Detecting Feature Duplication in a CRM Product Line

Amal Khtira*, Anissa Benlarabi, Bouchra El Asri
IMS Team, ADMIR Laboratory, Rabat IT Center, ENSIAS, Mohammed V University, Rabat, Morocco

* Corresponding author. Email: amalkhtira@gmail.com
Manuscript submitted December 19, 2020; accepted January 20, 2020.
doi: 10.17706/jsw.15.1.30-44

Abstract: Many defects may arise during software product lines such as inconsistency, incorrectness and ambiguity. In our work, we are interested in a specific defect, which is duplication in features since it has received little attention in literature. In previous work, we proposed a conceptual framework that describes the different processes to follow in order to produce duplication-free feature models. We also implemented a tool support called FDDetector based on this framework with the aim of detecting and correcting duplication introduced in feature models by new product line evolutions. To evaluate this tool, we have adopted a CRM product line as a case study. In this paper, we explain the motivations behind choosing a case study from the CRM field, and we present the results of a test performed on an evolution related to this product line.

Key words: Feature duplication, customer relationship management, software product line, case study.

1. Introduction

Duplication is among many other defects (e.g. inconsistency [1]-[3], ambiguity [4], unsafety [5], incorrectness [6]) that may occur in software product lines during their evolution [7]. This defect consists of having the same element repeated many times in a software artefact. In software product lines, duplication can appear in code level or in feature level. Many studies in literature have addressed the problem of code duplication (*code cloning*) [8]-[10], while the problem of feature duplication has not received big interest. To deal with feature duplication, we proposed in previous work, a framework that aims at detecting and correcting this defect in evolving software product lines [11], [12]. We have also implemented a tool support called FDDetector based on this framework [13], [14]. By correcting duplication at the stage of features, we ensure that this defect does not propagate to other artefacts, which helps reduce the development and maintenance cost and enhances the product line quality.

At the purpose of evaluating the accuracy of FDDetector, we have adopted a CRM software product line as a case study. In fact, duplication can easily happen in a CRM because this type of software needs to change constantly and rapidly to overcome several challenges. The objectives of this paper are thus to: i) explain the main reasons behind choosing a case study from the CRM field, ii) give an overview of the main functionality of this product line, iii) perform a test on a specific evolution related to a derived application, iv) expose the results achieved, and v) present the measures used to evaluate the tool accuracy.

The rest of the paper is structured as follows. Section 2 introduces the concept of duplication in Software product lines and explains the reasons behind the choice of a CRM product line as a case study. Section 3 gives an insight into the framework and the tool support proposed to detect and correct feature duplication in software product lines. Section 4 presents the adopted case study and the results of the test performed

on it using FDDetector. In Section 5, we conclude and outline future work.

2. Background and Motivation

In [7], we carried out a systematic review about model defects that may appear in software product lines during their evolution. As a result of this review, we listed the different model defects addressed in literature, such as inconsistency [1]-[3], incorrectness [6], ambiguity [4], and unsafety [5]. A complementary study based on field experience helped us decide to focus on a specific defect, which is feature duplication.

2.1. Duplication in Software Product Lines

According to [15], duplication is the fact of having the same thing expressed in two or more places. Duplication may occur in specifications, processes or programs. In this work, we are specifically interested to duplication in software product lines, which can affect many artefacts especially requirements and code. In literature, many studies have dealt with code duplication (or code cloning) [8]-[10], [16]. This problem was considered at the beginning as a problem of maintenance [17] because first, every modification of a code fragment implies the modification of all the related clones, and second, duplication increases the program size and consequently the effort spent in all maintenance activities. Other than that, duplication of source code can be the cause of other kinds of problems and leads to incorrect and unexpected system behaviors [18], [19]. However, source code is caused generally by a duplication that happens earlier in the project lifecycle, during the design or the requirement analysis activities. Indeed, as product lines are systems that last for a long time, they are subject to several changes, such as the modification of existing functions, the addition of new features or the correction of functional or non-functional defects. Before implementing these changes, an analysis of the new requirements must be performed, which triggers a number of decisions [20], [21] : i) If the demanded feature is already supported by the product line, it should only be derived from the domain model, ii) If the demanded feature is a new feature that belongs to the product line scope, it must be added to the product line platform, iii) If the feature is new but can't be implemented because of some environment, infrastructure or technical constraints, this feature could be deleted or replaced after discussion with the client, vi) If the feature is new in a particular application, a specific development should be considered. In large scale product lines that support an important number of features and include several stakeholders synchronizing between them, the requirement analysis becomes a complex and a time-consuming task and the project managers could decide to skip it sometimes, and that's how feature duplication happens.

