A Tool Support for Automatic Detection of Duplicate Features during Software Product Lines Evolution

Amal Khtira, Anissa Benlarabi, Bouchra El Asri

IMS Team, SIME Laboratory, ENSIAS Mohammed V University
Rabat, Morocco

Abstract
Software product lines are continuously changing systems that must evolve to meet new customers' needs and new business strategies. Due to this change, many defects impact both the core platform and the specific applications of the product line. Thus, the verification of feature models has become one of the most crucial issues related to software product line engineering. Many tools have been proposed in literature to verify product line models but few have focused on the problem of semantic feature duplication. In this paper, we introduce FDDetector, a tool that aims at optimizing the evolution of software product lines by detecting feature duplication between the existing feature models and the specifications of the new evolutions.
Keywords: *Software Product Line Evolution, Feature Duplication, Tool Support, Automatic Verification, Natural Language Processing.*

1. Introduction

The reduction of cost and the enhancement of product quality are today among the most crucial challenges in software engineering. To deal with these issues, many development paradigms have been proposed, among which, Software Product Line Engineering (SPLE) [1] has emerged as a discipline that aims mainly at producing products of high quality and reducing the time to market and the cost of development. SPLE consists of creating a core platform that contains the common and variable assets of a domain. Then, customized applications are generated based on this platform. Thus, two processes are separated in SPLE, namely the domain engineering and the application engineering [2]. The first process involves the determination of variability and commonality of the product line, while the second process is responsible for the derivation of individual applications that respond to specific needs of customers.

Software Product Lines are large scale systems that last over time. These systems have to evolve constantly to meet new customers' requirements and new technology. The evolution of SPLs is more complex than other systems because the change happens both in the domain assets of the core platform and the application assets related to derived products. Several papers in the literature have addressed issues related to SPL evolution such as requirements traceability [3], the co-evolution of domain and application models [4], the evolution modeling [5], and model defects [6].

Our area of research concerns model defects caused by SPL evolution, especially duplication [7]. Based on the literature review, we have identified some tools [8][9][10][11][12] dealing with the detection and correction of defects. An analysis of these tools has shown that most of them focus on specifications rather than feature models. In addition, these tools deal specifically with consistency checking, while duplication is not taken into account. To overcome these gaps, we propose a tool that aims at detecting duplication in feature models, and between feature models and the specifications of new SPL evolutions. The motivation for this work is that duplication causes the increase of cost and time-to-market and impacts negatively the quality of products, which contradicts the main objectives of SPLs.

The proposed tool is based on a two-process framework. The first process consists of transforming the existing feature models and the specifications of new evolutions into a more formal representation. The second process involves the detection of feature duplication through a set of algorithms.

The remainder of the paper is structured as follows. Section 2 gives an overview of the background of our work, namely software product line evolution and feature duplication. Section 3 describes our approach for detecting feature duplication when evolving software product lines. In Section 4, we present the architecture and functionalities of the proposed tool. The tool is evaluated in Section 5 using a case study. Section 6 compares out tool with related tools. Finally, Section 7 concludes the paper and describes future work.

2. Background and Motivations

In this section, we introduce the background of our study. First, we present the SPLE paradigm and we discuss SPL evolution challenges. Then, we highlight the problem of feature duplication when evolving SPLs.

2.1 Software Product Line Evolution

Formerly, the cost of hardware in a project was much higher than software. Thus, companies used to develop single software and little attention was given to reuse. Nowadays, software has become the most expensive component even in large projects such as aeronautics and nuclear field. The cost becomes more significant in the case of long living systems that evolve over time. At the aim of reducing the cost of development, deployment and maintenance, many paradigms have been proposed, among which, Software Product Line Engineering (SPLE) has emerged as an approach that promotes software reuse by using a core platform to create different products according to distinct needs of customers. The main objectives of the SPLE approach are the reduction of time to market, the reduction of cost and the enhancement of product quality [2].

A Software Product Line (SPL) is defined by [1] as a set of related systems that address a market segment and that are created from a common set of core assets. Software product lines are built around a domain model that contains the common and variable features of the system, and a multitude of specific applications generated by binding the domain variability.

In general, software product lines are long lived systems that incur significant evolution throughout their service life. This evolution allows companies to align their products with new business strategies, new customers' requirements and new technology challenges. Hence, a great importance must be assigned to the maintenance process of SPLs. For this, many studies in the literature have dealt with issues related to SPL evolution. Some issues are explained in what follows.

Evolution and Traceability [3][13][14]: The approaches dealing with traceability trace the links between the different assets of a SPL or between its subsequent releases. This enables to assess the change history and to highlight possible inconsistencies, which helps anticipate future steps of evolution and estimate the cost of changes.

Evolution modeling [15][16][17]: These studies consists of defining a strategy for change management and identifying rigorous and controlled steps of evolution, which simplifies the evolution process.

Co-evolution of artefacts [18]: Some papers have focused on the necessity of managing the co-evolution of different artefacts of a SPL. For instance, Passos [18] performed a thorough analysis of a specific Linux kernel release. He captured the co-evolution between the variability model, the makefiles which contains the mapping between features and compilation units, and the source code. As a result of this analysis, he proved that considering changes only in the variability model may lead to incorrect conclusions.

