Sādhanā Vol. 41, No. 12, December 2016, pp. 1381–1391
DOI 10.1007/s12046-016-0571-y

Improving software product line using an ontological approach

MEGHA BHUSHAN[1,*] and SHIVANI GOEL[2]

[1]Department of Computer Science and Engineering, Thapar University, Patiala 147004, India
[2]Department of Computer Science Engineering, School of Engineering and Applied Sciences, Bennett University, Greater Noida, India
e-mail: megha@thapar.edu; shivani.goel@bennett.edu.in

MS received 31 December 2015; revised 9 May 2016; accepted 21 June 2016

Abstract. Software product line (SPL) is an emergent strategy for generating software products. The variability and commonality of SPL is illustrated by feature models (FMs). The quality of software products relies on the correctness of SPL. The overall benefits of software product line engineering (SPLE) are reduced by various kinds of defects such as dead features and false optional features in an FM. These defects can be inherited in the software products built from a defective product line model (PLM). In this paper, the problem of enhancing the quality of software products derived from SPLE is handled. An ontological based approach is proposed following first-order logic (FOL) rules to identify defects namely dead features and false optional features. The classification of cases for these defects in FMs that represent variability of SPL is defined. The presented approach has been explained with the help of an FM derived from the standard case in product line (PL) community. The initial empirical evaluation of the proposed approach analyses 35 FMs with different sizes. The results obtained exhibit that the proposed approach is accurate, effective, scalable up to 200 features and therefore improves SPL.

Keywords. Feature model (FM); software product line (SPL); defects; ontologies; feature model ontology (FMO).

1. Introduction

A set of software products shares some common functions that address specific requirements of a certain market segment [1]. Software reusability is the objective of software product line engineering (SPLE) strategy. It allows numerous organizations to diminish cost, development time and simultaneously increase the quality of software products [2]. Feature model (FM) notation is used to represent variability in software product line (SPL). It illustrates the features and their relationships for developing valid products from a product line (PL) using an accurate and correct combination of features [3].

The successful development of software products in SPL entirely depends on the quality of FM. The modelers of feature modeling may inevitably introduce contradictory relationships into FM. The growing complexity of FMs may introduce inadvertent defects, which recedes the quality of FMs and further affects the yields of PL. Recently, defects in FM are considered as a critical issue in the community of SPL. However, manual inspection of defects in FMs is a tiring and cumbersome job. Therefore, this key issue should be solved suitably to provide the benefits of reusability in the industrial domain.

The intent of this research is to develop a generic framework for improving SPL. In this paper, the classification of cases for false optional and dead feature defects in FMs is defined. First-order logic (FOL) is used to define rules based on ontology for the identification of above-mentioned defects in the presented classification of cases for these defects. It provides information to the modelers and PL developers for defects fixation to improve SPL for attaining defect-free products. A standard method is used for modeling an SPL to provide formalization, which further allows reasoning of the model using few standard tools. The idea of using FOL rules in FMs has been explored earlier also. Few researches have used FOL rules in FMs as it formalizes an SPL [4–9]. Prolog [10] is used as a reasoning tool. The details of the SPL model used in this paper are also presented. In this paper, the generalization of all presented rules for dead and false optional feature defects is proved by implementing examples for all the cases.

The highlights of the proposed work are given below:

I. A generic framework is proposed for the identification of dead and false optional feature defects in the presented classification of cases for the above-mentioned defects in SPL.
II. FMO is constructed using predicate-based ontology to provide formalization using FOL. The ontological based approach following FOL rules is given for the identification of defects due to dead and false optional features. Seven rules are given for the identification of dead

*For correspondence

feature defects and four rules for the identification of false optional feature defects. These rules are defined based on the existing literature [5, 7, 9, 11, 12].

III. The outcome of preliminary evaluation indicates that the proposed approach is accurate, effective, and scalable up to 200 features in FMs and can surely improve SPL.

The paper is divided into following sections: Section 2 presents a brief description of the preliminary terms required for understanding the proposed generic framework illustrated in section 3. The implementation details and comparison with existing work is described in section 4. The related work is described in section 5. Section 6 provides the conclusion and future work.

2. Preliminary terms

2.1 *Software product line*

SPL is a family of related software intensive systems, sharing a managed and common set of features. It fulfills the exact requirements of an appropriate market segment built from a common set of core assets in a prerequisite fashion. The main focus of SPL is on software reuse in an attempt to improve the productivity and quality while reducing cost and time to market.

2.2 *Feature model with a running example*

In SPLE, feature modeling notation is used for representing the variabilities and commonalities in terms of relationships among the features. A feature is a unique element that is of interest to various stakeholders. FM can be represented in the form of a tree structure in which features are organized in a hierarchy by means of relationships. The root of the tree in FM represents the entire PL and it is mandatory to incorporate it in all valid products of the PL.

The PLM i.e., adapted version of the graph product line (GPL) [13] described by means of feature modeling notation is shown in figure 1. It is a standard case for the evaluation of PL methodologies among the communities of PL. The root feature is represented by "*gpl*" which is mandatory to incorporate in all the valid products of SPL. The proposed approach is illustrated by introducing 14 additional features along with eight cross tree constraints in the primary model which causes defects, i.e., seven dead features (*aa1*, *bb1*, *weighted*, *aa2*, *bb4*, *bfs*, and *cc1*) and four false optional features (*mst*, *cc1*, *cc2*, and *bb5*). All the features and cross tree constraints have a name for understanding. The various relationships used for feature modeling are

Mandatory: A child feature is said to be mandatory if it appears in every product whenever its parent feature is chosen. For example in figure 1, it is mandatory to select "*bb2*" whenever *search* is selected.

Optional: A child feature is optional if it may or may not be incorporated in valid products involving parent feature. For example in figure 1, it is optional to have feature *weighted.*

Group cardinality: This relationship describes an interval that limits the count of child features that are incorporated in a product whenever their parent feature is included.

The two cross tree constraints are as follows:

Implication: The inclusion of source feature of an implication relationship in a product implies the inclusion of its target feature in the same product. For example, the implication relation between features *mst* and *cc3*, describes that the feature *cc3* is implied by the feature *mst* as shown in figure 1.

Exclusion: The excluded features are not incorporated simultaneously in any valid product. For example, in figure 1, exclusion relationship between *aa1* and *graph_type* describes that no product will incorporate features *aa1* and *graph_type* simultaneously.

2.3 *Defects in feature models*

In PLM, the quality of the model is adversely affected by the undesirable properties known as defects [14]. Though, there are many types of defects in FMs, still, few defects are identified by the proposed approach mainly false optional and dead features. In general, these defects arise due to the incorrect usage of cross tree constraints in FMs.