According to an IBM study [22], it is one hundred times more expensive to correct a defect after the product is released. That is why we focus on our work on the detection and correction of duplication in the feature level to prevent the duplications from propagating to other artefacts, which guarantees high quality from the first beginning of a product and reduces the development and maintenance time and cost [23].

2.2. Motivating Example

The CRM or Customer Relationship Management is a strategy whose main objective is to manage and optimize interactions between a company and its customers. It also aims at enhancing productivity, increasing the company return and reducing the costs. A CRM system centralizes the company data centers used by the managers to have a visibility of customers' information and take the rights decisions. It can be adopted by big companies as well as small businesses since both need to manage their clients. During its lifetime, a CRM faces several challenges that cause its evolution. With the instable market of today, clients' needs are constantly changing and a CRM should necessarily be flexible to satisfy and retain its customers. Moreover, not all customers have the same preferences in terms of products and services. In the contrary,

they have different profiles, habits and needs. A CRM must take into account these differences by proposing customized services that satisfy the needs of every client and prospect. It must also offer a competitive advantage by optimizing the relationship with the clients and by anticipating or even creating new needs. In addition, with the continuous evolution of technology, CRMs have to evolve to keep up with new technologies. For example, communication with customers was limited in the past to direct contact or phone calls, while now other channels have emerged such us SMSs, emails, the Web or mobile applications.

In this instable context, different model defects and specifically feature duplication can arise in the system. In order to take profit of the several advantages offered by a CRM and to optimize its evolution, we decided to start by applying our deduplication tool on a CRM product line at the aim of detecting and correcting feature duplications introduced by the new evolutions. This solution allows the CRM not only to respond to new customers' needs but also to be as reactive as possible, to support new evolutions with low costs and to ensure high product quality despites the rapid change. In the next sections, we recall our proposed approach and the deduplication tool, we present the adopted CRM product line, and then we explain the details and results of the test performed on a specific evolution of this CRM.

3. The Proposed Approach for Feature Deduplication

At the aim of detecting and correcting duplication in software product lines, we first proposed a conceptual framework [11], [12] then implemented a support tool based on this framework which we called FDDetector (Feature Duplication Detector) [13], [14]. In this section, we give an overview of both the framework and the support tool.

3.1. The Deduplication Framework

The framework of feature deduplication is composed of three main processes: Inputs transformation, duplication detection and duplication correction [12].

The first process is responsible for transforming the feature models and the natural language specification related to evolutions into a more formal representation. The transformation of models is performed in two steps. First, we generate the XML source of the feature model using FeatureIDE [24], and then we use some mapping rules to create a variability-based tree structure. As regard specifications, we adopt an approach based on natural language processing by analyzing syntactically and semantically the specification sentences in order to extract the potential variation points and variants [25]. The repository of variants created from the product line general specification must be constantly updated by new evolutions to improve the activity of variant detection. The second process consists of detecting duplications introduced into a software product line during a new evolution. To achieve this, a number of algorithms have been proposed. The first algorithm has the objective of detecting duplications inside each of the framework inputs (models or specifications), which we call internal duplication, while the second algorithm aims at detecting duplication between feature models and specifications, which we call external duplication [12]. The output of this process consists of a list of all the potential duplications detected. The purpose of the third process of the framework is to correct the detected duplications and to generate a duplication-free specification or feature model. The analyst plays an important role in this process because the system relies on his decisions to generate the correct outputs.

