Software Product Line Testing Based on Feature Model Mutation

Johnny Maikeo Ferreira
Computer Science Department, Federal University of Paraná
CP:19081, CEP: 81531-970, Curitiba-PR, Brazil
johnnymaikeo@gmail.com

Silvia Regina Vergilio
Computer Science Department, Federal University of Paraná
CP:19081, CEP: 81551-970, Curitiba-PR, Brazil
silvia@inf.ufpr.br
http://www.inf.ufpr.br>

Marcos Quinaia
Computer Science Department, State University of Central West
Guarapuava, Brazil CEP: 85040-080
quinaia@unicentro.br

Received (February, 2015)
Revised ()
Accepted ()

The Feature Model (FM) is a fundamental artefact of the Software Product Line (SPL) engineering, used to represent commonalities and variabilities, and also to derive products for testing. However, the test of all features combinations (products) is not always possible in practice. Due to the growing complexity of the applications, only a subset of products is usually selected. The selection is generally based on combinatorial testing, to test features interactions. This kind of selection does not consider different classes of faults that can be present in the FM. The application of a fault-based approach, such as mutation based testing, can increase the probability of finding faults and the confidence that the SPL products match the requirements. Considering that, this paper introduces a mutation approach to select products for the feature testing of SPLs. The approach can be used similarly to a test criterion in the generation and assessment of test cases. It includes i) a set of mutation operators, introduced to describe typical faults associated to the feature management and to the FM; and ii) a testing process to apply the operators. Experimental results show the applicability of the approach. The selected sets are capable to reveal other kind of faults, not revealed in the pairwise testing.

Keywords: Software product line; feature model; testing criteria; mutation testing

1. Introduction

A Software Product Line (SPL) is defined as a set of software products that share common features and meet specific needs of a particular domain [1]. A feature represents a functionality that is visible for the user and can be mandatory, that is,
common for all SPL products, or can be designed like a variability, which represents a variable functionality that can be or not present in a product. In this way a product is given by a combination of features.

The feature model (FM) is the representation of all SPL products [2] and is an important artefact of the SPL engineering. It has been largely used by different SPL development methodologies to represent the features and their relationships [3, 4].

SPL contributes to decrease implementation costs and to reduce time to market. Due to this, we can observe a growing adoption of SPL in industry, and as a consequence, an increasing demand and interest in SPL testing. Testing an SPL is a difficult task, and has been subject of different surveys in the literature [5, 6, 7, 8]. Such works emphasize variability and feature testing as a fundamental key aspect. The goal is to ensure that the products derived from the FM match their requirements. To this end, ideally, all the products should be validated. But this is not always possible because the number of products grows exponentially to the number of features [9]. Due to the increasing complexity of the SPL applications, the exhaustive test, considering all combinations, is infeasible in practice and only a subset of products can be tested. Hence, a challenge related to the feature testing is how to select the subset of products for testing that can reveal many faults as possible.

To deal with this challenge, in the test of programs, testing criteria are used. They are predicates to be satisfied to help the tester in the selection, generation, of test cases, and (or) in the assessment of an existing test set, offering quality measures to evaluate test sets and to consider the test activity ended [10]. For example, an infinite number of paths in the program being tested is possible, and only the most interesting ones can be tested in practice. To select such paths, testing criteria are used, in order to find a great number of faults with minimal costs [10].

Criteria for the feature testing of SPLs are also found in the literature. The great majority is based on the combinatorial testing [9]. The idea is to select products that cover interactions between features [11, 12, 13, 14, 15, 16]. The most popular kind of combination test is the pairwise testing, which requires the test of pairs of features. Other works perform t-wise testing. But we can also find a structural approach, based on the basis path testing for control flow graphs [17]. This approach generates a set of independent paths (products) from a feature inclusion graph that represents the dependencies between features. However, we have not find works introducing a fault-based criterion, like the mutation analysis [18]. In the work of Henard et al [19] two mutation operators are proposed, however, they are used only to assess the quality of an existing test set. Other uses of a criterion, such as test data generation, are not explored in the work.

We can observe that fault-based criteria have not been enough investigated in the feature testing context. A motivation to do this is that, in the test of programs, experiments show that this criterion was the most efficacious to reveal faults [20]. Considering this, in a previous work [21], we introduced a set of mutation operators for the FM. Such operators were applied in a simple case study and promising results
were obtained, which encouraged us to implement a supporting tool, named FMTS. This tool allowed us to accomplish new experiments, which are now described in the present paper.