False optional feature: The feature declared as optional is incorporated in every product derived from the SPL is known as false optional feature.

Dead feature: A feature defined in an FM is never incorporated in any valid product derived from the SPL and is, therefore, termed as dead feature.

3. Proposed approach for improving SPL

This section explains the proposed approach to define as well as identify dead and false optional feature defects in the given classification of cases for the afore-mentioned defects in FMs. A set of rules is defined for representing certain cases of incorrect usage of relationships among features in an FM that causes these defects. The proposed rules are implemented using Prolog [15].

3.1 *Prerequisites for explaining the proposed approach for identification process*

The root feature is mandatory to be included in all products e.g., *gpl*. A parent can have more than one child. Each feature is assigned a unique name in FM. The cross tree

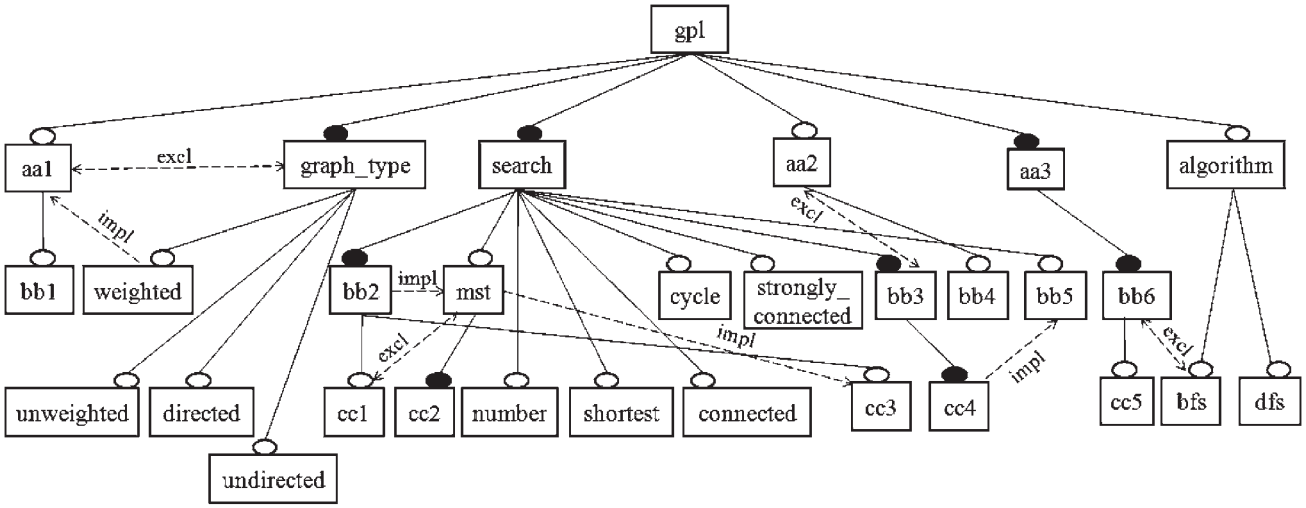

Figure 1. The FM of GPL (graph product line).

constraint relationships have been named for better understanding of the identification process. Each feature can be optional or mandatory where "o" represents an optional feature and "m" represents a mandatory feature. A relation (i.e., cross tree constraint relationships) can be exclusion or implication where "excl" represents exclusion relationship and "impl" represents implication relationship between features.

3.2 *Feature model ontology*

An ontology defines "a formal, explicit specification of a shared conceptualization" [16]. In context of expressiveness, ontology is more powerful and richer than a feature. This reason motivated us to merge them together for building a more effective feature meta-model. FOL is used to define formal semantics of the feature modeling that provides meticulous formal explanation for graphical notations. Formal languages based on FOL and unified modeling language (UML) class diagrams [17] have been used to represent ontologies. It is a common practice in software engineering. Ontology is applied to feature meta-modeling for a more influential description method. The meta-model is composed of ontology classes and their relationships.

The FMO is built using ontologies. The concepts of FM are represented in the shape of an ontology using FMO. The semantic relationships between the FM concepts are exploited using ontological representation. The FMO is constructed using FOL predicate-based ontology modeling [18] and conforming the FM meta-model based on UML recommended by Mazo *et al* [19] as shown in figure 2. The proposed FM meta-model is represented using ontology classes and properties. In the FMO, ontology classes are depicted as meta-model classes and the ontological properties are depicted using UML relationships. The meta-model class feature is separated in ontology classes i.e., Root, Feature, ParentFeature, and ChildFeature. It represents that an FM has single root feature. Meta-model is used for defining all the concepts that are used in FOL predicates for formalization.

The ontological representation of FMs allows us to infer interesting information about the FMs; for example, to get child features [20]. It further allows us to verify consistency among the FM and its meta-model [21].

3.3 *Generic framework*

The proposed generic framework broadly consists of input, identification process and output as shown in figure 3.

3.3a *Input:* It includes stage 1 of the framework given in figure 3. FM is transformed into FMO using FOL predicate-based ontology to provide formalization [22]. First-order language is a predicate-based ontology language in which classes of objects are represented using binary predicates and properties are represented using ternary predicates [18]. Following are the three types of predicates i.e., feature, relation, and parent which are used to represent FMO:

feature(x, m): It indicates feature x as a mandatory feature, e.g., feature(graph_type, m).

feature(y, o): It represents feature y as an optional feature, e.g., feature(algorithm, o).

parent(x, y): It shows feature x has parent feature y e.g., parent(unweighted, graph_type).

relation(x, y, excl): It indicates feature x excludes feature y, e.g., relation(graph_type, aa1, excl).

relation(x, y, impl) : It represents feature x implies feature y, e.g., relation(weighted, aa1, impl).

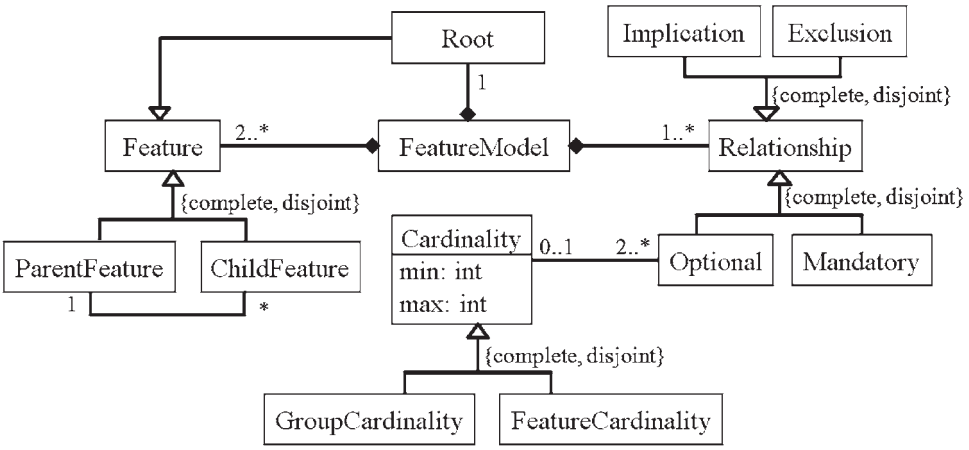

Figure 2. Proposed FM meta-model for feature PLMs.