3.2. FDDetector

FDDetector is a thick-client Java-based application built on Eclipse. Fig. 1 depicts the different tools used in the implementation of FDDetector.

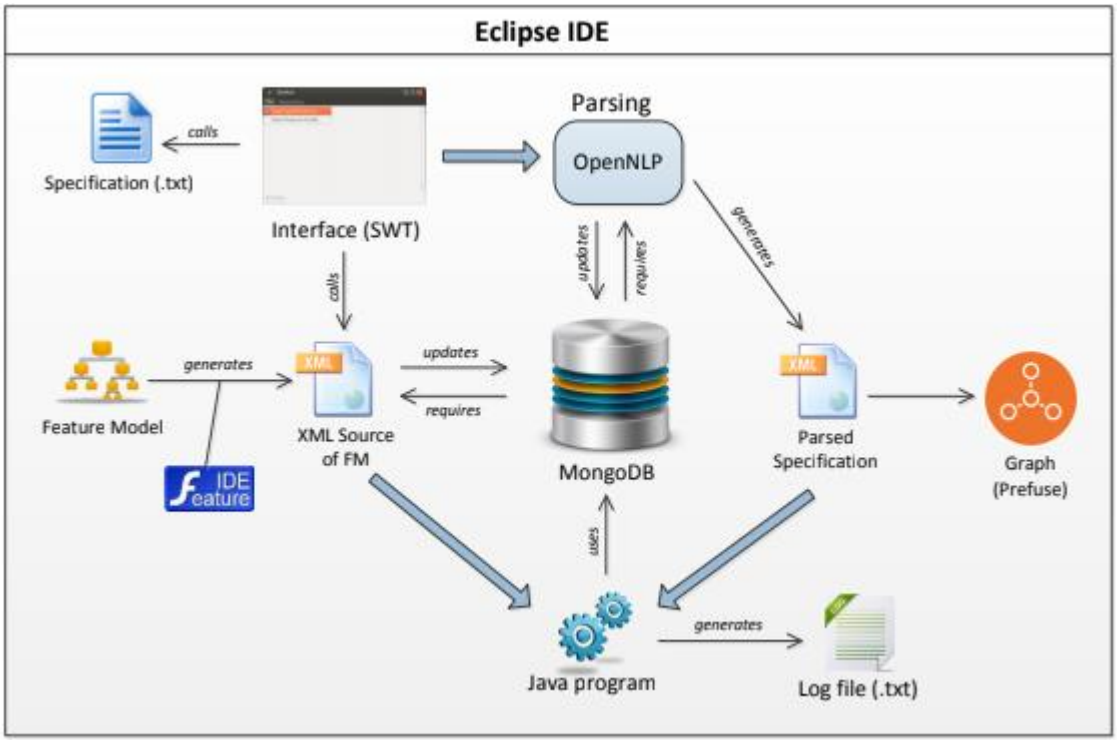

Fig. 1. FDDetector architecture.

In order to create feature models and to generate the XML source, we use FeatureIDE [24]. This open-source framework is based on Eclipse and supports several functions such as requirements analysis, domain implementation, and software generation. The HMI of our tool is created using SWT [26], which is an open source widget toolkit for Java that uses the user-interface facilities of the operating systems on which it is implemented. The analysis of natural language specifications is performed using Apache OpenNLP Library [27], which is a machine learning based toolkit that includes many NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, and parsing. In order to visualize the graph related to the analyzed specifications, we opt for Prefuse [28]. It is a Java-based open source toolkit that supports a rich set of features for data modeling, visualization, and interaction.

The content of the repository is stored using MongoDB [29]. It is an open source noSQL document-oriented database that makes data management easier and more flexible by storing data in JSON-like documents then mapping document models to the objects of the application. In addition, MongoDB is a distributed database that is capable of centralizing data retrieved from different places without requiring a predefined schema. In our database are stored all the features of the product line domain, the variants extracted from new evolutions, and the content of the dictionary. The mapping between the system classes and the data from the tables is conducted using the Java MongoDB Driver [30]. This driver provides both synchronous and asynchronous interaction with MongoDB and manipulates the data with Java.

