

Classifying and resolving software product line redundancies using an ontological first-order logic rule based method

Megha Bhushan^{a,*}, José Ángel Galindo Duarte^b, Piyush Samant^c, Ashok Kumar^d, Arun Negi^e

^a School of Computing, DIT University, Dehradun, Uttarakhand, India

^b University of Seville, Spain

^c Adventum Advanced Solution Pvt. Ltd. Bangalore, Karnataka, 560066, India

^d Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab 140401, India

^e Deloitte USI, Hyderabad, India

A B S T R A C T

Keywords:

Feature model

First-order logic

Ontologies

Quality

Redundancy

Software product line

Software product line engineering improves software quality and diminishes development cost and time by efficiently developing software products. Its success lies in identifying the commonalities and variabilities of a set of software products which are generally modeled using feature models. The success of software product lines heavily relies upon the quality of feature models to derive high quality products. However, there are various defects that reduce profits of software product line. One of such defect is redundancy. While the majority of research work focuses on the identification of redundancies, their causes and corrections have been poorly explored. Causes and corrections must be as accurate and comprehensible as possible in order to support the developer in resolving the cause of a redundancy. This research work classified redundancies in the form of a typology. An ontological first-order logic rule based method is proposed to deal with redundancies. A two-step process is presented for mapping model to ontology based on predicate logic. First-order logic based rules are developed and applied to the generated ontology for identifying redundancies, their causes and corrections to resolve redundancies. The proposed method is illustrated using a case study from software product lines online tools repository. The results of experiments performed on 35 models with varied sizes of real world models as well as automatically-generated models from the Software Product Line Online Tools repository and models created via FeatureIDE tool conclude that the method is accurate, efficient and scalable with FM up to 30,000 features. Thus, enables deriving redundancy free end products from the product line and ultimately, improves its quality.

1. Introduction

Software Product Line (SPL) is the popular software reuse approach in the communities that deal with reuse. SPL is a collection of software products that satisfy the needs of a specific domain (Clements & Northrop, 2001). The benefits of SPL include higher quality, shorter time and lower cost to deliver a new product. Many companies like Hewlett-Packard, Cummins, Inc., Boeing, McDonalds and Philips to exploit the benefits exhibited by SPL (Northrop, 2008). In SPL, Feature Model (FM) notation captures variant and common features along with the representation of variability within SPL (Kang, Cohen, Hess, Novak & Peterson, 1990). It represents features and their relationships as a tree like structured model or graphical feature diagram. Each product in SPL

is the distinctive and legal integration of features. A well known characteristic of a software system which is significant to the user is defined as a feature. The standard example of an e-shop incorporates features payment and security as its basic functionality for enabling payment and providing security for the electronic shopping, respectively.

The development of accurate software products in SPL depends on numerous activities including technical management, organizational management and software engineering activities (Northrop, 2008). For instance, interfaces, testing, reuse aspect, and overloaded software development eventually lead to the success of SPL approach. However, the quality of software product is one of the most influencing factors that impacts on the productivity and quality of an SPL. Defects may get inadvertently introduced in FMs with the increasing size as well as

* Corresponding author at: School of Computing, DIT University, Dehradun, Uttarakhand 248009, India.

E-mail addresses: dr.megha@dituniversity.edu.in, mb.meghabhushan@gmail.com (M. Bhushan), jagalindo@us.es (J. Ángel Galindo Duarte), piyushsamantpth@gmail.com (P. Samant), ashok.kr@chitkara.edu.in (A. Kumar), arun98765@gmail.com (A. Negi).

complexity of models and due to inaccurate combinations of features. These defects are the imperfections that impede the production of valid software products (Salinesi & Mazo, 2012) and diminish the quality of FM as well as benefits from the SPL. A defect due to redundancy in FM is one of such defects which are concerned to the redundant relationships in a model. A FM is redundant if one of its constraints can be removed without changing the set of derived products. Though redundancy does not affect the semantics of a model, however, it increases the complexity of the model as well as the computational efforts required for deriving valid software products, and simultaneously decreases the maintainability of the model. These defects can adversely affect the other derived software products from SPL. Moreover, defects due to redundancy yet have not been explored much like other defects (i.e. dead features, wrong cardinality, inconsistencies and false optional features etc.) in the existing literature.

In order to attain high quality SPL approach and to produce redundancy free products from FMs, it is necessary to avoid replicating the redundancy defects (occurred in prior software products) in new derived products. However, it is unfeasible to manually detect and fix redundancies. Therefore, the aforementioned information motivated us to propose an ontological First-Order Logic (FOL) rule based method to deal with redundancies in FMs in order to improve the quality of SPL. The results of experiments performed on 35 FMs with varied sizes conclude that the method is accurate, efficient and scalable with FM up to 30,000 features. FMs were selected from: (i) Software Product Line Online Tools (SPLOT), a FM repository of real FMs (available at https://ec2-52-32-1-180.us-west2.compute.amazonaws.com:8080/SPLOT/feature_model_repository_depot.html), (ii) SPLOT FM repository of automatically-generated FMs (available at <https://ec2-52-32-1-180.us-west2.compute.amazonaws.com:8080/SPLOT/ind ex.html>), and (iii) building them with the FeatureIDE tool (see <https://github.com/FeatureIDE/FeatureIDE/wiki/Tutorial>).

The novel contributions of our proposed method are as follow:

- I. Classification of FM redundancies in the form of various categories and their cases.
- II. Mapping of FM to ontology based on predicate logic as it presents a communication between FM and ontology
- III. Developing and applying FOL based rules to the generated ontology in order to deal with redundancies.
- IV. Identification of redundancies with their causes and suggesting corrections in natural language.
- V. Information given to SPL developers for resolving redundancies in order to derive redundancy free end products from the PL.
- VI. Enhancement in the quality of SPL as it deals with more types of redundancies (see Section 2).

The article’s structure incorporates a typology of redundancies in Section 2. The methodology followed by the proposed work is described in Section 3. A motivating example of FM is provided in Section 4. Sections 5 and 6 describe the proposed approach and the details of its experimental evaluation. The results followed by discussion are provided in Section 7. The related work has been given in Section 8. Finally, Section 9 provides conclusion of the proposed method and future work.

2. Typology of redundancies

In this Section, various FM redundancies that exist in the literature have been classified mainly into six categories as shown in the typology given in Fig. 1 (Elfaki, Fong, Aik & Johar, 2013; Felfernig, Benavides Cuevas, Galindo Duarte & Reinfrank, 2013; Mazo, 2011; Salinesi & Mazo, 2012; Salinesi, Mazo & Díaz, 2010). The redundancies include defects due to redundant relationships in models. These redundancies have been classified conforming to their level of significance. For example, redundancies associated with the quality improvement of models without varying their semantics. A unique number has been allotted to each category that describes the order in which the redundancies should be considered. The typology is shown as a tree where

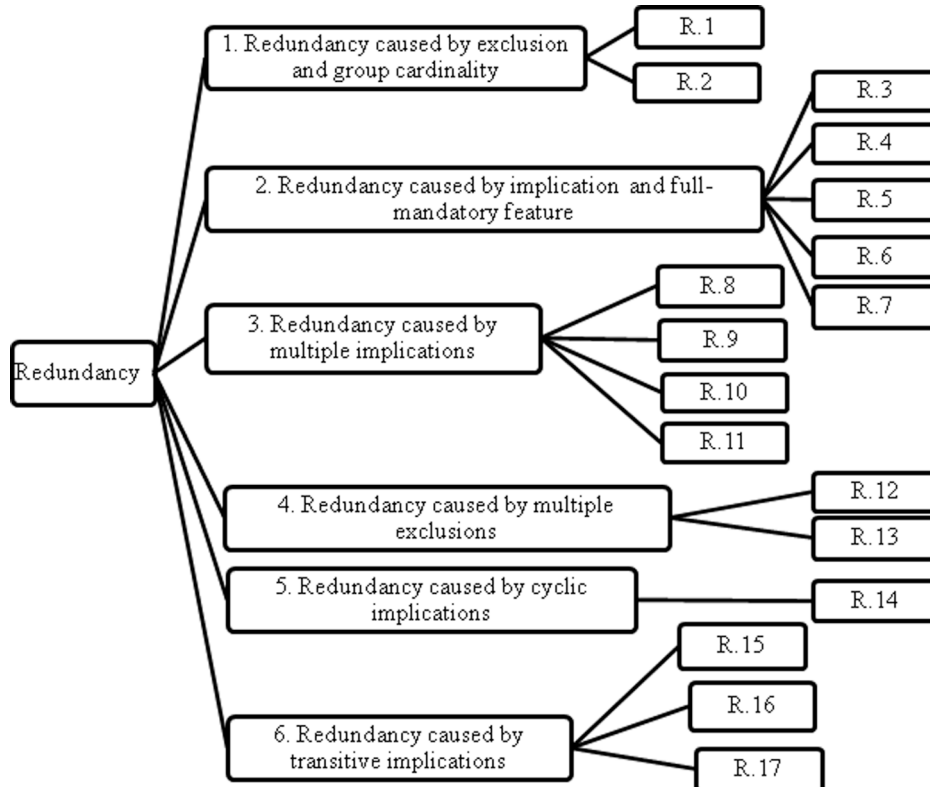


Fig. 1. Typology of redundancy.