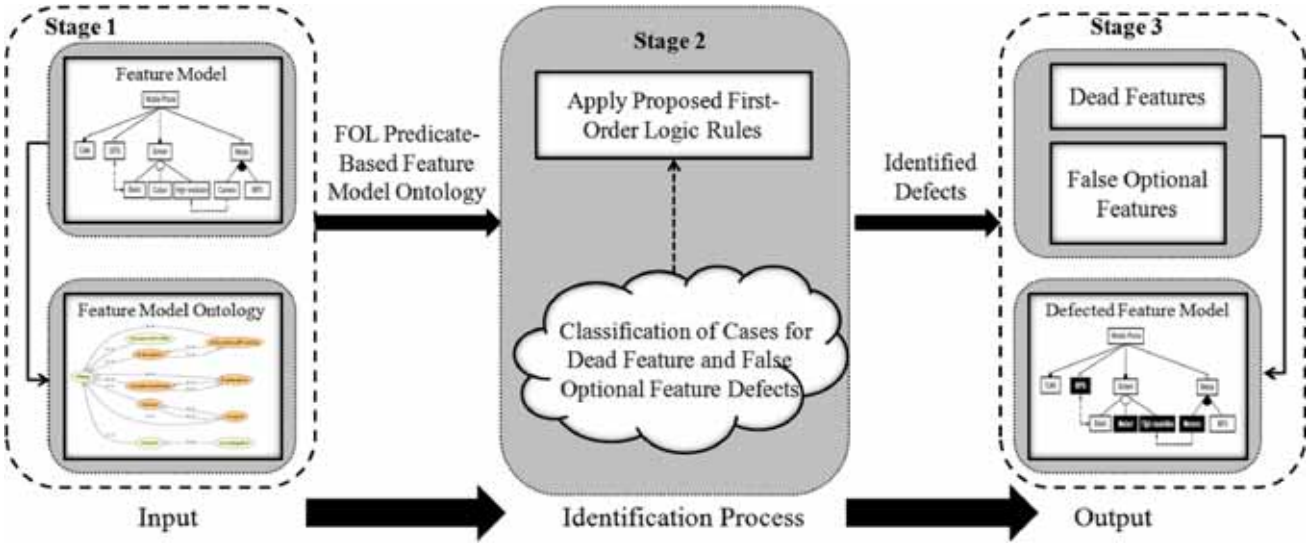

Figure 3. Overview of the generic framework for improving SPL.

3.3b *Identification process:* It includes Stage 2 of the framework, shown in figure 3. FOL rules have been developed and applied to the generated predicate-based FMO for the identification of defects. The identified defects are dead and false optional features in the given classification. Each rule is a source for originating the identified dead or false optional feature defects. The input predicate-based FMO in Prolog represents the existing facts. Following text illustrates the identification process for defects in SPL:

General criteria. The following are the general criteria for describing identification of dead and false optional feature defects:

1. Feature n1 is mandatory, feature n2 is optional and n1 excludes n2.
2. Feature n1 is mandatory, feature n2 is optional, n1 excludes n2 and optional feature n3 implies n2.
3. Feature n1 is mandatory, feature n2 is optional and n1 implies n2.
4. Mandatory feature n1 implies an optional feature n2 and n2 excludes an optional feature n3.
5. Mandatory feature n1 implies an optional feature n2 and n2 implies an optional feature n3.

Cases and rules for the identification of dead and false optional feature defects

The dead and false optional feature defects are classified into three cases, which are explained as follows: (a) exclusion and optional feature, (b) full-mandatory feature implies optional feature, and (c) implication and optional feature. The cases of these defects and the rules to represent

certain cases of incorrect usage of cross tree constraints in FM that causes these defects are explained below:

Case 1: Exclusion and optional feature: It is the case of dead feature defects, which is described with rules 1, 2, 3, 4, 5, and 6. In this case, a mandatory feature mutually excludes an optional feature. Accordingly, the optional feature can never be selected for the configuration of a valid product and therefore becomes a dead feature.

Rule 1: dead(Feature_2): feature(Feature_2, o), parent(Feature_2, Parent_1), feature(Feature_1, m), parent(Feature_1, Parent_1), relation(Feature_1, Feature_2, excl).

Input: feature(feature_1, m), parent(feature_1, parent_1), feature(feature_2, o), parent(feature_2, parent_1), relation(feature_1, feature_2, excl).

Output: Dead_Feature = feature_2.

Explanation: Figure 4(a) depicts rule 1, which identifies dead feature defect indicated by the general criteria number 1. In rule 1, mandatory feature *feature_1* which has a parent feature *parent_1* excludes optional feature *feature_2* that has same parent feature *parent_1*. Feature *feature_1* being mandatory must be incorporated in each product. According, to the exclusion relationship, *feature_1* and *feature_2* cannot be incorporated together in any product. It means *feature_2* is excluded from all products. Thus, *feature_2* becomes a dead feature defect.

Rule 2: dead(Feature_3): dead(Feature_2), feature(Feature_3, o), parent(Feature_3, Feature_2).

Input: feature(feature_1, m), parent(feature_1, parent_1), feature(feature_2, o), parent(feature_2, parent_1), relation(feature_1, feature_2, excl), feature(feature_3, o), parent (feature_3, feature_2).

Output: Dead_Feature = feature_3.

Explanation: Rule 2 is demonstrated using figure 4(b), which identifies dead feature defect indicated by the general criteria number 1. Rule 2 executes rule 1 to identify dead feature i.e., *feature_2*. However, rule 2 results in a dead feature defect *feature_3* as its parent feature *feature_2* is a dead feature also.

Rule 3: dead(Feature_3): dead(Feature_2), feature(Feature_3, o), parent(Feature_3, Feature_1), relation(Feature_3, Feature_2, impl).

Input: feature(feature_1, m), parent(feature_1, parent_1), feature(feature_2, o), parent(feature_2, parent_1), relation(feature_1, feature_2, excl), feature(feature_3, o), parent (feature_3, feature_1), relation(feature_3, feature_2, impl).

Output: Dead_Feature = feature_3.

Explanation: Figure 4(c) shows rule 3, which identifies dead feature defect indicated by the general criteria number 2. Rule 3 first executes rule 1 to identify dead feature i.e., *feature_2*. The output of rule 3 is a dead feature defect *feature_3* which has parent feature *feature_1* as *feature_3* implies dead feature *feature_2*.

