American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)

ISSN (Print) 2313-4410, ISSN (Online) 2313-4402

© Global Society of Scientific Research and Researchers

http://asrjetsjournal.org/

FDDetector: A Tool for Deduplicating Features in Software Product Lines

Amal Khtira[a]*, Anissa Benlarabi[b], Bouchra El Asri[c]

[a,b,c]IMS Team, ADMIR Laboratory, Rabat IT Center, ENSIAS, Mohammed V University, Rabat, Morocco

[a]Email: amalkhtira@gmail.com

[b]Email: a.benlarabi@gmail.com

[c]Email: b.elasri@um5s.net.ma

Abstract

Duplication is one of the model defects that affect software product lines during their evolution. Many approaches have been proposed to deal with duplication in code level while duplication in features hasn't received big interest in literature. At the aim of reducing maintenance cost and improving product quality in an early stage of a product line, we have proposed in previous work a tool support based on a conceptual framework. The main objective of this tool called FDDetector is to detect and correct duplication in product line models. In this paper, we recall the motivation behind creating a solution for feature deduplication and we present progress done in the design and implementation of FDDetector.

Keywords: Software Product Line; Feature Models; Duplication; Natural Language Processing; Tool Support.

1. Introduction

Software Product Lines encounter in their lifetime many problems that affect their different artefacts. In a systematic review that we carried out earlier [1], we investigated the different approaches proposed with respect to model defects in SPLs, we identified the nature of contributions and the different artefacts concerned by this issue, and we listed the different model defects of software product lines addressed in literature. As a result of this review, we have concluded that the most discussed model defect is inconsistency [2,3,4]. The other defects constantly addressed are ambiguity [5], unsafety [6] and incorrectness [7], while defects such as uncertainty, obsolescence and duplication haven't received enough attention. Another complementary study based on field experience enabled us to focus especially on the problem of duplication.

----

* Corresponding author

Duplication is a problem that has been discussed in many IT fields such as database management [8,9], Multimedia management [10,11,12], incident and request management [13,14,15], and also in academic research [16]. In software product lines, duplication can arise in requirement level as well as in code level. Many approaches have dealt with code duplication [17,18,19], but few have focused on feature duplication. In order to reduce development and maintenance cost and to enhance product quality, we have proposed, in previous work, a framework that aims at detecting and correcting feature duplication in software product lines [20,21]. In addition, we have introduced a tool support called FDDetector based on the framework [22]. The objective of this paper is to present the progress done so far in the development of FDDetector. This progress includes the proposed use cases and the description of the main functionality covered by the tool, the architecture and tools adopted in the development, and the implementation details of some functionality. The remainder of the paper is organized as follows. Section 2 highlights the conception of duplication as seen by many IT fields. Section 3 focuses on duplication in Software product lines and explains the motivation for a tool of feature deduplication. Section 4 presents the base deduplication framework and the work done so far in the development of FDDetector. In Section 5, we present some studies related to the problem of duplication in SPLs. Finally, Section 6 concludes the paper.

2. Concept of Duplication

In order to understand the duplication from different perspectives and have an idea about the different proposed approaches in relation with this subject, we have conducted a first general review of duplication. As a result of this review, we have identified several IT fields that have dealt with duplication-related issues.

2.1. Duplication in Databases

Duplication in databases is different from database duplication. Duplicate a database is a practice used in software development for backup purposes or to prepare development and test environments based on the production environment. It is thus considered to be a best practice, while duplication in databases is regarded as a defect. Data deduplication is part of data cleansing, which englobes inter-alia constraint verification, data fusion, format transformation, inconsistency correction and conformity verification [23]. This activity consists of identifying different instances representing the same entities of a database in order to delete or correct them. The studies addressing duplication in databases use basically similarity measures that allow automatic detection of duplication between two records [8,24,25,26], or propose algorithms to optimize the search of duplicates in a large volume of data [9,27,28].

2.2. Duplication in Multimedia Data

The rapid expansion of Internet and the success of the hosting and sharing services have explosively increased the volume of multimedia data in the web. Consequently, the number of duplicated images and videos increases rapidly, which makes the activities of data management and data recovery more difficult [29] and causes copyright issues as well. The approaches proposed to detect this kind of duplications can be classified into two main categories [10]. The first category consists of extracting and comparing the global features of an image or a video, such as the color feature [30], the edge-based feature [31], and the feature based on discrete cosine transformation (DCT) [11]. These features are easy to calculate and to compare, but they are sensitive to some serious changes such as the viewpoint changing and cropping. The second category of duplication detection approaches aims at extracting hundreds of local features, such as SIFT (Scale-Invariant Feature Transform) [32], PCA-SIFT (Principal Component Analysis-SIFT) [33], SURF (Speeded-Up Robust Feature) [34] and BOW (Bag-of-Visual-Words) [12,35].

2.3. Duplication related to Plagiarism

Plagiarism corresponds to the use of others' work without having their permission or citing the original reference, and includes all fields such as art, literature, sciences, etc. In software engineering, the plagiarism may occur in textual documents as well as in source code [36]. Chowdhury and Bhattacharyya [16] describe the different cases of plagiarism in each category. Plagiarism in documents includes deliberate copy-paste, simple or hybrid paraphrasing, borrowing others' ideas and claiming them as original, using its previous work in a new publication, etc. As regards the plagiarism in source code, it consists of manipulating the code produced by another person by adding, deleting or modifying some code segments, by changing the programming language, or by modifying the code structure, then claiming it as its own work.

2.4. Duplication Caused by Utilization

Users or developers using a system, a platform or an application meet in their daily work several issues that they report for verification in the form of bug reports or technical difficulties that they share with others via Community-based Question Answering (CQA) sites such as Quora and Stack Overflow. In the case of systems with large number of users or CQA sites with large number of subscribers, some questions and issues may be duplicated. If question or report management is done manually, users could wait a long time before receiving a response for their requests even for those that have already been resolved [37]. In order to improve the response time and reduce the effort done by the moderators and the support team, many studies have proposed methods and techniques to automate the detection of duplications in bug reports [13,14,38] and in CQA sites [15,39].

3. Duplication in Software Product Lines

Duplication in software as defined by [40] is to have the same thing expressed in two or more places. Duplication can happen in specifications, processes and programs. In this work, we focus specifically on duplication in software product lines.

3.1. Duplication in Code

Based on a research conducted about duplication in software products, the problem that received big interest in the last two decades is code cloning [17,18,19,41]. Clones, according to [42], are fragments of code that are similar based on a certain definition of similarity. The problem of duplication in source code was recognized first as a problem of maintenance [43] since it increases the modification effort. In fact, every modification of a code fragment must be applied to all clones. In addition, duplication may increase the program size and consequently the effort spent in the depending activities such as inspections. Another undesired effect of duplication is that inconsistent changes introduced unintentionally to the duplicated code may create errors and consequently incorrect or unexpected behaviors of the program [44,45].