Co-evolution of domain and application models [4]: SPL evolution is more complex than single software evolution because two levels of change have to be considered, the level of the core platform and the level of derived products. However, due to time pressure and tight deadlines, a specific product can evolve independently without taking into consideration the evolution of the platform. As a result, instead of having a set of applications derived from the same platform, we end up with a set of single applications. Thus, a management of the co-evolution of domain and application models becomes necessary to avoid the software aging phenomenon [19].

Model verification [17][20][21]: Software evolution consists of adding new features or modifying or deleting the existing ones. This evolution impacts all assets of the product line, but specifically the model, which is considered the input of the other assets. Therefore, a verification of the model is necessary to detect potential defects and ensure its integrity and correctness. In our study, we focus on this specific challenge which is the verification of SPL models during evolution and the detection of the resulted model defects.

2.2 Feature Duplication

As stated before, the SPL evolution can be the source of many model defects. In our work, we carried out a literature review whose objectives are to list the different model defects discussed in the literature, determine the different solutions proposed to deal with these defects, identify the limitations of these approaches and find potential area of research. As a result of this review, we listed the following model defects:

- **Inconsistency:** A contradiction between two or more features, requirements or functionalities [22].
- **Incompleteness:** The lack of necessary information related to a feature or requirement [9].

- **Incorrectness:** The non-correspondence of a feature with the requirements imposed in the specification [20].
- **Ambiguity:** When a feature has more than one interpretation [9].
- **Redundancy:** The presence of reusable elements and variability constraints among them that can be omitted from the Product Line Model (PLM) without loss of semantics on the PLM [6].
- **Unsafety:** It happens when the behavior of existing products is affected by a new evolution [23].
- **Duplication:** To have the same thing expressed in two or more places. Duplication can happen in specifications, processes and programs [24].

Compared to other defects, we found that feature duplication is the one who received little interest in literature, at least in the analyzed papers. Even when a paper addresses this issue, it focuses generally on code duplication or code cloning [25][26][27]. Consequently, we decided to deal with the problem of duplication in the feature level, which is an early stage of the product lifecycle, which enables to avoid duplication in the next steps of the project. Other motivations lead us to tackle this problem. Indeed, the introduction of duplication in a SPL causes a waste of time, money and effort by implementing repeatedly the same requirements, which contradicts the main goals of SPLs, namely the reduction of time-to-market and the reduction of cost. Moreover, the independent evolution of duplicate features may cause inconsistencies and contradictions in the model, which impacts negatively the product quality. In addition, feature duplication causes also code duplication, and results in many cloning-related problems such as the recurring-bug problem and the increase of maintenance effort [28]. A solution is thus necessary to detect duplicate features in an early stage of evolutions, which helps avoid their inclusion into the existing models from the very beginning.

3. Approach Description

In this section, we present the proposed framework for detecting and correcting feature duplication caused by SPL evolution. We first give a short definition of the basic concepts. Then, we present the overview of the framework and we describe its processes.

3.1 Basic Concepts

Before going any further, we will give an insight of the basic concepts used in the framework.

Feature: A feature is a distinctively identifiable functional abstraction that must be implemented tested, delivered, and maintained [29].

Feature Model: It is the description of all the possible features of a software product line and the relationships between them. The most common representation of feature models is the FODA feature diagram proposed by [30]. The feature diagram is a tree-like structure where a feature is represented by a node and sub-features by children nodes. In basic feature models, there are four kinds of relationships between features:

- Mandatory: It exists in all products.
- Optional: It is not present in all products.
- Alternative: Only one option can be chosen from a set of features.
- Or: One or more features may be included in the product.

Specification: Requirements specification is a description of the intended behavior of a software product. It contains the details of all the features that have to be implemented during an evolution of the system.

Variation Point: Variation points are places in a design or implementation that identify the locations at which variation occurs [31].

Variant: It is a single option of a variation point and is related to the latter using a variability dependency [32].

Duplication: We consider that two features are duplicated if they have the same semantics or that they satisfy the same functionality [7]. In our approach, duplication can happen in three levels: In feature models, in specifications or between specifications and feature models.

3.2 The Framework Overview

The objective of our work is to detect duplication when evolving SPL feature models. The solution we propose consists of a two-process framework [33]. The first process consists of transforming the existing feature models and the specifications into a formal presentation, while the second process involves the detection of feature