Redundancy is the root of tree. The main categories and various cases of redundancy are represented by the nodes and sub-nodes of tree, respectively. The typology includes categories from 1 to 6 which are further composed of various cases. This typology can be used to attain redundancy free valid products by identifying and resolving FM redundancies according to their impact on SPLs. The typology of redundancy directs to:

- I. The generation of a standard and reusable approach to handle redundancies.
- II. The classification of redundancies based on a perspective that allows identifying similarities and differences among various categories of redundancy.
- III. Various cases of redundancy which diminish the efforts to identify redundant relationships between reusable features (for this no cases have been reported at the same source in the existing work).
- IV. The categories of redundancy have different priority and order, as they have different impact. For instance, the category of redundancy is placed first in the typology that occurs due to the same cause(s) i.e., by exclusion and group cardinality. It includes less complexity and requires reduced computational efforts for deriving redundancy free valid products.
- V. The choice of redundancy case(s) that one wants to deal with according to the expected quality level of a model or the impact of these cases of redundancy. For instance, there is a possibility that some redundancies are actually intended so as to reinforce the relationship between features. Moreover, other possibility may include that some practitioners do not want to detect a specific type of redundancy.
- VI. SPL developers can employ the typology to identify and correct redundancy along with its various possible cases as per their impact on SPLs.

The categories of redundancy along with their cases are explained in Fig. 1. Further, Table 1 explains the FM notation.

2.1. Redundancy caused by exclusion and group cardinality (Salinesi et al., 2010)

Table 2 depicts the cases of redundancy caused by exclusion of an alternative-child feature.

Table 1
Types of relationships in feature model.

Notation	Type of relationship
	Mandatory (Kang et al., 1990): c (child feature) must be contained in each valid product whenever p (parent feature) is chosen and vice versa.
	Optional (Kang et al., 1990): c (child feature) may or may not be contained in the valid product(s) related with p (parent feature).
	Alternative (Benavides, Segura & Ruiz-Cortés, 2010): $c1$ and $c2$ (child features) are in alternative relationship with p (parent feature) when exactly one of the child features has to be incorporated for developing any valid product whenever p is incorporated.
	Or (Benavides, Segura & Ruiz-Cortés, 2010): $c1$ and $c2$ (child features) are in or-relationship with p (parent feature) when multiple child features are incorporated for developing any valid product whenever p is chosen.
	Implication (Von der Maßen & Lichter, 2004): Features $c1$ and $c2$ related with implication constraint represent that $c2$ should be included in each valid product with $c1$.
	Exclusion (Von der Maßen & Lichter, 2004): The exclusion constraint between features $c1$ and $c2$ will not allow incorporating these features concurrently in the same valid product.

Table 2
Cases of redundancy caused by exclusion and group cardinality.

Case (with rule no.)	Description of Rule
R.1 	p is the parent feature (either FMF or root) of alternative-child features $c1$ and $c2$. $c1$ and $c2$ are associated in group cardinality $< 1.0.1 >$ where $c2$ is excluded by $c1$. Thus, exclusion ($exc1$) becomes redundant.
R.2 	$c1$ and $c2$ are associated in group cardinality $< 1.0.1 >$ and FMF $c3$ has a parent $c1$. Further, $c2$ is excluded by FMF $c4$ which has a parent $c3$. It represents a multiple exclusion of an alternative-child feature $c2$ which is redundant.

FMF: Full-mandatory feature

2.2. Redundancy caused by implication and full-mandatory feature (Elfaki et al., 2013; Salinesi et al., 2010; Van Der Storm, 2007; Von der Maßen & Lichter, 2004; White et al., 2010)

Table 3 represents the cases of redundancy caused by implication constraint(s) and a FMF.

Table 3
Cases of redundancy caused by implication and full-mandatory feature.

Case (with rule no.)	Description of Rule
R.3 	p is the parent feature (either FMF or root) of child features $c1$ and $c2$. FMF $c1$ is implied by an optional feature $c2$, in any case, $c1$ is always selected and therefore, implication ($imp1$) from $c2$ to $c1$ is redundant.
R.4 	An optional feature $c3$ has a parent optional feature $c1$. Both, FMCF $c4$ and optional child feature $c5$ have same parent FMF $c2$ where $c1$ implies $c4$. In any respect, $c4$ is always incorporated in all the configurations and therefore, implication ($imp1$) from $c1$ to $c4$ is redundant.
R.5 	FMF $c2$ is implied by an optional feature $c1$ where $c2$ appears in all the configurations and thus, implication ($imp1$) from $c1$ to $c2$ is redundant. The FMCF $c3$ is implied by its parent feature $c2$. This implication ($imp2$) is also redundant as both $c2$ and $c3$ will always be selected in any case.
R.6 	An optional child feature $c1$ has a FMFP $p1$ and another FMCF $c2$ has a parent FMF $p2$. It means $c2$ is incorporated in every product and thus, $c1$ implies ($imp1$) $c2$ is redundant.
R.7 	Both FMCFs $c1$ and $c2$ have parent feature p (here, p can be either FMF or root), and these child features are implied by p . In any case, $c1$ and $c2$ appear in all the configurations and thus, implications ($imp1$ and $imp2$) are redundant.

FMF: Full-mandatory feature, FMCF: Full-mandatory child feature, FMFP: Full-mandatory parent feature.

2.3. Redundancy caused by multiple implications (Elfaki et al., 2011; Salinesi, Mazo & Diaz, 2010; Van Der Storm, 2007; Von der Maßen & Lichter, 2004

Table 4 shows various cases of redundancy caused by several implication constraints.

2.4. Redundancy caused by multiple exclusions (EElfaki et al., 2013; Mazo, 2011; Salinesi et al., 2010; Van Der Storm, 2007; Von der Maßen & Lichter, 2004

Table 5 depicts the cases of redundancy caused by numerous exclusion constraints.

2.5. Redundancy caused by cyclic implications (Salinesi et al., 2010; Von der Maßen & Lichter, 2004

Table 6 represents a case of redundancy caused by cyclic implication constraints.

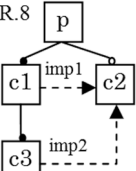
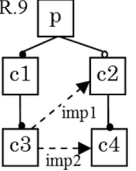
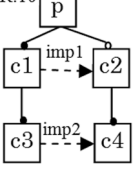
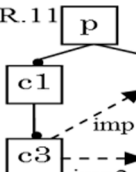
2.6. Redundancy caused by transitive implications (Salinesi et al., 2010; Trinidad et al., 2008; Von der Maßen & Lichter, 2004; White et al., 2010

Table 7 illustrates the cases of redundancy caused by transitive implication constraints.

3. Methodology

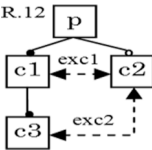
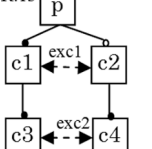
This section describes the methodology followed by the proposed approach which includes

Table 4 Cases of redundancy caused by multiple implications.

Case (with rule no.)	Description of Rule
R.8	 <p>p is the parent feature (either FMF or root) of child features $c1$ and $c2$. FMF $c1$ implies an optional feature $c2$, and $c1$ is a parent of FMCF $c3$, thus, the implication ($imp2$) from $c3$ to $c2$ is redundant.</p>
R.9	 <p>FMCF $c3$ has a FMFP $c1$ where $c3$ implies an optional feature $c2$ means that mandatory child feature $c4$ of parent $c2$ is also included. Therefore, implication ($imp2$) on $c4$ after $c2$ is implied is redundant.</p>
R.10	 <p>FMF $c1$ implies an optional feature $c2$ means that this implication ($imp1$) constraint will be reflected in each mandatory or FMCF of the two features $c1$ and $c2$. Simultaneously, FMCF $c3$ has a parent $c1$ implies another mandatory child feature $c4$ which has a parent $c2$ is redundant.</p>
R.11	 <p>FMFs $c1$, $c2$, $c3$ and $c4$ where $c3$ has a parent $c1$ implies $c2$ and its child feature $c4$. Both implications ($imp1$ and $imp2$) are redundant because $c2$ and $c4$ are already included in each product.</p>

FMF: Full-mandatory feature, FMCF: Full-mandatory child feature, FMFP: Full-mandatory parent feature

Table 5 Cases of redundancy caused by multiple exclusions.

Case (with rule no.)	Description of Rule
R.12	 <p>p is the parent feature (either FMF or root) of FMCF $c1$ and optional child feature $c2$. $c2$ is excluded by $c1$ where $c1$ is the parent of FMCF $c3$, thus, exclusion ($exc2$) from $c3$ to $c2$ is redundant.</p>
R.13	 <p>$c1$ excludes $c2$ means that all the FMCFs of $c1$ exclude by default all the mandatory child features of the excluded $c2$. Simultaneously, FMCF $c3$ has a parent $c1$ excludes another mandatory child feature $c4$ which has a parent $c2$ is redundant.</p>

FMF: Full-mandatory feature, FMCF: Full-mandatory child feature.

Table 6 Case of redundancy caused by cyclic implications.

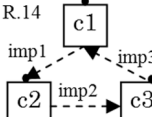
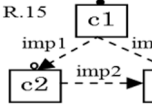
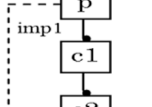
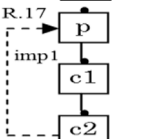
Case (with rule no.)	Description of Rule
R.14	 <p>Feature $c1$ implies feature $c2$, $c2$ implies feature $c3$ and $c3$ implies $c1$. The cycle can start from any feature. In this case, the selection of $c3$ leads to $c1$ is implied by $c3$ and $c2$ is implied by $c1$, therefore, $c1$ is implied by $c3$ is redundant since feature $c3$ is already chosen.</p>

Table 7 Cases of redundancy caused by transitive implications.