Rule 4: dead(Feature_2): feature(Feature_1, m), parent(Feature_1, Parent_1), feature(Feature_2, o), parent(Feature_2, Parent_1), feature(Feature_3, m), parent(Feature_3, Feature_1), relation(Feature_3, Feature_2, excl).

Input: feature(feature_1, m), parent(feature_1, parent_1), feature(feature_2, o), parent(feature_2, parent_1), feature(feature_3, m), parent(feature_3, feature_1), relation(feature_3, feature_2, excl).

Output: Dead_Feature = feature_2.

Explanation: Figure 4(d) depicts rule 4, which identifies dead feature defect indicated by the general criteria number 1. In rule 4, mandatory feature *feature_1* and optional feature *feature_2* have same parent feature *parent_1*. The mandatory feature *feature_3* has parent *feature_1*. Dead feature defect *feature_2* is the outcome of this rule since it is excluded by *feature_3*.

Rule 5: dead(Feature_4): dead(Feature_2), feature(Feature_4, o), parent(Feature_4, Feature_2).

Input: feature(feature_1, m), parent(feature_1, parent_1), feature(feature_2, o), parent(feature_2, parent_1), feature(feature_3, m), parent(feature_3, feature_1), feature(feature_4, o), parent(feature_4, feature_2), relation(feature_3, feature_2, excl).

Output: Dead_Feature = feature_4.

Explanation: Rule 5 is illustrated using figure 4(e), which identifies dead feature defect represented by the general criteria number 1. Rule 5 first executes rule 4 to identify dead feature i.e., *feature_2*. Dead feature defect *feature_4* is the outcome of rule 5 as its parent feature *feature_2* is already a dead feature.

Rule 6: dead(Feature_2): feature(Feature_1, m), parent(Feature_1, Parent_1), feature(Feature_2, o), parent(Feature_2, Parent_2), relation(Feature_1, Feature_2, excl).

Input: feature(feature_1, m), parent(feature_1, parent_1), feature(feature_2, o), parent(feature_2, parent_2), relation(feature_1, feature_2, excl).

Output: Dead_Feature = feature_2.

Explanation: Figure 4(f) depicts rule 6, which identifies dead feature defect indicated by the general criteria number 1. In rule 6, mandatory feature *feature_1* which has a parent feature *parent_1* excludes an optional feature *feature_2* that has a parent feature *parent _2*. Thus, *feature_2* becomes a dead feature defect.

Case 2: Full-mandatory feature implies optional feature: It is the case of false optional feature defects, which has been described through rules 7, 8, 9, and 10. In this case, a mandatory feature implies an optional feature. Accordingly, the optional feature is not optional further and thus emerges as a false optional feature.

Rule 7: false_option(Feature_2): feature(Feature_1, m), parent(Feature_1, Parent_1), feature(Feature_2, o), parent(Feature_2, Parent_1), relation(Feature_1, Feature_2, impl).

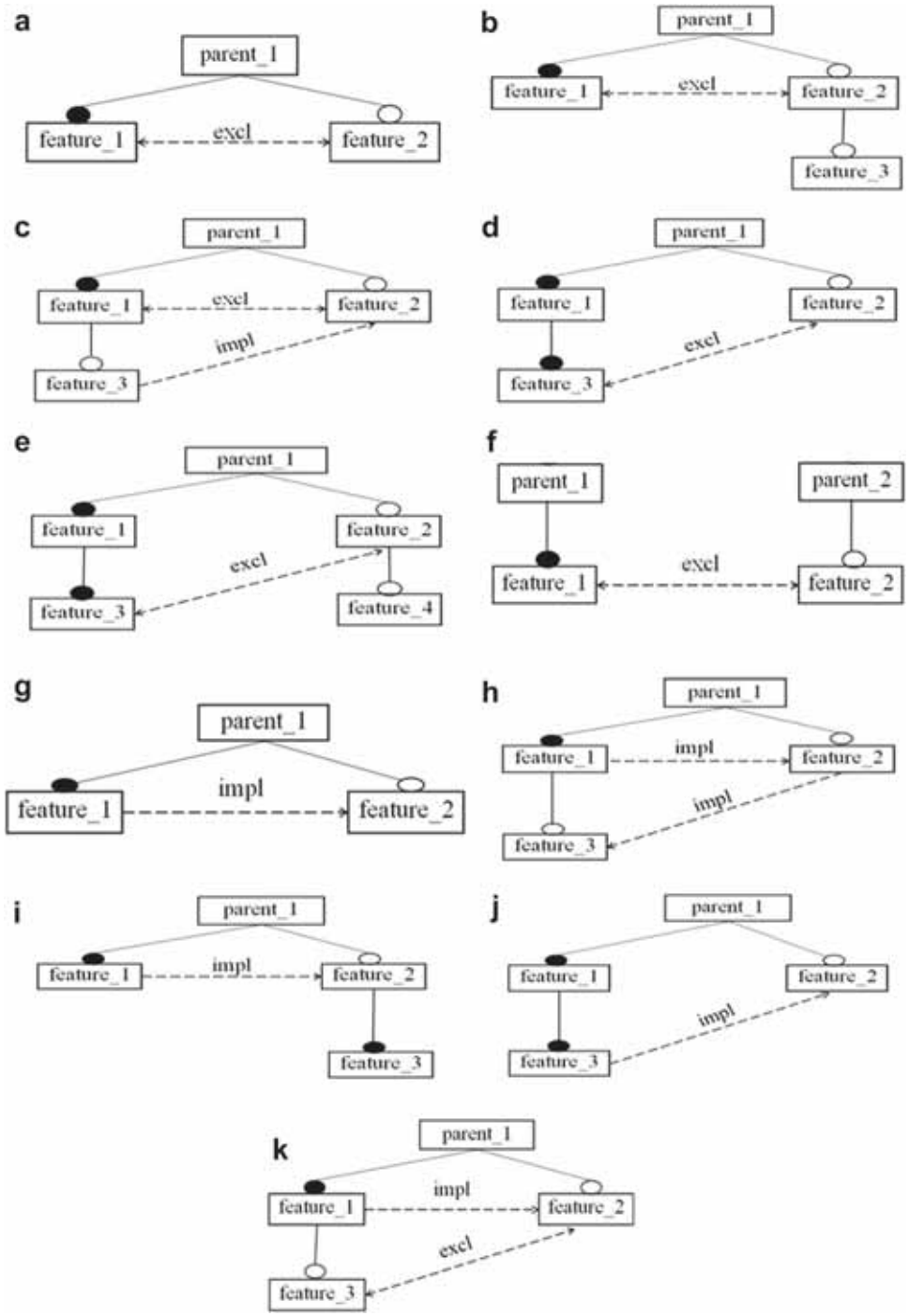

Figure 4. Demonstration of dead and false optional feature defects identification rules.