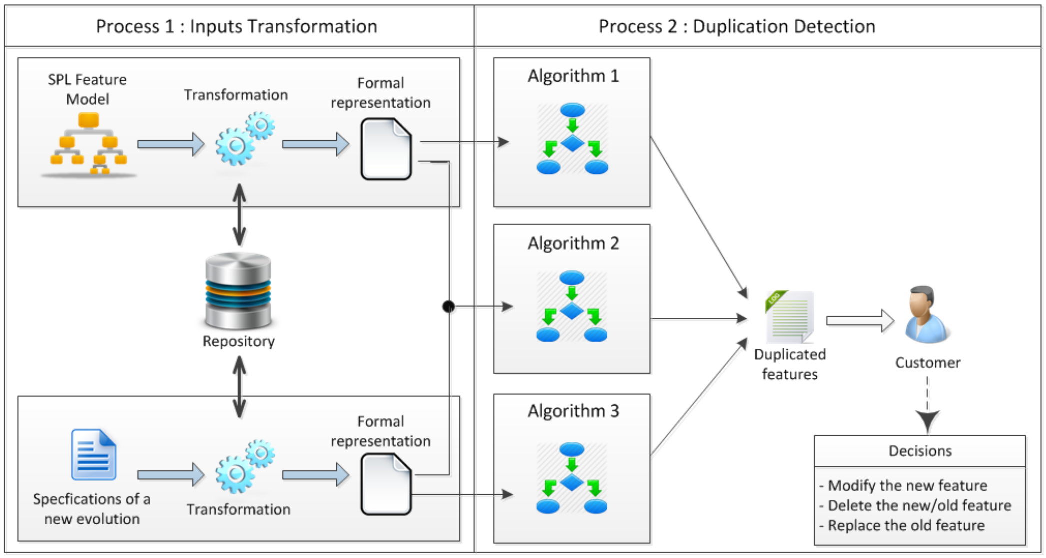


Fig. 1 The overview of the framework

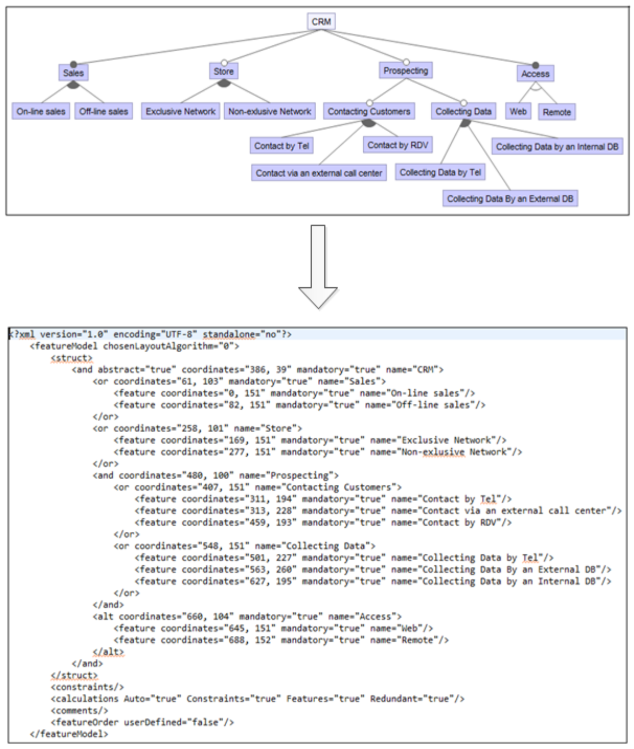
duplication using a set of algorithms. The overview of the framework is depicted in Fig. 1. In what follows, we will explain in details the framework processes.

3.2.1 Process 1: Inputs Transformation

In many software product lines, the variability is modeled using the FODA feature model [30] that we use in our approach. However, the requirements related to new evolutions of the system are documented using natural language because it is the easiest way for customers to express their needs. These two ways of modeling do not enable the detection and correction of defects. Thus, the aim of this process is to transform both the existing feature models of a SPL and the specifications of its evolutions to a more formal presentation.

Feature-oriented software development (FOSD) [34] is a paradigm based on the FODA method. The goal of this paradigm is to generate automatically software products based on the feature models. Hence, tools such as FeatureIDE [35] have been proposed to formalize the representation of feature models and allow the automatic selection of features of derived applications. In our approach, we opt for this tool to transform feature models to XML. Fig. 2 shows an example of a feature model created by FeatureIDE and its generated XML.

As for specifications, we adopt a natural language processing (NLP) approach to transform them into the same form of feature models. NLP is a branch of artificial intelligence that aims at analyzing and understanding human language in order to interface with computers in both written and spoken contexts. This approach allows the syntax and semantic parsing of a text. The syntax parsing analyzes the specifications and generates the syntactic tree based on the English grammar, while the semantic parsing extracts the meaning of the sentences.