Then, the goal of this work is twofold: i) to present a deeper description of our mutation based approach. This description includes the set of mutation operators, considering mistakes associated to the feature management and common faults that can be present in the FM; and a testing process that allows the use of the approach as a testing criterion. In this way the approach can be used either for selection (generation) of products, as well, as for assessment of an existing set; and ii) to present more complete experimental results, in order to evaluate the approach considering its applicability and related work based on pairwise testing.

The paper is organized as follows. Section 2 reviews FM concepts and adopted metamodel. Section 3 contains related work. Section 4 introduces the mutation approach, describing the mutation operators, mutation testing process, and usage examples. Section 5 describes how the evaluation was conducted. Section 6 presents and analyses the obtained results. Finally, Section 7 concludes the paper and points out future research works.

2. Background

According to Kang et al [3], a feature is defined as an user-visible aspect, quality, or characteristic of a software system. A feature model (FM) represents the features and relationships among them. A tree, named feature diagram (FD), is generally used to visualize the FM. The nodes represent the features of the system and the edges represent relations between them, which can be:

- mandatory: If a child feature is mandatory, it is included in all products in which its parents appear.
- optional: If a child feature is defined as optional, it can be optionally included in products in which its parents appear.
- alternative: A set of child features is defined as alternative if only one feature of this set can be selected when its parents are included in a the product.
- or-relation: A set of child features is said to have an or-relation with their parents when one or more of them can be included in the products in which its parents appear.

In addition to this, cross-tree constraints are possible. The most common are: requires (or depends) and excludes. In the first case if a feature A requires a feature B, the inclusion of A in a product implies the inclusion of B in such product. In the last one if a feature A excludes a feature B, both features cannot be appear in the same product.

FM extensions propose the use of UML-like multiplicities of the form \([n,m]\) with \(n\) being the lower bound and \(m\) the upper bound. These are used to limit the number of sub-features that can be part of a product whenever the parent is
selected. If the upper bound is \( m \) the feature can be cloned as many times as we want (as long as the other constraints are respected). This notation is useful for extensible products with an arbitrary number of components.

As a product is given by a combination of features, the feature model is a compact representation of all the products of an SPL [2] and has been largely used by different SPL development methodologies. FDs have been formalized to automatically perform SPL analysis [22]. FAMA\(^a\) [23] is a framework that allows automated FD analysis. It is capable to determine whether a feature model is void (case in which it represents no products); whether it contains dead features (that are not part of any product); the number of SPL valid products for a FD; whether a product is valid for a FD (the product can be generated from the FD); and so on. FAMA adopts the metamodel of Figure 1, and an XML representation to the FM [24] that we use in our approach.

![Fig. 1. FM metamodel (extracted from [24])](image)

To illustrate the model, consider the FM for the Car Audio System (CAS) of Figure 2. CAS is the root feature that has a set of constraints (depends and excludes, respectively RX and RY in the figure). CAS is composed by an optional set of relations. Relations can be of two different types: i) binary relations which includes mandatory (e.g. R1), optional (e.g. R2) and cardinality-based; or ii) set relations (e.g alternative choice R10, and or-relation R11). A feature can be of three different types and is composed by one or more relations. A set relation is composed by at least two grouped features (e.g. CD, Cassette and DVD). A binary relation is composed by one and only one solitary feature. In addition to this, a solitary

\(^a\)http://wwwisa.us.es/fama.
feature has a cardinality. Only cardinalities $n > 1$ are represented in the graph. The graph has no solitary features that are cloned a number $n$ of times. The cardinality of the solitary features Traffic Message Channel (not cloned mandatory feature) and Wheel Control (optional feature) are respectively $[1..1]$ and $[0..1]$, which are not usually represented in the graph [24].

Fig. 2. Feature diagram of the Car Audio System (adapted from [25])

3. Related work

As mentioned before, software product lines are usually modeled with a feature diagram [3, 4] that specifies constraints and relationships between these features [14]. From a feature diagram it is possible to derive products for the testing by selecting a set of features that satisfy all the constraints.

The products are software systems built by composing the software assets that implement each feature. For example, Figure 3 shows one example of test case (product) that can be derived from the FM of Figure 2. This product is usually represented in terms of its variabilities by (Navigation System, Map Data via CD, Map Data via USB, USB, CD, WMA, MP3). It is a valid product according to the FM. However other products such as (Navigation System, Map Data via CD, Map Data via USB, USB, CD, Cassette, WMA, MP3) are considered invalid, since the features USB and Cassette should not be present at the same time in a product according to the FD.