Case (with rule no.)	Description of Rule
R.15	 <p>Features $c1$ and $c2$ directly implies a feature $c3$ where $c2$ is implied by $c1$. The implication ($imp3$) from $c1$ to $c3$ is redundant as the transitive implication from $c1$ through $c2$ already implies $c3$.</p>
R.16	 <p>FMCF $c1$ has a parent feature p (either full-mandatory or root) and the FMCF $c2$ has a parent $c1$ where p implies $c2$. The implication ($imp1$) from p to $c2$ is redundant due to a transitive relationship between p and $c2$.</p>
R.17	 <p>FMCF $c1$ has a parent feature p (either FMF or root) and the FMCF $c2$ has a parent $c1$ where $c2$ implies p. The implication ($imp1$) is redundant due to a transitive relationship between $c2$ and p.</p>

FMF: Full-mandatory feature, FMCF: Full-mandatory child feature.

- (i) The input FM (in SXFM format is available in online SPLOT FM repository of real FMs or automatically-generated FMs) is automatically read and analyzed using FeatureIDE tool through importing its model from SPLOT (SXFM format). (Note: FMs can be manually encoded by using FeatureIDE tool).
- (ii) Additional features and cross-tree constraints are intentionally injected in the input model by enabling editing in FeatureIDE's model editor to cause redundancies.
- (iii) The modified input model (i.e. in XML format) generated from FeatureIDE is transformed to ontology (based on predicate logic) using an XML parser (available at <https://bit.ly/3aqRUZA>).
- (iv) To identify redundancies along with their causes and corrections, a set of redundancy rules based on FOL is developed and applied to the generated ontology.

- (v) The results obtained after applying all rules consist of identified redundancies along with their causes and provide corrections. The generated results enable PL developers to resolve redundancies, i.e. by eliminating cross-tree constraints incorporated in the source of redundancies.

M. Motivating example of FM

A running example of FM is used to demonstrate the proposed

method throughout the paper. The modified form of E-commerce system1 FM from SPLOT repository is used to describe the proposed method as shown in Fig. 2. The entire SPL is represented by the root feature "e-commerce system" which is required to be incorporated in each valid product of the PL. Further, unique name is assigned to each feature and cross-tree constraint relationship for better explanation. Thus, unique names are given to the features appearing twice. To demonstrate the presented method, 29 cross-tree constraints and 68 features are intentionally injected in the FM to describe redundancies.

5. Approach

This section explains the implementation of our method to deal with redundancies for deriving redundancy free valid products from SPLs. The running example of E-commerce system1 FM (explained using Fig. 2) is used for describing the presented method.

5.1. Mapping of E-commerce system1 PL to ontology

The mapping of FM to ontology based on predicate logic is a two-step process which is as follow:

5.1.1. SPLOT to FeatureIDE

FeatureIDE tool (Thüm, Kästner, Benduhn, Meinicke, Saake & Leich, 2014) automatically reads E-commerce system1 FM through importing its model in SXFM format from SPLOT repository. The corresponding model can be graphically represented by means of FeatureIDE's visualization functionality. The model editor of this tool further enables

editing. To illustrate the proposed method, additional features (i.e. a1-a68) and cross-tree constraints (i.e. imp1-imp23 and exc1-exc6) are intentionally inserted in the input E-commerce system1 FM file to cause defects due to redundancy as shown in Fig. 2

Following illustrates the relationships shown in Fig. 2 in order to comprehend the model developed in FeatureIDE:

A dashed arrow (starts from the source and finishes towards the target) represents an implication relationship which has been assigned a unique name such as imp1. A double headed dashed arrow represents an exclusion relationship which has been assigned a unique name such as exc1. The features associated in the group cardinality are basically optional features (Baader, Calvanese, McGuinness, Patel-Schneider & Nardi, 2003; Lesta, Schaefer & Winkelmann, 2015).

5.1.2. FeatureIDE to ontology

A parser (available at <https://bit.ly/3aqRUZA>) is developed that maps an XML file of modified input E-commerce system1 FM (available at <https://bit.ly/2XRgAHZ>) from FeatureIDE to ontology (based on predicate logic) as shown in Fig. 3. The features and their relationships in E-commerce system1 FM are represented in the generated ontology. This ontology comprises of following types of predicates: parent, exclusion, implication, feature and cardinality which are represented using "p", "e", "i", "f" and "c", respectively. Table 8 illustrates the mapping patterns used to attain the ontology from FM and examples for defining these predicates. In ontology, predicates represent properties and classes using ternary and binary predicates, respectively.

5.2. Analyze SPL redundancies

In this Section, redundancy rules are developed and applied as FOL queries to the generated ontology using SWI Prolog (Wielemaker, 2015). Following subsections incorporate redundancy rules to deal with defects due to redundancy and their results:

5.2.1. Redundancy rules

A set of redundancy rules based on FOL is developed as shown in Fig. 4, which represents particular cases of misuse among the

