84.02	83.64	67.68
	min	31	25	77	57	29	83	77	34
	max	81	72	85	84	73	86	85	77
Model Transformation	avg	68.69	52.33	83.17	80.24	54.21	84	83.99	58.94
	min	60	39	80	66	41	84	83	45
	max	74	65	84	84	66	84	84	73
Coche Ecologico	avg	72.66	59.21	83.93	80.40	60.90	88.94	88.32	66.46
	min	67	49	79	71	47	87	85	55
	max	78	68	88	86	69	89	89	74
Printers	avg	59.58	45.13	76.45	72.68	47.47	82.50	80.95	56.48
	min	38	21	72	60	31	80	77	38
	max	70	60	81	77	63	84	83	66
Electronic Shopping	avg	66.27	48.33	86.42	82.80	49.09	89.23	88.70	54.18
	min	55	38	80	76	38	88	85	40
	max	77	60	90	89	63	90	90	67
eCos 3.0 i386pc	avg	60.35	48.32	77.83	73.22	48.41	83.49	81.13	49.64
	min	49	38	74	65	38	77	76	40
	max	68	65	86	83	56	87	87	57
FreeBSD kernel 8.0.0	avg	26.82	18.62	40.91	36.55	18.89	46.89	45.28	19.14
	min	10	5	32	25	5	45	40	7
	max	37	32	46	43	29	48	48	32
Linux kernel 2.6.28.6	avg	10.14	7.14	15.72	13.97	7.40	23.21	21.40	7.29
	min	7	3	11	10	4	14	14	4
	max	17	16	24	22	16	35	34	10

B. Evaluation of the Mutation Score Towards the Similarity-driven Prioritization

Here, the objective is to assess whether the similarity-driven prioritization [14] is effective. In other words, we want to evaluate whether k products selected according to the prioritization technique presented in Section IV kill more mutants than k products randomly prioritized.

1) Setup: For each FM, we generated tests suites of 100 products containing both similar and dissimilar products. We executed the similarity-driven prioritization technique to prioritize each test suite. Then, we applied 100 times a random prioritization of the products in order establish a random ordering of them. Finally, for each number of k products selected between 0 and 100, we evaluated the mutation score achieved with these k products.

To compare the prioritization approaches, the area under curve is evaluated [14], [26]. This area is the numerical approximation of the integral of the coverage curve and is

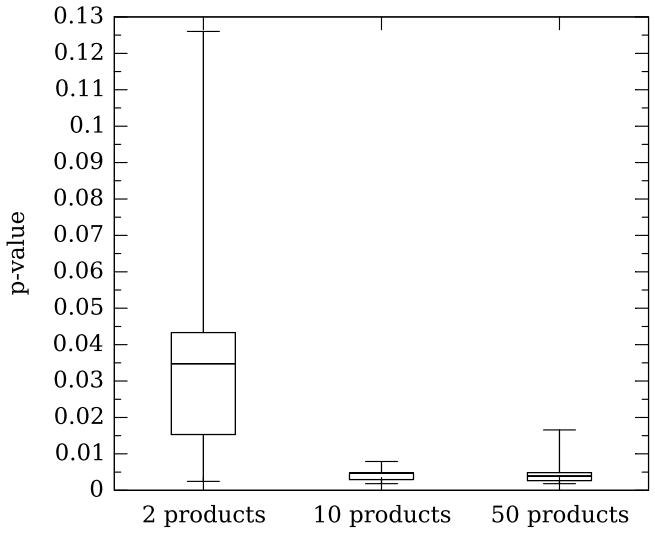

Figure 3. Distribution of the 100 p-values Resulting of the Mann-Whitney U Test between the Mutation Score Achieved by Similar and Dissimilar Test Suites for the 100 Executions.

