Received: 23 March 2016 | Revised: 7 August 2017 | Accepted: 29 September 2017

DOI: 10.1111/exsy.12256

WILEY Expert Systems

ORIGINAL ARTICLE

Improving quality of software product line by analysing inconsistencies in feature models using an ontological rule-based approach

Megha Bhushan[1] | Shivani Goel[2] | Ajay Kumar[1]

[1] CSED, Thapar University, Patiala, Punjab, India

[2] Department of Computer Science Engineering, School of Engineering and Applied Sciences, Bennett University, Greater Noida, U.P., India

Correspondence
Megha Bhushan, CSED, Thapar University, Patiala, Punjab, India.
Email: megha@thapar.edu

Funding information
University Grants Commission (UGC), New Delhi, Government of India, Rajiv Gandhi National Fellowship, Grant/Award Number: F117.1/201415/RGNF201415SCJAM66324

Abstract

In software product line engineering, feature models (FMs) represent the variability and commonality of a family of software products. The development of FMs may introduce inaccurate feature relationships. These relationships may cause various types of defects such as inconsistencies, which deteriorate the quality of software products. Several researchers have worked on the identification of defects due to inconsistency in FMs, but only a few of them have explained their causes. In this paper, FM is transformed to predicate-based feature model ontology using Prolog. Further, first-order logic is employed for defining rules to identify defects due to inconsistency, the explanations for their causes, and suggestions for their corrections. The proposed approach is explained using an FM available in Software Product Line Online Tools repository. It is validated using 26 FMs of discrete sizes up to 5,543 features, generated using the FeatureIDE tool and real-world FMs. Results indicate that the proposed methodology is effective, accurate, and scalable and improves software product line.

KEYWORDS

feature model, feature model ontology, inconsistency, ontological rule-based approach, software product line

1 | INTRODUCTION

A software product line (SPL) is a set of software products sharing some common functions for addressing specific requirements of a certain domain. The objective of SPL engineering is reusability of software products. It allows several organizations to amplify the quality of software products by diminishing development cost and time. For instance, Microsoft offers some well-known product lines (PLs) such as Microsoft Office, Windows, Visual Studio, and Skype. Organizations such as Motorola and Nokia have attained their business goals using SPL approach.

SPL is modelled using features and their relationships in feature model (FM) (Kang, Cohen, Hess, Novak, & Peterson, 1990). A feature is defined as a property, characteristic, or requirement of a software system relevant to the user. A feature can be a chunk of code (feature-oriented programming) (Prehofer, 2001), a component in architecture (Bernardo, Ciancarini, & Donatiello, 2002), a requirement (Jarzabek, Ong, & Zhang, 2003), or a non-functional characteristic (quality). For instance, *call waiting* and *call forwarding* are two basic features of a phone. Another instance, a car model incorporates features such as colour, which can be red, black, blue, and so forth. The variability of SPL is represented using a tree structure, namely, FM. A valid and distinct combination of features is used to represent each product in SPL, where a feature set characterizes each software product. A high-quality FM provides accurate combinations of features.

Software Engineering Institute identifies 29 practice areas for the success of SPL strategy (Northrop, 2008). These practice areas are grouped into technical management, software engineering, and organizational management, where each group includes several essential activities for the development of valid software products in SPL, for example, ensuring integrity of the PL and handling the challenge of customer satisfaction by eliminating poor customer interface. An SPL-based organization needs to manage several aspects of its customer interface, including identifying needs, providing technical support, providing information, and so forth. Testing is an important process for finding defects in software products,

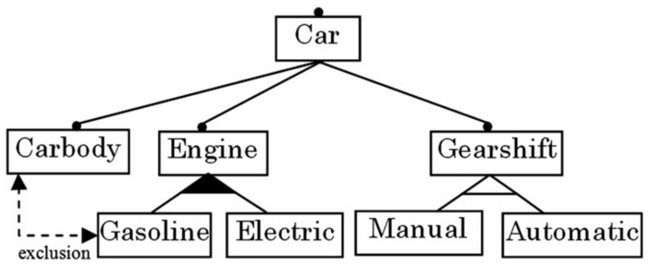


FIGURE 1 Car feature model from SPLOT repository

validating and verifying these products for a successful SPL. The maintenance of PL becomes more challenging due to high-level reusability. Growing PL consists of activities such as functionality evolution, bug fixing, and performance optimization, which diminishes reusability (Deng, Lenz, & Schmidt, 2006). Many issues are encountered during the development of SPL, such as

i. overloaded software development system,
ii. poor inter-group communication,
iii. division of personnel in multi-directions,
iv. underutilization and overutilization of resources,
v. limited awareness of new technological trends,
vi. painful transition,
vii. lack of management support,
viii. inadequate technical and administrative effort,
ix. excessive number of inter-module relationships within the architecture, and
x. multi-tier architectures in PLs, leading to complicated software systems.

However, the quality of software products is another essential part for developing the PL. Defects arising from the development process can affect all products in the PL. Therefore, it is of utmost importance to ensure quality during each phase of the development cycle. The quality of products can be improved by reducing the load of the development team. There can be some contradictory relationships (i.e., inconsistencies) when multiple modules are configured individually by several developer teams. The configuration of modules is based either on the selection of features by developer teams or the existing selection rules as per the goals of developer teams, their experience, and evaluation criteria to develop large-scale SPL. It leads to the misuse of business resources, reduced productivity, and conflicts with customer requirements. Further, the conflicts between developer teams restrict the software development process. It becomes more complex with changing customer requirements for product. The number of relations increases with the growing count of features in FM. Accordingly, the count of contradictory relations and complexity of FM increases.

Figure 1 depicts a car FM available in Software Product Line Online Tools (SPLOT[1]) repository. It consists of features, namely, *Carbody*, *Engine*, and *Gearshift*. *Engine* can be *Gasoline*, *Electric*, or both. *Gearshift* can be *Automatic* or *Manual*. For instance, a mutual exclusion between *Carbody* and *Gearshift* would result in an inconsistency as both cannot exist simultaneously. Another example includes the business units affected by inconsistencies at the Samsung Tech Advanced Research Center, Georgia Tech. Several case studies (Niu, Savolainen, & Yu, 2010; Pohl, Böckle, & van der Linden, 2005; van Gurp & Prehofer, 2008) have also highlighted the problems due to inconsistencies such as LG Industrial Systems (Pohl et al., 2005), Nokia (Ardis, Daley, Hoffman, Siy, & Weiss, 2000; D'Souza, Gopinathan, Ramesh, & Sampath, 2010; Thao, 2012), and so forth.

In another scenario, multiple architects work in parallel on their workstations and produce a single architectural model with different versions. These versions must be merged episodically to enable detection of errors and collaboration between these architects. For instance, merging of different software products or product variants (i.e., all products undergo consistent evolution to fulfil their relevant environmental changes and customer requirement changes) developed by a company to facilitate reusability between the products. The merging of the model without the help of a tool may result in inconsistent semantic or syntactic version.

Many intelligent methods such as logic-based methods (Fan & Naixiao, 2006; Hemakumar, 2008), constraint satisfaction techniques (White et al., 2010; White et al., 2014), formal semantics, optimization algorithms (Afzal, Mahmood, Rauf, & Shaikh, 2014), and so forth have been proposed till date to identify inconsistencies in an FM. First-order logic (FOL) (Elfaki, 2016; Elfaki, Phon-Amnuaisuk, & Ho, 2009; Osman, Amnuaisuk, & Ho, 2008) and ontology-based approaches (Guo, Wang, Trinidad, & Benavides, 2012; Noorian, Ensan, Bagheri, Boley, & Biletskiy, 2011; Wang, Li, Sun, Zhang, & Pan, 2007) are also used to deal with FM inconsistencies. FMs are represented by ontologies on the basis of FOL predicates (Àlvez, Lucio, & Rigau, 2012; Gruninger & Menzel, 2003; Pease & Sutcliffe, 2007; Ramachandran, Reagan, & Goolsbey, 2005;

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

Schneider, Carroll, Herman, & Patel-Schneider, 2009; Schneider & Sutcliffe, 2011). However, manually inspecting inconsistencies in large-sized FMs is a laborious task. Thus, identifying inconsistencies with explanations for their causes and suggestions for their corrections is a critical task for maintaining the quality of software products in SPL.

The proposed approach deals with FM defects due to inconsistency in SPL. We evaluated the proposed approach using 26 FMs available in SPLOT repository as well as models generated using a FeatureIDE tool (Thüm, Kästner, et al., 2014). Results indicate that the proposed work is effective, accurate, and scalable up to 5,543 features. The novelty of our work includes the (a) classification of defects due to inconsistency in the form of cases, (b) construction of predicate-based FM ontology (FMO) to represent FM, (c) identification of defects due to inconsistency using FOL-based rules, which provide explanations for their causes using natural language and suggest corrective solutions (i.e., alter or delete relationships related to the source of defects) in the presented classification, (d) provision of communication between FM representations and FOL predicate-based ontologies, (e) improvement in the quality of SPL as it identifies more types of defects due to inconsistency (see Section 4.2.2), and (f) information provided to PL modellers for resolving defects in order to produce defect-free products.

The paper is organized as follows: Section 2 discusses related work. Section 3 presents a concise description of the general terminology necessary for comprehending the proposed approach and its implementation details, which are illustrated in Section 4. The accuracy and scalability of our approach are estimated in Section 5. Finally, the conclusion and directions for future work are provided in Section 6.

2 | RELATED WORK

Relevant literature has been divided into representation of FMs using ontologies and validation of FMs to deal with inconsistencies in SPL.

An ontology is a formal and explicit specification of a shared conceptualization (Gruber, 1993). In the Artificial Intelligence domain, knowledge is represented in a formalism, and the inferencing capability is used to extract information from it. Lee, Kim, Song, and Baik (2007) worked on analysing the semantic similarity of the FM. They represented FM using ontology and analysed the variability and commonality in it. Wang et al. (2007) identified invalid configurations in FMs using web ontology language (OWL). Further, the causes of invalid configurations were explained using ontological terms and an OWL debugging tool. Moreover, they provided the analysis and visualization of FMs using CASE tool. Abo, Houben, De Troyer, and Kleinermann (2008) represented FM using an OWL-based ontology. The consistency of FM is validated with Semantic Web Rule Language rules. They worked on resolving conflicts and integrating FMs. However, their work did not include the analysis of FMs. Al Balushi, Sampaio, and Loucopoulos (2013) proposed a quality-driven requirement engineering framework. Furthermore, they developed a tool for knowledge management and building quality ontologies. This tool deals with requirement engineering activities, for instance, prioritization and elicitation of quality requirements.

OWL-DL is a formal language for representing knowledge in terms of ontologies. It is inefficient to deal with expressive ontologies like SUMO (Niles & Pease, 2001), DOLCE (Gangemi, Guarino, Masolo, Oltramari, & Schneider, 2002), and Cyc (Matuszek, Cabral, Witbrock, & Deoliveira, 2006). OWL is not expressive for (a) characterizing process models based on web service and (b) representation and interpretation of process model (Gruninger, Hull, & Mcilraith, 2008). Like other languages for process modelling, OWL should be more expressive for easy interpretation to fix ambiguities. Berardi, Grüninger, Hull, and Mcilraith (2004) developed FOL predicates to overcome these limitations because FOL is a more expressive logic-based language. Similar to OWL-DL, FOL is also a popular expressive formalism used for reasoning on ontologies (Schneider et al., 2009). FOL ontology is used to describe a process model that allows development of a framework for web service process modelling (Berardi et al., 2004). This framework automates the configuration and searching of web services. Later, an impressive growth of first-order (FO) theorem provers was observed in computing inferences that are not handled by description logic reasoners (Tsarkov, Riazanov, Bechhofer, & Horrocks, 2004). It facilitates better understanding and implementation of the reasoning tasks. Gruninger and Menzel (2003) extended the process specification language to develop a FOL-based ontology for web services (i.e., FLOWS). It provides a higher level of expressiveness than OWL-DL; for instance, it instantly incorporates *n*-ary predicates along with rich quantified sentences. Further, this ontology provides a framework to model the semantics of web services and enables reasoning on it (Gruninger et al., 2008). It represents different types of data flow from web services in a more refined manner using FOL than in OWL-S. FOL overcomes the issues related to the expressiveness of OWL-S through its rich expressiveness level and well-known semantics based on model-theoretic (Berardi et al., 2004). In other cases, ontology is translated into FOL such as SUMO ontology (Pease & Sutcliffe, 2007), a part of OWL 2 Full (Schneider et al., 2009; Schneider & Sutcliffe, 2011), Cyc ontology (Ramachandran et al., 2005), and Adimen-SUMO ontology (Àlvez et al., 2012). The consistency of FO ontology is checked using the model generation system on the basis of logic programming (Baumgartner & Suchanek, 2006).