3.2. Duplication in Requirements

Software product lines are long-living systems that evolve constantly. This evolution includes changes in existing functions, the introduction of new functionality, the correction of defects or changes due to external factors such as regulatory or organizational changes. As a consequence, functional documents and system models are modified. In most product lines, domain models are expressed using feature models [3,46,47], while the requirements related to new evolutions are documented in natural language specifications. In fact, customers prefer to express their needs in natural language because they believe it is the simplest and the easiest means of communication [48,49]. When new customer requirements are received, several decisions have to be studied [50,51] :

- If the feature is already supported by the product line, we need just to define a binding time for this feature.
- The feature could be deleted or replaced after discussion with the customer, because of some environment, infrastructure or technical constraints.
- The new feature must be integrated in the product line platform if it belongs to the product line scope.
- The new feature must be implemented in a specific application of the product line, which requires a specific development.

In large scale product lines with a significant number of features and several stakeholders synchronizing between them, the verification of new requirements becomes difficult and time-consuming and can be skipped in most cases. Consequently, many defects may be introduced to product line models as explained in the systematic review carried out in [1]. As a result of this review, we have found that many approaches have addressed defects such as inconsistency, ambiguity and incompleteness [2,3,47,52,53,54], but few have been interested to feature duplication.

3.3. Why Feature Deduplication?

As discussed in ***Section 3.1***, many approaches have proposed solutions for code duplication in order to improve the product quality. However, these approaches overlook the fact that if defects are not detected in an early stage of a project, they cause the deterioration of non-functional qualities and they propagate to other artefacts, which makes their correction more difficult, costly and time-consuming [55]. According to an IBM study [56], it is one hundred times more expensive to correct a defect after the product is released. Thus, programs should be oriented to detect and correct defects early in the development process, in order to reduce the cost of defect correction, to enhance productivity and to improve the product quality [57]. In the same vein, the operation of feature deduplication is considered a necessary activity that must be performed in an early step of the development lifecycle to achieve a satisfying level of quality for all project stakeholders and to reduce the implementation cost.

4. FDDetector

In order to solve the problem of feature duplication in Software Product Lines, we have proposed a framework that we presented in earlier work [20,21]. This framework was the base of an automated tool called FDDetector (Feature Duplication Detector) introduced in [22]. In this section, we give an insight of the base framework, then we present details about the analysis, the design and the implementation of FDDetector.

4.1. Framework Overview

As depicted in Figure 1, the framework of feature duplication detection is based on three main processes: Inputs transformation, duplication detection and duplication correction [21].

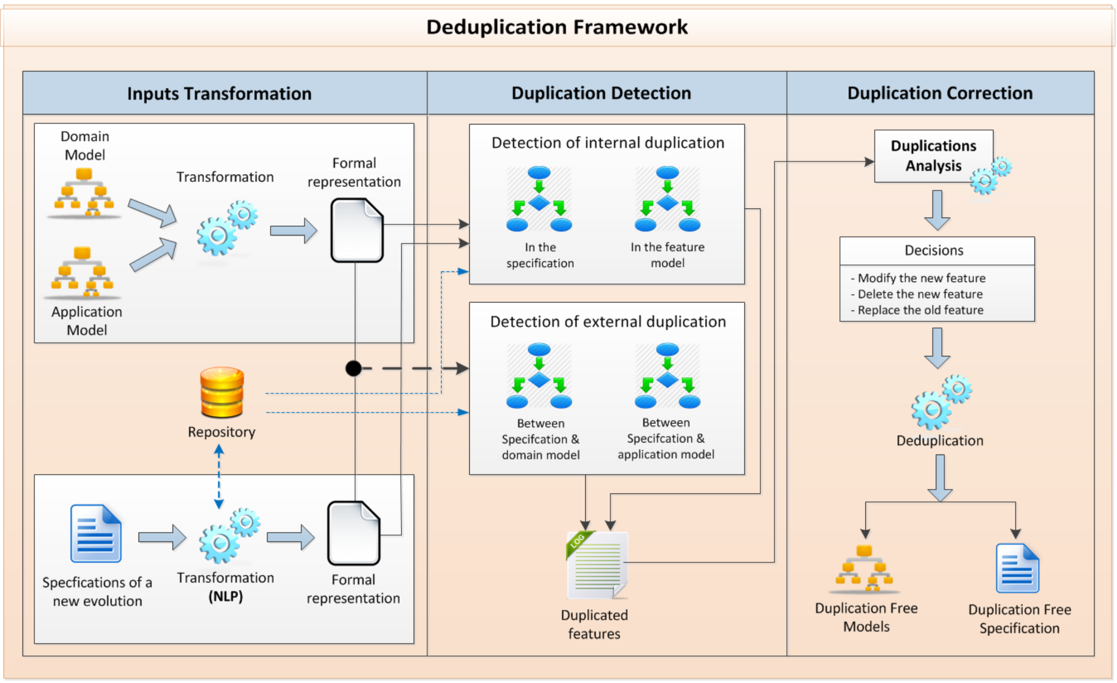


Figure 1: The Overview of the Deduplication Framework

The first process consists of transforming the framework inputs into a more formal representation. These inputs are: i) the domain model of the product line, ii) the application model of a derived product, and iii) the natural language specification of a specific evolution. In order to transform the domain and application models, we first generate the XML format of these models using FeatureIDE [58], and then we apply some mapping rules to create a variability-based tree structure. As for specifications, we perform a syntactic and semantic analysis of all the sentences to understand the user requirements and to extract the variation points and variants [59]. For this, we create a repository of variants that we fill based on the domain model. Then, the content of each sentence is compared against this repository to detect the potential variants existing in the specifications. When all the variants are detected, the corresponding tree is generated. This tree has the same structure as the tree generated from the transformation of domain and application models. This approach is based on machine learning since the repository is updated continuously to improve the activity of variant detection.

The second process is responsible for detecting duplications introduced into a software product line during a new evolution. This process includes two main activities. The first activity aims at detecting duplications inside each of the framework inputs, which we call *internal duplication*. And the second activity consists of detecting duplication between feature models and specification, which we call *external duplication* [21]. For this purpose, an algorithm for each activity was proposed. At the end of this process, the list of potential duplications is generated. The third and last process focuses on the correction of the detected duplications and involves two main activities, the analysis of detected duplications and the generation of a correct specification or feature model. During the first activity, the analyst, with the help of the customer, analyses the duplications to evaluate their relevance and validate or not their removal. The second activity relies on the analyst's decisions to provide a duplication-free specification or feature model.

4.2. Main Functionality

To have a clear overview of the system behavior, we have modelled the required functionality using the use case diagram as depicted in Figure 2.