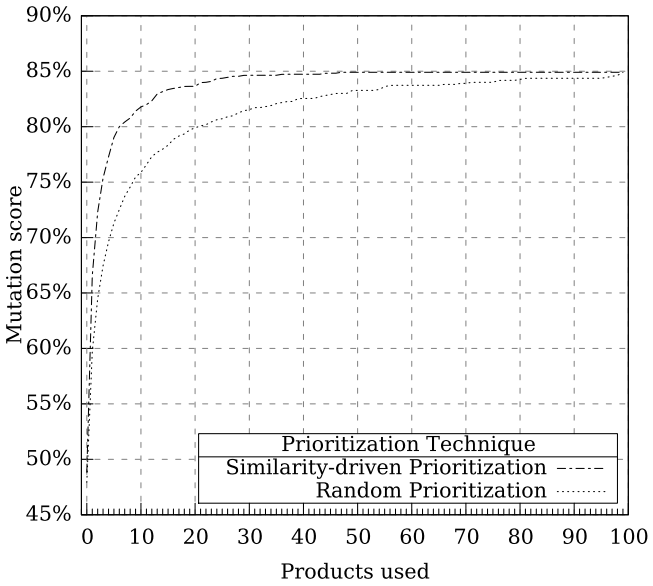

Figure 4. Mutation Score Achieved for the Prioritization Techniques Averaged on the 12 FMs. The Random Technique Is the Average of 100 Executions per FM.

computed using the trapezoidal rule:

$$\int_a^b g(x)dx \approx (b-a)\frac{g(a)+g(b)}{2}.$$

Thus, for each prioritization method, if $cov(x)$ denotes the mutation score achieved with the x-th product, then the area value is given by:

$$\sum_{i=1}^{99}\int_i^{i+1} cov(x)dx = \sum_{i=1}^{99}\frac{cov(i)+cov(i+1)}{2}.$$

A higher area under curve value expresses a more effective prioritization.

2) Results: Table IV presents the area under curve for the similarity and random prioritizations for each FM. From this table, one can see that the similarity-driven technique bestow a higher area under curve value than the random one, fact which demonstrates its effectiveness. Indeed, in any cases, the similarity-driven prioritization achieves the highest area under curve value. Figure 4 depicts the curve of the mutation score achieved for different number of products selected, averaged on the 12 FMs. This figure also shows the benefit of the similarity-driven prioritization. For instance, a mutation score of around 80% can be achieved with the similarity-driven prioritization with only around 5 products while the random one requires around 20 products. In addition, only around 30 similarity prioritized products are needed to achieve the maximum score of around 85% where the random prioritization requires 100 products.

To evaluate whether the differences between the similarity-driven prioritization technique and the random one are significant, we performed a Mann-Whitney U Test. For each FM, we compared the results of the similarity

Table IV
AREA UNDER CURVE OBSERVED FOR THE TWO PRIORITIZATION TECHNIQUES.

	Counter Strike Simple FM	DS Sample	Electronic Drum	Smart Home v2.2	Video Player	Model Transformation	Coche Ecologico	Printers	Electronic Shopping	eCos 3.0 i386pc	FreeBSD kernel 8.0.0	Linux kernel 2.6.28.6
Similarity-driven prioritization	8,346	8,567.5	8,177.5	9,276.5	8,208.5	8,280.5	8,730	8,028.5	8,777.5	8,080.5	4,540.5	2,304.5
Random prioritization (min)	7,995	7,666	7,983	9,117.5	8,009.5	8,040	8,289.5	7,614.5	8,342.5	7,476.5	4,014	1,473.5
Random prioritization (avg)	8,187	8,118.5	331,776	9,175	8,142.5	8,183	8,514	7,802	8,535.0	7,7728	4,239.5	1,883
Random prioritization (max)	8,313.5	8,473	331,776	9,268.5	8,205	8,274.5	8,684.5	8,021.5	8,729.5	8,068	4,530.5	2,272.5