Ideally, SPL developers should validate all products. But the space of possible combinations, in many cases, is likely to be enormous and exhaustive consider-
Table 1. Products required by pairwise testing (AT)

<table>
<thead>
<tr>
<th></th>
<th>Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wheel Control, Map Data via CD, CD, AAC, USB, MP3</td>
</tr>
<tr>
<td>2</td>
<td>Wheel Control, Map Data via USB, Cassette, WMA</td>
</tr>
<tr>
<td>3</td>
<td>Wheel Control, Map Data via USB, DVD, WAV, USB, MP3</td>
</tr>
<tr>
<td>4</td>
<td>Navigation System, Map Data via CD, CD, WMA, USB, MP3</td>
</tr>
<tr>
<td>5</td>
<td>Navigation System, Map Data via CD, Cassette, WAV, USB</td>
</tr>
<tr>
<td>6</td>
<td>Navigation System, Map Data via CD, DVD, AAC</td>
</tr>
<tr>
<td>7</td>
<td>Navigation System, Map Data via USB, CD, AAC, MP3</td>
</tr>
<tr>
<td>8</td>
<td>Media Format, Map Data via CD, Cassette, AAC, USB, MP3</td>
</tr>
<tr>
<td>9</td>
<td>Media Format, Map Data via CD, DVD, WMA, USB, MP3</td>
</tr>
<tr>
<td>10</td>
<td>Media Format, Map Data via USB, CD, WAV</td>
</tr>
</tbody>
</table>

Differently of the mentioned approaches, the work of Cabral et al [17] is not based on combinatorial testing. The approach named FIG Basis Path, transforms the OVM in a feature inclusion graph that represents the dependencies between features. The goal is to generate a set of independent paths (or products) for the graph, analogous to the basis path testing in control flow graphs of programs.

All the mentioned works offer a test criterion to cover as many interactions
among different features as possible, since the test of all combinations is not prac-
ticable. However, it is not possible to ensure that the subset of products selected is
capable to reveal all the feature management faults. To increase the probability of
finding faults and the confidence that the products represented by the FM match
their requirements, we introduce in the next section a fault based approach based
on mutation testing. The approach introduced can be viewed as complementary to
the existing ones.

In the literature the most related work to our approach is the work of Henard
et al [19]. In this work two mutation operators are proposed, however, they are
used only to examine the ability of existing test sets in detecting faults. These
operators are associated to the similarity test. The idea is to generate test sets
that are dissimilar to reveal more faults. The operators do not describe many faults
that can be present in the FM. They are not oriented to common mistakes that
are related to the feature management. The work does not explore the use of such
operators as a testing criterion, for generation and evaluation of a test set. To
overcome these limitations, our approach includes a broader set of operators and a
mutation testing process.

Fig. 3. Example of product (test data) to the FM of Figure 2

4. Mutation Approach
This section introduces a mutation based approach to the feature testing. It de-
cribes operators based on classes of possible FM faults, some examples of mutants,
and the mutation testing process. The operators are defined according to a formal
model described next.
4.1. Feature Model Definition

According to the metamodel of Figure 1, a feature model is denoted by $FM = (F, C, R)$ where:

- $F$ is the set of features, which is composed by the subsets $Root$ (root), $S$ (solitary), and $G$ (grouped). A feature $f \in F$ can be: (i) $r \in Root$; (ii) $s_{min,max} \in S$ where the cardinality associated to the solitaire feature is represented by $min$ and $max$, lower and upper bounds respectively; and (iii) $g \in G$.

- $C$ is a set of constraints $c(f_j, f_k)$ such as $f_j, f_k \in F$ and $j \neq k$. $C$ is composed by sets $D$ and $E$. $D$ is the set of depends constraints in the form $d(f_j, f_k)$, and $E$ is the set of excludes constraints in the form $e(f_j, f_k)$.

- $R$ is a set of relations composed by the sets $Bin$ (binary) and $Set$ (set). A binary relation $b(s_{min,max}) \in Bin$ is composed by a single solitary feature. A set relation $st_{min,max}(g_1, ..., g_n) \in Set$ is composed by grouped features. Similarly to solitary features, this relation comprises one or more cardinalities, represented by $min$ and $max$, for lower and upper bounds respectively, with $n > 1$, $min \leq max$, and $max \leq n$.

Mandatory and optional features are represented respectively as $b(s_1,1)$ and $b(s_0,1)$. A cardinality-based relation (cloned feature) is represented as $b(s_n,m)$ where $m > 1$, $m \geq n$. The or-relation is represented by $st_{1,1}(g_1, ..., g_n)$.