Several works dealing with defects arising from inconsistencies in FMs are discussed in the rest of the section. Inconsistency can be solved by debugging the inconsistent FM and by providing a consistent explanation to fix it. Fan and Naixiao (2006) formalized an FM to identify inconsistencies using description logics. Their technique only identified inconsistencies but did not find any solution to fix them. Hemakumar (2008) detected inconsistency in FMs using a connection between propositional logic and context-free grammar. The author identified a direct inconsistency in the configuration process of a software product and concluded that the time needed to identify an inconsistency even in a small-sized FM is significantly high in general.

Knowledge-based (KB) method along with FOL rules is used by (a) Osman et al. (2008) to identify inconsistency and redundancy (by looking for particular explanations for these defects), but they have not established its comprehensiveness and (b) Elfaki et al. (2009) and Elfaki (2016) to identify and prevent inconsistency in the domain engineering process without the requirement of configuration process. The independent FOL rules given by Elfaki (2016) identified three types of inconsistencies as well as prevented direct inconsistency (i.e., feature A implies feature B, and feature B excludes feature A). These methods have been validated using their own generated data sets.

White, Dougherty, Schmidt, and Benavides (2009) presented a methodology that detects and suggests changes in feature configurations to correct inconsistencies in FMs. However, it solves inconsistencies during configuration only. White et al. (2010) transformed FM into constraint satisfaction problem (CSP) (Tsang, 1993). The valid configurations are generated by implementing corrective explanations (i.e., by selecting or deselecting features) to fix invalid configurations. Their approach supports FMs, which include thousands of features. The limitation of their approach is that the time for diagnosing invalid configurations depends on the constraint's structure in the FM.

A dynamic-priority-based mechanism is used by Wang et al. (2010) to produce consistent FM on the basis of priorities assigned to dependencies. Their mechanism automatically suggests researchers a correction to fix inconsistency by removing one or more minimum subset of weaker (i.e., lower priority) dependencies. Because this mechanism only provides one correction per iteration, it must be repeated until the requirements of users are fulfilled. Noorian et al. (2011) transformed FM from simple XML FM (SXFM) format into OWL-DL file. They identified and provided corrections for inconsistencies in the model using Pellet reasoner (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2007). The minimal subsets of OWL axioms generated using this reasoner must be eliminated from OWL-DL file to attain model consistency. The corrections in the form of OWL-DL pure are not easy to understand by modellers, which is a limitation of their approach. Also, evaluation results concluded that the time to identify and fix inconsistencies increases with an increase in the count of features and inconsistencies in test cases.

Guo et al. (2012) formalized FM using ontology. The consistency of large FM is checked by diminishing the impact of changes to the concerned parts using an evolutionary change. Their approach has been evaluated using randomly generated models. Instead of considering the entire FM, only the changed part is considered up to the previous consistent version. This leads to reduced complexity of the problem. However, no automated support is provided for the same. Felfernig, Benavides, Galindo, and Reinfrank (2013) explained the causes of anomalies in FM. They discussed the FMCORE and FASTDIAG algorithms to determine minimal sets for non-redundant constraints and minimal diagnoses, respectively. The minimal sets of constraints (conflicts) had to be modified or removed from the FM to achieve model consistency. The entire set of diagnoses was determined using Hitting Set Directed Acyclic Graph algorithm proposed by Reiter (1987) to detect and fix a conflict. Although Felfernig et al. (2013) recommended corrections for anomalies in terms of inconsistencies and redundancies, they have not provided any implementation details. White et al. (2014) developed a solver to model and resolved problems due to the multi-step configuration of emerging FMs. The derivation of configurations can be automated by mapping the problem of SPL configuration to a CSP. Additionally, it enables researchers to automatically reason about the evolution of PL over time.

Various researchers used tools for handling inconsistencies in SPL. Mendonca, Branco, and Cowan (2009) created SPLOT at the University of Waterloo, Canada, in 2009, which is currently the largest repository of real SPL FMs (i.e., 430 models or 11,500 features). SPLOT allows researchers to analyse, edit, configure, debug, share, and download FMs (Mendonca, Branco, & Cowan, 2009). FMs in SXFM format are read and saved from FM repository available in SPLOT. One of the limitations of the tool is that it neither enables code generation nor provides any means to build a graphical representation of an FM. Another concern is the privacy of models, as models can be easily accessed and customized by anybody. Also, this tool allows researchers to generate only a single product configuration which cannot be stored in the SPLOT repository.

Segura, Benavides, and Ruiz-Cortés (2010) developed a FaMa tool suite, dedicated to the verification, editing, and automated analysis of FMs. It supports cardinality-based FMs, the import of FMs from XMI and export of FMs to XML, and analysis operations of FMs and GUI for representing models. This tool compares different solvers such as SAT, CSP, BDD, and Java. FaMa tool automatically selects the most proficient solver according to the user request for the operation. The tool enables FMs to incorporate numerical constraints and to derive optimal configurations. It also finds the overall number along with all the feasible products of an FM, validates models, and analyses the feature commonality. Limitations of the tool are that it neither includes feature interactions nor non-functional property values; also, instead of generating explanations in natural language, it generates a list of relationships incorporated in the defect that must be modified by modellers to fix the defect. Moreover, it does not provide corrective explanations for defects.

Research Group of GIRO developed a Feature Modelling Tool[2] at the University of Valladolid and Burgos, Spain, in 2008. It allows researchers to model the features graphically. Feature Modelling Tool models and configures FMs within Visual Studio IDE. It provides visualization of an indented list along with tree structure where nodes and links represent features and component hierarchy, respectively. The modeller design facilitates addition, deletion, or modification of features. Eclipse Modelling Framework[3] (EMF) FM project is a popular open source Eclipse plug-in under the Eclipse Modelling Framework Technology project. It is developed by a team at the IBM Cary NC lab. EMF is a standard representation of FMs within the platform of Eclipse. EMF defines

i. a feature meta-model,
ii. an extendible framework for evaluation engine,
iii. Feature Diagram editor for graphical representation of Feature Diagram,

[2]http://giro.infor.uva.es/FeatureTool.html
[3]http://www.eclipse.org/proposals/feature-model/

TABLE 1 Comparison of existing approaches with the proposed approach

	Fan and Naixiao (2006)	Wang et al. (2007)	Hemakumar (2008)	Osman et al. (2008)	Elfaki et al. (2009)	Segura et al. (2010)	White et al. (2010)	Wang et al. (2010)	Noorian et al. (2011)	Guo et al. (2012)	Felfernig et al. (2013)	White et al. (2014)	Thüm, Kästner, et al. (2014)	Elfaki (2016)	Proposed approach
Inconsistency	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Identification	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Explanation		✓		✓		✓	✓	✓	✓	✓	✓	✓			✓
Corrective explanation				✓			✓	✓	✓	✓		✓	✓	✓	✓
Wrong cardinality															✓
Classification															✓
Ontology based		✓							✓	✓					✓
Real world model						✓			✓		✓		✓		✓
Basic FM	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓		✓
Cardinality-based FM				✓	✓					✓					✓
Extended FM				✓											
Automated support						✓	✓	✓	✓		✓	✓	✓	✓	✓
Formalization	✓								✓	✓✓	✓	✓		✓	✓

Note. Here, cell marked with "✓" specifies that the corresponding author work supports the corresponding parameter.

FM = feature model.

iv. extendible visualizations as well as editors for the FMs, and
v. additional Eclipse projects.

The main limitation of EMF is the huge count and size of dependencies in the model at runtime.

Hydra[4] is an Eclipse plug-in as a standard feature modelling tool (Budinsky, Steinberg, Merks, Ellersick, & Grose, 2003; Stephan & Antkiewicz, 2008). It provides interoperability among other tools. This tool allows researchers to configure, validate, and automatically generate minimal configurations of cardinality-based FM with clonable features (Gamez & Fuentes, 2013). It provides support to the full graphical capabilities, both in configuring and editing FMs. Hydra creates a valid configuration by adding a minimum number of features to a configuration that is ensured as chosen configuration by its constraint solver. The time required for creating a configuration relies on the count of features chosen in the configuration, and it is tedious to model the configuration.

Leich, Apel, Marnitz, and Saake (2005) implemented FeatureIDE tool at the University of Magdeburg, Germany in 2005. It is integrated with Eclipse to directly produce code (C++ or Java) from an FM and to graphically represent model (Thüm, Kästner, et al., 2014). This tool provides support for the complete life cycle of an SPL (i.e., from domain engineering to feature-oriented software development). It allows researchers to edit model in its editor by importing and exporting different formats. FeatureIDE fixes void models, false optional as well as dead features in FM by providing corrective solutions. The exclusion and implication relationships involved in the defect are automatically identified by the tool to correct the identified defect. The limitation of the tool is that it does not support corrective solutions for the defect, which requires elimination of more than one relationship to fix it.

Table 1 depicts the comparison of existing approaches with the proposed approach on the basis of various parameters, and the details of these parameters are given in Appendix A. Most of the existing approaches (Asadi, Gröner, Mohabbati, & Gašević, 2014; Yang & Dong, 2013; Zhang & Møller-Pedersen, 2013) only identified a direct inconsistency in the configuration process, while the proposed approach additionally identified other types of inconsistencies without the requirement of the configuration process. Furthermore, the proposed approach suggested corrections for other cases of inconsistencies in contrast to the Elfaki (2016) approach where only a direct inconsistency is prevented. Thüm, Kästner, et al. (2014) developed a tool that suggests corrective solutions to fix defects. The major drawback of their tool is that it does not support corrections to fix the defects, which require deleting more than one relationship. The proposed approach also suggests the corrections that include deletion of more than one relationship. Hence, it allows modellers to create well-formed models in the development process, thereby, significantly reducing the time and cost for finding suggestions to correct inconsistencies in FM during early development stages.

3 | GENERAL TERMS

3.1 | Software product line

An SPL is a family of related software intensive systems, sharing a common and managed feature set to fulfil the particular requirements of a specific domain. These are built using a common set of core assets in a predefined manner (Clements & Northrop, 2001). Software reuse is the main goal behind SPL for enhancing the quality and productivity of software products by diminishing cost and time to market.

3.2 | FM with a running example

In SPL engineering, variability and commonality are represented in the form of relationships among the features using FM. A feature is a unique element that is of interest to stakeholders. FM is a tree structure within which features are hierarchically ordered using relationships. The entire SPL is represented using root of the tree, which is mandatory to be incorporated in each valid product of PL.

Figure 2 depicts an adapted version of celular FM available in SPLOT repository. The root feature "*mobilemedia*" is a prerequisite to be incorporated in each valid product of SPL. All the features have been assigned a unique name. The cross-tree constraint relationships are also assigned names for better understanding. Therefore, the features that appear twice are assigned new unique names. To illustrate the proposed approach, 23 features and 12 cross-tree constraints have been additionally introduced in the celular FM to generate inconsistencies. The following relationships used in feature modelling are explained using celular FM as shown in Figure 2.