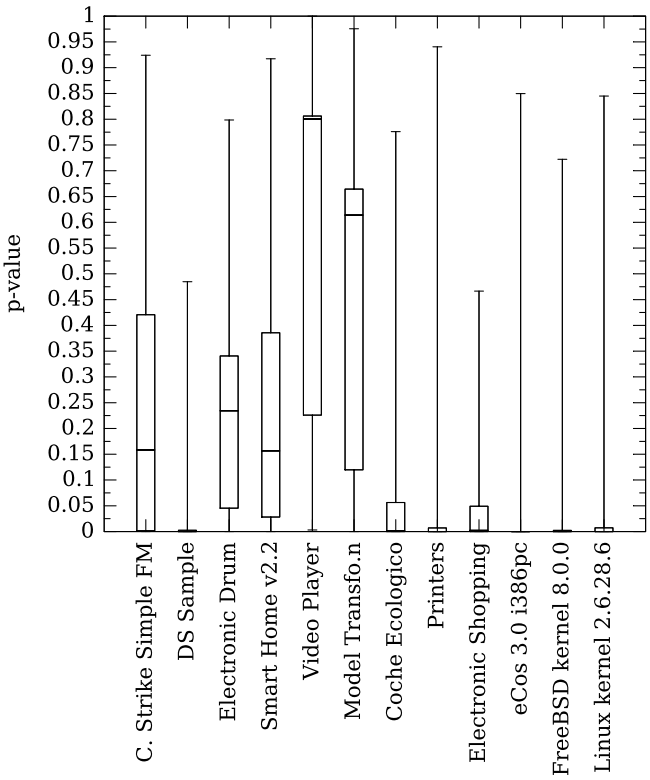

Figure 5. Distribution of the 100 p-values Resulting of the Mann-Whitney U Test between the Mutation Score Achieved by Products Prioritized with the Similarity Technique and the 100 Random Prioritization.

prioritzation with each of the 100 random executions. It thus leads to 100 p-values per FM, which are represented with box plot in Figure 5. From this figure, one can see that the results are not significantly significant for the small FMs. One explanation is that only a small number of products, e.g. 5 or 10 allows killing most of the mutants, and thus the remaining products don't kill any new mutants, leading to two samples which are almost the same. However, for the largest FMs, the difference is significant, with median values greatly below the significance level of 5%.

C. RQ1 and RQ2

The mutation testing approach proposed in this paper aims at evaluating the quality of the tests. We produced 100 erroneous FMs and we evaluated the fault detection power of different type of test suites generated from the correct FM. The approach uses mutation operators which perform on the boolean formula of the FM by altering clauses. The results obtained show that the tests are able to kill some mutants, which makes the approach interesting for testing FMs.

The impact of dissimilar and similar test suites on the mutation score is clear. The results obtained in this paper show that dissimilar test suites bestow a higher mutant detection rate than similar ones. Indeed, both the evaluation of the mutation score depending on the type of tests and the similarity-driven prioritization showed that dissimilar products kill more mutants than similar ones. In addition, we observed a significant statistical difference between the mutation score achieved by the different type of test suites, fact which confirm the similarity hypothesis.

D. Threats to Validity

First, there is an *external validity* threat. Indeed, we cannot ensure that the mutation analysis and prioritization approaches will output analogous results on different sets of FMs, e.g. larger or more constrained. To reduce this threat, we used 12 FMs of different sizes, from 24 to almost 7,000 features. Each of these FM bestow a different number and complexity regarding their constraints.

Besides, an *internal validity* threat could be due to potential errors in our implementation which could affect the presented results. To overcome these threats, we divided the implementation into sub stages. This practice allowed having a better control on each of the steps composing the proposed approaches. Besides, to avoid any risk due to random effects like coincidental selection of mutants or tests, we repeated the experiments 100 times.

Finally, whether the defects introduced in the mutants reflects real faults form a *construct validity* threat. Mutation has proven to be effective and the mutation operators used performs on the logical constraints of the feature model. These constraints linking the features represent a potential source of errors in the model's construction stage.

VI. Related Work

Generally, the application of mutation testing and analysis to test specifications or models has been widely used [19]. For instance, Li *et al.* applied mutation analysis on Finite State Machines [17]. Other models include Petri nets [18] or security policies metamodels [27]. In this paper, we focus on FMs, the boolean model of SPLs.

With respect to mutation, several work are related to the present one. In [28], [29], Kaminski *et al.* use a logic mutation approach to generate only subsuming higher order logic mutants. This approach works in the context of logic-based testing which aims at designing tests depending on logical expressions. In this paper, mutation operators are applied on the logic representing the FM. A similar approach is used in [30] to fix the formula of re-engineered FMs. In [11], Andrew *et al.* use mutation analysis to create mutants. They show that generated mutants can be used to predict the detection effectiveness of real fault. They investigate the relative cost and effectiveness of different testing coverage criteria. Here, we do not focus on whether or not the generated mutants are representative of real defects. Finally, Gargantini and Fraser propose a method that generate tests for the possible faults of boolean expressions [31]. Here,

we do not focus on test generation. We alter the boolean formula representing the FM.