Input: feature(feature_1, m), parent(feature_1, parent_1), feature(feature_2, o), parent(feature_2, parent_1), relation(feature_1, feature_2, impl).

Output: False_Optional_Feature = feature_2.

Explanation: Rule 7 is demonstrated using figure 4(g), which identifies false optional feature defect indicated by the general criteria number 3. In rule 7, mandatory feature *feature_1* which has a parent feature *parent_1* implies an optional feature *feature_2* that has same parent feature *parent_1*. According to the implication relationship, *feature_2* is incorporated in all the products in which *feature_1* is included. Thus, *feature_2* becomes false optional feature defect.

Rule 8: false_option(Feature_3): false_option(Feature_2), feature(Feature_3, o), parent(Feature_3, Feature_1), relation(Feature_2, Feature_3, impl).

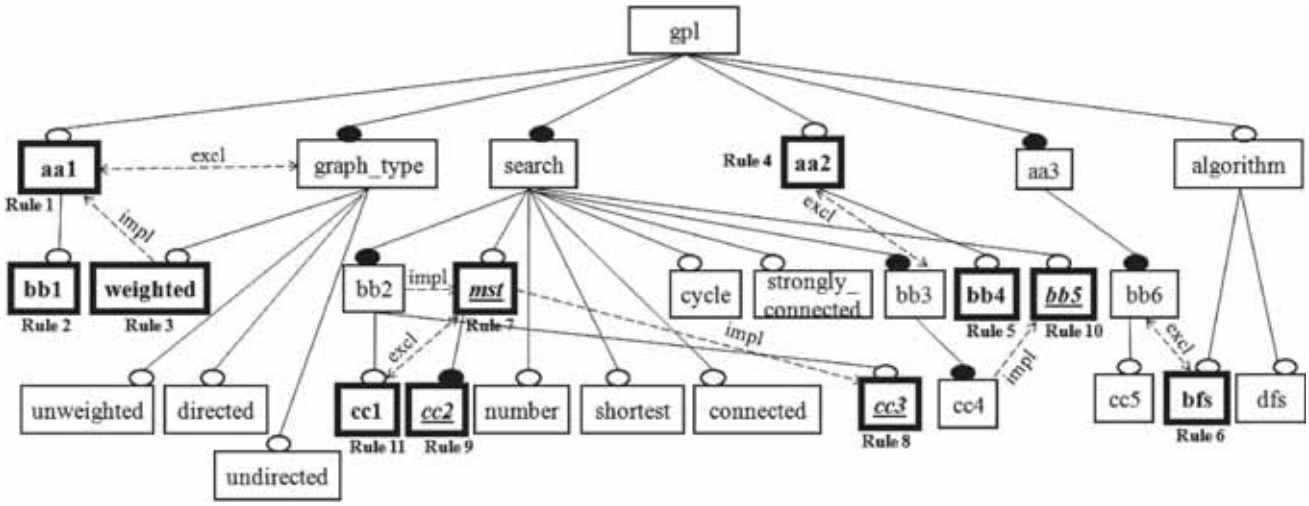

Figure 5. The features which are *bold*, *italic*, and *underlined* are false optional feature defects. *Bold* features are dead feature defects after applying rules to figure 1.

Input: feature(feature_1, m), parent(feature_1, parent_1), feature(feature_2, o), parent(feature_2, parent_1), relation(feature_1, feature_2, impl), feature(feature_3, o), parent(feature_3, feature_1), relation(feature_2, feature_3, impl).

Output: False_Optional_Feature = feature_3.

Explanation: Figure 4(h) depicts rule 8, which identifies false optional feature defect indicated by the general criteria number 5. Rule 8 first executes rule 7 to identify false optional feature i.e., *feature_2*. The output of rule 8 is false optional feature defect *feature_3* which has parent feature *feature_1* since *feature_3* is implied by *feature_2*.

Rule 9: false_option(Feature_3): false_option(Feature_2), feature(Feature_3, m), parent(Feature_3, Feature_2).

Input: feature(feature_1, m), parent(feature_1, parent_1), feature(feature_2, o), parent(feature_2, parent_1), feature(feature_3, m), parent(feature_3, feature_2), relation(feature_1, feature_2, impl).

Output: False_Optional_Feature = feature_3.

Explanation: Rule 9 is illustrated using figure 4(i), which identifies false optional feature defect indicated by the general criteria number 3. Rule 9 first executes rule 7 to identify false optional feature i.e., *feature_2*. The result of rule 9 is false optional feature defect *feature_3* as its parent feature *feature_2* is also a false optional feature.

Rule 10: false_option(Feature_2): feature(Feature_1, m), parent(Feature_1, Parent_1), feature(Feature_2, o), parent(Feature_2, Parent_1), feature(Feature_3, m), parent(Feature_3, Feature_1), relation(Feature_3, Feature_2, impl).

Input: feature(feature_1, m), parent(feature_1, parent_1), feature(feature_2, o), parent(feature_2, parent_1), feature(feature_3, m), parent(feature_3, feature_1), relation(feature_3, feature_2, impl).

Output: False_Optional_Feature = feature_2.

Explanation: Figure 4(j) demonstrates rule 10, which identifies false optional feature defect indicated by the general criteria number 3. In rule 10, a mandatory feature *feature_1* and optional feature *feature_2* have same parent feature *parent_1*. The mandatory feature *feature_3* has parent *feature_1*. False optional feature defect *feature_2* is the outcome of this rule since it is implied by *feature_3*.

Case 3. Implication and optional feature. It is the case of dead feature defect, which has been described by rule 11.

Rule 11: dead(Feature_3): false_option(Feature_2), feature(Feature_3, o), parent(Feature_3, Feature_1), relation(Feature_2, Feature_3, excl).

Input: feature(feature_1, m), parent(feature_1, parent_1), feature(feature_2, o), parent(feature_2, parent_1), relation(feature_1, feature_2, impl), feature(feature_3, o), parent(feature_3, feature_1), relation(feature_2, feature_3, excl).

Output: Dead_Feature = feature_3.

Explanation: Rule 11 is illustrated using figure 4(k), which identifies dead feature defect indicated by the general criteria number 4. Rule 11 first executes rule 7 to identify false optional feature defect i.e., *feature_2*. The output of rule 11 is a dead feature defect *feature_3* which has parent feature *feature_1* since *feature_3* is excluded by *feature_2*.

3.3c *Output:* It includes stage 3 of the framework shown in figure 3. The identified dead and false optional feature defects are produced as output in this stage.