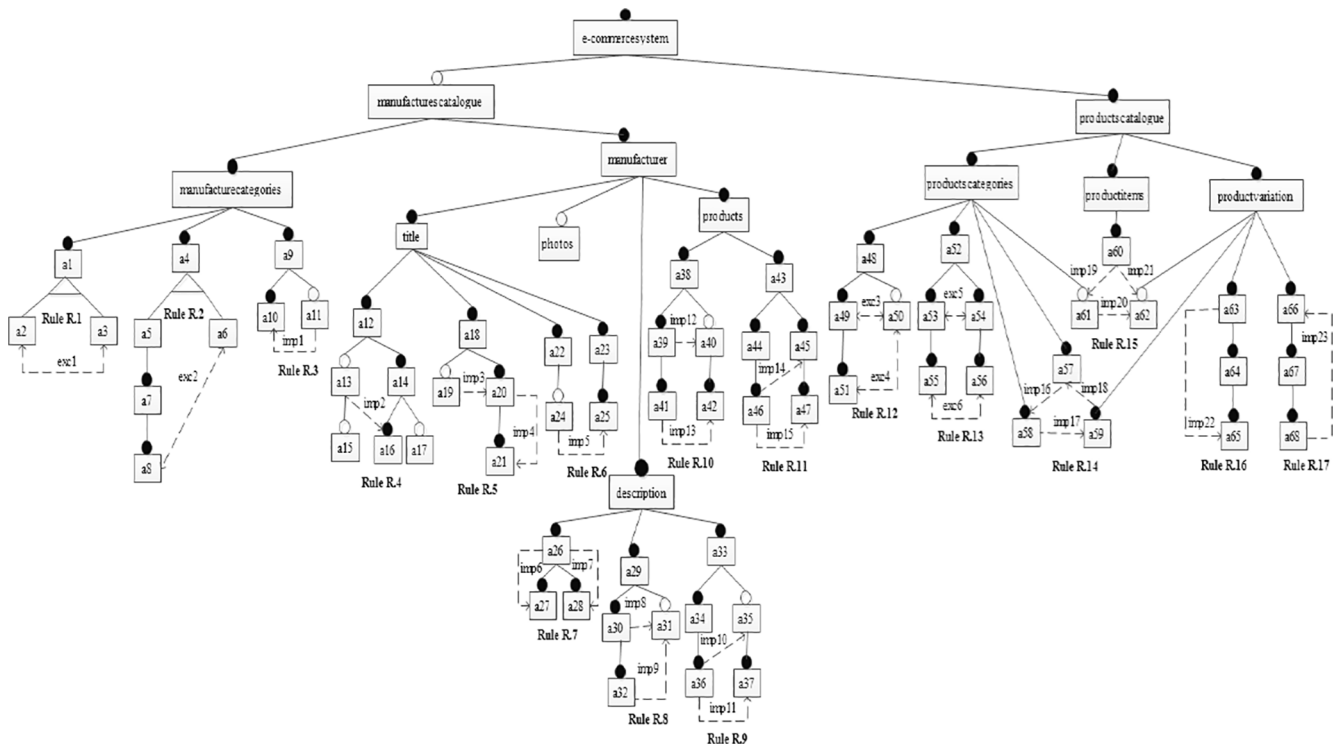


Fig. 2. E-commerce system1 feature model from software product lines online tools repository representing redundancies.

```

E-commerce system1 output.pl...
File Edit Format View Help
f(e_commercesystem,m).
f(manufacturerscatalogue,o).
f(manufacturerscategories,m).
f(a5,o).
f(a7,m).
f(a9,m).
f(manufacturer,m).
f(title,m).
f(a12,m).
f(a13,o).
f(a14,m).
f(a18,m).
f(a20,m).
f(a22,m).
f(a23,m).
f(description,m).
f(a26,m).
f(a29,m).
f(a30,m).
f(products,m).
f(a38,m).
f(a39,m).
f(a40,o).
f(a43,m).
f(a44,m).
f(a45,m).
f(a33,m).
f(a34,m).
f(a35,o).
f(productscatalogue,m).
f(productcategories,m).
f(a48,m).
f(a49,m).
f(a52,m).
f(a53,m).
f(a54,o).
f(productitems,m).
f(productvariations,m).
f(a63,m).
f(a64,m).
f(a66,m).
f(a67,m).
f(a1,m).
f(a4,m).
f(a2,o).
f(a3,o).
f(a8,m).
f(a6,o).
f(a10,m).
f(a11,o).
f(a15,o).
f(a16,m).
f(a17,o).
f(a19,o).
f(a21,m).
f(a24,o).
f(a25,m).
f(photos,o).
f(a27,m).
f(a28,m).
f(a32,m).
f(a31,o).
f(a41,m).
f(a42,m).
f(a46,m).
f(a47,m).
f(a36,m).
f(a37,m).
f(a51,m).
f(a50,o).
f(a55,m).
f(a56,m).
f(a61,o).
f(a58,m).
f(a60,m).
f(a62,o).
f(a59,m).
f(a65,m).



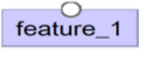
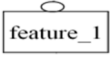
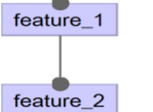

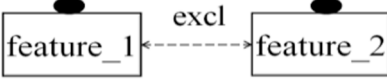
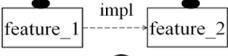
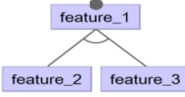
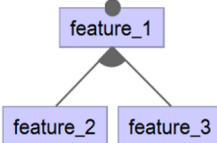
E-commerce system1 output.pl - Notepad
File Edit Format View Help
f(a68,m).
f(a57,m).
p(manufacturerscatalogue,e_commercesystem).
p(manufacturerscategories,manufacturerscatalogue).
p(a1,manufacturerscategories).
p(a2,a1).
p(a3,a1).
p(a4,manufacturerscategories).
p(a5,a4).
p(a7,a5).
p(a8,a7).
p(a6,a4).
p(a9,manufacturerscategories).
p(a10,a9).
p(a11,a9).
p(manufacturer,manufacturerscatalogue).
p(title,manufacturer).
p(a12,title).
p(a13,a12).
p(a15,a13).
p(a14,a12).
p(a16,a14).
p(a17,a14).
p(a18,title).
p(a19,a18).
p(a20,a18).
p(a21,a20).
p(a22,title).
p(a24,a22).
p(a23,title).
p(a25,a23).
p(photos,manufacturer).
p(description,manufacturer).
p(a26,description).
p(a27,a26).
p(a28,a26).
p(a29,description).
p(a30,a29).
p(a32,a30).
p(a31,a29).
p(products,manufacturer).
p(a38,products).
p(a39,a38).
p(a41,a39).
p(a40,a38).
p(a42,a40).
p(a43,products).
p(a44,a43).
p(a46,a44).
p(a45,a43).
p(a47,a45).
p(a33,manufacturer).
p(a34,a33).
p(a36,a34).
p(a35,a33).
p(a37,a35).
p(productscatalogue,e_commercesystem).
p(productcategories,productscatalogue).
p(a48,productcategories).
p(a49,a48).
p(a51,a49).
p(a50,a48).
p(a52,productcategories).
p(a53,a52).
p(a55,a53).
p(a54,a52).
p(a56,a54).
p(a61,productcategories).
p(a58,productcategories).
p(productitems,productscatalogue).
p(a60,productitems).
p(productvariations,productscatalogue).
p(a62,productvariations).
p(a59,productvariations).
p(a63,productvariations).
p(a64,a63).
p(a65,a64).
p(a66,productvariations).

E-commerce system1 output.pl - N...
File Edit Format View Help
p(a67,a66).
p(a68,a67).
p(a57,productscatalogue).
i(a11,a10,imp1).
i(a13,a16,imp2).
i(a19,a20,imp3).
i(a20,a21,imp4).
i(a24,a25,imp5).
i(a26,a27,imp6).
i(a26,a28,imp7).
i(a30,a31,imp8).
i(a32,a31,imp9).
i(a36,a35,imp10).
i(a36,a37,imp11).
i(a39,a40,imp12).
i(a41,a42,imp13).
i(a46,a45,imp14).
i(a46,a47,imp15).
i(a57,a58,imp16).
i(a58,a59,imp17).
i(a59,a57,imp18).
i(a60,a61,imp19).
i(a61,a62,imp20).
i(a60,a62,imp21).
i(a63,a65,imp22).
i(a66,a66,imp23).
e(a2,a3,exc1).
e(a8,a6,exc2).
e(a49,a50,exc3).
e(a51,a50,exc4).
e(a53,a54,exc5).
e(a55,a56,exc6).
c(a1,[a2,a3],[1,1]).
c(a4,[a5,a6],[1,1]).

```

Fig. 3. Ontology of E-commerce system1 feature model.

Table 8
Mapping patterns from feature model to ontology with description of predicates.

Type	FeatureIDE Notation	Feature Model Notation	Ontology Representation based on Predicate logic Type	Description	Example
Mandatory			$f(\text{feature_1}, m)$	Feature feature_1 is mandatory	$f(\text{manufacturer}, m)$
Optional			$f(\text{feature_1}, o)$	Feature feature_1 is optional	$f(\text{photos}, o)$
Parent			$p(\text{feature_1}, \text{feature_2})$	Parent feature_2 has child feature_1	$p(\text{photos}, \text{manufacturer})$
Exclusion	$\text{feature_1} \bowtie \text{feature_2}$		$e(\text{feature_1}, \text{feature_2}, \text{exc1})$	feature_2 is excluded by feature_1 where exc1 is an exclusion constraint	$e(a2, a3, \text{exc1})$
Implication	$\text{feature_1} \Rightarrow \text{feature_2}$		$i(\text{feature_1}, \text{feature_2}, \text{imp1})$	feature_2 is implied by feature_1 where imp1 is an implication constraint	$i(a11, a10, \text{imp1})$
Group cardinality	Alternative relationship		$c(\text{feature_1}, [\text{feature_2}, \text{feature_3}], [\text{Min}, \text{Max}])$	Child features feature_2 and feature_3 have same parent feature_1 belong to the group cardinality $\langle \text{min}, \text{max} \rangle$ where maximum (Max) and minimum (Min) describe the count of child features which are allowed to be specified in a cardinality relationship. In this case, child features can be more than two.	$c(a1, [a2, a3], [1, 1])$ represents that only one child feature can be selected from a set of features (i.e. a2 and a3) with group cardinality $\langle 1, 0, 1 \rangle$.
	Or-relationship				$c(p, [c1, c2], [1, 2])$ represents two child features c1 and c2 having same parent p is equivalent to a group cardinality $\langle 1, \dots, 2 \rangle$ where 2 is the maximum count of features.

```

Redundancy rules.pl - Notepad
File Edit Format View Help
rule1:- f(P,m), f(C1,o), f(C2,o), c(P,[C1,C2],[1,1]), e(C1,C2,E1), write('\nDefect: Redundancy1\nCause: exclusion between
alternative-child features '), write(C1), write(' and '), write(C2), write(' is redundant\nCorrection: eliminate '), write(E1).

rule2:- f(P,m), f(C1,o), f(C2,o), f(C3,m), f(C4,m), p(C3,C1), p(C4,C3), c(P,[C1,C2],[1,1]), e(C4,C2,E1), write('\nDefect:
Redundancy2\nCause: multiple exclusion of an alternative-child feature '), write(C2), write('\nCorrection: eliminate '), write
(E1).

rule3:- f(P,m), f(C1,m), f(C2,o), p(C1,P), p(C2,P), i(C2,C1,I1), write('\nDefect: Redundancy3\nCause: optional feature '),
write(C2), write(' implies the full-mandatory feature '), write(C1), write(' is redundant\nCorrection: eliminate '), write(I1).

rule4:- f(P,m), f(C1,o), f(C2,m), f(C3,o), f(C4,m), f(C5,o), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2), p(C5,C2), i(C1,C4,I1),
write('\nDefect: Redundancy4\nCause: optional feature '), write(C1), write(' implies the full-mandatory feature '), write(C4),
write(' is redundant\nCorrection: eliminate '), write(I1).

rule5:- f(P,m1), f(C1,o1), f(C2,m2),f(C3,m3), p(C1,P), p(C2,P), p(C3,C2), i(C1,C2,I1), i(C2,C3,I2), write('\nDefect:
Redundancy5\nCauses: optional feature '), write(C1), write(' implies a full-mandatory feature '),write(C2), write('\nand
mandatory child feature '), write(C3), write(' is implied by its parent '), write(C2), write(' are redundant'),write
('\nCorrections: eliminate '), write(I1), write(' and '), write(I2).

rule6:- f(P1,m), f(P2,m), f(C1,o), f(C2,m), p(C1,P1), p(C2,P2), i(C1,C2,I1), write('\nDefect: Redundancy6\nCause: optional
feature '), write(C1), write(' implies a full-mandatory feature'), write(C2), write(' is redundant\nCorrection: eliminate '),
write(I1).

rule7:- f(P,m), f(C1,m), f(C2,m), p(C1,P), p(C2,P), i(P,C1,I1), i(P,C2,I2), write('\nDefect: Redundancy7\nCauses: both
mandatory child features '), write(C1), write(' and '), write(C2), write(' are implied by their parent '), write(P), write(' are
redundant\nCorrections: eliminate '), write(I1), write(' and '), write(I2).

rule8:- f(P,m), f(C1,m), f(C2,o), f(C3,m), p(C1,P), p(C2,P), p(C3,C1), i(C1,C2,I1), i(C3,C2,I2), write('\nDefect:
Redundancy8\nCause: multiple implication of an optional feature '), write(C2), write('\nCorrection: eliminate '), write(I2).

rule9:- f(P,m), f(C1,m), f(C2,o), f(C3,m), f(C4,m), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2), i(C3,C2,I1), i(C3,C4,I2), write
('\nDefect: Redundancy9\nCause: implication on '), write(C4), write(' is redundant after '), write(C2), write(' is implied by a
mandatory feature '), write(C3), write('\nCorrection: eliminate '), write(I2).

rule10:- f(P,m), f(C1,m), f(C2,o), f(C3,m), f(C4,m), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2), i(C1,C2,I1), i(C3,C4,I2), write
('\nDefect: Redundancy10\nCause: mandatory feature '), write(C3), write(' implies another mandatory feature '), write(C4),
write(' is redundant\nCorrection: eliminate '), write(I2).

rule11:- f(P,m), f(C1,m), f(C2,m), f(C3,m), f(C4,m), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2), i(C3,C2,I1), i(C3,C4,I2), write
('\nDefect: Redundancy11\nCauses: implications on mandatory features '), write(C2), write(' and '), write(C4), write(' are
redundant '), write('\nCorrections: eliminate '), write(I1), write(' and '), write(I2).

rule12:- f(P,m), f(C1,m), f(C2,o), f(C3,m), p(C1,P), p(C2,P), p(C3,C1), e(C1,C2,E1), e(C3,C2,E2), write('\nDefect:
Redundancy12\nCause: multiple exclusion of an optional feature '), write(C2), write('\nCorrection: eliminate '), write(E2).

rule13:- f(P,m), f(C1,m), f(C2,o), f(C3,m), f(C4,m), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2), e(C1,C2,E1), e(C3,C4,E2),
write('\nDefect: Redundancy13\nCause: mandatory feature '), write(C3), write(' excludes another mandatory feature '),
write(C4), write(' is redundant\nCorrection: eliminate '), write(E2).

rule14:- i(C1,C2,I1), i(C2,C3,I2), i(C3,C1,I3), write('\nDefect: Redundancy15\nCause: implication from '), write(C3), write('
to '), write(C1), write(' is redundant\nCorrection: eliminate '), write(I3).

rule15:- i(C1,C2,I1), i(C2,C3,I2), i(C1,C3,I3), write('\nDefect: Redundancy14\nCause: implication from '), write(C1), write('
to '), write(C3), write(' is redundant\nCorrection: eliminate '), write(I3).

rule16:- f(P,m), f(C1,m), f(C2,m), p(C1,P), p(C2,C1), i(P,C2,I1), write(' \nDefect: Redundancy16\nCause: implication from
'), write(P), write(' to '),write(C2), write(' is redundant\nCorrection: eliminate '), write(I1).

rule17:- f(P,m), f(C1,m), f(C2,m), p(C1,P), p(C2,C1), i(C2,P,I1), write('\nDefect: Redundancy17\nCause: implication from
'), write(C2), write(' to '),write(P), write(' is redundant\nCorrection: eliminate '), write(I1).