In the context of similarity and model-based testing, Cartaxo *et al.* [32], [21] present a strategy for automatic test case selection based on the use of a similarity function. Labeled transition systems are the model from which test cases are generated. Hemmati *et al.* [33], [13] investigates and compares possible similarity functions that can be used for test cases selection in the context of state machine testing. The underlying model are UML state machines. In [14], Henard *et al.* use similarity to generate and prioritize t-wise test suites in the context of SPLs. The difference with this paper is that here similarity is used to assess the quality of the tests in terms of mutation score. Similarity is only used to compare two tests and is not used as a guide to generate the tests beforehand. Here, we generate the tests from the underlying FM using a SAT solver.

Most of the work on SPL testing was focused on providing scalable and efficient test generation techniques but less attention has been devoted to the evaluation of the bug detection ability of generated test suites, motivating this research. In [34], Steffens *et al.* provide an industrial account on the actual detection ability of t-wise techniques, showing that they actually detect bugs. Johansen *et al.* [35] applied such techniques on the Eclipse IDE and exhibited some interaction problems. Both did not consider issues occurring in the FM itself. Ensan *et al.* [36] developed an fault injection tool which associate errors to construct of the FM such as individual features, groups or constraints. This fault injection tool aims at simulating actual issues found in practice. To the best of our knowledge, our approach is the first to evaluate the ability of dissimilar test suites to detect FMs errors.

VII. Conclusion and Future Work

SPLs testing is difficult due to the large number of products that can be configured, number which can reach billions of billions with moderate size FMs. In this paper, we presented a mutation analysis approach for software product lines based on feature models. To the best of our knowledge, it is the first mutation analysis approach applied in the context of software product lines. In addition, this approach has been evaluated towards similar and dissimilar test suites to evaluate whether dissimilar test suites bestow a higher mutant detection rate than similar ones. The benefit of dissimilar test suite is that they allow to drastically decrease the number of products to test.

Our experiments, performed on 12 real feature models of different size demonstrate the effectiveness of the approach. In particular, the higher ability of dissimilar test suites to kill mutants has been proven with both the mutation score and prioritization evaluations. Indeed, dissimilar test suites are in some case able to kill two or three times more mutants than similar products. The prioritization results emphasized the benefit of this heuristic, showing that testing first dissimilar products rather than similar ones allow killing more mutants.

In future, further experiments based on additional and of various sizes FMs and mutant operators are scheduled. This will give more confidence on the findings of the present paper. We also plan to use mutation analysis as a guide towards generating test suites. By doing so, the testing process will be improved since the utilized tests will be capable of killing all the introduced mutants. Finally, an empirical comparison between the similarity-based and other approaches, e.g. t-wise [14] is also planned.

Acknowledgment

This work is supported by the Fonds National de la Recherche (FNR), Luxembourg, under the MITER project C10/IS/783852.

References

[1] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[2] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison Wesley, Reading, MA, USA, 2001.

[3] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., Nov. 1990.

[4] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Inf. Syst.*, vol. 35, no. 6, pp. 615–636, Sep. 2010.

[5] M. Mendonca, A. Wasowski, and K. Czarnecki, "Sat-based analysis of feature models is easy," in *Proceedings of the 13th International Software Product Line Conference*, ser. SPLC '09. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 231–240.

[6] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. L. Traon, "Automated and scalable t-wise test case generation strategies for software product lines," in *ICST*. IEEE Computer Society, 2010, pp. 459–468.

[7] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki, "Variability modeling in the real: a perspective from the operating systems domain," in *ASE*, 2010, pp. 73–82.

[8] J. McGregor, "Testing a software product line," in *Testing Techniques in Software Engineering*. Springer, 2010, vol. 6153, pp. 104–140.

[9] R. A. DeMillo and A. J. Offutt, "Constraint-based automatic test data generation," *IEEE Trans. Softw. Eng.*, vol. 17, no. 9, pp. 900–910, Sep. 1991.

[10] M. Papadakis and N. Malevris, "Mutation based test case generation via a path selection strategy," *Inf. Softw. Technol.*, vol. 54, no. 9, pp. 915–932, Sep. 2012.

[11] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Trans. Softw. Eng.*, vol. 32, no. 8, pp. 608–624, Aug. 2006.

[12] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" in *Proceedings of the 27th international conference on Software engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 402–411.

[13] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 1, 2012.

