The Journal of Systems and Software 000 (2017) 1–13

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Analyzing inconsistencies in software product lines using an ontological rule-based approach

Megha Bhushan[a,*], Shivani Goel[b], Karamjit Kaur[a]

[a] CSED, Thapar University, Patiala, Punjab, India
[b] Department of Computer Science Engineering, Bennett University, Greater Noida, U.P., India

ARTICLE INFO

Article history:
Received 23 March 2016
Revised 14 January 2017
Accepted 1 June 2017
Available online xxx

Keywords:
Feature model
Software product line
Rule-based approach
Ontology
Inconsistency
Defects

ABSTRACT

Software product line engineering (SPLE) is an evolving technical paradigm for generating software products. Feature model (FM) represents commonality and variability of a group of software products that appears within a specific domain. The quality of FMs is one of the factors that impacts the correctness of software product line (SPL). Developing FMs might also incorporate inaccurate relationships among features which cause numerous defects in models. Inconsistency is one of such defect that decreases the benefits of SPL. Existing approaches have focused in identifying inconsistencies in FMs however, only a few of these approaches are able to provide their causes. In this paper FM is formalized from an ontological view by converting model into a predicate-based ontology and defining a set of first-order logic based rules for identifying FM inconsistencies along with their causes in natural language in order to assist developers with solutions to fix defects. A FM available in software product lines online tools repository has been used to explain the presented approach and validated using 24 FMs of varied sizes up to 22,035 features. Evaluation results demonstrate that our approach is effective and accurate for the FMs scalable up to thousands of features and thus, improves SPL.

1. Introduction

A software product line (SPL) is a family of related software intensive systems, sharing a common and managed set of features that fulfill the exact requirements of an appropriate market segment (Clements and Northrop, 2001). The main focus of SPL is software reuse in an attempt to improve the quality and productivity while reducing cost as well as time to market.

Although other variability models for modeling variability and commonality exist, such as decision models (Schmid and John, 2004), dopler variability models (Dhungana et al., 2011), orthogonal variability models (Pohl et al., 2005) and textual variability language (Classen et al., 2011). However, feature model (FM) is the most popular variability model for SPL that illustrates the features and their relationships (Kang et al., 1990). There are various factors that impact the success of SPL such as (i) testing which includes to validate and verify the generated software products, (ii) poor interface among customers and organization hinders communication of information and customer satisfaction, (iii) reusability decreases with developing product line (PL) which includes bug fixing, performance etc., (iv) overloaded software development system becomes a challenge in the development and maintenance of the system, and (iv) low quality software products can lead to the development of defective new products. Therefore, ensuring quality in the software development paradigm is of utmost importance. The complexity of FMs increases with growing number of features and relationships which cause an increase in the contradictory relations. These contradictory relationships arise due to the combination of features derived from multiple software products in order to develop a new product. Presence of a single contradictory relation in a FM can cause an inconsistency defect which makes the model inconsistent. Inconsistency in FMs is a critical issue as software products are derived by reusing models. A FM with inconsistency defects has contradictory information i.e., the information which conflicts with other information in the similar model. It will not allow deriving products, as these products may include inconsistent configurations. Therefore, defect due to inconsistency is one of the major factors that deteriorates the reusability and quality of models. For example, various developers that belong to the same business unit working on a SPL of automotive assembly cause an inconsistency defect by adding automatic and manual gears where both gears cannot exist at the same time. Other examples include Nokia (Thao, 2012), LG Industrial Systems (Pohl et al., 2005) etc.

Several techniques exist for the identification of FM inconsistencies such as constraint satisfaction techniques, logic truth

* Corresponding author.
E-mail addresses: megha@thapar.edu (M. Bhushan), shivani.goel@bennett.edu.in (S. Goel), karamjit.kaur@thapar.edu (K. Kaur).

http://dx.doi.org/10.1016/j.jss.2017.06.002
0164-1212/

maintenance systems, formal semantics, logic based methods, genetic algorithms and many more. Few approaches based on first-order logic (FOL) provide an effective way to formalize FMs using ontology (Gruninger et al., 2008; Àlvez, 2012 and Bhushan and Goel, 2016), which is a formal, explicit specification of a shared conceptualization (Gruber, 1993). It improves the level of expressivity when compared to FMs and facilitates to infer interesting information related to the models; for instance, to verify consistency among the FM and its meta-model, and to retrieve child features. Other researchers have also used methods based on ontologies (Wang et al., 2007; Noorian et al., 2011 and Guo et al., 2012) and FOL (Mannion, 2002 and Elfaki, 2016) to deal with defects due to inconsistency.

Although several methods exist that have identified inconsistency (Mannion, 2002; Gheyi et al., 2011; Zhang and Moller-Perdersen, 2013; Yang and Dong, 2013; Asadi et al., 2014 and Thüm et al., 2014b) but only a few methods have provided explanation for their causes in a manner which is difficult to understand by developers (Segura et al., 2010; Noorian et al., 2011; Felfernig et al., 2013 and Lesta et al., 2015). However, Thüm et al. (2014a) have only detected the constraints involved in dead features along with false-optional features and Elfaki (2016) has only prevented indirect inconsistency. The lack of methods to explain the cause of inconsistency defects in a user friendly language and to recommend solutions for resolving inconsistencies motivated us to propose an effective approach. Moreover, manually inspecting larger FMs to detect inconsistencies is a laborious task. Therefore, handling defects due to inconsistency is a critical task in order to derive defect free valid products from SPL, to improve reusability and quality of SPL models.