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
    <featureModel chosenLayoutAlgorithm="0">
        <struct>
            <and abstract="true" coordinates="386, 39" mandatory="true" name="CRM">
                <or coordinates="61, 103" mandatory="true" name="Sales">
                    <feature coordinates="0, 151" mandatory="true" name="On-line sales"/>
                    <feature coordinates="82, 151" mandatory="true" name="Off-line sales"/>
                </or>
                <or coordinates="258, 101" name="Store">
                    <feature coordinates="169, 151" mandatory="true" name="Exclusive Network"/>
                    <feature coordinates="277, 151" mandatory="true" name="Non-exlusive Network"/>
                </or>
                <and coordinates="480, 100" name="Prospecting">
                    <or coordinates="407, 151" name="Contacting Customers">
                        <feature coordinates="311, 194" mandatory="true" name="Contact by Tel"/>
                        <feature coordinates="313, 228" mandatory="true" name="Contact via an external call center"/>
                        <feature coordinates="459, 193" mandatory="true" name="Contact by RDV"/>
                    </or>
                    <or coordinates="548, 151" name="Collecting Data">
                        <feature coordinates="501, 227" mandatory="true" name="Collecting Data by Tel"/>
                        <feature coordinates="563, 260" mandatory="true" name="Collecting Data By an External DB"/>
                        <feature coordinates="627, 195" mandatory="true" name="Collecting Data by an Internal DB"/>
                    </or>
                </and>
                <alt coordinates="660, 104" mandatory="true" name="Access">
                    <feature coordinates="645, 151" mandatory="true" name="Web"/>
                    <feature coordinates="688, 152" mandatory="true" name="Remote"/>
                </alt>
            </and>
        </struct>
        <constraints/>
        <calculations Auto="true" Constraints="true" Features="true" Redundant="true"/>
        <comments/>
        <featureOrder userDefined="false"/>
    </featureModel>
```

Fig. 2 A feature model created in FeatureIDE

In [36], we have explained in details the different steps of parsing, as depicted in Fig. 3:

- Sentence Detector: It enables the separation of sentences by putting each sentence in a different line.
- Tokenizer: It divides each sentence into tokens (e. g. noun, verb, number).
- Parser: It converts the sentence into a tree that represents the syntactic structure. Each word of the sentence is marked with a POS tagger (Part-Of-Speech tagger) that represents the role of this word in the English grammar.
- Entity Detector: It detects semantic entities in the sentences, especially variation points and variants in our case. This component is based on a repository where all the product line features are stored and categorized.

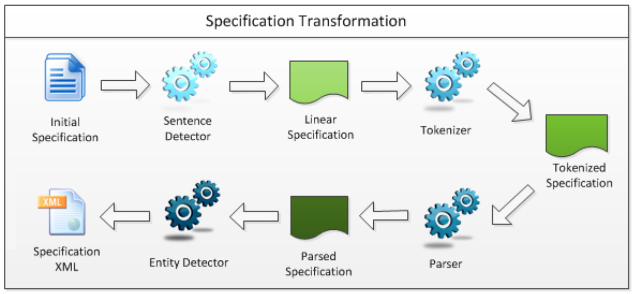


Fig. 3 Specification Transformation

3.2.2 Process 2: Duplication Detection

In a previous work [37], we stated that the operation of duplication detection has to be performed in three steps. In each step, we propose one or more algorithms. In what follows, we will detail each of these steps.

Detecting duplication in the existing feature models:
It is a one-shot operation that can be performed once the domain and application models are ready-to-use. The algorithm to detect duplication in feature models contains four steps:

- Generate an equivalent XML for the specification by replacing the name of every node (variation point or variant) with its associated key synonym in the dictionary.
- Put in alphabetical order the variation points and the variants of each variation point.
- Detect and remove duplicated variants for each variation point.
- Compare between the variants of all the variation points in order to detect duplication in the whole specification.

Detecting duplication in specifications:
This algorithm is the same as the previous algorithm because the representation of feature models and specifications was unified.

Detecting duplication between the feature models and the specifications:
In this level, two algorithms are proposed [33]. The first algorithm detects duplication between specifications and the application model, while the second algorithm detects duplication between specifications and the domain model. Fig. 4 illustrates the algorithm that corresponds to the second verification.

The two algorithms contain the following steps:

- Compare each variation point of the specification with the variation points of the domain feature model (or the application model).
- When an equivalent is found in the latter, compare between the variants corresponding to the variation point of the specification and the variants related to the equivalent variation point of the domain model (or the application model).
- If a variant is detected, this means that the feature corresponding to the pair (variation point, variant) is duplicated.

For each algorithm, duplications and their locations are stored in a log file that will be sent to the user so that he makes his decisions regarding the concerned features (e. g., Modify the new feature, delete the new/old feature, replace the old feature).

Algorithm 1 Detecting duplication between the specification and the domain model