Finally, the algorithms of duplication detection and the transformation of inputs to the new tree-like structure are all implemented using Java code.

4. Case Study: CRM Product Line

In this section, we first present the domain and application models of our CRM and the specification of an evolution. Then, we explain the test performed on this evolution using FDDetector and we present the

achieved results.

4.1. The CRM Presentation

The inputs of our test are the domain model, the application model and the specification of a new evolution.

4.1.1. Domain model

The domain model of the CRM presented in Fig. 2 contains 37 features that cover four main functions, namely sales force management, marketing, customer support and other additional services. The first function includes sales monitoring, customer management, the management of visits to the stores buying the company products and services, the management of sector headers' schedules, and the generation of reports used by managers to make business decisions. The sub-functions related to marketing are the prospection of new customers, the retention of the existing customers, advertising and the management of commercial operations. As for customer support, the sub-functions supported by the CRM are request management as well as pre-sales and after-sales services. The product line provides also additional functions that are used by the other functions. These functions are store management, offer management (an offer is a product or a service provided by the company), training management (training for the company sales force), and finally actor and profile management (to manage the users and their habilitations).

4.1.2. Application models

Based on the domain model, we can derive a variety of customized applications to respond to different customers' needs. In Fig. 3, we present two derived applications.

- **Application 1:** The client who demanded this application provides its services in stores. There are two kinds of stores managed by this client : i) the stores that belong to the company and buys exclusively its products and services, ii) the stores that are not part of the company and thus buys also the products and services of other companies. To access the application, users don't need an Internet connection. The application gives the possibility to sector headers to manage their visits to stores by helping them preparing their visits before they go and writing their reports after they finish. It also allows the management of visit schedules. Regarding prospection, the application facilitates the collection of potential customers' information via an internal database, by contacting them by phone directly, or by scheduling an appointment. As part of customer support, the system includes request management and after-sales service. Other functionality supported by the application is the management of existing and potential customers and also the management of commercial actions and advertisements.

- **Application 2:** The company that uses this application does not have stores. It provides its products and services via a website. All the sales are then made online. The application allows customer management and meeting scheduling. It also supports the management of commercial actions and customer prospection. Collecting information about potential customers can be done by accessing an external database, or by contacting the clients directly by telephone or by email. As for customer support, it is limited to the activity of request management.