We have developed an ontological approach based on FOL rules that handles FM inconsistencies to improve the quality of FMs in SPL and the results of evaluation using 24 FMs verified its effectiveness, accuracy and scalability with thousands of features in FMs. Following are the contributions of the proposed approach:

I. We classify FM defects due to inconsistencies in the form of cases.
II. We formalize FM using FOL predicate-based feature model ontology (FMO) which is one of the important contributions of the proposed approach that provides a correspondence among FM representations and FMO.
III. We define and implement a set of FOL rules in Prolog (Wielemaker, 2015) to deal with inconsistencies in FMs.
IV. We identify FM inconsistencies and their causes using a user friendly natural language in the presented classification.
V. The causes explained in natural language are easily understandable by developers and this information assists them to fix inconsistency defects by recommending solutions (i.e., by eliminating relationships involved in the defect).
VI. Evaluation results of the proposed approach using real-world FMs from online software product lines online tools (SPLOT[1]) repository as well as randomly generated FMs with thousands of features verifies our approach to be effective, accurate and scalable up to 22,035 features in FMs. Thus, it allows deriving defect free software products by subsequently enhancing the reusability and quality of FMs in SPL.

Following is the structure of remaining paper: Section 2 presents preliminaries required to understand the approach presented in Section 3. Section 4 analyzes performance and evaluates the scalability, execution time and accuracy of proposed approach. Section 5 compares our approach with related works. Section 6 describes concluding remarks and future directions.

[1] http://www.splot-research.org/.

2. Preliminaries

2.1. Inconsistencies

Various types of inconsistency defects discussed in this paper are explained with the help of Fig. 1 (Salinesi et al., 2010 and Elfaki, 2016), where *r* represents root feature which is mandatory to be included in each valid product of the PL, *p1* and *p2* represent parent child features, and f1 and f2 represent child features.

Rule 1: Mandatory child features *f1* and *f2* have same parent root feature *r*, and *f1* implies *f2* whereas *f2* excludes *f1*. Thus, it is an inconsistency as both features *f1* and *f2* can never be selected simultaneously for the configuration of a valid product.
Rule 2: Mandatory child features *f1* and *f2* have same parent root feature *r* where *f1* excludes *f2*. Thus, it is an inconsistency as both features *f1* and *f2* are mutually excluded and these features can never be selected together for the configuration of a valid product.
Rule 3: Mandatory parent features *p1* and *p2* have same root feature *r* where *p1* excludes *p2* and *p1* has a mandatory child feature *f1* which implies *p2*.
Rule 4: Mandatory parent features *p1* and *p2* have same root feature *r* where *p1* excludes *p2* and *p1* has a mandatory child feature *f1* which implies an optional child feature *f2* whose parent is *p2*.
Rule 5: A mandatory parent feature *p2* implies an optional parent feature *p1* where both features have same root feature *r* and *p1* has a mandatory child feature *f1* which excludes *p2*.
Rule 6: A mandatory parent feature *p1* implies an optional parent feature *p2* where both features have same root feature *r* and *p1* has a mandatory child feature *f1* which excludes the other mandatory child feature *f2* having parent *p2*.
Rule 7: A mandatory parent feature *p1* implies an optional parent feature *p2* where both features have same root feature *r* and *p1* has a mandatory child feature *f1* which excludes *p2*.
Rule 8: Mandatory child features *f1* and *f2* belong to the group cardinality <1..1> with a mandatory parent feature *p* where *f1* implies *f2*. In this particular case, *p* can also be connected to the root feature. The implication relationship among the child features exceeds the upper limit of the group cardinality and does not even allow to incorporate one sub feature.

2.2. Running example of FM

The variabilities and commonalities in SPLE are represented using feature modeling notation by means of relationships among the features. A feature is defined as a unique element that is of relevance to the user. It is a hierarchical tree structure comprising of features and relationships among them. The entire SPL is represented with the help of root of the FM tree. An adapted version of the address FM available in the SPLOT repository (Mendonca et al., 2009) is used as a running example in this paper. Fig. 2 describes the address FM using feature modeling notation where "*addressbook*" is the root feature. There can be more than one child feature associated with a parent feature. A unique name has been assigned to each feature in FM. The cross tree constraint relationships i.e., exclusion and implication among features have also been shown for better interpretation of the proposed approach. To illustrate the proposed approach, 14 cross tree constraints and 18 additional features were introduced in the primary model to inject inconsistency defects.

Following illustrates various feature modeling relationships using Fig. 2:

Please cite this article as: M. Bhushan et al., Analyzing inconsistencies in software product lines using an ontological rule-based approach, The Journal of Systems and Software (2017), http://dx.doi.org/10.1016/j.jss.2017.06.002