1. Mandatory: A child feature is mandatory if it is incorporated in all the valid products whenever its parent feature is included (Kang et al., 1990). For example, it is mandatory to choose *a1* whenever *mediamanagement* is chosen. Other mandatory features are *screensize*, *mediamanagement*, *mediaselection_1*, *mediaselection_2*, *mediaselection_3*, *screensize_1*, *screensize_2*, *screensize_3*, *setfavourites*, *viewfavourites*, *receivephoto*, *sendphoto*, *a1*, *b1*, *b3*, *b4*, *b5*, *b6*, *b7*, *b8*, *x1*, *c1*, *c2*, *c3*, *c4*, *c5*, *c6*, *c7*, *c8*, *c9*, *c10*, *c11*, *c12*, *c13*, and *c14*.
2. Optional: A child feature is optional if it may or may not be incorporated in the valid products associated with its parent feature (Kang et al., 1990). For example, it is optional to choose *copymedia* whenever *mediamanagement* is chosen. Other optional features are *screensize_3*, *favourites*, *smstransfer*, *music*, *b2*, *screen2*, and *screen3*.

[4]http://lcc.uma.es/spl/hydra

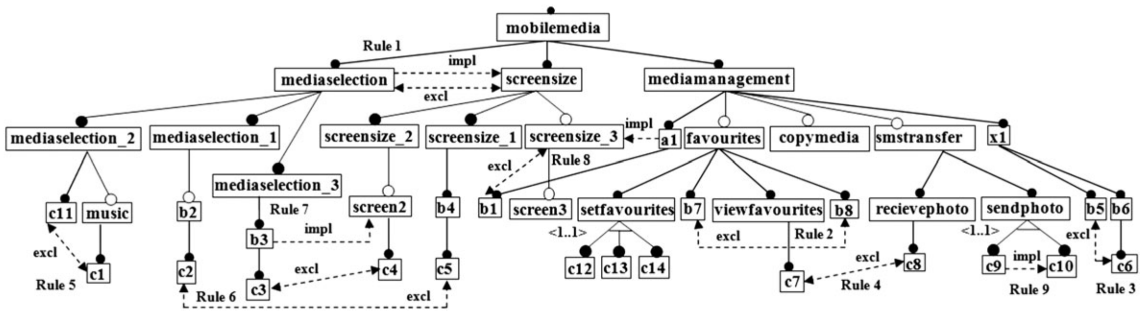


FIGURE 2 Celular feature model from SPLOT repository representing inconsistencies after applying all rules

3. Group cardinality: It is denoted by an interval <p..q>, where p and q represent the lowest and the highest count of the child features related in a group cardinality, respectively. This interval limits the count of child features, which are allowed to be selected in a product whenever their parent is incorporated (Czarnecki, Helsen, & Eisenecker, 2005). All the child features related in group cardinality must have the same parent feature. A child feature can only be included if its parent feature is selected. For instance, a photo can be sent in two ways, that is, *c9* and *c10*. There are two child features *c9* and *c10* that belong to a group cardinality <1..1> out of which only a single feature can be incorporated in a configuration.

   The following are the two cross-tree constraints:

1. Implication: The incorporation of a source feature of the implication relationship in a product implies the incorporation of its target feature in the particular product (Salinesi, Mazo, & Diaz, 2010). For instance, the implication between *mediaselection* and *screensize* represents that *mediaselection* implies *screensize*. There are four implication relationships in the celular FM.
2. Exclusion: The exclusion relationship between two features does not allow both features to be incorporated simultaneously in a valid product (Salinesi et al., 2010). For example, the exclusion relationship between *screensize_3* and *b1* represents that no product will incorporate features *screensize_3* and *b1* together. There are eight exclusion relationships in the celular FM.

3.3 | Feature model inconsistencies

Contradictory information (i.e., the information that conflicts with other information in the similar model) in an FM leads to defects due to inconsistency. An FM with inconsistency leads to an inconsistent product configuration. Thus, it is not feasible to derive a valid product from such FM. Therefore, defects due to inconsistency are considered as a critical issue, which reduce the reusability of products in an FM.

4 | PROPOSED APPROACH AND ITS IMPLEMENTATION

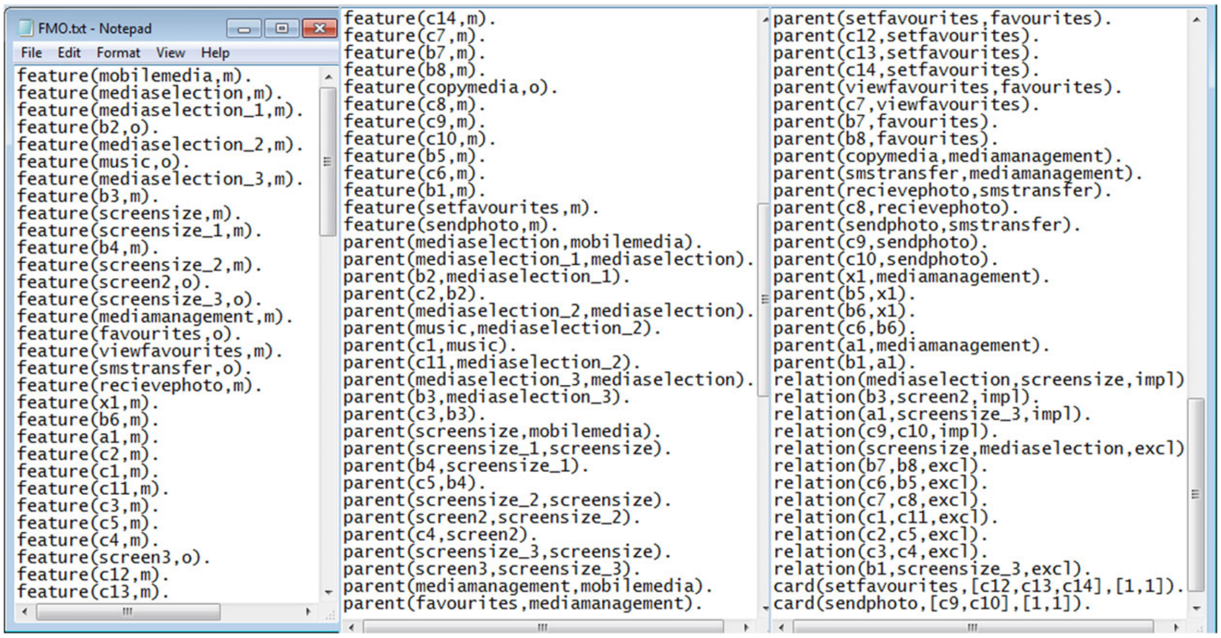
In this section, the proposed approach is used to define, identify, provide explanations for the causes, and suggest corrections for defects due to inconsistency in the presented classification. A rule set has been defined for specific cases of wrong employment of FM relationships that cause inconsistencies. The proposed rules are implemented with SWI-Prolog (Wielemaker, 2015), which is a popular and free environment of the Prolog programming language. It was developed by Jan Wielemaker in 1987 at the University of Amsterdam. In Prolog, a predicate is an elementary unit to express objects and their relationships. The implementation of the proposed approach is described using a running example of celular FM available in SPLOT repository (see Section 3.2). The implementation details are given in following subsections.

4.1 | Feature model ontology

Ontology is richer and powerful than a feature in terms of expressiveness. Like FMs, ontologies are also useful in identifying and defining the basic concepts of domain and their relationships. FO language is a predicate-based ontology language for modelling and formalizing an FM (Goldstein & Storey, 1991). It uses binary predicates to represent classes of objects and ternary predicates to represent properties (i.e., relationships among objects) (De Bruijn & Heymans, 2006).

Many researchers (Bhushan & Goel, 2016; Bhushan, Goel, & Kaur, 2017; Czarnecki, Kim, & Kalleberg, 2006; Lee et al., 2007; Sandkuhl, Thorn, & Webers, 2007) focused on ontologies for describing FMs. The ontological representation helps us in gathering useful information concerning FMs, for example, to acquire child features from a model (Abo et al., 2008). It allows verification of consistency between FM and its meta-model (Noorian et al., 2011). The concepts of FM and their semantic relationships are represented using FMO.

The construction of FMO includes a two-step model transformation. The transformation process is applied to the input FM in order to identify inconsistencies using FOL-based rules. An FM that represents the PL should be incorporated in a comprehensive formalism to enable



```
feature(mobilemedia,m).
feature(mediaselection,m).
feature(mediaselection_1,m).
feature(b2,o).
feature(mediaselection_2,m).
feature(music,o).
feature(mediaselection_3,m).
feature(b3,m).
feature(screensize,m).
feature(screensize_1,m).
feature(b4,m).
feature(screensize_2,m).
feature(screen2,o).
feature(screensize_3,o).
feature(mediamanagement,m).
feature(favourites,o).
feature(viewfavourites,m).
feature(smstransfer,o).
feature(recievephoto,m).
feature(x1,m).
feature(b6,m).
feature(a1,m).
feature(c2,m).
feature(c1,m).
feature(c11,m).
feature(c3,m).
feature(c5,m).
feature(c4,m).
feature(screen3,o).
feature(c12,m).
feature(c13,m).
feature(c14,m).
feature(c7,m).
feature(b7,m).
feature(b8,m).
feature(copymedia,o).
feature(c8,m).
feature(c9,m).
feature(c10,m).
feature(b5,m).
feature(c6,m).
feature(b1,m).
feature(setfavourites,m).
feature(sendphoto,m).
parent(mediaselection,mobilemedia).
parent(mediaselection_1,mediaselection).
parent(b2,mediaselection_1).
parent(c2,b2).
parent(mediaselection_2,mediaselection).
parent(music,mediaselection_2).
parent(c1,music).
parent(c11,mediaselection_2).
parent(mediaselection_3,mediaselection).
parent(b3,mediaselection_3).
parent(c3,b3).
parent(screensize,mobilemedia).
parent(screensize_1,screensize).
parent(b4,screensize_1).
parent(c5,b4).
parent(screensize_2,screensize).
parent(screen2,screensize_2).
parent(c4,screen2).
parent(screensize_3,screensize).
parent(screen3,screensize_3).
parent(mediamanagement,mobilemedia).
parent(favourites,mediamanagement).
parent(setfavourites,favourites).
parent(c12,setfavourites).
parent(c13,setfavourites).
parent(c14,setfavourites).
parent(viewfavourites,favourites).
parent(c7,viewfavourites).
parent(b7,favourites).
parent(b8,favourites).
parent(copymedia,mediamanagement).
parent(smstransfer,mediamanagement).
parent(recievephoto,smstransfer).
parent(c8,recievephoto).
parent(sendphoto,smstransfer).
parent(c9,sendphoto).
parent(c10,sendphoto).
parent(x1,mediamanagement).
parent(b5,x1).
parent(b6,x1).
parent(c6,b6).
parent(a1,mediamanagement).
parent(b1,a1).
relation(mediaselection,screensize,impl)
relation(b3,screen2,impl).
relation(a1,screensize_3,impl).
relation(c9,c10,impl).
relation(screensize,mediaselection,excl)
relation(b7,b8,excl).
relation(c6,b5,excl).
relation(c7,c8,excl).
relation(c1,c11,excl).
relation(c2,c5,excl).
relation(c3,c4,excl).
relation(b1,screensize_3,excl).
card(setfavourites,[c12,c13,c14],[1,1]).
card(sendphoto,[c9,c10],[1,1]).
```

FIGURE 3 Predicate-based feature model ontology of celular feature model from SPLOT repository

representation of the features and their relationships in the input model. Additionally, this formalism should enable the identification of defects due to inconsistency along with explanations for their causes. The transformation process has been explained in the following subsections using celular FM.

4.1.1 | SPLOT format to the FeatureIDE format

The celular FM is automatically read by FeatureIDE tool by importing its model from SPLOT (SXFM format). The graphical representation of the corresponding model can be obtained by using visualization functionalities of FeatureIDE. Additional cross-tree constraints and features (i.e., *a1*, *x1*, *b1*, *b2*, *b3*, *b4*, *b5*, *b6*, *b7*, *b8*, *c1*, *c2*, *c3*, *c4*, *c5*, *c6*, *c7*, *c8*, *c9*, *c10*, *c12*, *c13*, and *c14* as shown in Figure 2) are injected in the input celular FM file using model editor of FeatureIDE in order to generate defects due to inconsistency.