```
Principal Lookup :
for each $p_i \in P$ do
  for each $pd_k \in PD$ do
    if $Equiv(p_i, pd_k)$ then
      Secondary Lookup :
      for each $v_{ij} \in V_i$ do
        for each $vd_{kl} \in VD_k$ do
          if $Equiv(v_{ij}, v_{kl})$ then
            $NoticeDuplication(p_i, v_{ij}, pd_k, v_{kl})$
            $Continue$ Secondary Lookup
          end if
        end for
      end for
      $Continue$ Principal Lookup
    end if
  end for
end for
```

Fig. 4 The algorithm to detect duplication between the specification and the domain model

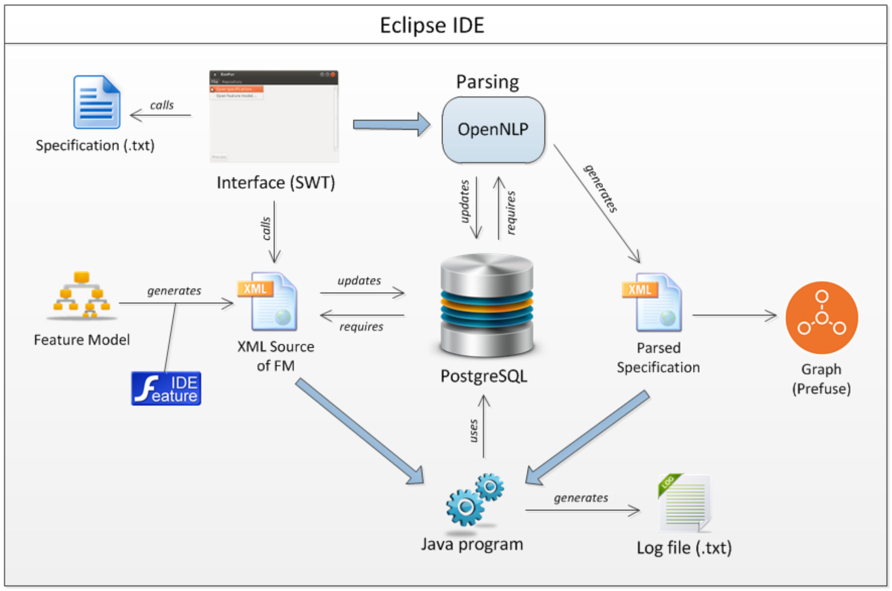


Fig. 5 The embedding of FDDetector

4. Tool Support for Duplication Detection

In order to instantiate the proposed framework, we have developed a tool that we called FDDetector (Feature Duplication Detector). In this section, we present the main functionalities supported by this tool and we describe its architecture.

4.1 Main Functionalities

As explained in the previous section, the goal of the proposed tool is to detect duplication during SPL evolution. Thus, the main functionalities provided by FDDetector are the following:

- The import of a textual specification in order to process it and transform it to XML.
- The detection of duplicate features in the processed specification.
- The import of the XML format of a feature model.
- The detection of duplicate features in the feature model.
- The comparison of features between a feature model and a specification to detect duplication between them.
- The creation of the repository and its initiation by a domain feature model.

Other functionalities are also required to enable an efficient duplication detection, such as:

- The generation of a graph that enables the visualization of duplicate features.
- The binding of new variants with the corresponding variation points in the repository.
- The refreshment of the repository after the binding of new features.
- The re-parsing of the specification after a modification of the repository.
- The manual update of the repository.
- The update of the repository based on a new specification or a new feature model.
- The generation of a log containing the detected duplicate features and their locations.
- The sending of the log to the user via email.

4.2 Design and Implementation

In order to implement the functionalities described earlier, we designed the embedding of FDDetector as depicted in Fig. 5.

The tool is a thick-client application built on Eclipse IDE, which is an open source IDE distinguished from its competitors by its adaptability and its large community of plugins creators. In our development, we will use a set of Eclipse plugins to implement different features. In addition, we will use java code to implement the algorithms of duplication detection.

For the creation of feature models and the generation of their XML source, we use FeatureIDE [35]. FeatureIDE is an open-source framework based on Eclipse. It supports all phases of FOSD, namely domain analysis, requirements analysis, domain implementation, and software generation.

The interface of our tool is created using SWT [38], which is an open source widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented. In our case, the development is done on Windows. The facilities of this OS are hence used to create the tool interface.

Both textual specifications and XML feature models can be opened via the main interface. The processing of specifications is performed using Apache OpenNLP Library [39], which is a machine learning based toolkit for the processing of natural language text. It supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and coreference resolution. It also includes an evaluation tool that measures the precision of entity recognition and provides information about the accuracy of the used model.