```

Fig. 4. Redundancy rules based on first-order logic.


```
SWI-Prolog -- f:/Experiments report/Case studies/Redundancy/E-commerce system1.pl
File Edit Settings Run Debug Help
A: 1498456951.23573

Defect: Redundancy1
Cause: exclusion between alternative-child features a2 and a3 is redundant
Correction: eliminate exc1

Defect: Redundancy2
Cause: multiple exclusion of an alternative-child feature a6
Correction: eliminate exc2

Defect: Redundancy3
Cause: optional feature a11 implies the full-mandatory feature a10 is redundant
Correction: eliminate imp1

Defect: Redundancy4
Cause: optional feature a13 implies the full-mandatory feature a16 is redundant
Correction: eliminate imp2

Defect: Redundancy5
Causes: optional feature a19 implies a full-mandatory feature a20
and mandatory child feature a21 is implied by its parent a20 are redundant
Corrections: eliminate imp3 and imp4

Defect: Redundancy6
Cause: optional feature a24 implies a full-mandatory feature a25 is redundant
Correction: eliminate imp5

Defect: Redundancy7
Causes: both mandatory child features a27 and a28 are implied by their parent a26 are redundant
Corrections: eliminate imp6 and imp7

Defect: Redundancy8
Cause: multiple implication of an optional feature a31
Correction: eliminate imp9

Defect: Redundancy9
Cause: implication on a37 is redundant after a35 is implied by a mandatory feature a36
Correction: eliminate imp11

Defect: Redundancy10
Cause: mandatory feature a41 implies another mandatory feature a42 is redundant
Correction: eliminate imp13

Defect: Redundancy11
Causes: implications on mandatory features a45 and a47 are redundant
Corrections: eliminate imp14 and imp15

Defect: Redundancy12
Cause: multiple exclusion of an optional feature a50
Correction: eliminate exc4

Defect: Redundancy13
Cause: mandatory feature a55 excludes another mandatory feature a56 is redundant
Correction: eliminate exc6

Defect: Redundancy14
Cause: implication from a59 to a57 is redundant
Correction: eliminate imp18

Defect: Redundancy15
Cause: implication from a60 to a62 is redundant
Correction: eliminate imp21

Defect: Redundancy16
Cause: implication from a63 to a65 is redundant
Correction: eliminate imp22

Defect: Redundancy17
Cause: implication from a68 to a66 is redundant
Correction: eliminate imp23

B: 1498456951.31573
Time: 0.08000016212463379
true .
```

Fig. 5. Results attained after implementing rules to E-commerce system1 ontology.

Table 9

Description of REAL feature models taken from software product lines online tools repository (Available at <https://doi.org/10.5281/zenodo.3820862>).

Feature models	Total number of features	Total number of cross-tree constraints	Identified number of redundancies	Identified type of redundancies
Isolation	21	2	2	1
E-commerce system1	80	29	17	All
Computers	53	9	7	1,2
e-Event	78	17	11	1-3
ATM Software	95	21	13	1-4
Tankwar	107	24	14	1-5
WebCollaboration	111	24	14	1-5
GerenciaLojaVirtual	128	26	16	1-5
BioFM	138	32	18	All
Berkley DB	148	43	19	All
E-science application	148	32	18	All
Nákladný automobile	160	277	19	All
Windows	157	29	17	All
FISH	171	32	19	All
ModelTransformation	203	38	24	All
SmartTV	231	46	28	All
Xtext	283	51	30	All
Total Informatica	300	58	34	All
BattleofTanks	339	67	41	All
FM_Test	363	113	41	All
Printers	412	87	51	All
android60	431	115	53	All
android510	486	134	58	All
Electronic Shopping	586	125	62	All
A Model for Decision-making for Investments on Enterprise Information Systems	686	308	68	All
windows80	792	256	70	All

Type of redundancies: There are a total of six categories of FM redundancies as shown in the typology in Fig. 1, where “All” represents all six categories.

Table 10

Description of automatically-generated 3-CNF Feature Models from software product lines online tools repository (Available at <https://doi.org/10.5281/zenodo.3820862>).

Feature models	Total number of features	Total number of cross-tree constraints	Identified number of redundancies	Identified type of redundancies
FM-1	916	175	75	All
FM-2	1495	233	79	All
FM-3	2658	254	92	All
FM-4	6016	319	99	All
FM-5	11,580	283	109	All

Type of redundancies: There are a total of six categories of FM redundancies as shown in the typology in Fig. 1, where “All” represents all six categories.

relationships in a model that generated redundancies. Each rule is developed corresponding to each case of redundancy shown in Tables 2–7. These rules (as FOL queries) can be implemented independently as well as simultaneously to the developed ontology of E-commerce system1 FM (see Fig. 3) in Prolog.

5.2.2. Results

Fig. 5 represents the results obtained after applying each redundancy rule to the ontology of E-commerce system1. The results comprise of identified redundancies along with their causes and provide corrections in natural language which is easily comprehensible by users. SPL developers can use the generated results to resolve redundancies, i.e. by eliminating cross-tree constraints incorporated in the source of redundancies.

Table 11

Description of Feature Models generated using FeatureIDE tool (Available at <https://doi.org/10.5281/zenodo.3820862>).

Feature Models	Total number of features	Total number of cross-tree constraints	Identified number of redundancies	Identified type of redundancies
FM-15000	15,000	203	119	All
FM-20000	20,000	212	126	All
FM-25000	25,000	232	136	All
FM-30000	30,000	256	150	All

Each generated FM comprises of mandatory feature, optional feature and cross-tree constraints.

Type of redundancies: There are a total of six categories of FM redundancies as shown in the typology in Fig. 1, where “All” represents all six categories.

6. Experiment evaluation

The presented method was evaluated by performing experiments on 35 FMs up to 30,000 features. These models were selected from SPLOT FM repository (Tables 9 and 10) and others were built by FeatureIDE tool (Table 11). The goal was to measure the accuracy, scalability, completeness as well as minimalism of the set of redundancy rules and finally, to compare the proposed method with existing work. Following discusses these measurements:

6.1. Experimental environment

The environment for the evaluation of our method includes a Dell workstation T-5600 with Windows 7 Professional N of 64 bits, RAM memory of 8.00 GB 1600 MHz, processor Intel® Xeon® CPU E5-2650 @2.60 GHz, HDD-SATA 500 GB @7200RPM, online SPLOT repository, Eclipse Luna SR2 (5.5.2), SWI-Prolog (Version 7.4.2, 64 bits) and FeatureIDE 2.7.4.

6.2. Accuracy

The accuracy of proposed method is based on the correct (i) mapping of FMs into ontologies, and (ii) identification of redundancies along with their causes and corrections.

6.2.1. Accuracy of the mapping

As discussed in Section 5.1, in order to check that models have been accurately mapped, the number of mandatory features, optional features, implications, exclusions, and cardinalities in the results obtained with our mapping were compared against the input XML files from FeatureIDE (by searching the corresponding tags used in the parser). These comparisons were made over all 35 FMs with root feature, features, group cardinalities and cross-tree constraints, and these models were of varied sizes up to 30,000 features. The outcomes of these comparisons are same, resulting in 100% accuracy in the mapping of models into ontologies with 0% false positives.

6.2.2. Accuracy of the identification of redundancies with their causes and corrections

The ratio of the number of redundancies with their causes and corrections that are accurately identified by the proposed method to the total number of redundancies (with their causes and corrections) in the FM is defined as accuracy. The proposed method identified 100% of the redundancies with their causes and corrections in 35 FMs by considering our set of rules for redundancies with 0% false positives. Additionally, it was observed that all rules simultaneously and individually, identified the expected redundancies with their causes and corrections signifying 100% accuracy to handle redundancies considered in the proposed method.

Table 12

Comparing the accuracy of FeatureIDE tool with the proposed method using E-commerce system1 Feature Model for redundancies.