For a better understanding of the celular FM generated in FeatureIDE, relationships in the graphically represented model shown in Figure 2 are illustrated as follows:

1. implication relationship (*impl*) is depicted using a dashed arrow that begins with source feature and ends at target feature,
2. exclusion relationship (*excl*) is represented using a double-headed dashed arrow,
3. all the child features related in a group cardinality with the same parent feature are represented as mandatory features.

4.1.2 | FeatureIDE format to FMO (facts in Prolog)

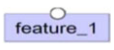
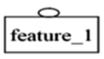
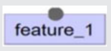
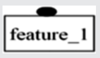
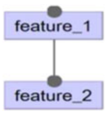
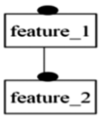
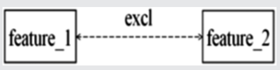
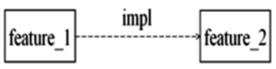
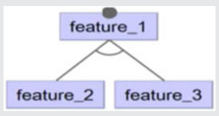
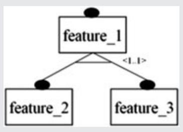
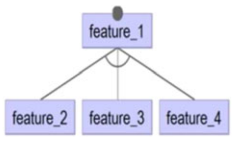
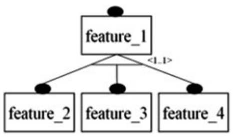
In this step, the modified input celular FM file[5] in XML format is transformed to FOL predicate-based FMO (shown in Figure 3) using an XML parser.[6] The generated ontology corresponds to the features and their relationships represented in the celular FM. The XML parser translates input XML file to ontology. This ontology is based on four types of predicates defined in Prolog, namely, feature, parent, relation, and card.

Table 2 represents the transforming patterns followed to acquire predicate-based ontology from FM. The predicates are explained using Figure 3 as follows:

feature(feature_1,o):	It represents that *feature_1* is an optional feature, for example, feature(copymedia,o).
feature(feature_1,m):	It indicates that *feature_1* is a mandatory feature, for example, feature(mediaselection,m).
parent (feature_1,feature_2):	It shows that child *feature_1* has a parent *feature_2*, for example, parent(screensize_1,screensize).
relation(feature_1,feature_2,excl):	It represents that *feature_1* excludes *feature_2* where *excl* describes an exclusion relationship, for example, relation(screensize,mediaselection,excl).

[5]http://bit.ly/2gBSnRQ
[6]http://bit.ly/2opasTs

TABLE 2 Transforming patterns from feature model into first-order logic predicate-based ontology

Name	FeatureIDE representation	Feature model representation	Predicate-based ontology representation
Optional			feature(feature_1,o)
Mandatory			feature(feature_1,m)
Parent			parent(feature_2,feature_1)
Exclusion	feature_1 ⇔ feature_2		relation(feature_1,feature_2,excl)
Implication	feature_1 ⇒ feature_2		relation(feature_1,feature_2,impl)
Group cardinality with two child features			card(feature_1,[feature_2, feature_3],[1,1])
Group cardinality with three child features			card(feature_1,[feature_2,feature_3, feature_4],[1,1])

relation(feature_1,feature_2,impl):	It represents that *feature_1* implies *feature_2* where *impl* describes an implication relationship, for example, relation(a1,screensize_3,impl).
card(feature_1,[feature_2,feature_3],[min, max]):	It indicates that both child features *feature_2* and *feature_3* are related in a <min..max> group cardinality with parent *feature_1*. It determines the minimum (min) and maximum (max) number of child features allowed to be selected in a cardinality relationship. For instance, card(sendphoto,[c9,c10],[1,1]) illustrates that single child feature can be chosen from a set of features (i.e., *c9* and *c10*) with group cardinality <1..1>.
card(feature_1,[feature_2,feature_3,feature_4],[min,max]):	Three child features, namely, *feature_2*, *feature_3*, and *feature_4* are related in a <min..max> group cardinality with parent *feature_1*. For example, card(setfavourites,[c12,c13,c14],[1,1]) describes that only one child feature can be included from a set of three child features *c12*, *c13*, and *c14*, which are related in a <1..1> group cardinality.

Ontology is applied to FM meta-modelling to provide an improved description methodology. This reason motivated the present research work to integrate them together for generating a potent FM meta-model. As shown in Figure 4, the FMO is built using FOL predicate-based ontology modelling, and conforming FM meta-model based on UML recommended by Mazo, Lopez-Herrejon, Salinesi, Diaz, and Egyed (2011). The proposed FM meta-model is depicted via ontology properties and classes. In the FMO, ontology classes and their properties are represented using meta-model classes and UML relationships, respectively. FMO represents the concepts and their relationships in FM meta-model.

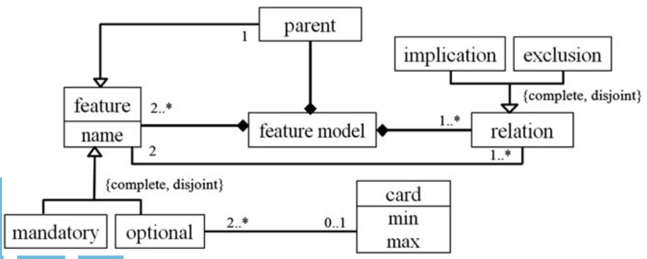


FIGURE 4 Proposed feature model meta-model

The predicate-based ontology for proposed FM meta-model is given below:

feature(name,optional).

feature(name,mandatory).

parent (name1,name2).

relation(name1,name2,implication).

relation(name1,name2,exclusion).

card(name1,[name2,name3, ...,namen],[min,max]).

The meta-model class *feature* exhibits attribute *name*, which is a string illustrating the name of a feature. It also describes that a feature can be either *optional* or *mandatory*. The class *parent* includes two attributes *name1* and *name2* where *name1* represents the name of child feature, which has parent feature *name2*. Two attributes, *name1* and *name2*, refer to the class *relation*, and these, respectively, represent the name of source and destination features intermediary in a relationship. The relationship between two features can be either *implication* or *exclusion*. The meta-model class *card* describes child features (i.e., *name2*,*name3*, ...,*namen*) related in a <*min..max*> group cardinality with parent feature (i.e., *name1*). It determines the minimum (min) and maximum (max) count of child features allowed to be selected in a cardinality relationship. The group cardinality can have two or more child features.

4.2 | Inconsistency defect identification process

It describes the process to deal with inconsistencies in SPL using an ontological rule-based approach. It is based on cross-tree constraints, that is, implication and exclusion relationships. The predicate-based FMO is a group of Prolog predicates that represents the existing facts. A rule set based on FOL is developed. This set represents nine particular cases of wrong employment of relationships among features in an FM that lead to inconsistencies. Further, FOL rules are implemented to FMO as FOL queries using SWI-Prolog. In the present classification, results include identified inconsistencies along with the explanation for their causes, which suggest corrections to the modellers. Every rule is considered as a source of the originating identified inconsistency. The proposed rules provide integration of (a) existing facts, (b) rules in FOL, (c) corresponding results after applying each rule, which describes identified defect and its causes, and (d) illustrations. This can be represented by a combined example of celular FM given in subsection 4.2.2. The following subsections illustrate the process to deal with inconsistencies in SPL.

4.2.1 | Prerequisites for identification process

The root feature is mandatory to be incorporated in every product, for example, *mobilemedia*. A parent feature can have more than one child features. Each feature has a unique name in FM. The cross-tree constraint relationships have also assigned names for a better understanding of the identification process (as mentioned in Section 4.1.1). Each feature can be either optional or mandatory where "*o*" and "*m*" represent an optional and mandatory feature, respectively.

4.2.2 | Rules for identification

Inconsistencies in FM are classified in the following cases: (a) implication and exclusion simultaneously, (b) exclusion and mandatory features, (c) exclusion and optional features, and (d) implication among alternative child features. These cases are represented using rules that include facts, a rule, and a result as given below:

Case 1. Implication and exclusion simultaneously.

In this case, Rule 1 describes a mandatory feature, which mutually excludes and implies another mandatory feature at the same time. Accordingly, it is an inconsistency as both features cannot be chosen simultaneously for configuring a valid product.

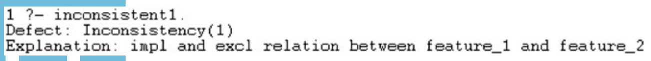
Rule 1.

Facts: feature(feature_1,m). feature(feature_2,m). relation(feature_1,feature_2,impl). relation (feature_2,feature_1,excl).

Rule: inconsistent1:-feature(F_1,m), feature(F_2,m), relation(F_1,F_2,impl), relation(F_2, F_1, sexcl),write('\nDefect: Inconsistency(1) \nExplanation: impl and excl relation between'), write(F_1), write('and'), write(F_2).

Result: Figure 5 gives the result produced by applying Rule 1.

Illustration: Figure 6(a) shows Rule 1, which depicts a mandatory feature *feature_1* implies another mandatory feature *feature_2*, while *feature_2* excludes *feature_1*. Thus, it is an inconsistency as both features cannot be incorporated for configuring a valid product.



```
1 ?- inconsistent1.
Defect: Inconsistency(1)
Explanation: impl and excl relation between feature_1 and feature_2
```

FIGURE 5 Result of Rule 1

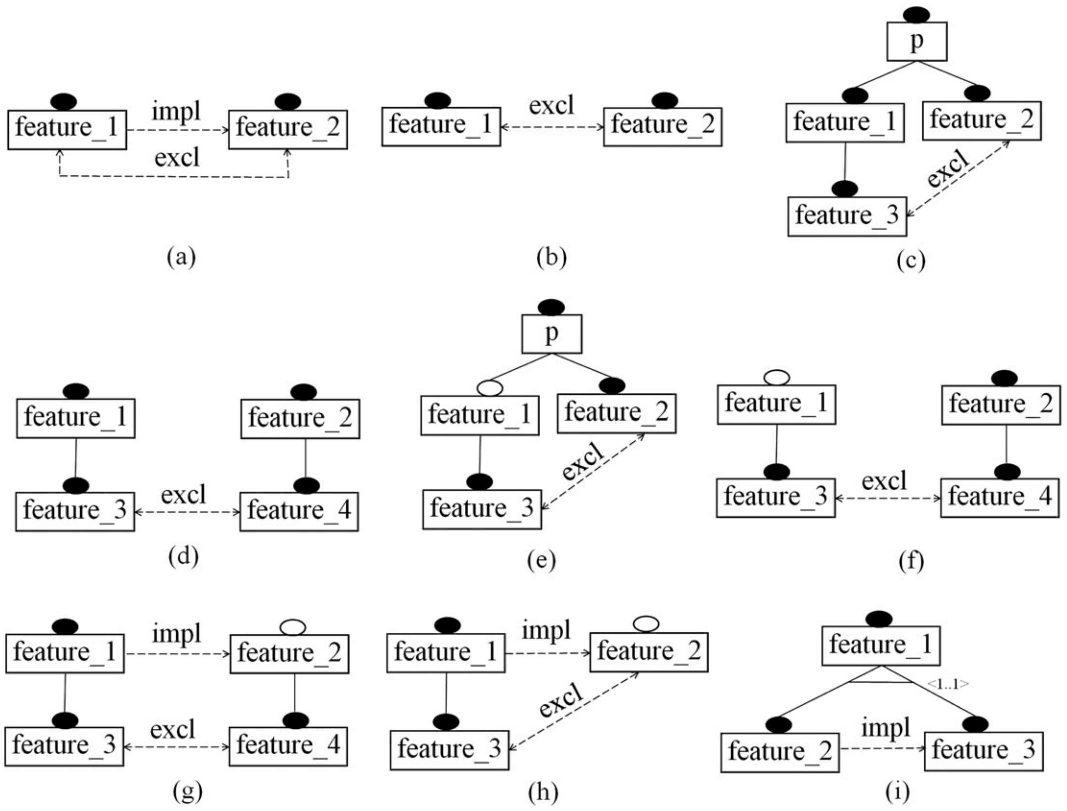


FIGURE 6 Demonstration of inconsistency defects identification rules

Case 2. Exclusion and Mandatory

This case describes a mandatory feature excludes another mandatory feature. Rules 2, 3, 4, 5, and 6 are used to identify inconsistencies in this case.