The content of the repository is stored using PostgreSQL [40], which is an open source object-relational database system. In the repository are stored all the domain features, their categories, and their synonyms. For a set of synonyms, we define a key synonym that we will use in the comparison between features.

In order to visualize the processed specifications and the duplicate features, we opt for Prefuse [41]. It is an open source toolkit that provides a visualization framework for the Java programming language. It supports a rich set of features for data modeling, visualization, and interaction.

5. Evaluation

To evaluate our solution, we use a CRM (Customer Relationship Management) product line, and specifically, we consider one of its derived applications. The feature model of this application is depicted in Fig. 6.

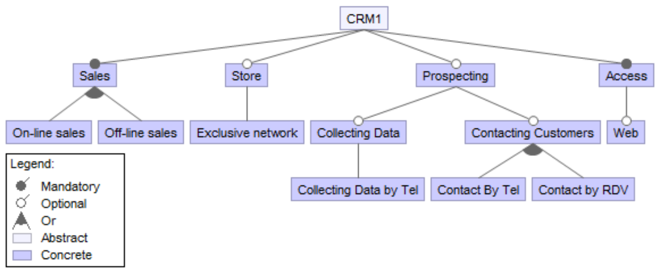
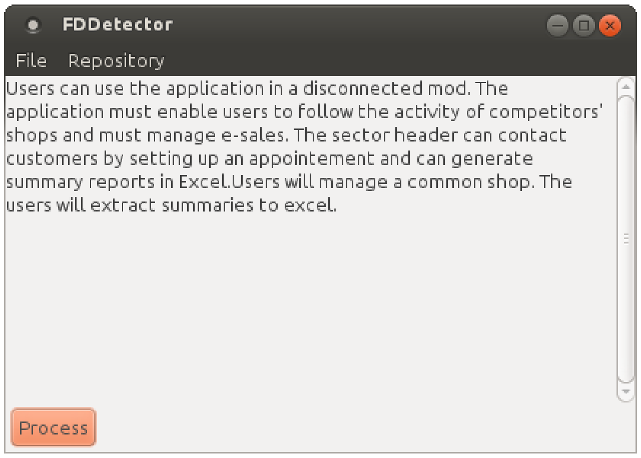


Fig. 6 The feature model of the CRM application

It is a web application that enables the management of on-line and off-line sales and the management of exclusive network stores. The sector header collects customers' information using phone calls, and he contacts them by telephone or by setting up an appointment.

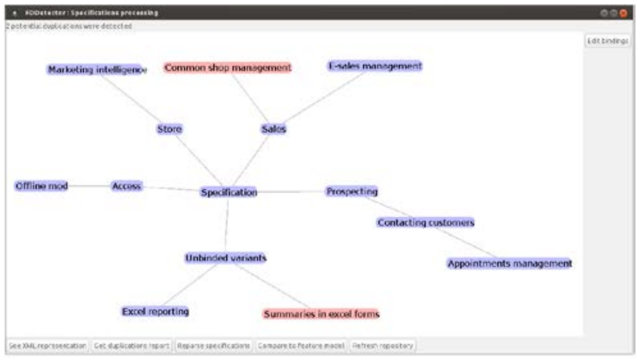
In our test, we use the specification of a new evolution of this application. In order to verify the algorithm of duplication detection in specifications, we added intentionally two duplicate features in the specification illustrated in Fig. 7.



Users can use the application in a disconnected mod. The application must enable users to follow the activity of competitors' shops and must manage e-sales. The sector header can contact customers by setting up an appointement and can generate summary reports in Excel.Users will manage a common shop. The users will extract summaries to excel.

Fig. 7 The specification of the new evolution

After the specification is processed and the algorithm is applied, we obtain the generated XML of the specification and a graph corresponding to this XML. This graph is depicted in Fig. 8. This representation facilitates the visualization of the different new features introduced by the specification and distinguishes the duplicate features by presenting them in a different color.



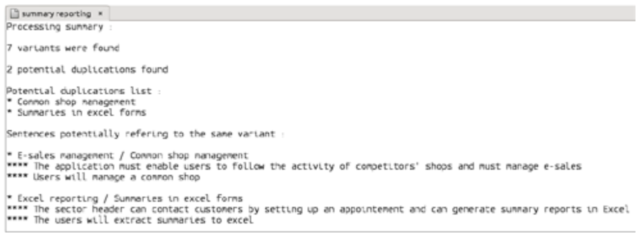
2 potential duplications were detected

Fig. 8 The graph corresponding to the specification

As a result of the algorithm, two duplications are detected ("Summaries in Excel form" and "Common shop management"). The number of duplications is displayed in the top left corner of the interface. Moreover, the duplicate features are colored red in the graph.

The tool also distinguishes the new variants added by the specification by relating them to the node "unbinded variants". The interface gives the possibility to the user to bind these variants with a variation point from the repository. In case of binding new variants, the user can also refresh the repository and re-parse the specification.

The log generated for our example is presented in Fig. 9. It contains the number of variants in the specification and the number of detected duplications. It also displays the sentences of the specification that contain duplication. This log will be sent to the customer by email so that he can make his decisions about the duplicate features (Remove the new/old feature, replace the existing feature, etc…).