Rules	Constraints description	FeatureIDE tool	Proposed method	Status (S)
R.1	(i) a2 excludes a3	(i) Dead features: manufacturerscatalogue, manufacturercategories, a1-a10, a12, a14, a16, title, a18, a20-a23, manufacturer, description, a25-a30, a32-a34, products, a36-a39, a41-a47 False-optional features: a40, a35, a31, a24, a17, a15, a13, a11, photos	Defect: Redundancy1 Cause: exclusion between alternative-child features a2 and a3 is redundant Correction: eliminate exc1	0
R.2	(i) a8 excludes a6	(i) CR	Defect: Redundancy2 Cause: multiple exclusion of an alternative-child feature a6 Correction: eliminate exc2	1
R.3	(i) a11 implies a10	(i) CR	Defect: Redundancy3 Cause: optional feature a11 implies the full-mandatory feature a10 is redundant Correction: eliminate imp1	1
R.4	(i) a13 implies a16	(i) CR	Defect: Redundancy4 Cause: optional feature a13 implies the full-mandatory feature a16 is redundant Correction: eliminate imp2	1
R.5	(i) a19 implies a20 (ii) a20 implies a21	(i) CR (ii) CR	Defect: Redundancy5 Causes: optional feature a19 implies a full-mandatory feature a20 and mandatory child feature a21 is implied by its parent a20 are redundant Corrections: eliminate imp3 and imp4	1
R.6	(i) a24 implies a25	(i) CR	Defect: Redundancy6 Cause: optional feature a24 implies a full-mandatory feature a25 is redundant Correction: eliminate imp5	1
R.7	(i) a26 implies a27 (ii) a26 implies a28	(i) CR (ii) CR	Defect: Redundancy7 Causes: both mandatory child features a27 and a28 are implied by their parent a26 are redundant Corrections: eliminate imp6 and imp7	1
R.8	(i) a30 implies a31 (ii) a32 implies a31	(i) CR (ii) CR	Defect: Redundancy8 Cause: multiple implication of an optional feature a31 Correction: eliminate imp9	0.5
R.9	(i) a36 implies a35 (ii) a36 implies a37	(i) CR (ii) CR	Defect: Redundancy9 Cause: implication on a37 is redundant after a35 is implied by a mandatory feature a36 Correction: eliminate imp11	0.5
R.10	(i) a39 implies a40 (ii) a41 implies a42	(i) CR (ii) CR	Defect: Redundancy10 Cause: mandatory feature a41 implies another mandatory feature a42 is redundant Correction: eliminate imp13	0.5
R.11	(i) a46 implies a45 (ii) a46 implies a47	(i) CR (ii) CR	Defect: Redundancy11 Causes: implications on mandatory features a45 and a47 are redundant Corrections: eliminate imp14 and imp15	1
R.12	(i) a49 excludes a50 (ii) a51 excludes a50	(i) CR (ii) CR	Defect: Redundancy12 Cause: multiple exclusion of an optional feature a50 Correction: eliminate exc4	0.5
R.13	(i) a53 excludes a54 (ii) a55 excludes a56	(i) CR (ii) CR	Defect: Redundancy13 Cause: mandatory feature a55 excludes another mandatory feature a56 is redundant Correction: eliminate exc6	0.5

Table 12 (continued)

Rules	Constraints description	FeatureIDE tool	Proposed method	Status (S)
R.14	(i) a57 implies a58 (ii) a58 implies a59 (iii) a59 implies a57	(i) CR (ii) CR (iii) CR	Defect: Redundancy14 Cause: implication from a59 to a57 is redundant Correction: eliminate imp18	0.5
R.15	(i) a60 implies a61 (ii) a61 implies a62 (iii) a60 implies a62	(i) CR (ii) CR (iii) CR	Defect: Redundancy15 Cause: implication from a60 to a62 is redundant Correction: eliminate imp21	0.5
R.16	(i) a63 implies a65	(i) CR	Defect: Redundancy16 Cause: implication from a63 to a65 is redundant Correction: eliminate imp22	1
R.17	(i) a68 implies a66	(i) CR	Defect: Redundancy17 Cause: implication from a68 to a66 is redundant Correction: eliminate imp23	1

CR: Constraint is redundant and could be removed.

6.2.3. Comparing accuracy of the proposed method with FeatureIDE tool

The proposed method is compared with FeatureIDE tool for accuracy. The experiments were carried out for redundancies by considering the practicability of the proposed method with FeatureIDE. Table 12 shows the results obtained after analyzing E-commerce system1 FM with redundancies using (i) FeatureIDE tool (as shown in column 3), and (ii) proposed method (as shown in column 4). The accuracy is represented with Status (S). When FeatureIDE finds accurate redundancy or constraint(s) for the cause of redundancy, the status is indicated with 1. Similarly, when it only identified one or half of the constraint out of two or more constraints involved in the cause of redundancy, the status is indicated by 0.5. Further, when FeatureIDE is unable to find accurate redundancy or constraints involved in the cause of redundancy or did not identify any redundancy, the status is indicated by 0. The number of models, number of rules and the status are represented by m , r , and $S_{m,r}$, respectively, $S_{m,r} \in [0, 1]$.

The computed values of S for E-commerce system1 FM including redundancies are demonstrated in Table 12. The value of Status (S) was calculated for each of the considered 17 rules for redundancies, which further were executed for 35 models as illustrated in Eq. (1). The final accuracy of FeatureIDE is 73.53% for redundancies which was computed based on the average of the values obtained from Eq. (1).

However, our method handled 100% of the considered redundancies with 0% false positives. Further, all rules individually and jointly identified as well as provided causes and corrections of all the expected redundancies, indicating 100% accuracy of the proposed method.

$$accuracy = \frac{1}{m} \sum_{m=1}^{35} \left(\frac{1}{r} \sum_{r=1}^{17} S_{m,r} \right) \quad (1)$$

6.3. Computational scalability

As shown in Fig. 6, the average execution time (in seconds) was calculated after executing entire rules to deal with redundancies in each one of the six FMs with features 5000, 10,000, 15,000, 20,000, 25,000 and 30,000 for testing the performance of proposed method. X-axis and Y-axis represent the number of features in all models and time respectively. The scalability of proposed method to deal with redundancies is determined by the plot.

It describes the minimum and the maximum time required by the queries to deal with redundancies in models with 5000 features and large-sized FMs with 30,000 features respectively. Results conclude that queries take a reasonable time of 0.185 sec to deal with redundancies in huge FM up to 30,000 features. The reliable and valid measures of execution time are acquired by executing the entire rules for 50 times on each of the 35 FMs. The overall execution time considered is the average of 50 executions for the entire rules over all the FMs.

6.4. Completeness, consistency and consistency gain of the set of redundancy rules

To measure the quality of rules, usage of fitness function is considered as an interesting issue in rule-based approaches (Zhou, Xiao, Tirpak & Nelson, 2003). The focus is on proving the completeness and consistency of the developed set of redundancy rules. In literature, various formulas exist which integrate completeness and consistency as shown in Eqs. (2) and (3) respectively (Bruha, 1997). Instead of using consistency metric, we are using consistency gain of rule (Michalski & Kaufman, 2001), as it considers the distribution of positive examples and negative examples in the training set (TS).

Here, a set of rules (a *ruleset*) represents the redundancy rules, where “ruleset” and “rule” are used as “rule” and as a component of a rule respectively. The number of negative examples and positive examples in the complete TS are represented using *NE* and *PE* respectively. Let r be a rule (or a ruleset) developed for that TS to cover its examples where ne



Fig. 6. Execution time, for the entire rules, per number of features, to deal with redundancies.

Table 13

Description of rule for the minimalism of set of redundancy rules.

Related rule	Canonical rule	Time (in sec)
R.3	R.5	0.005

and pe are the number of negative examples and positive examples covered by r , called negative and positive support respectively.

For r , the *completeness* (relative support or relative coverage), *consistency* and *inconsistency* (training error rate) are defined by Eqs. (2)–(4) respectively.

$$comp(r) = \frac{pe}{PE} \quad (2)$$

$$con(r) = \frac{pe}{pe + ne} \quad (3)$$

$$incon(r) = \frac{ne}{pe + ne} \quad (4)$$

A complete cover of the training examples is obtained if there is 100% *completeness* of a ruleset for a single class and *consistent* cover is obtained if there is 0% *inconsistency* of the ruleset.

The distribution of positive examples and negative examples in the TS is measured by the ratio $PE/(PE + NE)$. The distribution of positive examples and negative examples in the set covered by the rule is measured by the consistency $pe/(pe + ne)$. The consistency gain of the rule over the dataset distribution is given by the difference between $(pe/(pe + ne)) - (PE/(PE + NE))$.

To normalize the expression, we have divided it by $(1 - (PE/(PE +$

$NE)))$, or equivalently by $NE/(PE + NE)$. After rearranging the normalized expression, the consistency gain ($cons(r)$) is defined by Eq. (5).

$$cons(r) = \left(\left(\frac{pe}{pe + ne} \right) - \left(\frac{PE}{PE + NE} \right) \right) \times \frac{PE + NE}{NE} \quad (5)$$

This expression determines the value of consistency gain as (i) Zero, when the distribution of positive examples and negative examples covered by a rule is identical to the distribution in the entire TS (as a random guess), (ii) One, in case of perfect consistency, and (iii) Negative, this turns the rule to be less accurate than a pure random guess.

The normalized expression obtained above calculates the benefits of using the rule over making random guesses. If the rule generates poor results then the value of the benefit becomes negative. The *fitness function* for evaluating the rules is defined by Eq. (6).

$$fitness(r) = \begin{cases} 0, & \text{if } cons(r) < 0 \\ cons(r) \times \exp(comp(r) - 1), & \text{otherwise} \end{cases} \quad (6)$$

where $comp(r) = pe/PE$ is the completeness of rule.