Rule 2.

Facts: feature(feature_1,m). feature(feature_2,m). relation(feature_1,feature_2,excl).

Rule: inconsistent2:-feature(F_1,m), feature(F_2,m), relation(F_1,F_2,excl), write('\nDefect: Inconsistency(2)\nExplanation: excl relation between'), write(F_1), write('and'), write(F_2).

Result: The result obtained by applying Rule 2 is shown in Figure 7.

Illustration: Figure 6(b) depicts Rule 2, which describes a mandatory feature *feature_1* excludes another mandatory feature *feature_2*. Accordingly, it is an inconsistency as both features are mutually excluded, and these features cannot be incorporated together for configuring a valid product.

Rule 3.

Facts: feature(p,m). feature(feature_1,m). feature(feature_2,m). Feature (feature_3, m). parent (feature_1,p). parent(feature_2,p). parent(feature_3,feature_1). relation(feature_2,feature_3, excl).

Rule: inconsistent3:-feature(F_1,m), feature(F_2,m), feature(F_3,m), parent(F_3,F_1), parent (F_1,P), parent(F_2,P), relation(F_2,F_3,excl), write('Defect: Inconsistency(3)\n Explanation: excl relation between'), write(F_2), write('and'), write(F_3).

Result: Figure 8 shows the result generated by implementing Rule 3.

Illustration: Figure 6(c) shows Rule 3, which explains a mandatory feature *feature_3* has a parent mandatory feature *feature_1* excludes another mandatory feature *feature_2* where *feature_1* and *feature_2* have the same parent mandatory feature *p*.

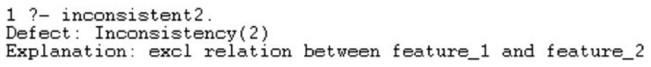

```
1 ?- inconsistent2.
Defect: Inconsistency(2)
Explanation: excl relation between feature_1 and feature_2
```

FIGURE 7 Result of Rule 2

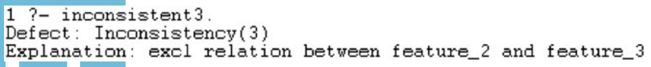

```
1 ?- inconsistent3.
Defect: Inconsistency(3)
Explanation: excl relation between feature_2 and feature_3
```

FIGURE 8 Result of Rule 3

Rule 4.

Facts: feature(feature_1,m). feature(feature_2,m). feature(feature_3,m). feature(feature_4,m). parent (feature_3,feature_1). parent(feature_4,feature_2). relation(feature_3,feature_4,excl).

Rule: inconsistent4:-feature(F_1,m), feature(F_2,m), feature(F_3,m), feature(F_4,m), parent (F_ 3,F_1), parent(F_4,F_2), relation(F_3,F_4, excl), write('\nDefect: Inconsistency(4)\n Explanation: excl relation between'), write(F_3), write('and'), write(F_4).

Result: The result produced by applying Rule 4 is given in Figure 9.

Illustration: Figure 6(d) depicts Rule 4, which elucidates a mandatory feature *feature_3* has a parent mandatory feature *feature_1* excludes another mandatory feature *feature_4* that has a parent mandatory feature *feature_2*.

Rule 5.

Facts: feature(p,m). feature(feature_1,o). feature(feature_2,m). feature(feature_3,m). parent(feature_1,p). parent(feature_2,p). parent(feature_3,feature_1). relation(feature_3,feature_2, excl).

Rule: inconsistent5:-feature(P,m), feature(F_1,o), feature(F_2,m), feature(F_3,m), parent(F_1, P), parent(F_2,P), parent(F_3,F_1), relation(F_3,F_2,excl), write('Defect: Inconsistency(5)\n Explanation: excl relation between'), write(F_3), write('and'), write(F_2).

Result: Figure 10 shows the result obtained by implementing Rule 5.

Illustration: Figure 6(e) depicts Rule 5, which describes a mandatory feature *feature_3* has a parent optional feature *feature_1* excludes another mandatory feature *feature_2* where *feature_1* and *feature_2* have the same parent mandatory feature *p*.

Rule 6.

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

Rule: inconsistent6:-feature(F_1,o), feature(F_2,m), feature(F_3,m), feature(F_4,m), parent(F_3, F_1), parent(F_4,F_2), relation(F_3,F_4, excl), write('\nDefect: Inconsistency(6)\nExplanation: excl relation between'), write(F_3), write(' and'), write(F_4).

Result: Figure 11 represents the result generated by applying Rule 6.

Illustration: Figure 6(f) shows Rule 6, which explains a mandatory feature *feature_3* has a parent optional feature *feature_1* excludes another mandatory feature *feature_4* that has a parent mandatory feature *feature_2*.

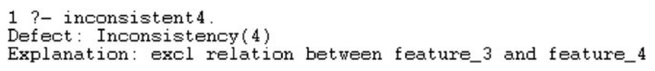
Rule 7.

Facts: feature(feature_1,m). feature(feature_2,o). feature(feature_3,m). feature(feature_4,m). parent(feature_3,feature_1). parent(feature_4, feature_2). relation(feature_1,feature_2,impl). relation(feature_3,feature_4,excl).

Rule: inconsistent7:-feature(F_1,m), feature(F_2,o), feature(F_3,m), feature(F_4,m), parent(F_3, F_1), parent(F_4,F_2), relation(F_3,F_4, excl), relation(F_1,F_2,impl), write('\nDefect: Inconsistency(7)\nExplanation: impl relation between'), write(F_1), write('and'), write(F_2), write('\nand excl relation between'), write(F_3), write('and'), write(F_4).

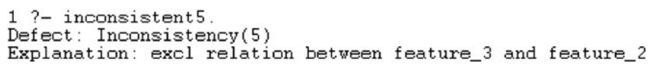
Result: The result produced by applying Rule 7 is given in Figure 12.

Illustration: Figure 6(g) depicts Rule 7, which explains a mandatory feature *feature_3* has a parent mandatory feature *feature_1* excludes another mandatory feature *feature_4* that has a parent optional feature *feature_2*, where *feature_2* is implied by *feature_1*.



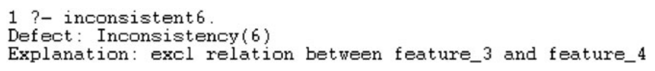
```
1 ?- inconsistent4.
Defect: Inconsistency(4)
Explanation: excl relation between feature_3 and feature_4
```

FIGURE 9 Result of Rule 4



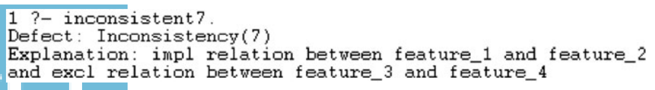
```
1 ?- inconsistent5.
Defect: Inconsistency(5)
Explanation: excl relation between feature_3 and feature_2
```

FIGURE 10 Result of Rule 5



```
1 ?- inconsistent6.
Defect: Inconsistency(6)
Explanation: excl relation between feature_3 and feature_4
```

FIGURE 11 Result of Rule 6



```
1 ?- inconsistent7.
Defect: Inconsistency(7)
Explanation: impl relation between feature_1 and feature_2
and excl relation between feature_3 and feature_4
```

FIGURE 12 Result of Rule 7

Case 3. Exclusion and Optional.

In this case, a mandatory feature excludes an optional feature. Rule 8 is used to identify defect due to inconsistency.

Rule 8.

Facts: feature(feature_1,m). feature(feature_2,o). feature(feature_3,m). parent(feature_3,feature _1). relation(feature_1,feature_2,impl). relation(feature_3,feature_2,excl).

Rule: inconsistent8:-feature(F_1,m), feature(F_2,o), feature(F_3,m), parent(F_3,F_1), relation(F _1,F_2,impl), relation(F_3,F_2,excl), write('\nDefect: Inconsistency(8)\nExplanation: impl relation between'), write(F_1), write('and'), write(F_2), write('\nand excl relation between'), write(F_3), write('and'), write(F_2).

Result: Figure 13 shows the result obtained by implementing Rule 8.

Illustration: Figure 6(h) depicts Rule 8, which describes a mandatory feature *feature_3* has a parent mandatory feature *feature_1* excludes an optional feature *feature_2*, where *feature_1* implies *feature_2*.

Case 4. Implication among alternative child features.

This case describes implication relationship between two child features grouped in a cardinality. Rule 9 is used to identify defect due to inconsistency.

Rule 9.

Facts: feature(feature_1,m). feature(feature_2,m). feature(feature_3,m). card(feature_1,[feature _2, feature_3],[1,1]). relation(feature_2,feature_3,impl).

Rule: inconsistent9:-feature(F_1,m), feature(F_2,m), feature(F_3,m), card(F_1,[F_2,F_3],[1,1]), relation(F_2,F_3,impl), write('\nDefect: Inconsistency(9)\nExplanation: impl relation between'), write(F_2), write('and'), write(F_3).

Result: The result generated by applying Rule 9 is shown in Figure 14.

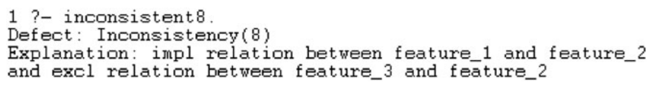
Illustration: Figure 6(i) shows Rule 9, which describes *feature_2* implies *feature_3*, and these features belong to the group cardinality <1..1> with a mandatory parent *feature_1*. The implication relationship among the child features exceeds the upper bound of the group cardinality <1..1> and does not allow the selection of one sub-feature.

Results obtained after applying all the above mentioned nine rules (see Section 4.2.2) to celular FMO are shown in Figure 15. These results suggest modellers to alter or delete relationships related to the source of inconsistencies for resolving defects.

The flow chart shown in Figure 16 provides step-by-step details of the proposed approach, and also, a detailed description is explained in this section.

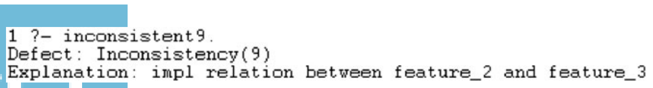
5 | PRELIMINARY EVALUATION

The accuracy as well as scalability of the proposed approach was evaluated using 26 models, out of which 10 models were standard cases for evaluating PL methodologies (Felfernig et al., 2013; Segura et al., 2010; Wang et al., 2010), 11 were taken from SPLOT repository, and 5 were generated using FeatureIDE tool. The models used from the SPLOT repository ranged from 17 to 366 features with cross-tree constraints ratio (CTCR) varying between 0% and 93%, and cross-tree constraints clause density varying from 0.5 to 0.8. CTCR represents a fraction of features in the FM, which participate in cross-tree constraints. However, the four models did not have any cross-tree constraints (0% CTCR) and clause density. The five models included constraints involving two features, and two models had constraints relating eight features, which represented more complex relations. The eight models included features involving mandatory, optional, and group cardinality, and only one model included mandatory and optional features, whereas two models incorporated features involving mandatory and group cardinality.