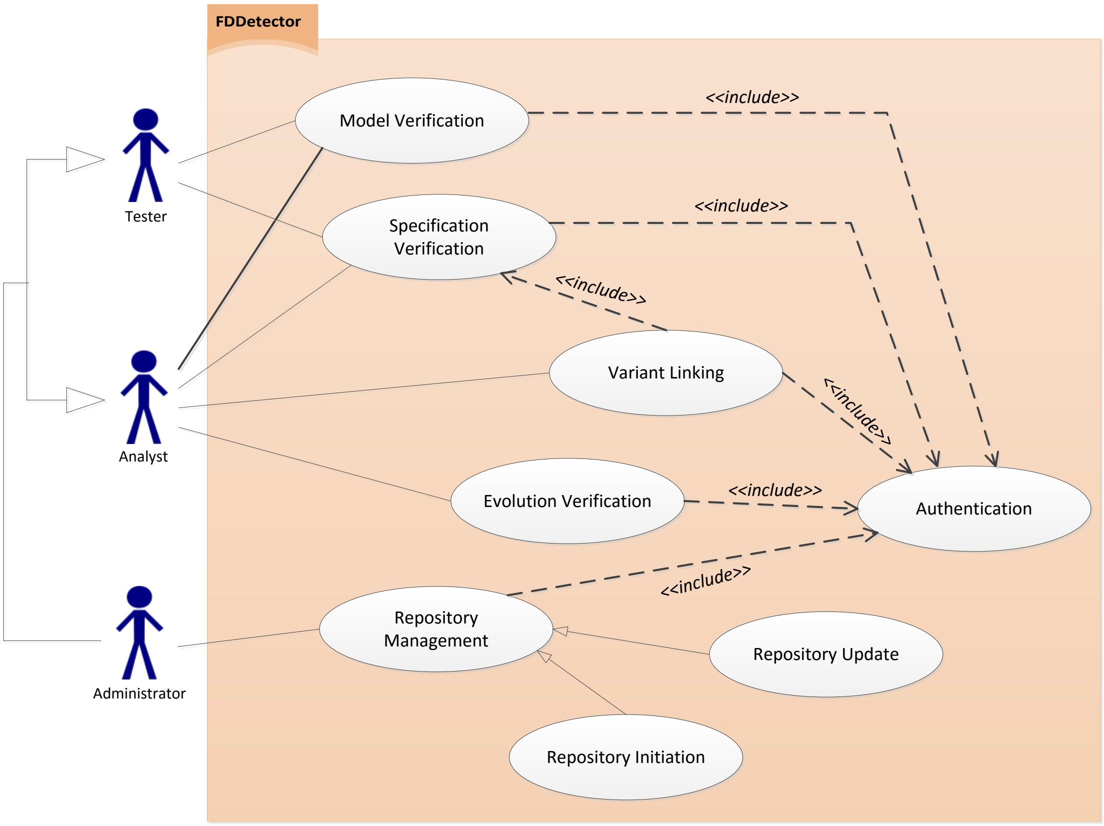


Figure 2: The Use Case Diagram of FDDetector

We distinguish three actors: The analyst, the tester and the administrator. The tester is responsible for performing reviews of the project documents, in our case the models and the specifications. The analyst is responsible for creating general and detailed specifications that he sends to developers. These specifications must be of good quality to prevent the propagation of errors to other artefacts. Thus, the analyst, like the tester, can verify the models and the specifications, but he also can compare between the two artefacts to detect duplications and deliver correct specifications. As for the administrator, he can carry out the same actions as the analyst and the tester, but he is also responsible for managing the repository. Table 1 presents the main scenarios related to the different use cases.

Table 1: Scenarios of FDDetector

Ref.	Scenario	Objective
SC1	Initiate the repository	Create the repository and fill the dictionary and the base model.
SC2	Update the repository	Update the model content based on the new features.
SC3	Verify a model	Detect internal duplications inside a model.
SC4	Verify a specification	Detect internal duplications inside a specification.
SC5	Verify an evolution	Detect external duplications between a specification and a model.
SC6	Link variants	Link new variants with existing variation points.

4.3. Software Architecture

FDDetector is a thick-client Java-based application built on Eclipse. Working with this IDE allowed us to use a set of plugins to implement different features. To design the application, we have adopted a multi-tiers architecture that separates different layers. Each layer is managed independently to anticipate technology evolution, to guarantee a good maintainability of the system and to reduce maintenance costs. Figure 3 presents a description of the different layers constituting the system.

- Presentation Layer: This layer represents the visible part of the application. Its role is to manage user interfaces and to enable the communication between the user and the application. To create user interfaces, we used SWT [60], which is an open source Java toolkit that reuses the facilities of the operating system on which it is implemented (Windows in our case). Another aspect managed by this interface is the visualization of the analyzed specifications in the form of graphs. For this, we used Prefuse [61], which is an open source toolkit that provides a visualization framework for the Java programming language and includes other functionality of data modeling, visualization and interaction.

- Business Layer: This layer is the heart of the application as it includes the implementation of the system functionality and defines the business rules. Its role is to receive the user requests from the presentation layer, process the required operations on data retrieved from the data access layer, and send the results again to the presentation layer. Regarding our tool, this layer supports specification analysis, models and specification transformation, and the detection of internal and external duplications. To perform the syntactic and semantic analysis of specifications, we use the Apache OpenNLP Library [62]. OpenNLP is a machine learning-based toolkit for the processing of natural language text. It enables the tokenization, chunking, part-of-speech tagging, entity extraction, and co-reference resolution. In addition, it includes an evaluation tool that measures the accuracy of entity recognition. The other operations supported by the tool are all implemented using Java code.

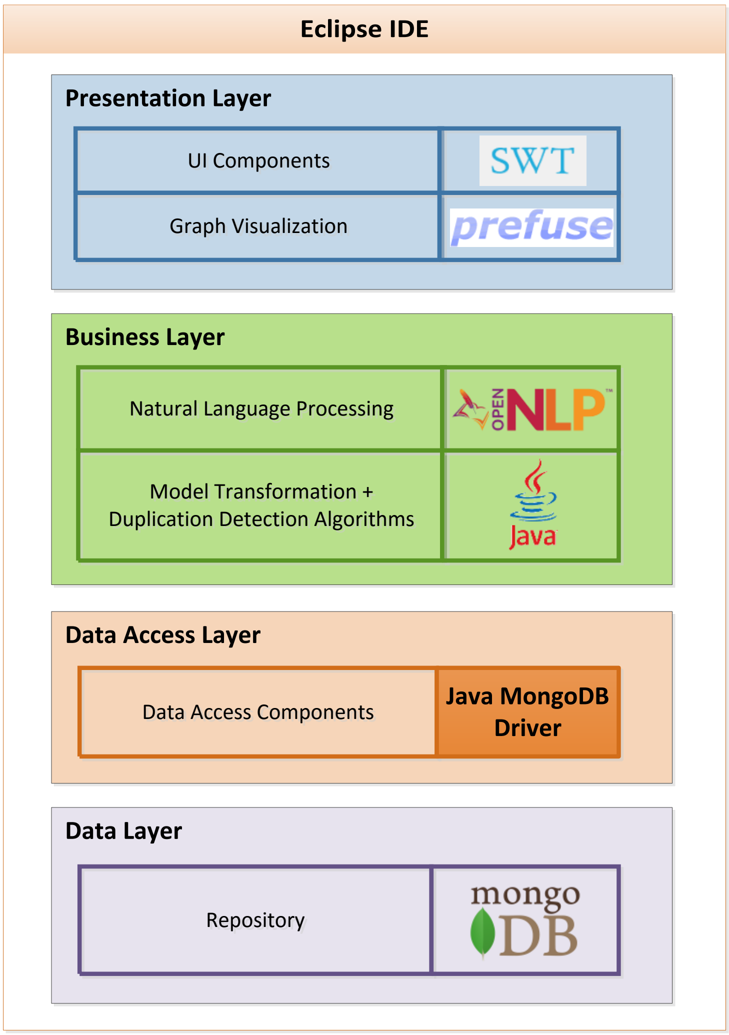


Figure 3: The architecture of FDDetector

- Data Access Layer: This layer manages the access to all the internal and external data needed by the system and ensures a weak coupling between the data layer and the business layer. In order to perform the mapping between the system classes and the data from the tables, we use in the new version of our tool Java MongoDB Driver [63]. This driver provides both synchronous and asynchronous interaction with MongoDB and manipulates the data with Java.