The $\exp()$ function gives preference to the use of consistency gain of the rule, for measuring the quality of rule. The fitness function given above is simple and returns a normalized value between 0 and 1.

6.5. Minimalism of the set of redundancy rules

A rule should be comprised of at least (a) two features; one of these should be the root feature, and (b) one relationship that relate both features. It means that there is no model with a single feature, as a single feature does not lead to redundancy in FM. As our domain is FM, the elements features, relationships and cross-tree constraints can represent an entire FM. The notations (see Section 5.1.2) used for developing the set of redundancy rules can represent a FM where each developed rule comprises of at least two features represented in the form of *mandatory* or *optional* features and one cross-tree constraint represented in the form of *implication* or *exclusion*. Additionally, *alternative* and *or* relationships are represented by *cardinalities*.

The minimalism of proposed set of rules can be achieved by minimizing the number of related rules. The elimination of a related rule from the set of redundancy rules does not affect the expected outcome which comprises of identified redundancies with their causes and corrections. For instance, Table 13 represents that the elimination of related Rule R.3 from the set of redundancy rules does not affect the outcome, as the same redundancy can be identified using Rule R.5. Eliminating Rule R.3 from the set will reduce the overall execution time (0.08 sec) by 0.005 sec which in turn will improve the performance of proposed method.

The accuracy is defined by Eq. (7).

Table 14

Comparing proposed method with existing methods on the basis of number of rules and execution time to deal with redundancies.

Article	Description of features, constraints and defects	Total rules or criteria	Number of rules or criteria per defect type	Execution time (in sec) Existing methods	Proposed method
(Mazo, Lopez-Herrejon, Salinesi, Diaz & Egyed, 2011)	10,000	9 Rules	R (1), NR (3), Others (5)	Execution time of each rule < 0.14	Execution time of each rule < 0.008
(Salinesi & Mazo, 2012)	2000	15 verification criteria	R (6), Others (9)	Execution time of each verification operation < 19	Execution time of each rule < 0.008
(Felfernig et al. (2013))	172	–	–	HSDAG algorithm (100)	0.126
(Elfaki, Fong, Aik & Johar, 2013)	20,000	13 Rules	R (9), Others (4)	FASTDIAG algorithm (10.54)	Identification with cause (0.058)
(White, Benavides, Schmidt, Trinidad, Dougherty & Ruiz-Cortes, 2010)	2513 features, 2,833 constraints, R (563), NR (204)	–	–	Detection with cause (221.516)	Identification with cause and correction (0.07)
Proposed method	30,000	17 Rules	R (17)	16,546.44	3.475

R: redundancies, NR: defects other than redundancy, Others: Rules not for defects. Here, R (1) signifies that there is 1 rule or criteria for redundancy.

$$A_{max,r} - B_{rel,r} = C_{min,r} \quad (7)$$

$$\text{where, } B_{rel,r} = \begin{cases} \emptyset, & \text{if there is no related rule in the set} \\ B_{rel,r} \in A_{max,r} & \text{otherwise} \end{cases}$$

Here,

$A_{max,r}$: a set of maximum number of redundancy rules handled by our method

$B_{rel,r}$: a set of related rules

$C_{min,r}$: a minimal set of rules that include minimum number of rules which are enough to identify redundancies with their causes and corrections handled by the proposed method.

Moreover, the minimal set of rules leads to lower execution times and higher accuracies than a larger randomly chosen set of rules which suggests that there is no gain using additional rules than the minimum set of rules. However, rule R.3 is still required to deal with particular redundancy independently, although this redundancy is being handled in collaboration with rule R.5 as shown in [Table 13](#).

7. Results and discussion

The results of implementing our method to deal with redundancies are discussed in this section. We compare the results of our method with existing methods and threats to validity are also discussed.

7.1. Comparison with existing methods

This subsection illustrates the results obtained after comparing existing methods with our method based on various factors. Salinesi and Mazo ([Salinesi, Mazo & Diaz, 2010](#)) presented a typology of FM verification criteria. It includes 15 verification criteria where each criterion represents a case of defect. Their typology provided only 6 verification criteria for redundancies while our work classified FM redundancies into 17 cases (i.e., broadly into six major categories) in the form of a typology (as shown in [Fig. 1](#)). [Salinesi and Mazo \(2012\)](#) identified redundancy based on their verification criteria. Few researchers worked on the identification and cause of redundancies ([Elfaki, Fong, Aik & Johar, 2013](#); [White, Benavides, Schmidt, Trinidad, Dougherty & Ruiz-Cortes, 2010](#)), but none of them resolved redundancy. [Felfernig et al. \(Felfernig, Benavides Cuevas, Galindo Duarte & Reinfrank, 2013\)](#) have not given any details related to the implementation of recommended solutions for redundancies. Moreover, their explanations comprise of constraint sets which increase the difficulty to understand the anomaly for developers. However, the proposed method, in addition to identifying redundancies with their causes, also identified corrections in a user-friendly language ([Section 5.2.2](#)) which are comprehensible by SPL developers in resolving redundancies. A tool was developed by [Thüm, Kästner, Benduhn, Meinicke, Saake and Leich \(2014\)](#) that recommends solutions to resolve defects including redundancies. Their tool does not support solutions which require elimination of multiple relationships. However, our method provides corrections which incorporate elimination of multiple relationships. It is worth noting that the proposed method detected actual defective features and the cross-tree constraints for their causes and corrections, and not defective FMs. Additionally, correction provided by our method is minimal, as it targets the cross-tree constraints itself which are involved in the source of redundancy.

According to [Table 14](#), the proposed method not only deals with redundancies considered by existing works (column 1) but also handled other cases of redundancy ([Section 2](#)). It provides more rules (17) to deal with redundancies when compared to ([Elfaki, Fong, Aik & Johar, 2013](#); [Felfernig et al., 2013](#); [Mazo, Lopez-Herrejon, Salinesi, Diaz & Egyed, 2011](#); [Salinesi, Mazo & Diaz, 2010](#); [White, Benavides, Schmidt, Trinidad, Dougherty & Ruiz-Cortes, 2010](#)). The presented method takes less time to deal with redundancies (column 6) for particular cases (columns 2–4) when compared to existing methods (column 5). The results indicate the improved performance as compared to existing methods.

Further, it handled redundancies in huge FMs up to 30,000 features (column 2) in 0.185 s when compared to prior methods (column 2) resulting in enhanced scalability (in an extremely reasonable time). Our method is validated using real FMs and automatically-generated FMs available in SPLIT repository, as well as models generated using FeatureIDE tool in contrast to [Elfaki, Fong, Aik and Johar \(2013\)](#) who have used their own generated data sets for the validation of their method.

7.2. Threats to validity

Following are the validity threats that may affect the experimental results of presented method:

7.2.1. External validity

The FMs used in the experiments are mostly real-world models from SPLIT repository, some of them are automatically-generated models, which may cause a threat to external validity as these FMs are not reflecting real-world models. The problem size and model's complexity may vary with real-world FMs. Thus, the use of models created using FeatureIDE tool and automatically generated 3-CNF-FMs from SPLIT minimizes the aforementioned effect. To diminish the impact of other threads (for instance, threads of operating system) on the computed execution time, each of the FM is analysed independently. This impact is minimized by using the average of results attained after executing entire rules for 50 times on all models.

7.2.2. Internal validity

Additional cross-tree constraints and features are inserted in the input FMs to cause redundancies, and it is one of the threats to internal validity. Results obtained after transforming FMs comprise of these additional features and cross-tree constraints. The threat to internal validity is diminished as the proposed method identified all redundancies with their causes as well as corrections generated due to the additional features and cross-tree constraints, in a reasonable performance time of 0.185 sec in huge FMs up to 30,000 features. Further, FMs are automatically mapped into ontologies using parser. The threat to internal validity is diminished by obtaining the same results (the number of mandatory features, optional features, implications, exclusions, and cardinalities) by computing the accuracy of the mapping manually for all 35 FMs.

8. Related work

Numerous existing work focuses on dealing with redundancies in FMs is discussed in this section. For instance, [Salinesi and Mazo \(2012\)](#) analyzed FMs by representing them in Constraint Program (CP). A series of algorithms is proposed that verified the models against the typology of verification criteria ([Salinesi, Mazo & Diaz, 2010](#)) in single-view and multi-view Product Line Models (PLMs). Their approach only identified void models, false-optional features, dead features, false PLM, redundant constraints and wrong cardinality in FMs. The approach is limited to FMs up to 2000 features, as the solver used in this approach does not allow accommodating more than 5000 variables.

Later, the causes of FM anomalies are explained by [Felfernig et al. \(2013\)](#). The minimal diagnoses and minimal sets for non-redundant constraints are determined by discussing the FASTDIAG and FMCORE algorithms respectively. The model consistency can be attained by modifying or removing the minimal sets of constraints from the FM. FASTDIAG, independent of any solver is used to explain each defect. Reiter proposed Hitting Set Directed Acyclic Graph algorithm for detecting and fixing a conflict by determining the complete set of diagnoses ([Guo, Wang, Trinidad & Benavides, 2012](#)). Further, corrections for anomalies have been recommended by the authors in terms of redundancies and inconsistencies, however, no implementation details have been given for the same. Further, no information was provided for the number or types of defects in the FM. The evaluation of their approach is limited to FMs with 172 features available at SPLIT

repository. Additionally, understanding the anomaly becomes challenging for developers as explanations consist of constraint sets which are indirectly concerned to the model's structural information. The scalability of FASTDIAG for large-scale FMs is missing.