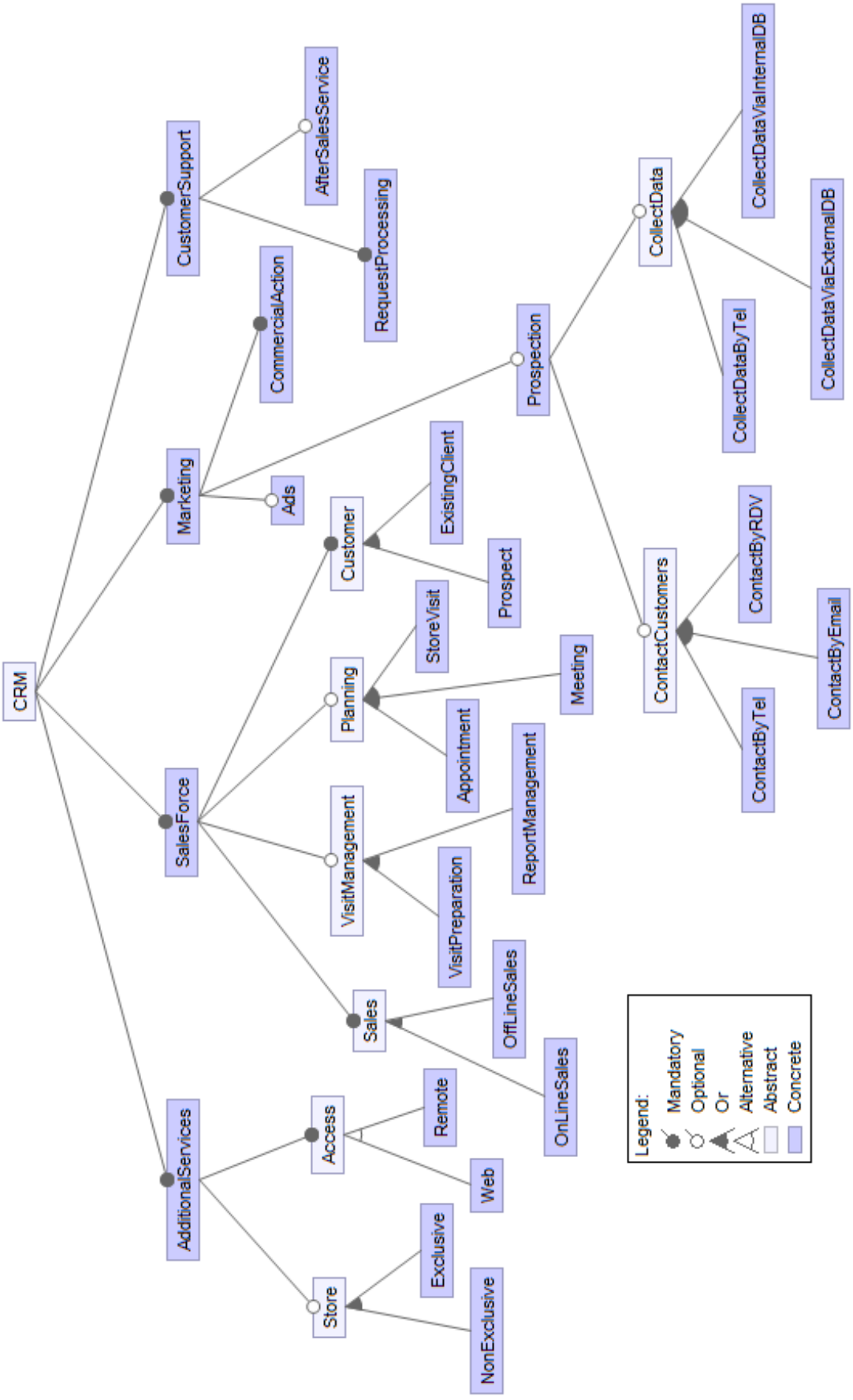

Fig. 2. The CRM domain model

Fig. 3. The configurations of the two applications

4.1.3. Evolution specification

As we notice in the second application, many features from the domain model are absent. The objective of this evolution is to enrich the application by adding features from the domain model and other new features that are not covered by the current version of the product line. For this, we considered the textual specification depicted in Fig. 4.

The specification

The new application must support stores management. The stores can belong to the company itself or to another chain. The sales of the company include both sales on the company's website and stores' sales. The visits to stores must be managed by allowing the sector header to produce visits reports. The sector header is also able to manage his planning of visits and meetings, and to generate summary reports in Excel and Business Object. As part of prospecting, the customers can be contacted by telephone or email, or by setting up an appointment. In addition, activities of customers' retention must be taken into account. Regarding Customer support, the system has to support the new activity of pre-sales service. Another new need concerns the addition of a module that deals with internal and external training. The reporting module will be responsible for the extraction of summaries to Excel and BO. The new version of the application must also handle customer loyalty as part of Marketing activities.

Fig. 4. The specification of the new evolution

4.2. Test and Results

The aim of our test is to detect the internal duplications inside the specification of the new evolution and the external duplications between this specification and the existing feature models.

4.2.1. Model verification

In order to upload the XML file corresponding to the domain model, we use the option « Open Domain Model » of FDDetector main interface. Fig. 5 presents the uploaded model. The displayed interface gives also the possibility to load the model features into the repository if it is not already done. To detect internal duplications in the domain model, we use the function « Get potential inner duplications ». The output of this operation is a log file that contains the list of duplicate features and that has the same format as the log fileof Fig. 8. To add the application model shown at the right of Fig. 3, we use the option « Open Application Model » of the main interface. This option enables to upload the configuration file related to this application in the .txt format. In our test, both the domain and application models do not contain duplications.