- Data Layer: This layer represents the application database. The DBMS we opted for is MongoDB [64], which is an open source noSQL document-oriented database that stores data in JSON-like documents and maps document models to the objects of the application, which makes data management easier and more flexible. In addition, MongoDB is a distributed database that is capable of retrieving data from different places and doesn't require a predefined schema. In the database we built, we have stored all the features of the product line domain, the variants extracted from new evolutions, and also the content of the dictionary that includes the description and synonyms of all the system concepts.

4.4. Implementation

The main interface of FDDetector is presented in Figure 4. This interface contains two entries « File » and « Repository ». In the first entry, there are 3 options: 1) « Open Specifications » enables the user to import textual specifications in txt or doc format, 2) « Open domain model » is used to import the domain model in XML format, and 3) « Open application model » is used to import the configuration file of a specific application.

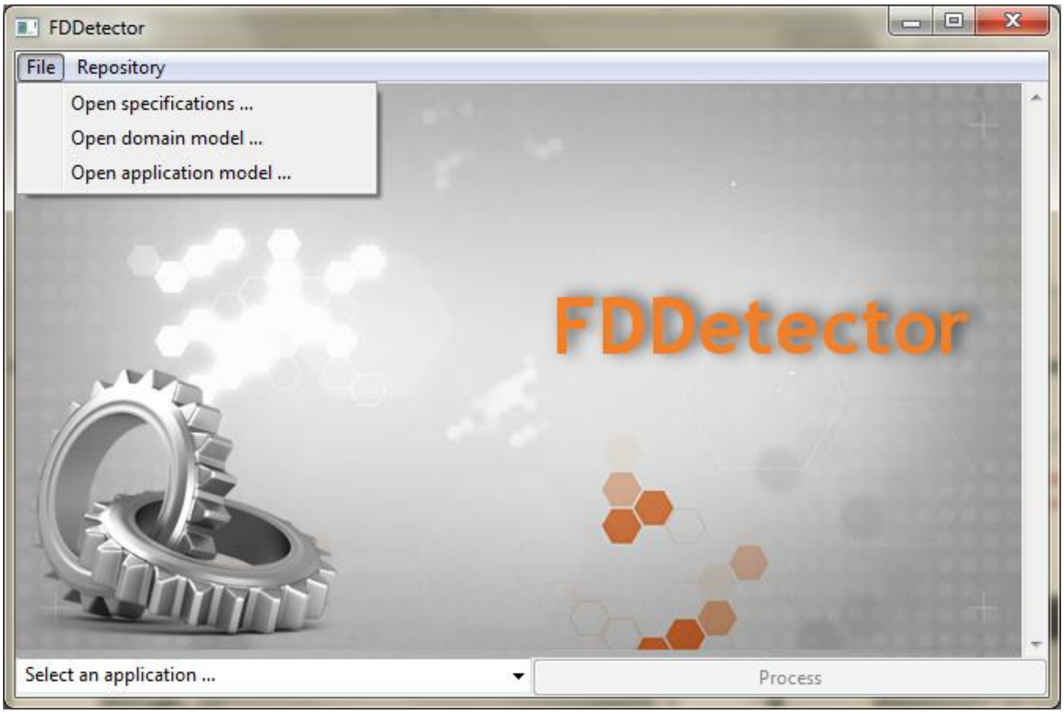
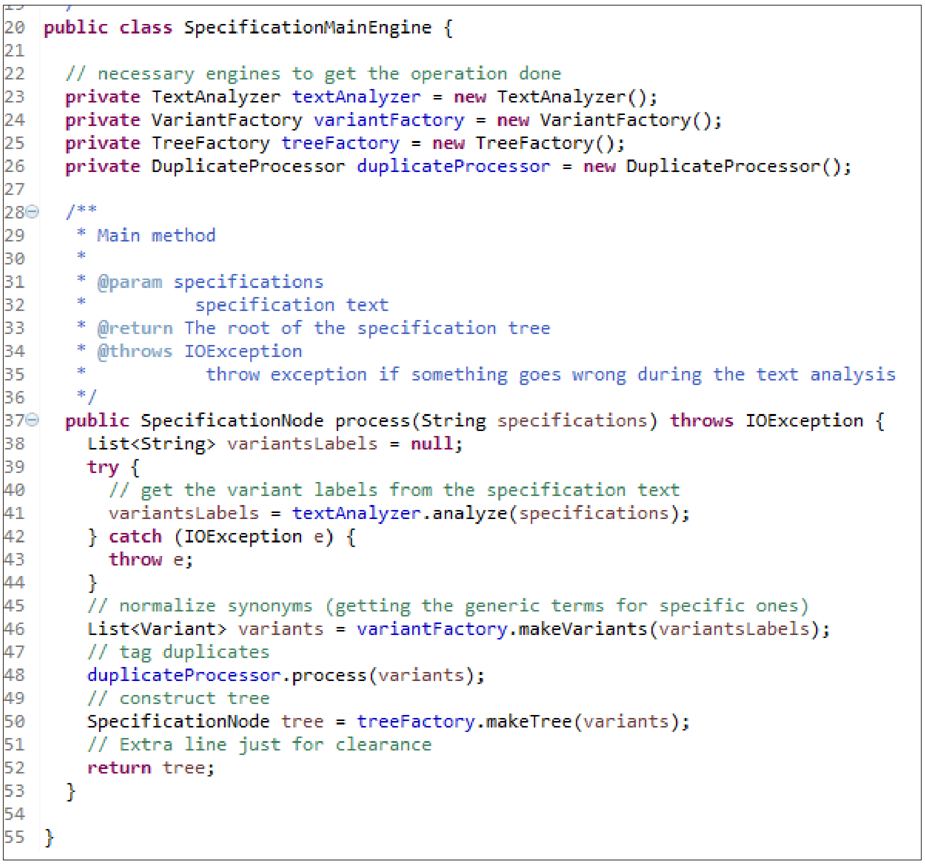


Figure 4: The main interface of FDDetector

The second entry is responsible for the repository management. The repository is a central element of the base framework as it is used both in the transformation of specifications and models, and in the detection of duplications. Thus, this functionality allows the initiation of the repository from the domain model and the update of the repository based on the new evolutions.