4.2. Mutation Operators

To propose the operators, possible faults in the FM were identified. These faults were classified into four categories, described below. The operators in each category are also defined and examples of generated mutants are presented.

1. Incorrect Cardinality of a Solitary Feature: a solitary feature is mistakenly defined. Once solitary feature cardinality accepts different value ranges, a possible fault occurs if a required mandatory feature receives cardinality values 0 and 1 for lower and upper bounds respectively. In this case the feature was incorrectly defined as optional;

   (a) DFL (Decrease solitary feature lower bound, change mandatory feature to optional): $DFL(s_{min,max}) = s_{min-1,max}$, if $min > 0$;
   (b) IFL (Increase solitary feature lower bound, change optional feature to mandatory): $IFL(s_{min,max}) = s_{min+1,max}$, if $min < max$;
   (c) DFU (Decrease solitary feature upper bound): $DFU(s_{min,max}) = s_{min,max-1}$, if $min < max$;
   (d) IFU (Increase solitary feature upper bound): $IFU(s_{min,max}) = s_{min,max+1}$.

Figure 4 presents examples of mutants generated to FD of Figure 2. Due to space restrictions only the changed part is shown, in bold. This figure contains two FM mutants generated by DFL and IFL operators. In Figure 4(a) the
mandatory feature *Traffic Message Channel* was changed to optional by applying operator DFL. In Figure 4(b) the optional feature *Wheel Control* is changed to mandatory by applying operator IFL.

![DFL](a) DFL ![IFL](b) IFL

Fig. 4. Examples of mutants - Incorrect Cardinality of a Solitary Feature

2. **Incorrect Elements of a Grouped Relation**: one of the grouped features should not belong to the set relation, or a solitary feature should be included into a set relation;

(a) AFS (Add feature to a set relation, solitary feature to grouped): 

\[
AFS(st_{\min,\max}(f_1, ..., f_{k-1}, f_k, f_{k+1}, ..., f_n), b(f_{k,i})) = st_{\min,\max}(f_1, ..., f_{k-1}, f_k, f_{k+1}, ..., f_n);
\]

(b) RFS (Remove features from a set relation, grouped to solitary and set relation to binary): 

\[
RFS(st_{\min,\max}(f_1, ..., f_{k-1}, f_k, f_{k+1}, ..., f_n)) = st_{\min,\max}(f_1, ..., f_{k-1}, f_{k+1}, ..., f_n) \text{ and } b(f_{k,i});
\]

Figure 5 presents examples of mutants generated by AFC and RFS operators. In Figure 5(a) the feature *USB* was added to the set relation, previously composed only by features *CD*, *Cassette* and *DVD*. In Figure 5(b), the feature *DVD* was removed from the set relation and became an optional feature (binary relation).

![AFS](a) AFS ![RFS](b) RFS

Fig. 5. Examples of mutants - Incorrect Elements of a Grouped Relation
3. **Existence of a Set Relation**: faults associated to a wrong set relation. Some child features belong to a set relation with an appropriate cardinality but they should be defined as solitary features;
   - **RSR** (Remove a set relation, create solitary and new binary optional relations): 

   \[ RSR(st_{\min,\max}(f_1, \ldots, f_n)) = b(f_{i_1,1}), \ldots, b(f_{i_n,1}); \]

   Figure 6 presents an example of mutant generated by the operator RSR. The set relation was removed, and the features CD, Cassette and DVD were changed to an optional binary relation.

![Diagram of a set relation](image)

*Fig. 6. Examples of mutants - Existence of a Set Relation*

4. **Incorrect Cardinality of a Set Relation**: a set relation represents that a certain number of grouped features is included in a product. Once cardinality is responsible by defining such number, a mistake on setting the max or min values can result in a diagram that allows products with more or less features than required.
   - **DRL** (Decrease set relation lower bound): 

   \[ DRL(st_{\min,\max}(f_1, \ldots, f_n)) = st_{\min-1,\max}(f_1, \ldots, f_n), \text{ if } \min > 0; \]
   - **IRL** (Increase set relation lower bound): 

   \[ IRL(st_{\min,\max}(f_1, \ldots, f_n)) = st_{\min+1,\max}(f_1, \ldots, f_n), \text{ if } \min < \max; \]
   - **DRU** (Decrease set relation upper bound): 

   \[ DRU(st_{\min,\max}(f_1, \ldots, f_n)) = st_{\min,\max-1}(f_1, \ldots, f_n), \text{ if } \min < \max; \]
   - **IRU** (Increase set relation upper bound): 

   \[ IRU(st_{\min,\max}(f_1, \ldots, f_n)) = st_{\min,\max+1}(f_1, \ldots, f_n), \text{ if } \max < n; \]

   Figure 7 presents examples of mutants generated by operators related to the cardinality faults. The mutants in Figures 7(a) and 7(b) were generated by the operators IRL and IRU, which increase the cardinalities of, respectively, the lower and upper bounds of the set relation. In Figure 7(a), the selection of two features is necessary among AAC, WMA, and WAV. In Figure 7(b), it is possible to select one, two or the three features.