Running example: A running example of GPL is used to explain the identification of dead and false optional feature defects as output. The root feature "*gpl*" is mandatory to be included in every product. According to rule 1, optional feature *aa1* turns into dead feature defect since it is excluded by mandatory feature *graph_type*. The output of rule 2 is dead feature defect *bb1* as it has a dead parent feature *aa1*. The result of rule 3 is a dead feature defect

weighted since it implies dead feature *aa1*. Dead feature defect *aa2* is the output of rule 4 as it is excluded by dead feature defect *aa2*. However, the output of rule 5 is dead feature defect *bb4* since its parent feature *aa2* is a dead feature defect (figure 5).

According to rule 6, optional feature *bfs* is a dead feature defect as (i) mandatory feature *bb6* has a parent mandatory feature *aa3*, and (ii) feature *bb6* excludes feature *bfs* that has a parent optional feature *algorithm*. False optional feature defect *mst* is the output of rule 7 since *mst* is implied by mandatory feature *bb2*. After implementing rule 8, optional feature *cc3* becomes false optional feature defect as it is implied by another false optional feature defect *mst*. False optional feature defect *cc2* is the output of rule 9 as its parent feature *mst* is a false optional feature too. The output of rule 10 is a false optional feature defect *bb5* since it is implied by mandatory feature *cc4* which has a parent mandatory feature *bb3*. Dead feature defect *cc1* is the output of rule 11 as it is excluded by false optional feature defect *mst*.

4. Comparative analysis

On the basis of implementation details, FM is transformed into FMO based on FOL predicates for representing concepts of FM meta-model. SWI-Prolog 7.2.3 is used for implementing rules in FOL to identify defects. The preliminary evaluation was done on a 2.40 GHz, Intel(R) Core(TM) i7 processor machine with 8GB RAM.

The proposed approach for the identification of dead and false optional features is scalable up to 200 features as compared to the one given by Rincón *et al* [9] which is scalable up to 150 features only. Rincón *et al* [9] have proposed six rules for the identification of dead features and three rules for the identification of false optional features. However, the proposed approach provides (i) classification of cases for these defects, (ii) seven rules for the identification of dead feature defects, and (iii) four rules for the identification of false optional feature defects.

Elfaki *et al* [5] have represented variability in SPL by merging FM and orthogonal variability model (OVM) [23] while the proposed approach completely deals with ontology based feature modeling notation as it is more expressive and understandable for PL modelers. The author Elfaki *et al* [5] validated the detection method for defects by using own generated data sets while in the proposed approach, validation was done using real world FMs from SPLOT repository.

5. Related work

Many studies have been proposed so far associated with the representation of FMs in SPL and with the identification of defects e.g., false optional and dead features.

The varied contributions have been proposed for representing FMs in SPL. One way to represent FMs is using ontologies for improving SPL [17, 24, 25] and there is a requirement to add more semantics to FMs. Lee *et al* [26] have proposed that FM could be expressed using ontologies for analyzing their variabilities and commonalities based on semantic analysis criteria. A framework based on Web Ontology Language-Description Logic (OWL-DL) has been used to represent FMs, their configurations and Pellet as a reasoner by Sirin *et al* [27]. Zaid *et al* [20] have represented FMs using an approach based on OWL which facilitated integration of different FMs. Wang *et al* [28] have proposed DL and RACER tool to deal with dead features and void feature models. Later, Wang *et al* [29] have provided explanations using OWL debugging tool.

Matcha *et al* [30] have constructed FMs in ontology and evaluated for the consistency of the feature configuration with an appropriate example. Zaid *et al* [31] have analyzed FMs based on semantic web technologies using Pellet reasoner which focused on conformance checking [19] and further detects inconsistencies in FMs [21]. FOL is proved to be useful for representing FMs. Mannion [8] was the first to connect FOL with FMs. Salinesi *et al* [11] have provided FOL formalization and explanation for the verification criteria of PLMs. Elfaki *et al* [5] have represented variability in SPL using FM and OVM together. In comparison with the afore-mentioned works, the proposed approach transforms FM into FMO for representation using FOL predicate-based ontology modeling.

Several works exist in the literature related to the identification of defects in FMs e.g., false optional and dead features. Many authors indicated the automated identification of dead feature [12, 14, 32, 33]. Trinidad *et al* [34] proposed a method based on finding each product and then exploring unused features to detect dead features. The solvers are used to validate FMs by Trinidad *et al* [35]. They aimed to automate error detection (i.e., full mandatory features, dead features, and void features) based on theory of diagnosis. FMs are mapped to diagnose models and analyzed using constraint satisfaction problems (CSPs). It is based on the configuration of all products in SPL. Finding each solution is a complex process and solvers may take inconceivable amount of time to explore each product to validate large-sized FMs. The set of dependencies and their explanation for dead and false optional features have been identified by automating their approach [35] in FaMa by Trinidad *et al* [36].

Other than above-mentioned approaches, many other ways are used by researchers to represent and analyze defects in SPL. FMs are transformed into generalized feature trees using algorithms [37]. Their work deals only with dead features, explanation, and minimal set of conflicting constraints. Approaches based on contradictory relationship sets (CFRs) [38] and FODA maturity model [39] identifies and explains the causes of dead and false optional features in FMs. Ripon *et al* [40] have detected and provided

explanations for void FMs, dead features, and invalid products using FOL and UML. However, they have detected false optional feature but there is no explanation given for the same. For all features of an SPL, two finite state machines (FSMs) are built, one for the requirements and another for the design level, after which it is specified how to check their conformance [41]. FSMs are extended based on transitions in variables. The prototype SPLEnD uses SPIN model checker for implementing conformance checking. FeatureIDE is a tool to implement explanation for various defects [33].

Minimal correction subsets (MCSes) of constraints have been computed that could be removed from the constraint program (CP) to produce accurate model by Trinidad and Ruiz-Cortes [42]. There is no information provided to PL modelers regarding why these constraints led to the occurrence of defects in PLMs. Transforming models into CPs could loss structure required for these explanations. Thus, techniques based on constraint satisfaction are insufficient for providing these explanations. Therefore, the semantics and structure of PLMs are transformed into ontologies as both properties are of utmost importance for explaining defects.

A constraint-based framework is proposed for SPL that uniformly expressed formalism to explain the constraints within and across perspectives [43]. Their work shows that the problems of liveness (i.e., dead features), consistency, and commonness can be reduced to problems of constraint solving using a realistic case study. The proposed approach identifies false optional features as compared to the work by Millo *et al* [43]. Knowledge-based (KB) method is used to identify inconsistency and dead feature using particular explanations for these defects [44]. Giraldo *et al* [45] have used ontologies and Semantic Query-enhanced Web Rule Language (SQWRL) to identify causes of dead features. However, an ontological rule-based approach has been used to identify and provide causes for false optional and dead features [9]. Elfaki *et al* [5] have deduced false optional features using FOL but did not deduce dead features. Rincón *et al* [46] have identified these defects based on MCSes, CPs, and detected possible corrections. Further, Elfaki [7] has detected and prevented inconsistencies in the process of domain engineering using FOL rules. They have used their own generated data sets to validate these methods [5, 7, 44].