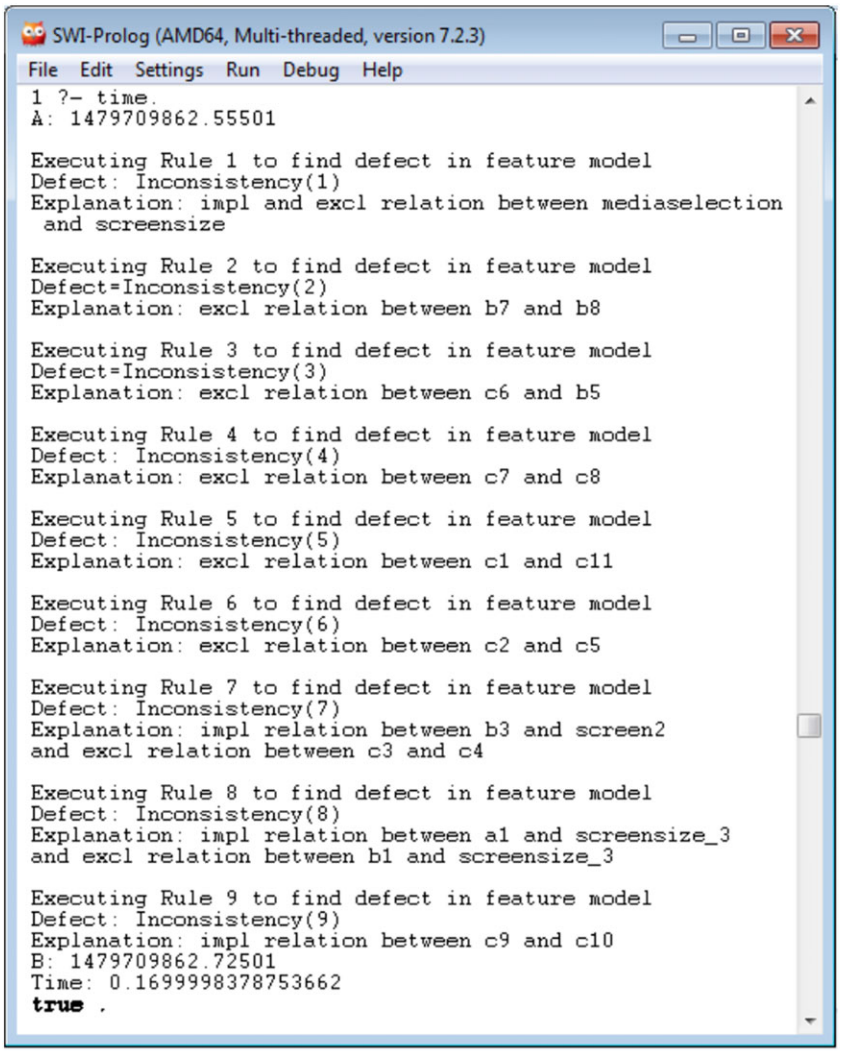
```
1 ?- inconsistent8.
Defect: Inconsistency(8)
Explanation: impl relation between feature_1 and feature_2
and excl relation between feature_3 and feature_2
```

FIGURE 13 Result of Rule 8



```
1 ?- inconsistent9.
Defect: Inconsistency(9)
Explanation: impl relation between feature_2 and feature_3
```

FIGURE 14 Result of Rule 9


SWI-Prolog (AMD64, Multi-threaded, version 7.2.3)
File Edit Settings Run Debug Help

```
1 ?- time.
A: 1479709862.55501

Executing Rule 1 to find defect in feature model
Defect: Inconsistency(1)
Explanation: impl and excl relation between mediaselection
 and screensize

Executing Rule 2 to find defect in feature model
Defect=Inconsistency(2)
Explanation: excl relation between b7 and b8

Executing Rule 3 to find defect in feature model
Defect=Inconsistency(3)
Explanation: excl relation between c6 and b5

Executing Rule 4 to find defect in feature model
Defect: Inconsistency(4)
Explanation: excl relation between c7 and c8

Executing Rule 5 to find defect in feature model
Defect: Inconsistency(5)
Explanation: excl relation between c1 and c11

Executing Rule 6 to find defect in feature model
Defect: Inconsistency(6)
Explanation: excl relation between c2 and c5

Executing Rule 7 to find defect in feature model
Defect: Inconsistency(7)
Explanation: impl relation between b3 and screen2
and excl relation between c3 and c4

Executing Rule 8 to find defect in feature model
Defect: Inconsistency(8)
Explanation: impl relation between a1 and screensize_3
and excl relation between b1 and screensize_3

Executing Rule 9 to find defect in feature model
Defect: Inconsistency(9)
Explanation: impl relation between c9 and c10
B: 1479709862.72501
Time: 0.1699998378753662
true .
```

FIGURE 15 Results generated after implementing all rules to celular feature model from SPLOT repository

5.1 | Evaluation environment

A system with Windows 8.1 of 64 bits, processor Intel(R) Core(TM) i7-3630QM, CPU 2.40 GHz, RAM memory of 8.00 GB, SPLOT repository, Eclipse 4.2, FeatureIDE 2.7.2, and SWI-Prolog (multi-threaded, 64 bits, version 7.2.3) were used for the evaluation of our proposed approach.

5.2 | Accuracy

The evaluation of our approach was carried out over 26 FMs of discrete sizes from 10 to 5,543 features. As shown in Table 3, the results include identifying inconsistency defects along with their causes in all FMs. Table 4 shows the results obtained after analysing celular FM (a) using the Thüm, Kästner, et al. (2014) approach (column 3) and (b) using the proposed approach (column 4).

The *Status (S)* represents accuracy, where

i. 0 indicates that the approach used by Thüm, Kästner, et al. (2014) is unable to find constraints involved in the cause of defect (i.e., identified dead feature or false optional feature or did not identify any defect)
ii. 0.5 indicates that out of two constraints involved in the defect, the approach used by Thüm, Kästner, et al. (2014) only identified one constraint involved in the cause of defect
iii. 1 indicates that their method finds accurate constraints for the cause of defect

$$\text{accuracy} = \frac{1}{m}\sum^{m=126}\left(\frac{1}{r}\sum_{r=1}^{9} S_{m,r}\right), \quad (1)$$

where m is the number of models, r is the number of rules, and $S_{m,r}$ represents the status and $S_{m,r} \in [0,1]$.

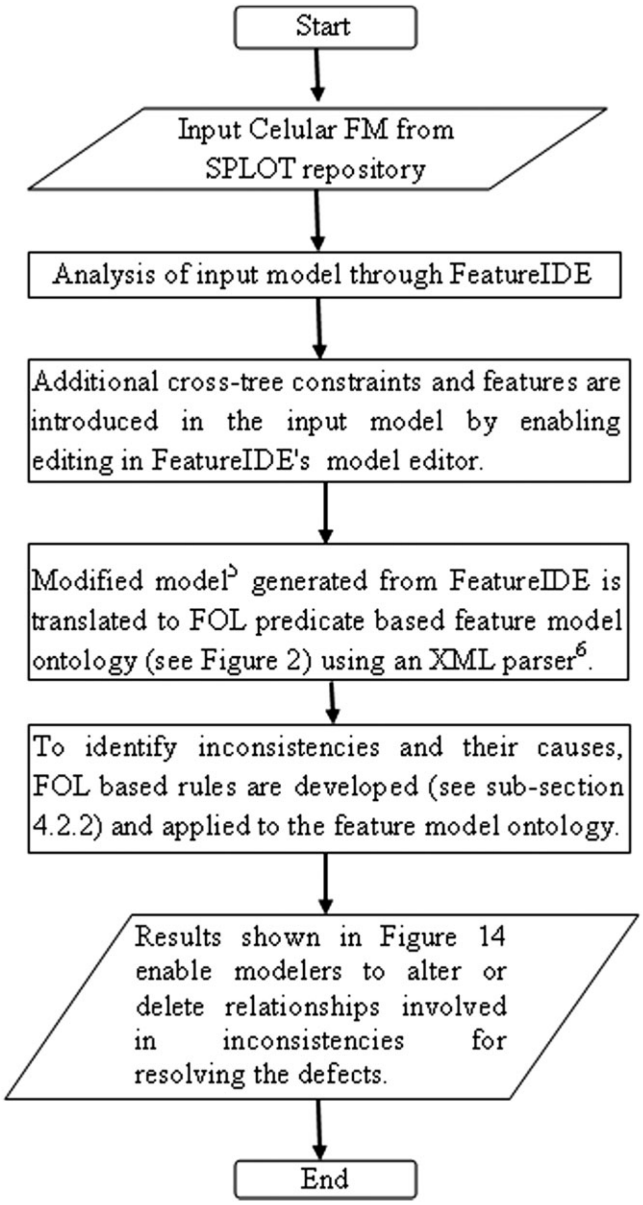

Start
Input Celular FM from SPLOT repository
Analysis of input model through FeatureIDE
Additional cross-tree constraints and features are introduced in the input model by enabling editing in FeatureIDE's model editor.
Modified model5 generated from FeatureIDE is translated to FOL predicate based feature model ontology (see Figure 2) using an XML parser6.
To identify inconsistencies and their causes, FOL based rules are developed (see sub-section 4.2.2) and applied to the feature model ontology.
Results shown in Figure 14 enable modelers to alter or delete relationships involved in inconsistencies for resolving the defects.
End

FIGURE 16 Flow chart of the proposed approach

Table 4 demonstrates the calculation of *S* for one model, namely, celular FM. Similarly, all remaining 25 models were also analysed using the Thüm, Kästner, et al. (2014) approach in the same way. As illustrated in Equation 1, the value of Status ($S_{m,r}$) was calculated for each of the considered nine rules, which further were executed for all of the 26 models. The final accuracy of the Thüm, Kästner, et al. (2014) approach is 69.96%, which was computed on the basis of the average of the values as shown in Equation 1. However, the proposed approach identified 100% of the inconsistency defects and their explanations with 0% false positives. Additionally, all nine rules individually and jointly identified and provided explanations of all the expected defects, indicating that the proposed approach is 100% accurate.

5.3 | Scalability

The performance of our approach was tested by calculating the average execution time (in seconds) after executing all nine rules (refer to Section 4.2.2) on 26 FMs with size from 10 to 5,543 features, for identifying inconsistency defects with their explanations as shown in Figure 17. The time is represented on the *x*-axis, and the number of features of all models is represented on the *y*-axis. Results indicate that the queries took 4.8 s on large-sized FMs with 5,543 features.

To obtain the reliable measures of execution time, each rule was executed five times for each of the 26 models, that is, 1,170 (9*26*5) queries in total were executed. The average of five execution time for each of the nine rules on each model is presented as time in the

TABLE 3 Defects due to inconsistency and corresponding execution time for their identification in feature models

Case study	OF	AF	TNF	NI	NE	NIC	Rules
Chat	10	0	10	1	1	2	5, 9
E-shop2	11	0	11	2	2	3	4,8,9
Car	9	7	16	1	3	4	4, 5, 6, 9
FM (faulty elements)	10	7	17	4	4	5	1, 2, 7, 8, 9
Mobile media	16	2	18	2	3	4	1, 2, 5, 9
E-shop1	27	3	30	3	3	4	1, 5, 7, 9
Tourist guide	12	19	31	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Mobile phone	8	24	32	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Mobile agent	17	20	37	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Graph product line	16	22	38	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Celular	17	19	39	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Bike-sharing systems	33	9	42	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Bcmscontext	35	11	46	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Experiment environment	35	24	59	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Car selection	72	10	82	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Reuso—UFRJ—Eclipse1	72	11	83	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Model_Transformation	88	10	98	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
BattleofTanks	144	20	164	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
BankingSoftware	176	6	182	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Printers	172	13	185	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
A model for decision-making for investments on enterprise information systems	366	14	380	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Model with 1,539 features	1,500	39	1,539	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Model with 2,562 features	2,500	62	2,562	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Model with 3,551 features	3,500	51	3,551	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Model with 4,538 features	4,500	38	4,538	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9
Model with 5,543 features	5,500	43	5,543	4	8	9	1, 2, 3, 4, 5, 6, 7, 8, 9

Note. AF = additional features; FM = feature model; NE = no. of exclusions; NI = no. of implications; NIC = no. of inconsistencies; OF = original features; TNF = total no. of features.