5. **Incorrect Constraint**: this class is associated to depends and excludes constraints, and includes the following cases: (i) the two features of the constraint were in-
correctly selected; (ii) the constraint should not exist; (iii) a constraint is absent in the FM.

(a) FDC (Change depends constraint): \( FDC(d(f_a, f_b), f_k \in F) = d(f_b, f_a)|k \neq a, k \neq b; \)

(b) RDC (Remove a depends constraint): \( RDC(d(f_a, f_b)) = D - \{d\}; \)

(c) REC (Remove an excludes constraint): \( REC(e(f_a, f_b)) = E - \{e\}; \)

(d) CDC (Create a depends constraint): \( CDC(f_1, ..., f_n \in F)|\{f_k = s_{0, max} \in S \text{ or } f_k = g \in G\} = d(f_i, f_j)|i, j \leq n, i \neq j \text{ and } c(f_i, f_j), c(f_j, f_i) \notin C; \)

(e) CEC (Create an excludes constraint): \( CEC(f_1, ..., f_n \in F)|\{f_k = s_{0, max} \in S \text{ or } f_k = g \in G\} = e(f_i, f_j)|i, j \in [1, n], i \neq j \text{ and } c(f_i, f_j), c(f_j, f_i) \notin C; \)

Figure 8 presents examples of mutants related to the incorrect constraint fault class. This figure contains three FM mutants generated by FDC, RDC and REC operators. In Figure 8(a) the operator FDC changed the depends relation between Map Data via CD and CD. In Figure 8(b) the operator RDC removed this relation. In Figure 8(c), the excludes relation between USB and Cassette was removed by operator REC.

4.3. Testing Process

The operators can be used similarly to the traditional mutation testing of programs. The steps of a process to the mutation based testing of features include: generation of mutants, execution of the mutants with a test set, and mutation score production.

First of all, the mutants are generated. The tester needs to select the operators to be applied, as well as, a percentage of mutants to be generated by each operator. For example if the operator DFL (Decrease solitary feature lower bound) is selected with a 10%, only 10% of the existing solitary features will be mutated. Observe that all the mutants must be valid FMs, if an inconsistent mutant diagram is obtained by applying an operator, it is considered anomalous and is discarded. An example of anomalous mutant in the program testing is a division by zero produced in execution time. An example of anomalous mutant in the feature testing is the production of a void FD, which is not associated to any valid products.
The mutants and test cases (products) are “executed” with a FM analyser. A mutant is considered dead if the validation of a product by using the mutant produces a different result from the validation of the same product against the original FM. At the end a mutation score given by the number of generated and dead mutants is calculated. Among the mutants, there can be some ones that are equivalent to the original FM, that is, both generate the same set of products. In such case, the tester needs to determine the equivalence of the models. Equivalent mutants are not counted to calculate the score.

To exemplify, consider the test case of Figure 3. The test case kills the mutant of Figure 5(a), generated by operator AFS. The product is valid for the original FM, however it is not valid for the mutant. This does not happen for the mutant of Figure 6, generated by operator RSR. The product is valid for both diagrams, hence, the test case is not capable to distinguish the diagrams. The analysis of the valid and invalid products can point out modeling faults. At the end, a set of products to be tested is available.

Uses to the approach are similar to testing criterion uses. The mutants can be used to 1) to guide the selection of products, and 2) to evaluate the quality of the test set (set of products). In the first use, no set $T$ of products is available, a task of test data generation is performed to find products to kill the mutants, until the desired score is obtained (ideally score $= 1$). In the second one, the mutation score is used as an adequacy measure. In such case the tester has a set $T$ and wants to know how good it is, or it is possible to use the score to compare two test sets $T_1$ and $T_2$.

Notice that these uses are not exclusive. If a test set $T$ is available, and if its score is not adequate, this set can be improved by generation of additional test...
cases. The use of such initial test set allows effort reduction in the application of the mutation approach. In this way, our approach can be considered complementary to existing ones, such as pairwise testing. It improves efficacy in terms of revealed faults and offers a coverage measure to evaluate a set of products.