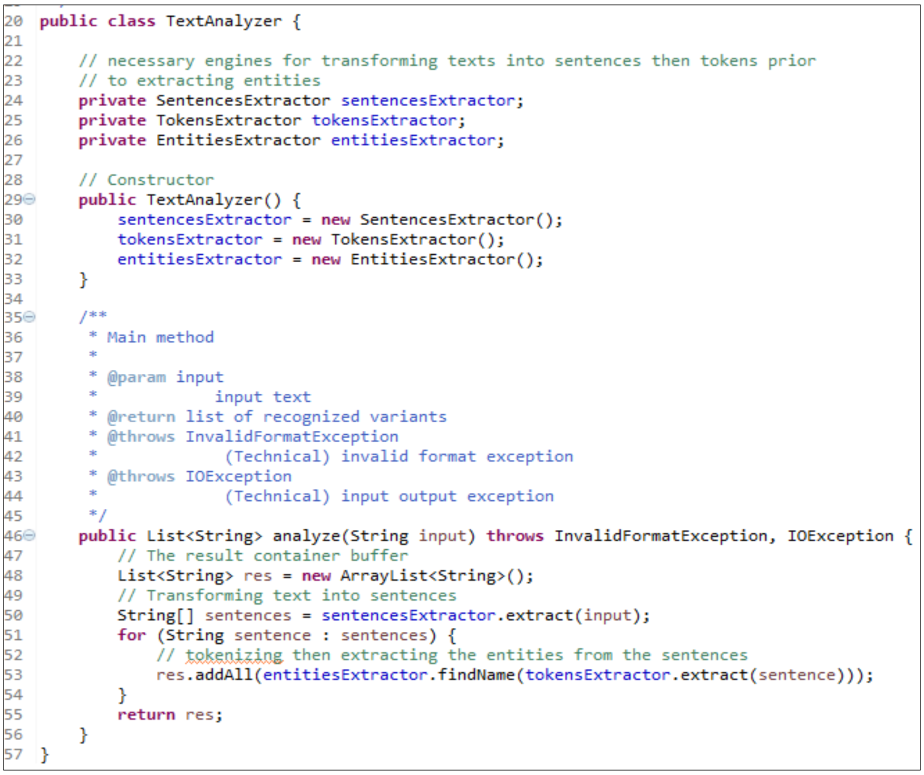
- **Specification Processing**

After importing the specification corresponding to a specific application, the button « Process » is used to analyze the specification by calling the class **SpecificationMainEngine** presented in the code of Figure 5.



```java
public class SpecificationMainEngine {

  // necessary engines to get the operation done
  private TextAnalyzer textAnalyzer = new TextAnalyzer();
  private VariantFactory variantFactory = new VariantFactory();
  private TreeFactory treeFactory = new TreeFactory();
  private DuplicateProcessor duplicateProcessor = new DuplicateProcessor();

  /**
   * Main method
   *
   * @param specifications
   *          specification text
   * @return The root of the specification tree
   * @throws IOException
   *           throw exception if something goes wrong during the text analysis
   */
  public SpecificationNode process(String specifications) throws IOException {
    List<String> variantsLabels = null;
    try {
      // get the variant labels from the specification text
      variantsLabels = textAnalyzer.analyze(specifications);
    } catch (IOException e) {
      throw e;
    }
    // normalize synonyms (getting the generic terms for specific ones)
    List<Variant> variants = variantFactory.makeVariants(variantsLabels);
    // tag duplicates
    duplicateProcessor.process(variants);
    // construct tree
    SpecificationNode tree = treeFactory.makeTree(variants);
    // Extra line just for clearance
    return tree;
  }

}
```

Figure 5: The SpecifcationMainEngine Class



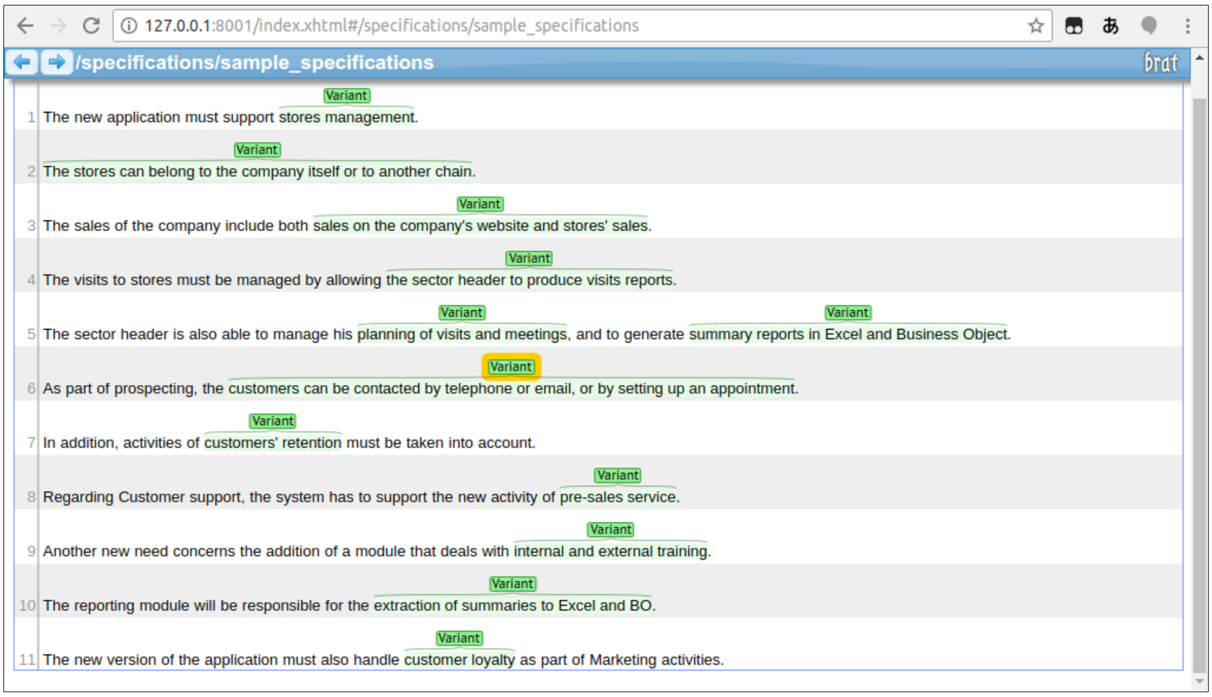
```java
public class TextAnalyzer {

    // necessary engines for transforming texts into sentences then tokens prior
    // to extracting entities
    private SentencesExtractor sentencesExtractor;
    private TokensExtractor tokensExtractor;
    private EntitiesExtractor entitiesExtractor;

    // Constructor
    public TextAnalyzer() {
        sentencesExtractor = new SentencesExtractor();
        tokensExtractor = new TokensExtractor();
        entitiesExtractor = new EntitiesExtractor();
    }

    /**
     * Main method
     *
     * @param input
     *            input text
     * @return list of recognized variants
     * @throws InvalidFormatException
     *             (Technical) invalid format exception
     * @throws IOException
     *             (Technical) input output exception
     */
    public List<String> analyze(String input) throws InvalidFormatException, IOException {
        // The result container buffer
        List<String> res = new ArrayList<String>();
        // Transforming text into sentences
        String[] sentences = sentencesExtractor.extract(input);
        for (String sentence : sentences) {
            // tokenizing then extracting the entities from the sentences
            res.addAll(entitiesExtractor.findName(tokensExtractor.extract(sentence)));
        }
        return res;
    }
}
```

Figure 6: The TextAnalyzer Class

The **TextAnalyzer** class is responsible for analyzing the specification and extracting its content through three main methods as shown in Figure 6. These methods are: **sentencesExtractor** that enables the detection of sentences of the specification, **tokenExtractor** that allows the extraction of tokens from the specification, and **entitiesExtractor** that extracts potential variants existing in the specification. Apart from the textual analysis, the **SpecificationMainEngine** class calls other classes, namely the **VariantFactory** class whose function is to search the generic variants previously determined and link them with the variants detected in the specification, the **DuplicateProcessor** class that detects potential internal duplications, and finally the **TreeFactroy** class that generates the graph of variants related to the processed specification.