```
Processing summary :

7 variants were found

2 potential duplications found

Potential duplications list :
* Common shop management
* Summaries in excel forms

Sentences potentially refering to the same variant :

* E-sales management / Common shop management
**** The application must enable users to follow the activity of competitors' shops and must manage e-sales
**** Users will manage a common shop

* Excel reporting / Summaries in excel forms
**** The sector header can contact customers by setting up an appointement and can generate summary reports in Excel
**** The users will extract summaries to excel
```

Fig. 9 The log of the performed test

So far, we have developed the duplication detection in specifications. In our future work, we intend to implement the algorithms of detecting duplication in feature models and between the specifications and the feature models. We also intend to apply the different algorithms to a product line with a large number of requirements in order to obtain more accurate results.

6. Related Tools

Many tools have been proposed to support the verification of models and the detection of defects. In this section, we present some of these tools.

Marama [8] is an automated tracing tool that enables users to capture their requirements and automatically generate the Essential Use Cases (EUC). This tool supports the inconsistency checking between the textual requirements, the abstract interactions and the EUCs. However, it does not support the detection of semantic duplication and focuses only on specifications, while our tool detects duplication between specifications and feature models.

QuARS (Quality Analyzer for Requirement Specifications) is a tool proposed by Lami et al [9]. This tool performs an initial parsing of the specifications in order to detect automatically specific linguistic defects, namely inconsistency, incompleteness and ambiguity. Although QuARS converges with our tool in the fact that it is based on parsing of natural language requirements to detect defects. However, it is limited to syntax-related issues of a natural language document, while our approach focuses on a semantic problem which is feature duplication.

CIRCE [10] is a system that uses natural language processing to extract information from natural language requirements and allows the checking and measurement of requirements consistency and produces functional metric reports. Our tool is different because, on the one hand, it focuses on semantic duplication in SPLs and not on inconsistency in general, and on the other hand, because our final goal is not only to detect duplication in textual requirements, but between the existing feature models and the specifications of new evolutions.

Requiline [11] is a requirement engineering tool for the management of software product lines. This tool provides many capabilities such as feature modeling, product configuration and consistency checking. Apart from the consistency checking, RequiLine does not perform any of the other analysis operations identified on feature models, especially duplication detection.

VMWare [12] is a tool that enables the verification of structural and semantic correctness of models derived from the FORE metamodel. This tool allows the structural verification of constraints, but does not support all semantic correctness properties, especially feature uniqueness.

Table 1 presents a comparison between the analyzed tools based on the functionalities they provide.

Table 1: Comparison of related tools

	Marama	QuARS	CIRCE	Requiline	VMWare	FDDetector
Specifications	+	+	+	+	-	+
Feature Models	-	-	-	+	+	+
Semantic correction	-	-	-	-	~	~
Duplication Detection	-	-	-	-	-	+
Consistency checking	+	+	+	+	+	-
NLP	-	+	+	-	-	+
XML	-	-	-	+	+	+
Machine Learning	-	-	-	-	-	~

7. Conclusion and Future Work

In this paper, we introduced the first prototype of FDDetector, which is a tool for the detection of feature duplication in the feature models and specifications related to software product lines. Our goal is to optimize the evolution of software product lines by detecting the duplication in an early stage of evolutions, which enables to reduce the time to market and the cost of development. So far, we have implemented the detection of duplication in natural language specifications and we evaluated the efficacy of the solution through a product line that contains a limited number of features. In future work, we intend to implement other functionalities such as: (i) detecting duplication in feature models; (ii) detecting duplication between specifications and feature models; (iii) using machine learning to update the repository; and (iv) supporting other formats of specifications and feature models. In addition, we will perform further evaluation through large scale and real-life software to ensure the effectiveness of our solution.

References

[1] P. Clements, and L. Northop, Software Product Lines - Practices and Patterns, Boston: Addison-Wesley, 2002.

[2] K. Pohl, G. Böckle, and F. Van Der Linden, Software Product Line Engineering Foundations, Principles, and Techniques, Berlin, Germany: Springer-Verlag, 2005.

[3] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J. C. Royer, A. Rummler, and A. Sousa, "A model-driven traceability framework for software product lines", Software and Systems Modeling, Vol. 9, No. 4, 2010, pp. 427-451.

[4] A. Benlarabi, B. El Asri, and A. Khtira, "A co-evolution model for software product lines: An approach based on evolutionary trees", in the 2d World Conference on Complex Systems (WCCS), IEEE, Nov. 2014, pp. 140-145.

[5] A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski, "Model-driven support for product line evolution on feature level", Journal of Systems and Software, Vol. 85, No. 10, 2010, pp. 2261-2274.

[6] R. Mazo, "A generic approach for automated verification of product line models", Ph.D. thesis, Pantheon-Sorbonne University, Paris, France, 2011.

[7] A. Khtira, A. Benlarabi, and B. El Asri, "Towards Duplication-Free Feature Models when Evolving Software Product Lines", in the 9th International Conference on Software Engineering Advances (ICSEA'14), Oct. 2014, pp. 107-113.

[8] M. Kamalrudin, J. Grundy, and J. Hosking, "Managing consistency between textual requirements, abstract interactions and Essential Use Cases", in the 34th Computer Software and Applications Conference (COMPSAC), IEEE, July 2010, pp. 327-336.

[9] G. Lami, S. Gnesi, F. Fabbrini, M. Fusani, and G. Trentanni, "An automatic tool for the analysis of natural language requirements", Informe tcnico, CNR Information Science and Technology Institute, Pisa, Italia, Sept. 2004.

[10] V. Ambriola, and V. Gervasi, "Processing Natural Language Requirements", in the 12th IEEE Conference on Automated Software Engineering (ASE'97), IEEE Computer Society Press, Nov. 1997, pp. 36-45.