4.4. Implementation aspects

To allow the application of the proposed approach, we implemented a tool named (FMTS) (Feature Mutation based Test Suite). This tool works with the framework FAMA, which is responsible to validate the original FM and the FM mutants. This framework is also used to evaluate the test cases.

The tool receives as input the FMs in an XML format. It generates automatically the mutants and, given a test case (or a set of test cases) provided by the tester it produces the mutation score, and the set of alive and dead mutants. The tool has a procedure to help in the identification of equivalent mutants. Given a set of products $P$ generated by FAMA, which are valid for a mutant $M$, to determine the equivalence between the mutants, the procedure compares $P$ with the set of products generated by FAMA for the original FM. If both sets are the same, $M$ is considered equivalent. However, due to some FAMA restrictions, it is possible to generate the set of all valid products only for small FMs. For huge FMs the determination needs to be done manually.

In addition to this, a procedure is available to help the tester in the generation of test cases (products). This procedure receives as input a product $p$ (valid according to the original FM), and considering the structures present in a mutant, changes $p$ in order to kill $m$. In many cases, this procedure only generates an initial set that needs to be complemented by the tester.

The tester can also provide a test case (or a set of test cases) in XML format to check the corresponding score, as well as, to mark a mutant as equivalent.
Table 2. Information about the FMs used.

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>SPL</th>
<th>CAS</th>
<th>JAMES</th>
<th>Weather Station</th>
<th>E-Shop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory Features</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Optional Features</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Grouped Features</td>
<td>8</td>
<td>8</td>
<td>15</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Binary Relations</td>
<td>12</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Grouped Relations</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Include Constraints</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Exclude Constraints</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Number of Products</td>
<td>449</td>
<td>67</td>
<td>503</td>
<td>1151</td>
<td></td>
</tr>
</tbody>
</table>

5. Evaluation Description

This section describes the experiment conducted to evaluate the proposed approach. It was guided by the following general research questions.

1. How is the applicability of the operators proposed? This question aims at investigating if the operators and process are applicable in practice. To evaluate this question we analysed the number of mutants generated by each operator, as well, as the number of equivalent mutants. Other factor to be evaluated is the number of required test cases (products).

2. How are the results in comparison with pairwise testing? The idea of this question is to perform a comparison with pairwise testing. To do this we considered two factors generally used in the literature to compare test criteria: the number of test cases generated and the strength. The strength is related to an inclusion relation between criteria. We check if the test case set generated to satisfy the pairwise testing is capable to kill the mutants generated by our approach. In this analysis the mutation score is used as basis.

The FMs used and the main steps conducted are presented next.

5.1. Target FMs

We used FMs of four SPLs: CAS [25], described in Section 2; JAMES [24], an SPL for collaborative web systems; Weather Station [30], an SPL for weather systems and forecasting; and E-Shop [2] an E-commerce SPL. Information about the FM of these SPLs is presented in Table 5.1. We can observe that all of them have constraints. E-Shop has the greatest number of products. Such number is related to the number of grouped features.

5.2. Conducted steps

To answer the posed research questions, for each FM, we executed the following steps.

1. Mutants generation: to generate the mutants, the operators were applied with
the application percentage of 100%. Two operators, DFU and IFU, associated to the cloned features, could not be used. FAMA can not deal with this kind of feature. Anomalous mutants and other ones that could be their equivalence automatically determined, were discarded. After this step a set M of mutants were generated.

2. Test Case Generation and Execution: by using the FMTS procedure, an initial set of products (test cases) were generated and an initial score was obtained. After this, the set was manually updated by adding other test cases, in order to kill all the remaining alive mutants. If necessary, other equivalent mutants were detected and discarded in this step. At the end, a set, named FMT, of test cases (products) with a score of 100% was obtained.

3. Pairwise testing: to generate the set of pairs required, it was used the AETG algorithm and the framework Combinatorial Tool. The set of products to cover all the pairs of features was manually generated. This set is named here AT.

4. Comparison with pairwise: The sets ATs generated in the last step were submitted to the tool FMTS, executed, and the corresponding scores were obtained. In this step the number DM of dead mutants by each operator was obtained, as well as, the set RP of products in AT that contributed to get such score.

The mutants and products were generated in the sequence that they were obtained, without the goal of minimizing the size of the sets FMT and AT.

5.3. Threats to Validity

The main threat of this experiment is the size of the SPLs used. They are small. Another limitation is related to the FAMA framework used in the implementation and evaluation. It was not possible to apply and evaluate the operators DFU and IFU. The procedures, implemented by FMTS, to help in the generation of test cases, as well in the determination of equivalent mutants also, are dependent of this framework and can not be always used, mainly for huge SPLs. The effects of these threats and the scalability of the approach should be evaluated in future experiments.