- **Repository Management**

In order to make the extraction of variants from specifications more effective, the repository model must be initially filled by a large number of domain specifications that describe the features of the product line and these specifications must be annotated by adding tags representing the variants. This approach based on machine learning allows the extractor to detect potential variants existing in a new specification. To prepare such a model, we have opted for Brat. Brat is web-based collaborative environment dedicated to text annotation by adding notes to words or sentences of a document [65]. It is basically designed for structured annotations where the notes respect a specific form that can be processed and interpreted by a computer. This tool is used in many applications, namely entity mention detection, event extraction, meta-knowledge extraction, terminology normalization, and chunking. In our case, we use Brat to extract variants which represent metadata of a specification. Figure 7 shows an example of a textual specification annotation.


127.0.0.1:8001/index.xhtml#/specifications/sample_specifications
/specifications/sample_specifications
brat
Variant

1 The new application must support stores management.

2 The stores can belong to the company itself or to another chain.

3 The sales of the company include both sales on the company's website and stores' sales.

4 The visits to stores must be managed by allowing the sector header to produce visits reports.

5 The sector header is also able to manage his planning of visits and meetings, and to generate summary reports in Excel and Business Object.

6 As part of prospecting, the customers can be contacted by telephone or email, or by setting up an appointment.

7 In addition, activities of customers' retention must be taken into account.

8 Regarding Customer support, the system has to support the new activity of pre-sales service.

9 Another new need concerns the addition of a module that deals with internal and external training.

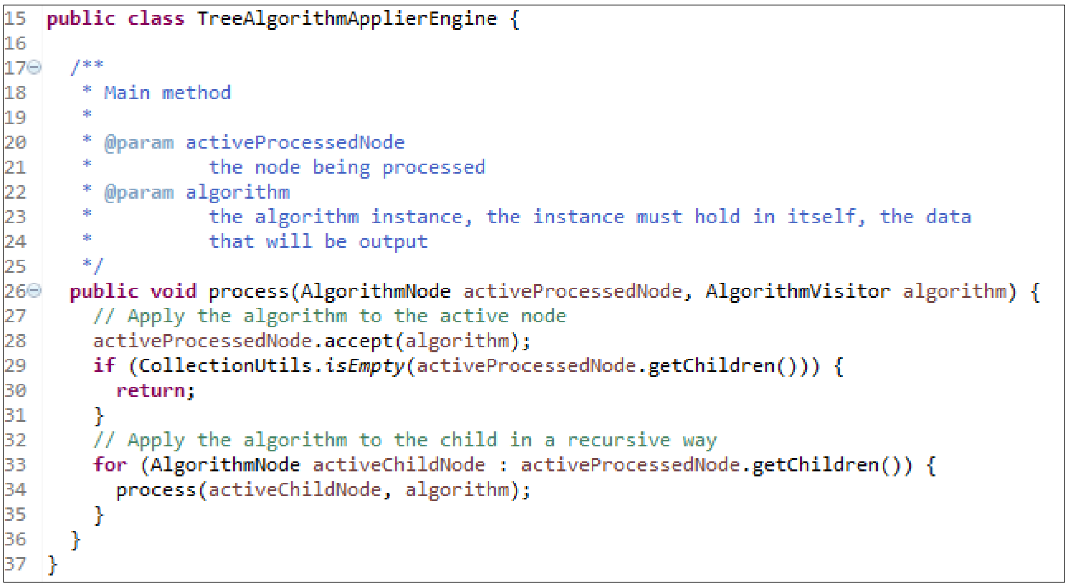
10 The reporting module will be responsible for the extraction of summaries to Excel and BO.

11 The new version of the application must also handle customer loyalty as part of Marketing activities.

Figure 7: Specification Annotation Using Brat

- **Duplication Detection**

At the purpose of implementing the algorithms of duplication detection, we decided to use the Visitor design pattern [66]. It is a behavioral pattern that separates between an operation and an object structure, by giving the possibility to define an operation without changing the classes on which it operates. In FDDetector, we intend to detect duplication in different inputs (specifications, feature models, or both). Thus, this pattern allows us to focus on the operation itself instead of focusing on the inputs. The source code of Figure 8 presents the implementation of the Visitor design pattern in our tool. The ***process*** method of the **TreeAlgorithmApplierEngine** class has two parameters: *AlgorithmNode* that corresponds to the input model and *AlgorthmVisitor* that corresponds to the algorithm used to detect duplication.



```
public class TreeAlgorithmApplierEngine {

  /**
   * Main method
   *
   * @param activeProcessedNode
   *            the node being processed
   * @param algorithm
   *            the algorithm instance, the instance must hold in itself, the data
   *            that will be output
   */
  public void process(AlgorithmNode activeProcessedNode, AlgorithmVisitor algorithm) {
    // Apply the algorithm to the active node
    activeProcessedNode.accept(algorithm);
    if (CollectionUtils.isEmpty(activeProcessedNode.getChildren())) {
      return;
    }
    // Apply the algorithm to the child in a recursive way
    for (AlgorithmNode activeChildNode : activeProcessedNode.getChildren()) {
      process(activeChildNode, algorithm);
    }
  }
}
```

Figure 8: The TreeAlgorithmApplierEngine Class

5. Related Work

According to the systematic review performed in [1] and a complementary review on software duplication, we have found some papers that address the problem of duplication in software product lines. For example, the objective of [67] is to solve the inconsistencies introduced in a product line due to code forking. For this, a PL-CDM model that includes features, their implementations and the dependencies between them has been defined. This model helps both managers and developers in the detection of inconsistencies between duplicated products and in the synchronization of developments. Schmorleiz and Lämmel [68] propose a process to manage the similarity of cloned variants during software evolution. The aim of this process is to save developers' intentions using annotations in order to anticipate automatic change propagation. Hellebrand and his colleagues [69] focus on the coevolution between feature models and code. More specifically, they define metrics that enable the detection of variability erosion between the two artefacts during SPL evolution. Rubin and his colleagues [70] address the management of cloned software product variants by proposing seven conceptual operators. The validation of these operators was performed through three case studies from the automotive industry. Ghofrani and his colleagues [71] introduce a machine learning based framework dedicated to detect code clones. The similarity between code snippets is calculated using summaries generated by deep neural networks. Störrle [72] proposes a formal definition of model clones, presents a specific algorithm of clone detection for UML domain models, and implements a prototype to evaluate the algorithm.