[11] T. von der Maßen, and H. Lichter. "Requiline: A requirements engineering tool for software product lines", Lecture notes in computer science, 2004, pp. 168-180.

[12] C. Salinesi, C. Rolland, and R. Mazo. "VMWare: Tool support for automatic verification of structural and semantic correctness in product line models", in International Workshop on Variability Modelling of Software-intensive Systems (VaMos), 2009.

[13] L. Passos, K. Czarnecki, S. Apel, A. Wasowski, C. Kästner, and J. Guo, "Feature-oriented software evolution", in the 7th International Workshop on Variability Modelling of Software-intensive Systems, ACM, 2013, p. 17.

[14] A. Goknil, I. Kurtev, K. van den Berg, and J. Veldhuis, "Semantics of trace relations in requirements models for consistency checking and inferencing, Software Systems Modeling", Springer, Vol. 10, No. 1, 2011, pp. 31-54.

[15] S. Urli, M. Blay-Fornarino, P. Collet, and S. Mosser, "Using composite feature models to support agile software product line evolution", in the 6th International Workshop on Models and Evolution, 2012, pp. 21-26

[16] S. A. Ajila, and A. B. Kaba, "Evolution support mechanisms for software product line process", Journal of Systems and Software, Elsevier, Vol. 81, No 10, 2008, pp. 1784-1801.

[17] D. Romero, S. Urli, C. Quinton, M. Blay-Fornarino, P. Collet, L. Duchien, and S. Mosser, "SPLEMMA: A generic framework for controlled-evolution of software product lines", in the 17th International Software Product Line Conference, 2013, pp. 59-66.

[18] L. Passos, J. Guo, L. Teixeira, K. Czarnecki, A. Wasowski, and P. Borba, "Coevolution of variability models and related artifacts: A case study from the linux kernel", in the 17th International Software Product Line Conference, 2013, pp. 91-100.

[19] D. L. Parnas, "Software aging", in the 16th international conference on Software engineering, 1994, p. 279-287.

[20] D. Zowghi, and V. Gervasi, "On the interplay between consistency, completeness, and correctness in requirements evolution", Information and Software Technology, Vol. 46, No. 11, 2004, pp. 763-779.

[21] J. Guo, and Y.Wang, "Towards consistent evolution of feature models", in Software Product Lines: Going Beyond, Springer Berlin Heidelberg, 2010, pp. 451-455.

[22] B. Nuseibeh, "To be and not to be: On managing inconsistency in software development", in the 8th International Workshop on Software Specification and Design, IEEE, Mar. 1996, pp. 164-169.

[23] L. Neves, L. Teixeira, D. Sena, V. Alves, U. Kulezsa, and P. Borba, "Investigating the safe evolution of software product lines", ACM SIGPLAN Notices, Vol. 47, No. 3, 2012, pp. 33-42.

[24] A. Hunt, and D.Thomas, "The pragmatic programmer: from journeyman to master", Addison-Wesley Professional, 2000.

[25] S. Schulze, "Analysis and Removal of Code Clones in Software Product Lines", Ph.D. thesis, Magdeburg University, 2012.

[26] Y Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An exploratory study of cloning in industrial software product lines", in the 17th European Conference on Software Maintenance and Reengineering (CSMR'13), IEEE, Mar. 2013, pp. 25-34.

[27] T. Mende, F. Beckwermert, R. Koschke, and G. Meier, "Supporting the grow-and-prune model in software product lines evolution using clone detection", in the 12th European Conference on Software Maintenance and Reengineering (CSMR'08), IEEE, Apr. 2008, pp. 163-172.

[28] L. Aversano, L. Cerulo, and M. Di Penta, "How clones are maintained: An empirical study", in the 11th European Conference on Software Maintenance and Reengineering (CSMR'07), IEEE, March 2007, pp. 81-90.

[29] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A feature-oriented reuse method with domain-specific reference architectures", Annals of Software Engineering, Vol. 5, No. 1, 1998, pp. 143-168.

[30] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21, Carnegie Mellon University, Software Engineering Institute, Nov. 1990.

[31] I. Jacobson, M. Griss, and P. Jonsson, Software Reuse. Architecture, Process and Organization for Business Success, Addison-Wesley, ISBN: 0-201-92476-5, 1997.

[32] S. Creff, "Une modélisation de la variabilité multidimensionnelle pour une évolution incrémentale des lignes de produits", Doctoral dissertation, University of Rennes 1, 2003.

[33] A. Khtira, A. Benlarabi, and B. El Asri, "An Approach to Detect Duplication in Software Product Lines Using Natural Language Processing", in the Mediterranean Conference on Information and Communication Technologies (MEDICT'14), May. 2015, *under publication*.

[34] S. Apel, and C. Kästner, "An Overview of Feature-Oriented Software Development", Journal of Object Technology (JOT), Vol. 8, 2009, pp. 49-84.

[35] C. Kästner, T. Thüm, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, and S. Apel, "FeatureIDE: A Tool Framework for Feature-Oriented Software Development", in the 31st International Conference on Software Engineering, 2009, pp. 611-614.

[36] A. Khtira, A. Benlarabi, and B. El Asri, "Detecting Feature Duplication in Natural Language Specifications when Evolving Software Product Lines", in the 10th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'15), Apr. 2015.

[37] A. Khtira, "Towards a Framework for Feature Deduplication during Software Product Lines Evolution", in the 27th International Conference on Advanced Information Systems Engineering (CAISE'15 Doctoral Consortium), June 2015, pp. 65-73.

[38] The Eclipse Foundation, "SWT: The Standard Widget Toolkit", eclipse.org/swt/ [retrieved: July, 2015].

[39] The Apache software foundation, "OpenNLP", opennlp.apache.org [retrieved: July, 2015].

[40] The PostgreSQL Global Development Group, "About PostgreSQL", postgresql.org/about/ [retrieved: July, 2015].

[41] "The prefuse visualization toolkit", prefuse.org/ [retrieved: July, 2015].

Amal Khtira received a degree in software engineering from National High School of Computer Science and Systems Analysis (ENSIAS) in 2008. She is currently a PhD student in the IMS (Models and Systems Engineering) Team of SIME Laboratory at ENSIAS. Her research interests include Software Product Line Engineering, Requirements Engineering, Feature Modeling and Software Evolution.

Anissa Benlarabi obtained a degree in software engineering from National High School of Computer Science and Systems Analysis (ENSIAS) in 2010. She is currently a PhD candidate in the IMS (Models and Systems Engineering) Team of SIME Laboratory at ENSIAS. Her research interests include Software Evolution, Biology-Based Software Approaches and Software Product Line Engineering.

Bouchra El Asri is a Professor in the Software Engineering Department and a member of the IMS (Models and Systems Engineering) Team of SIME Laboratory at National High School of Computer Science and Systems Analysis (ENSIAS), Rabat. Her research interests include Service-Oriented Computing, Model-Driven Engineering, Cloud Computing, Component-Based Systems and Software Product Line Engineering.