[14] 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," *CoRR*, vol. abs/1211.5451, 2012. [Online]. Available: http://arxiv.org/abs/1211.5451

[15] M. Mendonca, M. Branco, and D. Cowan, "S.p.l.o.t.: software product lines online tools," New York, NY, USA, pp. 761–762, 2009.

[16] J. Offutt, "A mutation carol: Past, present and future," *Information & Software Technology*, vol. 53, no. 10, pp. 1098–1107, 2011.

[17] J.-h. Li, G.-x. Dai, and H.-h. Li, "Mutation analysis for testing finite state machines," in *Proceedings of the 2009 Second International Symposium on Electronic Commerce and Security - Volume 01*, ser. ISECS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 620–624.

[18] S. C. P. F. Fabbri, J. C. Maldonado, P. C. Masiero, M. E. Delamaro, and E. Wong, "Mutation testing applied to validate specifications based on petri nets," in *Proceedings of the IFIP TC6 Eighth International Conference on Formal Description Techniques VIII*. London, UK, UK: Chapman & Hall, Ltd., 1996, pp. 329–337.

[19] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *Software Engineering, IEEE Transactions on*, vol. 37, no. 5, pp. 649 –678, sept.-oct. 2011.

[20] M. Papadakis and Y. L. Traon, "Using mutants to locate "unknown" faults," *Software Testing, Verification, and Validation, 2008 International Conference on*, vol. 0, pp. 691–700, 2012.

[21] E. G. Cartaxo, P. D. L. Machado, and F. G. O. Neto, "On the use of a similarity function for test case selection in the context of model-based testing," *Software Testing, Verification and Reliability*, vol. 21, no. 2, pp. 75–100, 2011.

[22] D. Le Berre and A. Parrain, "The sat4j library, release 2.2, system description," *Journal on Satisfiability, Boolean Modeling and Computation(JSAT)*, vol. 7, pp. 59–64, 2010.

[23] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Softw. Test. Verif. Reliab.*, vol. 22, no. 2, pp. 67–120, Mar. 2012.

[24] http://code.google.com/p/linux-variability-analysis-tools/source/browse/?repo=formulas.

[25] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 1–10.

[26] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *IEEE Trans. Softw. Eng.*, vol. 32, no. 9, pp. 733–752, Sep. 2006.

[27] T. Mouelhi, F. Fleurey, and B. Baudry, "A generic metamodel for security policies mutation," in *Software Testing Verification and Validation Workshop, 2008. ICSTW '08. IEEE International Conference on*, april 2008, pp. 278 –286.

[28] G. K. Kaminski, U. Praphamontripong, P. Ammann, and J. Offutt, "A logic mutation approach to selective mutation for programs and queries," *Information & Software Technology*, vol. 53, no. 10, pp. 1137–1152, 2011.

[29] G. Kaminski, P. Ammann, and J. Offutt, "Improving logic-based testing," *J. Syst. Software*, 2012.

[30] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le Traon, "Towards automated testing and fixing of re-engineered feature models," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE 2013, 2013.

[31] A. Gargantini and G. Fraser, "Generating minimal fault detecting test suites for general boolean specifications," *Information & Software Technology*, vol. 53, no. 11, pp. 1263–1273, 2011.

[32] E. G. Cartaxo, F. G. O. Neto, and P. D. L. Machado, "Automated test case selection based on a similarity function." in *GI Jahrestagung (2)'07*, 2007, pp. 399–404.

[33] H. Hemmati and L. Briand, "An industrial investigation of similarity measures for model-based test case selection," in *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering*, ser. ISSRE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 141–150.

[34] M. Steffens, S. Oster, M. Lochau, and T. Fogdal, "Industrial evaluation of pairwise spl testing with moso-polite," in *VaMoS*, 2012, pp. 55–62.

[35] M. F. Johansen, Ø. Haugen, F. Fleurey, E. Carlson, J. Endresen, and T. Wien, "A technique for agile and automatic interaction testing for product lines," in *ICTSS*, ser. Lecture Notes in Computer Science, B. Nielsen and C. Weise, Eds., vol. 7641. Springer, 2012, pp. 39–54.

[36] F. Ensan, E. Bagheri, and D. Gasevic, "Evolutionary search-based test generation for software product line feature models," in *CAISE*. Gdansk, Poland: Springer, 2012.