6. Conclusion

In order to keep up with new evolutions, software product lines are constantly changing and become subject to many defects, especially duplication. In a previous work, we have introduced the first prototype of FDDetector, which is a tool for the detection and correction of feature duplication in software product lines. The purpose of this tool was to reduce the time to market and the cost of development of software product line evolutions by detecting the duplication in the stage of specification analysis. In this paper, we presented the progress done so far in the implementation of FDDetector. For this, we described the different use cases taken into account in the current version, we presented the system architecture and the tools used in the different system layers, and we provided implementation details of some functionality. In future work, we intend to improve the tool and evaluate it on a large scale software product line in order to ensure its effectiveness and scalability.

References

[1] A. Khtira, A. Benlarabi, and B. El Asri, "Model Defects in Evolving Software Product Lines: A Review of Literature," American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS), vol. 41, no. 1, July 2018.

[2] A. O. Elfaki, "A rule-based approach to detect and prevent inconsistency in the domain engineering process." Expert Systems, vol. 33, no. 1, pp. 3-13, 2016.

[3] M. Alférez, R. E. Lopez-Herrejon, A. Moreira, et al., "Consistency Checking in Early Software Product Line Specifications-The VCC Approach." Journal of Universal Computer Science, vol. 20, no. 5, pp. 640-665, 2014.

[4] C. Quinton, A. Pleuss, D. L. Berre, et al., "Consistency checking for the evolution of cardinality-based feature models," in Proc. 18th International Software Product Line Conference-Volume 1, ACM, Sept. 2014, pp. 122-131.

[5] Z. Stephenson, K. Attwood, and J. McDermid, "Product-Line Models to Address Requirements Uncertainty, Volatility and Risk," in Relating Software Requirements and Architectures, Springer Berlin Heidelberg, pp. 111-131, 2011.

[6] L. Neves, P. Borba, V. Alves, et al., "Safe evolution templates for software product lines." Journal of Systems and Software, vol. 106, pp. 42-58, 2015.

[7] I. Groher, A. Reder, and A. Egyed, "Incremental consistency checking of dynamic constraints," in: Rosenblum D.S., Taentzer G. (eds) Fundamental Approaches to Software Engineering (FASE 2010), Lecture Notes in Computer Science, Springer Berlin Heidelberg, vol. 6013, pp. 203-217, 2010.

[8] L. Zhu, M. Amsler, T. Fuhrer, et al., "A fingerprint based metric for measuring similarities of crystalline structures," The Journal of chemical physics, 2016, vol. 144, no. 3, p. 034203.

[9] T. Papenbrock, A. Heise, and F. Naumann, "Progressive duplicate detection," IEEE Transactions on knowledge and data engineering, 2015, vol. 27, no. 5, pp. 1316-1329.

[10] Z. Zhou, QM. J. Wu, F. Huang, et al. "Fast and accurate near-duplicate image elimination for visual sensor networks," International Journal of Distributed Sensor Networks, 2017, vol. 13, no. 2, pp. 1-12.

[11] Z. Zhang, D. Wang, C. Wang, et al., "Detecting Copy-move Forgeries in Images Based on DCT and Main Transfer Vectors," KSII Transactions on Internet & Information Systems, 2017, vol. 11, no. 9.

[12] J. Yao, B. Yang, and Q. Zhu, "Near-duplicate image retrieval based on contextual descriptor," IEEE signal processing letters, 2015, vol. 22, no. 9, pp. 1404-1408.

[13] K. Aggarwal, F. Timbers, T. Rutgers, et al. "Detecting duplicate bug reports with software engineering domain knowledge," Journal of Software: Evolution and Process, 2017, vol. 29, no. 3, p. e1821.

[14] A. Hindle, A. Alipour, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection and ranking," Empirical Software Engineering, 2016, vol. 21, no. 2, pp. 368-410.

[15] Y. Zhang, D. Lo, X. Xia, et al., "Multi-factor duplicate question detection in stack overflow," Journal of Computer Science and Technology, 2015, vol. 30, no. 5, pp. 981-997.

[16] H. A. Chowdhury and D. K. Bhattacharyya, "Plagiarism: taxonomy, tools and detection techniques," 2018, arXiv preprint arXiv:1801.06323

[17] H. Sajnani, "Large-Scale Code Clone Detection," Doctoral thesis, UC Irvine, 2016.

[18] Y. Dang, D. Zhang, S. Ge, et al., "Transferring code-clone detection and analysis to practice," in Proc. IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), IEEE, 2017, pp. 53-62.

[19] J. T. Svajlenko, "Large-scale clone detection and benchmarking," Doctoral thesis, University of Saskatchewan, 2018.

[20] A. Khtira, A. Benlarabi, and B. El Asri, "Duplication Detection when evolving Feature Models of Software Product Lines," Information Science Journal (ISJ), vol. 6, no. 4, pp. 592-612, Oct. 2015.

[21] A. Khtira, A. Benlarabi, and B. El Asri, "Modelling and Correcting Duplication in Evolving Software Product Lines," IJCSI International Journal of Computer Science Issues, vol. 15, no. 4, July 2018.

[22] A. Khtira, A. Benlarabi, and B. El Asri, "A Tool Support for Automatic Detection of Duplicate Features

during Software Product Lines Evolution," IJCSI International Journal of Computer Science Issues, vol. 12, no. 4, pp. 1-10, July 2015.

[23] H. Müller, J-C. Freytag, Problems, methods, and challenges in comprehensive data cleansing, Professoren des Inst. Für Informatik, 2005.

[24] M. Bilenko and R. J. Mooney, "Adaptive duplicate detection using learnable string similarity measures," in Proc. 9th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2003, pp. 39-48.

[25] J. LY. Koh, M. L. Lee, A. M. Khan, et al., "Duplicate detection in biological data using association rule mining," in Proc. 2nd European Workshop on Data Mining and text Mining in Bioinformatics, 2004, pp. 35-41.

[26] L. Leitão, P. Calado, and M. Weis, "Structure-based inference of XML similarity for fuzzy duplicate detection," in. Proc. 16th ACM conference on Conference on information and knowledge management, ACM, 2007, pp. 293-302.

[27] L. Leitão, P. Calado, and M. Herschel, "Efficient and Effective Duplicate Detection in Hierarchical Data," IEEE Transactions on knowledge and data engineering, 2013, vol. 25, no. 5, pp. 1028-1041.

[28] U. Draisbach and F. Naumann, "A generalization of blocking and windowing algorithms for duplicate detection," in. Proc. International Conference of Data Knowledge Engineering, 2011, pp. 18-24.

[29] W. Jun, Y. Lee, and B-M. Jun, "Duplicate video detection for large-scale multimedia," Multimedia Tools and Applications, 2015, vol. 75, no. 23, pp. 15665-15678.

[30] B. Chen, H. Shu, G. Coatrieux, et al., "Color image analysis by quaternion-type moments," Journal of mathematical imaging and vision, 2015, vol. 51, no. 1, pp. 124-144.

[31] C. Lin and S. Wang, "An edge-based image copy detection scheme," Fundam Inform, 2008, vol. 83, no. 3, pp. 299-318.