The limitation of work by Trinidad *et al* [35] motivated for developing an approach to identify dead and false optional features that work without solvers. In the proposed approach, the search process is time efficient and has low-cost for large-sized FMs as it searches only for the predefined cases to identify defected features instead of the set of dependencies. It improves semantics and expressivity of FMs by using predicate-based ontology modeling. FOL based rules are used to identify both defects and the proposed approach has been validated using real world cases from SPLOT repository.

6. Conclusion and future work

In this paper, the proposed approach has analyzed the problem of defects in SPL. The dead feature and false optional feature defects have been identified. The cases classified under these defects are considered as a medium issue. These must be managed to verify that efficacious products are developed. In the proposed approach, FM is transformed to FMO and rules are developed using FOL predicates to identify defects. These rules are implemented independently as well as simultaneously in Prolog. The outcome of these rules is defects in an FM that can be used by PL modelers to identify the incorrect relationships which cause these defects. The proposed approach has been validated with a standard case study of GPL and with 35 real world FMs from SPLOT repository up to 200 features.

It should be considered that, presently, the proposed approach is restricted to function in a specific environment i.e., in all the cases where cross tree constraints are well defined in basic FMs only. The existing set of rules in the proposed approach could be maximized by adding new rules to identify dead and false optional feature defects in other FM notations also e.g., extended FMs and cardinality-based FMs. Future line of research could be focused on extending the proposed method to provide explanations for the causes and corrective solutions for each defect. Further, work can be done on execution time to increase the time efficiency.

Acknowledgement

One of the authors, Megha, gratefully acknowledges the University Grants Commission (UGC), New Delhi, Government of India, for awarding her the Rajiv Gandhi National Fellowship (Grant No. F117.1/201415/RGNF201415SCJAM66324) to carry out this research work. We would like to thank Vikram Jeet Singh, Research Scholar, Department of Computer Science, Himachal Pradesh University, Shimla, India.

References

[1] Clements P and Northrop L 2001 *Software product lines: practices and pattern*, Addison-Wesley Professional

[2] Bosch J 2002 Maturity and evolution in software product lines: approaches, artefacts and organization. In: *Software Product Lines, Lecture Notes in Computer Science*, Springer. 257–271. doi:10.1007/3-540-45652-X_16

[3] Kang K C, Cohen S G, Hess J A, Novak W E and Peterson A S 1990 Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, ESD-90-TR-222, SEI

[4] Elfaki A O, Fong S, Aik K and Johar M M D 2013 Towards detecting redundancy in domain engineering process using

first order logic rules. *Int. J. Knowl. Eng. Soft Data Paradigms* 4(1): 1–20. doi:10.1504/IJKESDP.2013.052716

[5] Elfaki A O, Fong S L, Vijayaprasad P, Johar M G M and Fadhil M S 2014 Using a rule based method for detecting anomalies in software product line. *Res. J. Appl. Sci. Eng. Technol.* 7(2): 275–281

[6] Elfaki A O, Phon-Amnuaisuk S and Ho C K 2009 Using first order logic to validate feature model. In: *Proceedings of the 3rd International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS)*. pp. 169–172

[7] Elfaki A O 2016 A rule-based approach to detect and prevent inconsistency in the domain-engineering process. *Expert Syst.* 33(1): 3–13. doi:10.1111/exsy.12116

[8] Mannion M 2002 Using first-order logic for product line model validation. In: *Proceedings of the 2nd International Conference on Software Product Line Conference (SPLC2), Springer: Verlag*. pp. 176–187. doi:10.1007/3-540-45652-X_11

[9] Rincón L F, Giraldo G L, Mazo R and Salinesi C 2014 An ontological rule-based approach for analyzing dead and false optional features in feature models. *Electron. Notes Theoret. Comput. Sci.* 302: 111–132. doi:http://dx.doi.org/10.1016/j.entcs.2014.01.023

[10] Wielemaker J 2007 *SWI-Prolog (Version 5.6.36), free software*, Amsterdam, University of Amsterdam

[11] Salinesi C, Mazo R and Diaz D 2010 Criteria for the verification of feature models. In: *Proceedings of the 28th INFORSID (INFormatique Des ORganisations et Syst'emes d'Information et de Décision)*. pp. 293–308

[12] Von Der Massen T and Lichter H 2004 Deficiencies in feature models. In: *Proceedings of the Workshop on Software Variability Management for Product Derivation – Towards Tool Support*. pp. 59–62

[13] Lopez-Herrejon R and Batory D 2001 A standard problem for evaluating product-line methodologies. In: *Proceedings of the 3rd International Conference on Generative and Component-Based Software Engineering, Springer: Berlin Heidelberg*. pp. 10–24. doi:10.1007/3-540-44800-4_2

[14] Salinesi C and Mazo R 2012 Defects in product line models and how to identify them. *Software product line – advanced topic, InTech*. 97–122. doi:10.5772/35662

[15] Segura S 2008 Automated analysis of feature models using atomic sets. In: *Proceedings of the 1st Workshop on Analyses of Software Product Lines (ASPL 2008), SPLC'08*. pp. 201–207

[16] Gruber T R 1993 A translation approach to portable ontology specifications. *Knowl. Acquisit.* 5(2): 199–220. doi:10.1006/knac.1993.1008

[17] Czarnecki K, Kim C H P and Kalleberg K T 2006 Feature models are views on ontologies. In: *Proceedings of the 10th International on Software Product Line Conference, IEEE Computer Society*. pp. 41–51

[18] De Bruijn J and Heymans S 2006 Translating ontologies from predicate-based to frame-based languages. In: *Proceedings of the 2nd International Conference on Rules and Rule Markup Languages for the Semantic Web, IEEE*. pp. 7–16. doi:10.1109/RULEML.2006.23

[19] Mazo R, Lopez-Herrejon R, Salinesi C, Diaz D and Egyed A 2011 Conformance checking with constraint logic programming: the case of feature models. In: *Proceedings of the 35th Annual International Computer Software and Applications Conference (COMPSAC), IEEE Press*. pp. 456–465

[20] Zaid L A, Houben G, De Troyer O and Kleinermann F 2008 An OWL-based approach for integration in collaborative feature modelling. In: *Proceedings of the 4th Workshop on Semantic Web Enabled Software Engineering (SWESE2008)*. pp. 93–100