Mazo (2011) proposed the conformance checking of PLMs (namely FMs) based on Constraint Logic Programming (CLP). Nine conformance checking rules were identified and applied using GNU Prolog constraints solver. The elements related to each particular conformance rule are tested using CLP in extended feature models (EFMs). His approach is efficient and scalable to industry size FMs, as it was validated by implementing these rules on 50 FMs up to 10,000 features. Later, the variability in SPL is represented using FM and Orthogonal Variability Model (OVM) together, and further, Elfaki, Fong, Aik and Johar (2013) identified redundancies and dead features in Domain Engineering (DE) process using FOL rules. The efficiency of their work lies in the direct detection of redundancy in the DE. Their method has been validated using own generated data sets. Results conclude that these methods are scalable up to 20,000 features in a reasonable time.

Rincón, Giraldo, Mazo, Salinesi and Diaz (2015) transformed FMs into CPs. They identified false PLs, false-optional features, dead features and redundancies by analyzing CP using algorithms given by Salinesi and Mazo (2012). They identified MCSes (i.e. a minimal subset of relationships) of an unsolvable CP for detecting potential corrections of FM defects. MCSes should be filtered out of the FM to fix minimum single defect. Their method identified all MCSes for defects by systematically removing relationships from the FM. The identified corrections for redundancies include the elimination of constraints without changing the semantics of the model. Designers would easily understand the identified corrections, even without knowing constraint programming. Designers can decide on the correction according to their interests and possible MCSes. However, their method only provides corrections by removing relationships from the FM. Further, the method is evaluated using 78 models with a varied number of features, up to 120 dependencies.

Kowal, Ananieva & Thm (2016) proposed a generic algorithm to explain various anomalies such as void model, dead features, false-optional features and redundant constraints in FMs based on predicate logic. It explains each type of anomaly encoded in a Conjunctive Normal Form (CNF) and a set of assumptions based on initial truth values. These explanations are in a user-friendly way. They computed short explanations and benefited developers by highlighting the most significant parts of them that may be the possible source of an anomaly. Due to the support of their open-source tool in FeatureIDE, the scalability is evaluated with industrial-size FMs. However, the evaluation is restricted to models with 2,513 features and 2,833 constraints.

9. Conclusion

One of the major factors behind the successful derivation of defect free valid software products from SPL is the quality of FM. Defect due to redundancy in FMs is one of such defects which hinder the derivation of high-quality valid software products in SPL. An ontological FOL rule based method is proposed to deal with redundancies. FM redundancies are classified in the form of a typology (various categories and their cases). FM has been mapped to ontology based on predicate logic. FOL rules are developed and applied to the generated ontology using Prolog that identified redundancies with their causes and corrections in natural language. This information helps SPL developers to resolve redundancies. Further, these rules can be implemented independently and simultaneously as they are independent in nature. The method has been validated using 35 models with varied sizes up to 30,000 features which conclude that it is efficient, accurate and scalable. Thus, allows deriving redundancy free valid end products from the SPL and subsequently, improves its quality.

In future, the set of rules can be modified by adding new rules to deal with redundancies in different FM notations (EFM, decision model, OVM and textual variability language). Further, SWRL based rules can also be developed for the auto identification and correction of redundancies.

The presented method only resolves redundancies by eliminating redundant relationships. Consequently, an improvement in corrections includes eliminating redundant features in model.

CRedit authorship contribution statement

Megha Bhushan: Conceptualization, Methodology, Software, Validation, Funding acquisition. **José Ángel Galindo Duarte:** Investigation, Validation. **Piyush Samant:** Writing - original draft, Visualization. **Ashok Kumar:** Supervision, Writing - review & editing. **Arun Negi:** Project administration, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

Corresponding author Megha Bhushan, gratefully acknowledges the University Grants Commission (UGC), New Delhi, Government of India, for awarding her the RGNF (Grant no. F117.1/201415/RGNF201415SCJAM66324) to carry out this research work.

References

- Baader, F., Calvanese, D., McGuinness, D., Patel-Schneider, P., & Nardi, D. (Eds.). (2003). *The description logic handbook: Theory, implementation and applications*. Cambridge university press.
- Benavides, D., Segura, S., & Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6), 615–636.
- Bruha, I. (1997). Quality of decision rules: Definitions and classification schemes for multiple rules. *Machine learning and statistics, the interface*, 107–131.
- Clements, P., & Northrop, L. (2001). *Software Product Lines: Practices and Pattern* (3rd ed.). Addison-Wesley Professional.
- Elfaki, A. O., Fong, S. L., Aik, K. L. T., & Johar, M. G. M. (2013). Towards detecting redundancy in domain engineering process using first order logic rules. *International Journal of Knowledge Engineering and Soft Data Paradigms*, 4(1), 1–20.
- Felfernig, A., Benavides Cuevas, D. F., Galindo Duarte, J. Á., & Reinfrank, F. (2013). Towards anomaly explanation in feature models. Paper presented at the 15th International Configuration Workshop (ConfWS-2013) (pp. 117–124). CEUR-WS.
- 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>
- 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.
- Kowal, M., Ananieva, S., & Thüm, T. (2016, October). Explaining anomalies in feature models. Paper presented at the International Conference on Generative Programming: Concepts and Experiences (GPCE). ACM SIGPLAN Notices (Vol. 52, No. 3, pp. 132–143). ACM. New York.
- Lesta, U., Schaefer, I., & Winkelmann, T. (2015). Detecting and explaining conflicts in attributed feature models. arXiv preprint arXiv:1504.03483.
- Mazo, R. (2011). A generic approach for automated verification of product line models (Doctoral dissertation). Université Panthéon-Sorbonne-Paris.
- Mazo, R., Lopez-Herrejon, R. E., Salinesi, C., Diaz, D., & Egyed, A. (2011, July). Conformance checking with constraint logic programming: The case of feature models. Paper presented at the 35th Annual Computer Software and Applications Conference (pp. 456–465). IEEE.
- Michalski, R. S., & Kaufman, K. A. (2001). *Learning Patterns in Noisy Data: The AQ Approach*. Machine Learning and Its Applications.
- Northrop, L. (2008). *Software product lines essentials*. Pittsburgh: Software Engineering Institute Carnegie Mellon University.
- Rincón, L., Giraldo, G., Mazo, R., Salinesi, C., & Diaz, D. (2015). Method to identify corrections of defects on product line models. *Electronic notes in theoretical computer science*, 314, 61–81. <https://doi.org/10.1016/j.entcs.2015.05.005>
- Salinesi, C., Mazo, R., & Diaz, D. (2010, May). Criteria for the verification of feature models. In *Paper presented at the 28th INFORSID* (pp. 293–308).
- Salinesi, C., & Mazo, R. (2012). Defects in product line models and how to identify them. In *Software Product Line-Advanced Topic* (pp. 97–122). IntechOpen. <https://doi.org/10.5772/35662>.
- 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.
- Trinidad, P., Benavides, D., Durán, A., Ruiz-Cortés, A., & Toro, M. (2008). Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, 81(6), 883–896. <https://doi.org/10.1016/j.jss.2007.10.030>

Van Der Storm, T. (2007). In *Generic feature-based software composition* (pp. 66–80). Berlin, Heidelberg: Springer.

Von der Maßen, T., & Lichter, H. (2004, August). Deficiencies in feature models. In workshop on software variability management for product derivation-towards tool support (Vol. 44, p. 21).

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. <http://dx.doi.org/10.1016/j.jss.2010.02.017>.

Wielemaker, J. SWI-Prolog (Version 7.2.3), free software (2015). University of Amsterdam, Amsterdam.

Zhou, C., Xiao, W., Tirpak, T. M., & Nelson, P. C. (2003). Evolving accurate and compact classification rules with gene expression programming. *IEEE Transactions on Evolutionary Computation*, 7(6), 519–531.



Mr. Piyush Samant is currently working as an AI and Biomedical expert in Adventum Advanced Solution Pvt. Ltd. Bangalore, Karnataka, India. He has done M. E. in Electronics and Instrumentation Engineering from Thapar University, Patiala. He is also pursuing Phd from Thapar University, Patiala. His research interests include machine learning and expert systems.



Dr. Megha received her Ph.D. degree from Thapar University, Punjab, India. She is currently an Assistant professor in the School of Computing, DIT University, Dehradun, India. She has worked as Junior Research Fellow under UGC, New Delhi, Government of India from 2014 to 2016. She has also worked as Senior Research Fellow under UGC, New Delhi, Government of India from 2016 to 2018. She was awarded with fellowship by University Grants Commission (UGC), Government of India, in 2014. In 2017, she was a recipient of Grace Hopper Celebration India (GHCI) fellowship. She has filed 4 patents and published many research articles in international journals and conferences of repute. Her research interest includes Software quality, Software reuse, Ontologies, Artificial Intelligence, and Expert systems. She is also the reviewer and editorial board member of many international journals.



Dr. Ashok Kumar is currently an Assistant Professor in Chitkara University Research and Innovation Network (CURIN) Department, Punjab, India. He is PhD in Computer Science and Engineering from Thapar University, Punjab, India. He has 15+ years of teaching and research experience. He has filed 3 patents and published many articles in International Journals and Conferences of repute. His current areas of research interest include Cloud Computing, Internet of Things, and Mist Computing. His teaching interest includes Python, Haskell, Java, C/C++, Advanced Data structures and Data mining



José Ángel Galindo Duarte is with University of Seville, Spain. His interests include Recommender systems, software visualization, and variability-intensive systems.



Major Arun Negi is presently working with Deloitte USI, Hyderabad, India. Before joining Deloitte, he served as a Major in Indian Army, Government of India, India. He has research publications in conferences of repute. His research conferences of repute. His research research interests include Artificial Intelligence, Software product line, Software reuse and Expert systems.