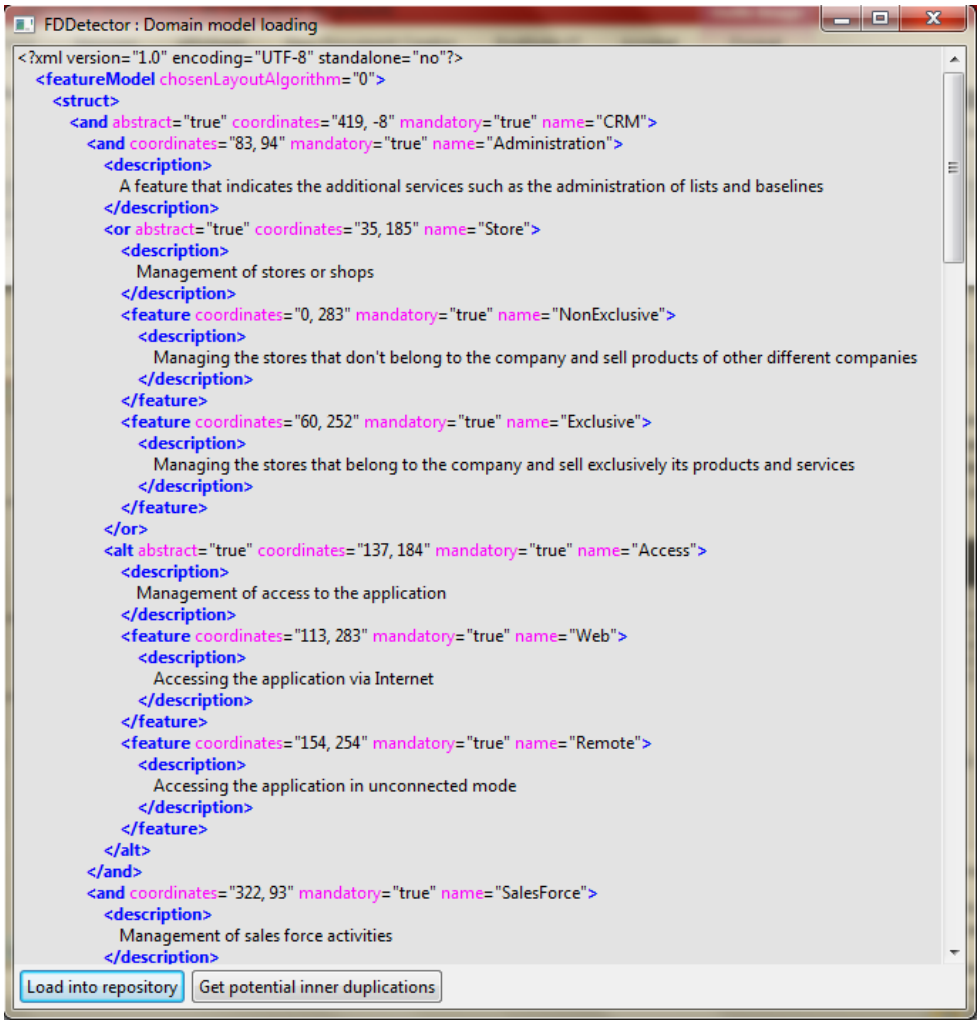

Fig. 5. The domain model loading

4.2.2. Specification verification

The second input of our test is the specification of the new evolution in question. As explained earlier, we have introduced in the specification some requirements that represent the same functions. In addition, some functions demanded in the specification are already implemented in the system. To load the specification, we use the option « Load the specification » of the main interface. The result is shown in Fig. 6.