Another threat is related to the sets FMT and AT. The generation of these sets may be influenced by the tester. Other ones could generate other test sets. As mentioned before, these sets were not minimized. The set RP also can depends on the execution order of the products in FMTS.

In spite of these threats we think that the evaluation conducted can point out positive and negative aspects of the approach. The results are valid for these and other similar SPLs and can be used for future investigation and improvements.

6. Results and Analysis

In this section the obtained results are presented, and analysed in order to answer the research questions.
Table 3. Totals of mutants and test cases per operator.

<table>
<thead>
<tr>
<th>Operator</th>
<th>CAS</th>
<th>JAMES</th>
<th>Weather Station</th>
<th>E-Shop</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#M</td>
<td>#FMT</td>
<td>#M</td>
<td>#FMT</td>
<td>#M</td>
</tr>
<tr>
<td>DFL</td>
<td>10</td>
<td>10</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>IFL</td>
<td>12</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>AFS</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RFS</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>RSR</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>DRL</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>IRL</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>DRU</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>IUR</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FDC</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RDC</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>REC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CDC</td>
<td>150</td>
<td>26</td>
<td>68</td>
<td>6</td>
<td>240</td>
</tr>
<tr>
<td>CEC</td>
<td>75</td>
<td>1</td>
<td>34</td>
<td>6</td>
<td>120</td>
</tr>
<tr>
<td>Total</td>
<td>268</td>
<td>27</td>
<td>129</td>
<td>15</td>
<td>387</td>
</tr>
</tbody>
</table>

6.1. RQ1: Applicability of the operators

To answer this question we generated Table 6.1. This table presents the number of mutants generated by each operator for each SPL, as well as the number of test cases required to kill these mutants.

We can observe that a total of 268 mutants were generated for CAS and 27 products were required. A greater number of mutants and consequently a greater number of test cases, were generated to FMs with a greater number of products.

The operators CDC and CEC generated the greatest number of mutants. These operator add dependencies (depends and excludes constraints) in the diagram. They are followed by the operators RFS, DFL, IFL and RSR. RFS and RSR change the properties of grouped relations. IFL and DFL change the cardinalities of solitary features.

On the other hand, the operators REC and RDC, which are also related with faults in constraints, generated the lower number of mutants. The first ones (CDC and CEC) deal with absent constraints, and have a lot of possibilities. The last ones (REC and RDC) deal with the constraints that are present in the diagram, which are few, in most cases.

The operator that generated the lowest number of mutants is AFS. This operator adds grouped features in the diagram. In general we observe operators that change cardinality of relations or features generate a number of mutants similar to the number of corresponding structures in the diagram.

For most operators, there is a correspondence between the number of mutants and required test cases. However, we can observe that for the operators CDC and CEC, which generate more mutants, this does not happen. In fact this shows that the number of required test cases does not grow proportionally to the number of
Table 4. Mutation score of the ATs.

<table>
<thead>
<tr>
<th>Operator</th>
<th>CAS</th>
<th>JAMES</th>
<th>Weather Station</th>
<th>E-Shop</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>0.819</td>
<td>0.688</td>
<td>0.52</td>
<td>0.189</td>
<td>0.472</td>
</tr>
<tr>
<td>#DM (dead mutants)</td>
<td>217</td>
<td>86</td>
<td>195</td>
<td>86</td>
<td>584</td>
</tr>
<tr>
<td># AT</td>
<td>12</td>
<td>9</td>
<td>11</td>
<td>11</td>
<td>43</td>
</tr>
<tr>
<td>#RP (required products)</td>
<td>11</td>
<td>9</td>
<td>11</td>
<td>8</td>
<td>39</td>
</tr>
</tbody>
</table>

generated mutants (or to the number of products of the original FM).

Other important information related to the practical use of the approach is the number of equivalent mutants, which can imply in an additional cost. Only the operators CDC and CEC generated equivalent mutants, respectively the number of 10 and 17 (6% of the total).

These last two analyses contribute to answer RQ1, and point out that the approach is applicable in practice. However, the operators CDC and CEC can be redefined. Maybe the application of the operators related to faults in constraints should take into account some conditions to be satisfied. In this way, a lower number of mutants can be generated.