[32] W. Zhou, H. Li, Y. Lu, et al., "SIFT match verification by geometric coding for large-scale partial-duplicate web image search," ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 2013, vol. 9, no. 1, p. 4.

[33] K. Yan and R. Sukthankar, "PCA-SIFT: A more distinctive representa- tion for local image descriptors," in Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit., Jun./Jul. 2004, vol. 4, pp. 506-513.

[34] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," Computer vision and image understanding, Jun. 2008, vol. 110, no. 3, pp. 346-359.

[35] Z. Li and X. Feng, "Near Duplicate Image Detecting Algorithm based on Bag of Visual Word Model," Journal of Multimedia, 2013, vol. 8, no. 5.

[36] V. T. Martins, D. Fonte, P. R. Henriques, et al., "Plagiarism detection: A tool survey and comparison," in 3rd Symposium on Languages, Applications and Technologies, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[37] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, et al. "Mining duplicate questions in stack overflow," in Proc. 13th International Conference on Mining Software Repositories, ACM, 2016. pp. 402-412.

[38] K. K. Sabor, A. Hamou-lhadj, and A. Larsson, "Durfex: a feature extraction technique for efficient detection of duplicate bug reports," in. Proc. IEEE international conference on software quality, reliability and security (QRS), IEEE, 2017, pp. 240-250.

[39] W. E. Zhang, Q. Z. Sheng, J. H. Lau, et al., "Detecting duplicate posts in programming QA communities via latent semantics and association rules," in Proc. 26th International Conference on World Wide Web, 2017, pp. 1221-1229.

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

[41] K. Tonscheidt, "Leveraging code clone detection for the incremental migration of cloned product variants to a software product line: An explorative study," Bachelorarbeit, Otto-von-Guericke-Universität Magdeburg, 2015, pp. 4-16.

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

[43] R. Koschke, "Identifying and removing software clones," in T. Mens and S. Demeyer (eds.), Software Evolution, Springer, 2008.

[44] R. Koschke, "Survey of research on software clones," in Proc. Dagstuhl Seminar on Duplication, Redundancy, and Similarity in Software, 2007.

[45] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?", in Proc. 31st International Conference on Software Engineering, IEEE, May 2009, pp. 485-495.

[46] S. Apel, D. Batory, C. Kästner, and G. Saake, "Analysis of Software Product Lines," in Feature-Oriented Software Product Lines, Berlin: Springer, 2013, pp. 243-282.

[47] R. M. de Mello, E. Nogueira, M. Schots, et al., "Verification of Software Product Line Artefacts: A Checklist to Support Feature Model Inspections," Journal of Universal Computer Science, 2014, vol. 20, no. 5, pp. 720-745.

[48] D. Berry, "Natural language and requirements engineering-Nu? ," in International Workshop on Requirements Engineering, Imperial College, London, UK. 2001.

[49] M. Luisa, F. Mariangela, and N. I. Pierluigi, "Market research on requirements analysis using linguistic tools," Requirements Engineering, 2004, vol. 9, no. 1, pp. 40–56.

[50] M. Schubanz, A. Pleuss, G. Botterweck, et al., "Modeling rationale over time to support product line evolution planning," in Proc. 6th International Workshop on Variability Modeling of Software-Intensive Systems, ACM, 2012, pp. 193-199.

[51] F. Van Der Linden, K. Schmid, and E. Rommes, Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, Springer, 2007.

[52] M. Cordy, P. Y. Schobbens, P. Heymans, and A. Legay, "Beyond boolean product-line model checking: dealing with feature attributes and multi-features," in Proc. International Conference on Software Engineering, IEEE, May 2013, pp. 472-481.

[53] D. Romero, S. Urli, C. Quinton, et al., "SPLEMMA: A generic framework for controlled-evolution of software product lines," in Proc. 17th International Software Product Line Conference Collocated Workshops, Tokyo, Japan, 26-30 August 2013, pp. 59–66.

[54] L. Neves, P. Borba, V. Alves, et al., "Safe evolution templates for software product lines," Journal of Systems and Software, 2015, vol. 106, pp. 42-58.

[55] S. McConnell, "Software quality at top speed," Software Development, 1996, vol. 4, no. 8, pp. 38-42.

[56] D. Edwards, "DevOps: Shift left with continuous testing by using automation and virtualization," Sep. 18th, 2014, https://www.ibm.com/developerworks/community/blogs/invisiblethread/entry/enabling_devops_success_with_shift_left_continuous_testing?lang=en [retrieved: October, 2016]

[57] T. Kasse, Practical insight into CMMI, Artech House, 2008.

[58] 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 Proc. 31st International Conference on Software Engineering (ICSE'09), IEEE, Washington, DC, USA, 2009, pp. 611–614.

[59] A. Khtira, A. Benlarabi, and B. El Asri, "Detecting Feature Duplication in Natural Language Specifications when Evolving Software Product Lines," in Proc. 10th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'15), Barcelona, Spain, Apr. 2015, pp. 257-262.

[60] The Eclipse Foundation, "SWT: The Standard Widget Toolkit," eclipse.org/swt/ [retrieved: December,

2019]

[61] "The prefuse visualization toolkit," https://web.archive.org/web/20181226190156/http://prefuse.org/ [retrieved: December, 2019]

[62] The Apache Software Foundation, "OpenNLP," opennlp.apache.org [retrieved: December, 2019]

[63] MongoDB, "Java MongoDB Driver," https://docs.mongodb.com/ecosystem/drivers/java/ [retrieved: December, 2019].

[64] MongoDB, "What is MongoDB?" https://www.mongodb.com/what-is-mongodb [retrieved: December, 2019].

[65] Brat, "brat rapid annotation tool," http://brat.nlplab.org/index.html [retrieved: December, 2019]

[66] Sourcemaking, "Visitor Design Pattern," https://sourcemaking.com/design_patterns/visitor [retrieved: December, 2019]

[67] J. Rubin, A. Kirshin, G. Botterweck, and M. Chechik, "Managing forked product variants," in Proc. 16th International Software Product Line Conference-Volume 1, ACM, Sept. 2012, pp. 156-160.

[68] T. Schmorleiz and R. Lämmel, "Similarity management of 'cloned and owned' variants," in Proc. 31st Annual ACM Symposium on Applied Computing, ACM, Apr. 2016, pp. 1466-1471.

[69] R. Hellebrand, A. Silva, M. Becker, et al., "Coevolution of variability models and code: an industrial case study", in 18th International Software Product Line Conference, ACM, Sept. 2014, Vol. 1, pp. 274-283.

[70] J. Rubin, K. Czarnecki, and M. Chechik, "Cloned product variants: from ad-hoc to managed software product lines", International Journal on Software Tools for Technology Transfer, Vol. 17, No. 5, 2015, pp. 627-646.

[71] J. Ghofrani, M. Mohseni, and A. Bozorgmehr, "A conceptual framework for clone detection using machine learning", in 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI), Dec. 2017, pp. 0810-0817.

[72] H. Störrle, "Towards clone detection in UML domain models," Software & Systems Modeling, 2013, Vol. 12, no 2, pp. 307-329.