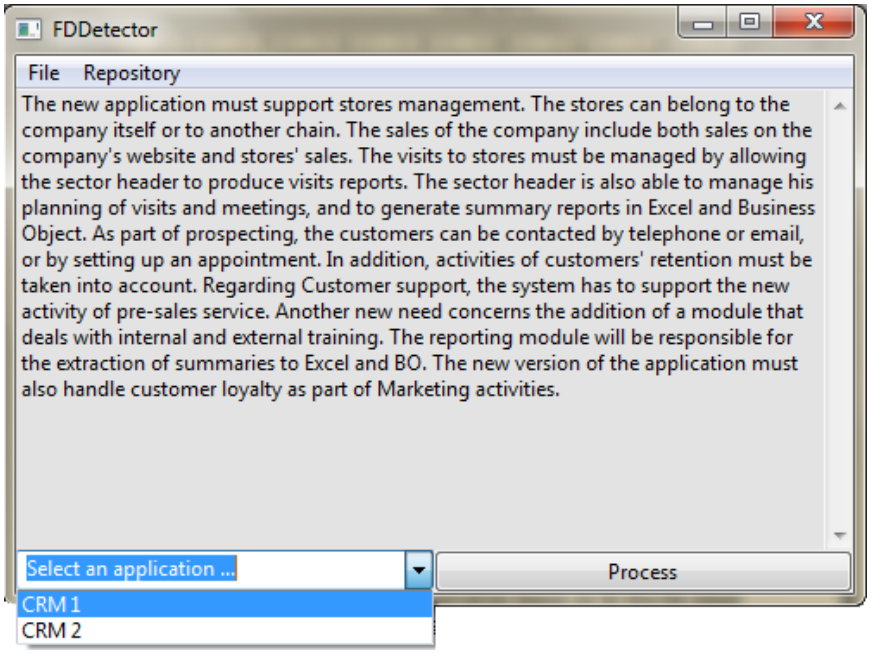

Fig. 6. The specification loading

After choosing the evolving application, we use the function « Process » that triggers the specification analysis to extract the inner features and to generate the tree-like structure. Fig. 7 presents the graph constructed from the extracted features. This presentation facilitates the visualization of the different features introduced by the specification as well as the identification of duplicated features by displaying them in a different color (red). In this test, three internal duplications were detected as shown in the upper-left corner of the window. In addition, the graph helps identifying the new variants added by the specification by associating them to the node « unbound variants », which enables the user to link them with existing variation points from the repository.