6.2. RQ2 - Comparison with pairwise testing

The sets generated to satisfy the pairwise testing were executed in FMTS. The mutation score obtained and the cardinalities of the sets AT, DM and RP are in Table 4. For CAS, a score of 0.819 was obtained by the AT set that killed 217 (out 268) mutants. AT contains 12 products, among them, only 11 were required (RP), that is, if all 12 products were executed the score will be the same. Then one test case did not contributed to improve the score in the executed sequence. But we can observe that for the other SPLs all the products in the AT sets were necessary.

We notice that the AT sets require a lower number of products to be satisfied. With respect to the mutation score, the ATs obtained a mean score (considering all SPLs) of 0.472. The best performance was for SPL CAS (0.819) and the worst for E-Shops (0.189), which has the greatest number of products. For CAS the AT set did not killed 18% of the mutants, for E-Shop, 81% of the mutants were not killed. For JAMES and Weather Station these numbers are 31% and 49%, respectively. This means that the test cases (products) generated by pairwise testing can not reveal faults described by these alive mutants.

To better analyse the main fault classes that are not covered by the pairwise testing, Table 5 presents the number of dead mutants (DM) by each operator, as well as, the number of products (RP) in the AT sets that were necessary to kill those mutants.

The best results are for operators related to the diagram constraints and other ones that generated a great number of mutants. The worst results obtained by the ATs are for operators DFL, DRL, DRU and IRU. Such operators are related to cardinality faults. It seems that the pairwise testing may not reveal faults from
Table 5. Totals of mutants dead by the ATs per operator.

<table>
<thead>
<tr>
<th>Operator</th>
<th>CAS</th>
<th>JAMES</th>
<th>Weather Station</th>
<th>E-Shop</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#DM</td>
<td>#RP</td>
<td>#DM</td>
<td>#RP</td>
<td>#DM</td>
</tr>
<tr>
<td>DFL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IFL</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AFS</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RFS</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>RSR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DRL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IRL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DRU</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IRU</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FDC</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RDC</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>REC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CDC</td>
<td>127</td>
<td>7</td>
<td>57</td>
<td>6</td>
<td>142</td>
</tr>
<tr>
<td>CEC</td>
<td>75</td>
<td>1</td>
<td>21</td>
<td>6</td>
<td>43</td>
</tr>
<tr>
<td>Total</td>
<td>217</td>
<td>11</td>
<td>86</td>
<td>9</td>
<td>195</td>
</tr>
</tbody>
</table>

such classes. On the other hand, to kill the mutants, for all SPLs, except CAS, all the products in the AT were necessary. This points out that the sets generated using the mutation approach can, in most cases, also satisfy the pairwise testing. Most of them were necessary to kill mutants generated by operator CDC, maybe this indicates that these operators describe faults generally revealed by pairwise testing.

Answering RQ2, we can observe that the mutation approach requires a greater number of test cases, but can reveal other kind of faults not revealed by the pairwise testing. Considering the factor strength, we observe that the FMTS sets empirically include the AT sets in three of the SPLs studied.

7. Conclusions

This paper presented a mutation based approach for feature testing of SPL. The idea is to select or evaluate a subset of products from the FM, considering common faults related to feature management and that can be present in the FM. These faults are described by mutant operators.

To apply the operators, a mutation process, similar to the mutation testing of programs, is used. A FM mutant is considered dead if it validates a test case (product) in a different way that the original FM does. At the end a mutation score is obtained that can be used to guide the generation of products, or to evaluate the quality of an available set of products.

The process was implemented in a tool named FMTS. By using FMTS an experiment was conducted to evaluate the proposed approach. The results show the applicability of the operators. The number of test cases does not grow proportionally to the number of generated mutants. In addition to this, we observe that 53% of mutants were not killed by the set of products generated with pairwise testing. The
pairwise testing was not capable to reveal faults associated mainly to the cardinality of features and relations. The main faults revealed by this kind of test are related to the use of mandatory and optional features, and constraints (depends and excludes). We can then conclude that the approach should be used in a complementary way to the combinatorial testing. It aims at revealing other kind of faults and increasing the confidence that the products of a FM are according to the specification.

The operators related to faults in constraint relations should be refined to produce a lower number of mutants. They do not generate invalid mutant FMs, but they do not consider semantic aspects that can be incorporated in conditions to be satisfied during the application of such operators. This can reduce the number of generated mutants and allow scalability of the approach. These and other improvements in the tool FMTS should be evaluated in future experiments. Some works on mutation test of programs investigate ways to reduce mutation cost without decreasing the global score based on a set of sufficient operators. Studies like this could be conducted in the context of FM mutation testing.

8. Acknowledgments

This work is supported by CNPq (Productivity grant number: 305358/2012-0 and Universal Project grant number: 473899/2013-2).

References