[21] Noorian M, Ensan A, Bagheri E, Boley H and Biletskiy Y 2011 Feature model debugging based on description logic reasoning. In: *Proceedings of the 17th International Conference on Distributed Multimedia Systems (DMS'11)*. pp. 158–164

[22] Goldstein R C and Storey V C 1991 Database and expert systems applications. In: *Proceedings of the International Conference in Berlin, Federal Republic of Germany, Springer: Vienna Wien Gmb*. pp. 124–129. doi:10.1007/978-3-7091-7555-2_21

[23] Pohl K, Bockle G and Van Der Linden F 2005 *Software product line engineering foundations principles and techniques*. Springer: Verlag New York, ISBN: 3540243720

[24] Johansen M F, Fleurey F, Acher M, Collet P and Lahire P 2010 Exploring the synergies between feature models and ontologies. In: *SPLC Workshops*. pp. 163–171

[25] Kim C H P 2006 *On the relationship between feature models and ontologies*. University of Waterloo, Master's Thesis. (Available at http: //gsd.uwaterloo.ca/2006/05/11/peter-kims-masc-thesis/.14)

[26] Lee S, Kim J, Song C and Baik D 2007 An approach to analyzing commonality and variability of features using ontology in a software product line engineering. In: *Proceedings of the 5th International Conference on Software Engineering Research, Management and Applications*. pp. 727–734

[27] Sirin E, Parsia B, Grau BC, Kalyanpur A and Katz Y 2007 Pellet: a practical OWL-DL reasoner. *Web Semant. Sci., Services Agents World Wide Web*. 5: 51–53. doi:10.1016/j.websem.2007.03.004

[28] Wang H, Li Y F, Sun J, Zhang H and Pan J 2005 A semantic web approach to feature modeling and verification. In: *Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*. Galway, Ireland

[29] Wang H H, Li Y F, Sun J, Zhang H and Pan J 2007 Verifying feature models using OWL. *Web Semant. Sci., Services Agents World Wide Web*. 5: 117–129. doi:http://dx.doi.org/10.1016/j.websem.2006.11.006

[30] Matcha V B, Reddy P V G D P, Hari C V M K, Srinivas G, Rao N S, Jayachand B, Kumar J N V R S, SriRamGanesh G, Krishna N V R V V, Pradeep I K and Ramesh C 2009 Software reuse: ontological approach to feature modeling. *Int. J. Comput. Sci. Netw. Security* 9(8): 262–268

[31] Zaid L A, Kleinermann F and Troyer O D 2009 Applying semantic web technology to feature modeling. In: *Proceedings of the 2009 ACM symposium on Applied Computing (SAC '09)*. ACM, New York, NY, USA. pp. 1252–1256. doi:http://doi.acm.org/10.1145/1529282.1529563

[32] Mazo R, Salinesi C and Diaz D 2012 VariaMos: a tool for product line driven systems engineering with a constraint based approach. In: *Proceedings of the 24th International Conference on Advanced Information Systems Engineering (CAiSE Forum '12)*. Springer Press, Gdansk-Poland. pp. 25–29

[33] Thüm T, Kästner C, Benduhn F, Meinicke J, Saake G and Leich T 2014 FeatureIDE: an extensible framework for

feature-oriented software development. *Sci. Comput. Program.* 79: 70–85. doi:http://dx.doi.org/10.1016/j.scico.2012.06.002

[34] Trinidad P, Benavides A and Ruiz-Cortés A 2006 Isolated features detection in feature model. Paper presented at the *Advanced Information Systems Engineering (CAiSE), Luxembour*. doi:http://www.ceur-ws.org/Vol-231/Paper19.pdf

[35] Trinidad P, Benavides D, Duran A, Ruiz-Cortés A and Toro M 2008a Automated error analysis for the agilization of feature modelling. *J. Syst. Softw*. 81: 883–896. doi:10.1016/j.jss.2007.10.030

[36] Trinidad P, Benavides D, Ruiz-Cortés A, Segura S and Jimenez A 2008b FAMA framework. In: *Proceedings of the 12th International Software Product Line Conference (SPLC'12), IEEE Computer Society*. pp. 359. doi:10.1109/SPLC.2008.50

[37] Broek P and Galvão I 2009 Analysis of feature models using generalised feature trees. In: *Proceedings of the Third International Workshop on Variability Modeling of Software-intensive Systems (VaMoS'09)*, University of Sevilla, Spain. pp. 71–76. doi:10.1.1.216.5377

[38] Zhang G, Ye H and Lin Y 2013 An approach for validating feature models in software product lines. *J. Softw. Eng*. 7(1): 1–29. doi:10.3923/jse.2013.1.29

[39] Javed M, Naeem M and Wahab H A 2014 Towards the maturity model for feature oriented domain analysis. *Comput. Ecol. Softw*. 4(3): 170–182

[40] Ripon S, Hossain J and Bhuiyan T 2013 Managing and analysing software product line requirements. *Int. J. Softw. Eng. Appl.* 4(5): 63–75. doi:10.5121/ijsea.2013.4505

[41] Millo J-V, Ramesh S, Krishna S N and Narwane G K 2013 Compositional verification of software product lines. In: *Proceedings of the 10th International Conference on Integrated Formal Methods, Springer, Lecture Notes in Computer Science*. vol. 7940, pp. 109–123. doi:10.1007/978-3-642-38613-8_8

[42] Trinidad P and Ruiz-Cortés A 2009 Abductive reasoning and automated analysis of feature models: How are they connected. In: *Proceedings of the 3rd International Workshop on Variability Modelling of Software-Intensive Systems*. pp. 145–153. doi:10.13140/2.1.4955.0400

[43] Millo J-V, Mohalik S K and Ramesh S 2011 Integrated analysis of software product lines: a constraint based framework for consistency, liveness, and commonness checking. In: *Proceedings of the 4th India Software Engineering Conference, ISEC '11*, New York, NY, USA. ACM. pp. 41–50. doi:10.1145/1953355.1953361

[44] Osman A, Amnuaisuk S P and Ho C K 2008 Knowledge based method to validate feature models. In: *Proceedings of the 12th International Conference Software Product Lines Conference (SPLC* 2008), Limerick, Ireland. pp. 217–225

[45] Giraldo G L, Rincón-Perez L and Mazo R 2013 Identifying dead features and their causes in product line models: an ontological approach. *Revista DYNA*. 81: 68–77. doi:10.15446/dyna.v81n183.36348

[46] Rincón L, Giraldo G L, Mazo R, Salinesi C and Diaz D 2015 Method to identify corrections of defects on product line models. *Electron. Notes Theoret. Comp. Sci*. 314: 61–81. doi:10.1016/j.entcs.2015.05.005