TABLE 4 Comparing accuracy of the Thüm, Kästner, et al. (2014) approach with the proposed approach using celular feature model

Rules	Constraints description	Thüm, Kästner, et al. (2014) approach	Proposed approach	Status (*S*)
Rule 1	(a) mediaselection implies screensize (b) screensize excludes mediaselection	(a) Redundant constraint (b) Redundant constraint	Defect: Inconsistency(1) Explanation: impl and excl relation between mediaselection and screensize	1
Rule 2	(a) b7 excludes b8	(a) Redundant constraint	Defect: Inconsistency(2) Explanation: excl relation between b7 and b8	1
Rule 3	(a) c6 excludes b5	(a) Redundant constraint	Defect: Inconsistency(3) Explanation: excl relation between c6 and b5	1
Rule 4	(a) c7 excludes c8	(a) No defect/nil	Defect: Inconsistency(4) Explanation: excl relation between c7 and c8	0
Rule 5	(a) c1 excludes c11	(a) Music is false optional feature	Defect: Inconsistency(5) Explanation: excl relation between c1 and c11	0
Rule 6	(a) c2 excludes c5	(a) b2 is false optional feature	Defect: Inconsistency(6) Explanation: excl relation between c2 and c5	0
Rule 7	(a) b3 implies screen2 (b) c3 excludes c4	(a) Screen2 is false optional feature (b) Redundant constraint	Defect: Inconsistency(7) Explanation: impl relation between b3 and screen2 and excl relation between c3 and c4	0.5
Rule 8	(a) a1 implies screensize_3 (b) b1 excludes screensize_3	(a) Redundant constraint (b) Redundant constraint	Defect: Inconsistency(8) Explanation: impl relation between a1 and screensize_3 and excl relation between b1 and screensize_3	1
Rule 9	(a) c9 implies c10	(a) c9 is dead feature	Defect: Inconsistency(9) Explanation: impl relation between c9 and c10	0

Note. False optional feature is a feature defined as optional but exists in each valid product of the product line; dead feature is a feature that cannot be chosen in any valid product of the product line.

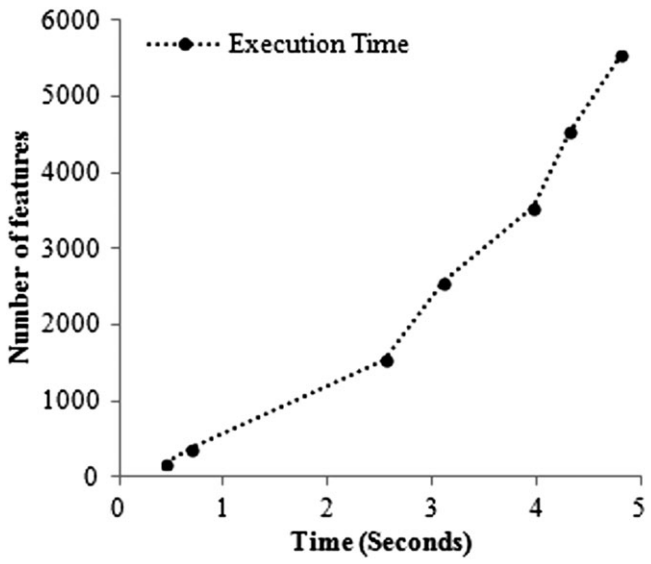


FIGURE 17 Execution time, for all nine rules, per number of features

proposed work (i.e., 234 combined results). Figure 15 shows the calculated scalability of the running model used. Felfernig et al. (2013) stated that the execution time for evaluating the developed algorithms (to deal with inconsistencies) with 172 features is 10.54 s for FASTDIAG and is more than 100 s for Hitting Set Directed Acyclic Graph. Meanwhile, the execution time for the debugging process with 31 features is 2.45 min in Noorian et al. (2011). Table 5 compares the approach given by Wang et al. (2010) with the proposed approach on the basis of execution time required to deal with inconsistencies. By considering the statistics of the discussed approaches, the proposed approach handled inconsistencies in FMs up to 5,543 features in 4.8 s. It demonstrates attainment of better scalability and that too in a very reasonable execution time.

Each rule can be checked using Spearman rank correlation coefficient and integral square error as shown in Table 6. These are calculated between the number of features in the FMs and the execution time for each rule. Spearman rank correlation coefficient provided good suggestions of their dependency or independency, and it is closer to 0 in the case of Rule 2. Although the complexity of this type of problem related to predicate-based FMO is nondeterministic polynomial time, the rules appeared to be scalable with large-sized FMs by the application of the proposed method. The best-suited rule was determined by integral square error criteria. It was calculated using a curve fitting technique. Table 6 shows that Rules 7 and 9 have minimum errors, which turned it to be the best prediction for this approach. Each rule can be examined in a polynomial time (1, 2, 3, 5, 6, 7, 8, and 9 rules with a 5th degree polynomial) and a linear time (Rule 5 with linear) because our approach evaluated the features concerned with the rule only, instead of the entire model.

TABLE 5 Comparison of the Wang et al. (2010) approach with the proposed approach

	Number of features and constraints	Time (s)	
		Wang et al. (2010)	Proposed approach
(a)	3,751features 290 constraints	>56 (FMs with the same priorities) >25 (FMs with the different priorities)	4.01
(b)	4,433 features 330 constraints	>64 (FMs with the different priorities)	4.22

Note. FM = feature model.

TABLE 6 Spearman rank correlation coefficients and ISEs per each rule over 15 feature models

Rules	1	2	3	4	5	6	7	8	9
Spearman rank correlation coefficient	0.68	0.26	0.68	0.59	0.73	0.57	0.97	0.77	0.94
ISE	29,990.2	38,347	32,484.9	26,577.3	25,518.4	27,399.1	705.73	19,039.1	1,614.73

Note. ISE = integral square error.

6 | CONCLUSION AND FUTURE DIRECTIONS

Inconsistency in FMs is a critical problem that hinders the derivation of high-quality software products in SPL. In this paper, an ontological rule-based approach is proposed to analyse defects due to inconsistency in FMs. The classification of inconsistencies is defined in the form of cases. FOL predicate-based FMO is constructed to represent FM as it improves expressivity of the model as well as provides formalization. FOL-based rules are developed, which identify defects due to inconsistency in FMs, explain their causes, and suggest corrections to resolve defects in the given classification. These rules are implemented individually and all together in Prolog. The developed approach is validated using the maximum size of real-world FMs available in the SPLOT repository and the models generated using FeatureIDE (i.e., up to 5,543 features). It is found to be 100% accurate, efficient, and scalable.

To our knowledge, the proposed approach is unique in dealing with inconsistencies that not only identified defects but also explained their causes and suggested corrective solutions to fix them using predicate-based FMO together with FOL rules. The presented approach also handled inconsistencies in cardinality-based FMs. Moreover, the explanation for the cause of defects is an improvement on earlier work by Zhang and Møller-Pedersen (2013), Yang and Dong (2013), Asadi et al. (2014), and Thüm, Meinicke, et al. (2014), where inconsistency detection is done, but no explanation for their causes is given. The use of natural language for explaining the cause of defects due to inconsistency helps modellers to detect the incorrect relationships that lead to these inconsistencies. It enables correction of defects with much ease and clear interpretation, in order to create well-formed models for producing defect-free end products, thereby improving the quality of SPL.

The proposed approach is limited in dealing with inconsistencies for the prespecified cases and rules. These cases and rules may vary according to other FM notations (for instance, extended FMs) for handling defects, and therefore, the existing cases and set of rules could be broadened by adding more cases and rules correspondingly. Further, Semantic Web Rule Language-based rules can also be developed for the auto identification and correction of inconsistencies. The proposed approach only supports inconsistencies due to cross-tree constraints. Other defects caused by the misuse of features, attributes, and so forth are not yet considered. It only suggests deleting relationships involved in the cause of defect to fix defects, but it may be helpful to develop a method for identifying corrections for defects based on modifying the model (i.e., by adding new relationships or features to the model). The proposed approach can be evaluated with randomly generated large FMs (with over thousands of features), which may improve scalability of the approach. Additionally, our future line of research is to reduce the execution time to increase the efficiency. There is a need for developing a tool to identify the causes and corrections for the identified inconsistencies.

ACKNOWLEDGEMENTS

This work is funded in part by a grant from the University Grants Commission (UGC), New Delhi, Government of India, Rajiv Gandhi National Fellowship (grant F117.1/201415/RGNF201415SCJAM66324). We would like to thank Vikram Jeet Singh, Research Scholar, Department of Computer Science, Himachal Pradesh University, Shimla, India.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest regarding the publication of this manuscript.

ORCID

Megha Bhushan http://orcid.org/0000-0003-4309-875X

REFERENCES

Abo, L., Houben, G., De Troyer, O., & 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), Germany.

Afzal, U., Mahmood, T., Rauf, I., & Shaikh, Z. A. (2014). Minimizing feature model inconsistencies in software product lines. In Proceedings of the Multi-Topic Conference (INMIC), 2014 IEEE 17th International, IEEE, 137–142. https://doi.org/10.1109/INMIC.2014.7097326

Al Balushi, T. H., Sampaio, P. R. F., & Loucopoulos, P. (2013). Eliciting and prioritizing quality requirements supported by ontologies: A case study using the ElicitO framework and tool. *Expert Systems*, *30*(2), 129–151. https://doi.org/10.1111/j.1468-0394.2012.00625.x

Àlvez, J., Lucio, P., & Rigau, G. (2012). Adimen-SUMO: Reengineering an ontology for first-order reasoning. *Journal on Semantic Web and Information Systems*, *8*(4), 80–116. https://doi.org/10.4018/jswis.2012100105

Ardis, M., Daley, N., Hoffman, D., Siy, H., & Weiss, D. (2000). Software product lines: A case study. *Software-Practice and Experience*, *30*(7), 825–847. https://doi.org/10.1002/(SICI)1097-024X(200006)30:7%3C825::AID-SPE322%3E3.0.CO;2-1

Asadi, M., Gröner, G., Mohabbati, B., & Gašević, D. (2014). Goal-oriented modeling and verification of feature-oriented product lines. *Software & Systems Modeling*, *15*, 257–279. https://doi.org/10.1007/s10270-014-04028

Baumgartner, P., & Suchanek, F.M. (2006). Automated reasoning support for first-order ontologies. In Proceedings of the International Workshop on Principles and practice of semantic web reasoning, Springer Berlin Heidelberg, 18–32.

Berardi, D., Grüninger, M., Hull, R., & Mcilraith, S. (2004). Towards a first-order ontology for semantic web services. In Proceedings of the W3C Workshop on Constraints and Capabilities for Web Services.

Bernardo, M., Ciancarini, P., & Donatiello, L. (2002). Architecting families of software systems with process algebras. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *11*(4), 386–426. https://doi.org/10.1145/606612.606614

Bhushan, M., & Goel, S. (2016). Improving software product line using an ontological approach. *Sādhanā*, *41*(12), 1381–1391. https://doi.org/10.1007/s12046-016-0571-y

Bhushan, M., Goel, S., & Kaur, K. (2017). Analyzing inconsistencies in software product lines using an ontological rule-based approach. *Journal of Systems and Software*. ISSN 0164–1212. https://doi.org/10.1016/j.jss.2017.06.002

Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., & Grose, T. J. (2003). *Eclipse Modeling Framework*. Reading: Addison Wesley Professional.

Clements, P., & Northrop, L. M. (2001). *Software product lines: Practices and patterns*. Addison-Wesley Professional.

Czarnecki, K., Helsen, S., & Eisenecker, U. (2005). Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, *10*(1), 7–29. https://doi.org/10.1002/spip.213

Czarnecki, K., Kim, C. H. P., & Kalleberg, K. T. (2006). Feature models are views on ontologies. In Proceedings of the 10th International on Software Product Line Conference, IEEE Computer Society, 41–51.

De Bruijn, J., & 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, Athens, IEEE, 7–16. https://doi.org/10.1109/RULEML.2006.23.

Deng, G., Lenz, G., & Schmidt, D. C. (2006). Addressing domain evolution challenges in software product lines. *Lecture Notes in Computer Science*, *3844*, 247–261. https://doi.org/10.1007/11663430_26

D'Souza, D., Gopinathan, M., Ramesh, S., & Sampath, P. (2010). Conflict-tolerant specifications for hybrid systems.

Elfaki, A., Phon-Amnuaisuk, S., & Ho, C. K. (2009). Investigating inconsistency detection as a validation operation in software product line. *Studies in Computational Intelligence*, *Springer*, *253*, 159–168. https://doi.org/10.1007/978-3-642-05441-9_14

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

Fan, S., & Naixiao, Z. (2006). Feature model based on description logics. *Knowledge-Based Intelligent Information and Engineering Systems*, *Springer*, *4252*, 1144–1151. https://doi.org/10.1007/11893004_145

Felfernig, A., Benavides, D., Galindo, J., & Reinfrank, F. (2013). Towards anomaly explanations in feature models. In Proceedings of the 15th International Configuration Workshop (ConfWS-2013), Vienna, Austria, 117–124. https://doi.org/10.1.1.428.5517

Gamez, N., & Fuentes, L. (2013). Architectural evolution of FamiWare using cardinality-based feature models. *Information and Software Technology*, *56*(3), 563–580. https://doi.org/10.1016/j.infsof.2012.06.012

Gangemi, A., N. Guarino, C. Masolo, A. Oltramari, & Schneider, L. (2002). Sweetening ontologies with DOLCE. In Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW 2002), Springer Berlin Heidelberg, 166–181.

Goldstein, R. C., & Storey, V. C. (1991) Database and expert systems applications. In Proceedings of the International Conference in Berlin, Federal Republic of Germany: Springer, Vienna Wien GmbH, 124–129. https://doi.org/10.1007/978-3-7091-7555-2_21.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, *6*(2), 199–220. https://doi.org/10.1006/knac.1993.1008

Gruninger, M., Hull, R., & Mcilraith, S. A. (2008). A short overview of flows: A first-order logic ontology for web services. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, *31*(3), 3–7.

Gruninger, M., & Menzel, C. (2003). The process specification language (PSL) theory and applications. *AI Magazine*, *24*(3), 63–74.

Guo, J., Wang, Y., Trinidad, P., & Benavides, D. (2012). Consistency maintenance for evolving feature models. *Expert Systems with Applications*, *39*(5), 4987–4998. https://doi.org/10.1016/j.eswa.2011.10.014

Hemakumar, A. (2008). Finding contradictions in feature models. In S. Thiel, & K. Pohl (Eds.), (pp. 183–190). Ireland: SPLC (2), Lero Int. Science Centre, University of Limerick.

Jarzabek, S., Ong, W. C., & Zhang, H. (2003). Handling variant requirements in domain modeling. *The Journal of Systems and Software*, *68*(3), 171–182. https://doi.org/10.1016/S0164-1212(03)00060-8

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

Lee, S., Kim, J., Song, C., & 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, Busan: IEEE, 727–734. https://doi.org/10.1109/SERA.2007.41

Leich, T., Apel, S., Marnitz, L., & Saake, G. (2005). Tool support for feature-oriented software development: FeatureIDE: An eclipse-based approach. In Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange (eclipse'05), ACM, 55–59. https://doi.org/10.1145/1117696.1117708

Matuszek, C., Cabral, J., Witbrock, M., & Deoliveira, J. (2006). An introduction to the syntax and content of Cyc. In Proceedings of the AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, 44–49.

Mazo, R., Lopez-Herrejon, R., Salinesi, C., Diaz, D., & 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), Munich: IEEE, 456–465. https://doi.org/10.1109/COMPSAC.2011.66

Mendonca, M., Branco, M., & Cowan, D. (2009). S.P.L.O.T.: Software product lines online tools. In Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications (OOPSLA '09), ACM, New York, USA, 761–762. https://doi.org/10.1145/1639950.1640002

Niles, I., & Pease, A. (2001). Towards a standard upper ontology. In Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS 2001), ACM, 2–9.

Niu, N., Savolainen, J., & Yu, Y. (2010). Variability modeling for product line viewpoints integration. In Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference (COMPSAC), IEEE, 337–346.

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

Northrop L. M. (2008). *Software product lines essentials*. Pittsburgh: Software Engineering Institute Carnegie Mellon University.

Osman, A., Phon-Amnuaisuk, S., & 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, 217–225.

Pease, A., & Sutcliffe, G. (2007). First-order reasoning on a large ontology. In Proceedings of the Workshop on Empirically Successful Automated Reasoning in Large Theories (CADE-21), 257, 59–69.

Pohl, K., Böckle, G., & van der Linden, F. J. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. New York, USA: Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/3-540-28901-1

Prehofer, C. (2001). Feature-oriented programming: A new way of object composition. *Concurrency and Computation: Practice and Experience*, *13*(6), 465–501. https://doi.org/10.1002/cpe.583

Ramachandran, D., Reagan, P., & Goolsbey, K. (2005). First-orderized ResearchCyc: Expressivity and efficiency in a common-sense ontology. In P. Shvaiko, J. Euzenat, A. Leger, D. L. McGuinness, & H. Wache (Eds.), in *Proceedings of the AAAI 2005 Workshop on Contexts and Ontologies: Theory, Practice and Applications, Pittsburgh, Pennsylvania, USA* (pp. 33–40). Menlo Park, California: Technical report WS-05-01, AAAI Press.

Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, *32*(1), 57–95. https://doi.org/10.10%2016/0004-3702(87)90062-2

Salinesi, C., Mazo, R., & 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 D'ecision), 293–308.

Sandkuhl, K., Thorn, C., & Webers, W. (2007). Enterprise ontology and feature model integration – Approach and experiences from an industrial case. In *Proceedings of the Second International Conference on Software and Data Technologies, ICSOFT (PL/DPS/KE/MUSE)* (pp. 264–269). Barcelona, Spain: INSTICC Press.

Schneider, M., Carroll, J., Herman, I., & Patel-Schneider, P. F. (2009). OWL 2 web ontology language: RDF-based semantics. W3C Recommendation.

Schneider, M., & Sutcliffe, G. (2011). Reasoning in the OWL2 full ontology language using first-order automated theorem proving. In Proceedings of the 23rd International Conference on Automated Deduction (CADE-23), Springer Berlin Heidelberg, 461–475.

Segura, S., Benavides, D., & Ruiz-Cortés, A. (2010). FaMa Test Suite v1.2. Technical Report ISA-10-TR-01, Applied Software Engineering Research Group, University of Seville, Spain.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoned. *Web Semantics: Science, Services and Agents on the World Wide Web*, *5*(2), 51–53. https://doi.org/10.1016/j.websem.2007.03.004

Stephan, M., & Antkiewicz, M. (2008). Ecore.fmp: A tool for editing and instantiating class models as feature models. Technical Report, University of Waterloo.

Thao, C. (2012). *A configuration management system for software product lines*. Milwaukee: PhD dissertation, University of Wisconsin.

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., & Leich, T. (2014). FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming*, *79*, 70–85. https://doi.org/10.1016/j.scico.2012.06.002

Thüm, T., Meinicke, J., Benduhn, F., Hentschel, M., von Rhein, A., & Saake, G. (2014). Potential synergies of theorem proving and model checking for software product lines. In Proceedings of the 18th International Software Product Line Conference (SPLC '14), Florence, Italy, 177–186. https://doi.org/10.1145/2648511.2648530

Tsang, E. (1993). *Foundations of constraint satisfaction: The classic text*. London: Academic Press.

Tsarkov, D., Riazanov, A., Bechhofer, S., & Horrocks, I. (2004). Using vampire to reason with OWL. In S. A. McIlraith, D. Plexousakis, & F. van Harmelen (Eds.), in *Proceedings of the 3rd International Semantic Web Conference (ISWC 2004). Lecture Notes in Computer Science* 3298 (pp. 471–485). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-540-30475-3_33

Van Gurp, J., & Prehofer, C. (2008). From SPLs to open, compositional platforms. In Dagstuhl Seminar, Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Wang, B., Xiong, Y., Hu, Z., Zhao, H., Zhang, W., & Mei, H. (2010). A dynamic-priority based approach to fixing inconsistent feature models. In D. C. Petriu, N. Rouquette, & Ø. Haugen (Eds.), in *Proceedings of the 13th international conference on Model Driven Engineering Languages and Systems: Part I (MODELS'10)* (pp. 181–195). Oslo, Norway: Springer-Verlag. https://doi.org/10.1007/978-3-642-16145-2_13

Wang, H. H., Li, Y. F., Sun, J., Zhang, H., & Pan, J. (2007). Verifying feature models using OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, *6*(2), 117–129. https://doi.org/10.1016/j.websem.2006.11.006

White, J., Benavides, D., Schmidt, D. C., Trinidad, P., Dougherty, B., & Ruiz-Cortés, A. (2010). Automated diagnosis of feature model configurations. *Journal of Systems and Software*, *83*(7), 1094–1107. https://doi.org/10.1016/j.jss.2010.02.017

White, J., Dougherty, B., Schmidt, D. C., & Benavides, D. (2009). Automated reasoning for multi-step feature model configuration problems. In *Proceedings of the 13th International Software Product Line Conference (SPLC '09)* (pp. 11–20). Pittsburgh, PA, USA: Carnegie Mellon University.

White, J., Galindo, J. A., Saxena, T., Dougherty, B., Benavides, D., & Schmidt, D. C. (2014). Evolving feature model configurations in software product lines. *Journal of Systems and Software*, *87*, 119–136. https://doi.org/10.1016/j.jss.2013.10.010

Wielemaker, J. (2015). SWI-Prolog (version 7.2.3), free software, Amsterdam, VU University Amsterdam, University of Amsterdam. Available at: http://www.swi-prolog.org

Yang, D., & Dong, M. (2013). Applying constraint satisfaction approach to solve product configuration problems with cardinality-based configuration rules. *Journal of Intelligent Manufacturing*, *24*(1), 99–111. https://doi.org/10.1007/s10845-011-0544-2

Zhang, X., & Møller-Pedersen, B. (2013). Towards correct product derivation in model-driven product lines. *System Analysis and Modeling: Theory and Practice, Lecture Notes in Computer Science*, *7744*, 179–197. https://doi.org/10.1007/978-3-642-36757-1_11

Megha Bhushan obtained her B.Tech. in Information Technology from UIIT, HPU, Shimla, H.P. in 2010. She obtained her Master of Engineering Degree in Software Engineering from Thapar University, Patiala, Punjab in 2012. Presently, she is pursuing PhD (Computer Science and Engineering) degree from Thapar University, Patiala. She is currently a Senior Research Fellow in RGNF, University Grants Commission (UGC), New Delhi, Government of India. Her research interests are software product line, software reuse, and feature models. She is an active member of ACM.

Dr Shivani Goel is a Professor in the Department of Computer Science and Engineering, School of Engineering and Applied Sciences, Bennett University, Greater Noida, U.P., India. She has done her B.Tech in Computer Science and Engineering, ME Software Engineering and PhD in 2012 from Thapar University, Patiala. She has guided 35 ME/MTech students, is presently guiding 8 PhD students, and has more than 60 publications in international journals and conferences of repute. She is a member of various professional organizations. Her research interests are software reuse and artificial intelligence.

Dr Ajay Kumar is an Assistant Professor at the Computer Science and Engineering Department, Thapar University, Patiala, India. He obtained his PhD degree in Theoretical Computer Science from Thapar University and M.Tech. from Kurukshetra University in 2013 and 2004, respectively. He has 13 years of teaching experience in the area of theory of computation, software testing, and programming languages. His research interest focuses on theoretical computer science, quantum computing, and cognitive science.

How to cite this article: Bhushan M, Goel S, Kumar A. Improving quality of software product line by analysing inconsistencies in feature models using an ontological rule-based approach. *Expert Systems*. 2018;35:e12256. https://doi.org/10.1111/exsy.12256

APPENDIX A

PARAMETERS USED IN TABLE 1

i. Inconsistency: It deals with the defects arising due to contradictory information in feature models (FMs).
ii. Identification: It deals with the identification of inconsistencies.
iii. Explanation: It provides an explanation for the cause of inconsistencies.
iv. Corrective explanation: It provides corrective solutions to fix inconsistencies.
v. Wrong cardinality: Inconsistencies due to wrong cardinality occur whenever one or more values of cardinality used by a set-relationship are not attainable.
vi. Classification: It classifies the defects due to inconsistency in FMs.
vii. Ontology based: It deals with the inconsistencies using ontology.
viii. Real world model: In real world models, inconsistencies are evaluated using real-life FMs.
ix. Basic FM: Inconsistencies in FM occur while defining the basic relationships among features.
x. Cardinality-based FM: Inconsistencies occur in FM extended with cardinalities.
xi. Extended FM: Inconsistencies in FM occur while defining additional information related to its features using attributes.
xii. Automated support: It handles inconsistencies using an automated approach.
xiii. Formalization: A formal approach is used to deal with inconsistencies.