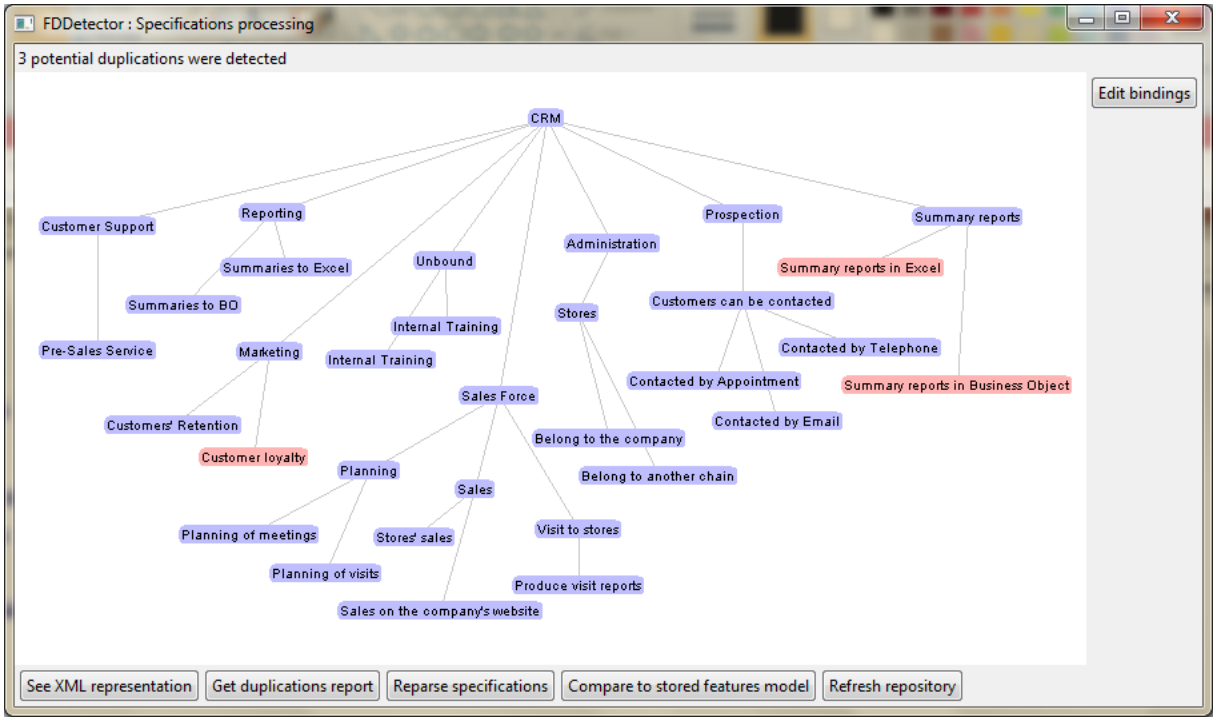

Fig. 7. The graph generated for the specification

This interface provides also other functions, namely: generating the XML file corresponding to the new specification, re-processing the specification after the modification of the repository, refreshing the repository after linking new features, and generating duplication logs. The log generated in our test is depicted in Fig. 8. It displays the total number of variants existing in the specification and the number of the detected duplicates. It also contains the sentences of the specification where the duplications were detected.

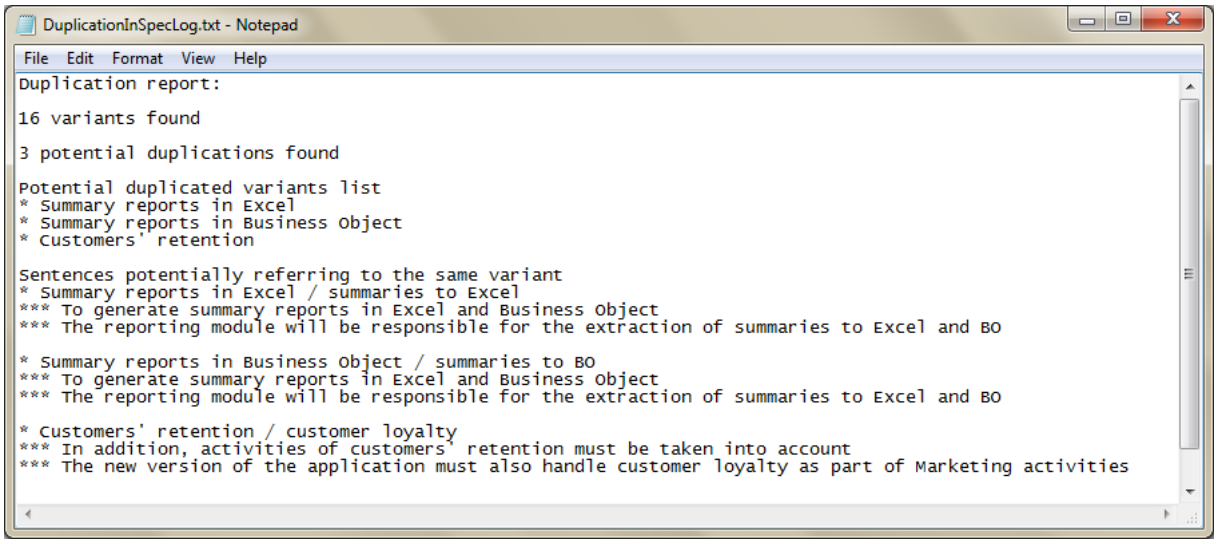

Fig. 8. The log of duplications detected in the specification

4.2.3. Duplication detection

At the aim of comparing the new features introduced by the specification with the existing features in the domain and application models, we use the function « Compare to feature models ». The output of this operation is the list of detected duplications against the two models. The potential duplications that should be detected are presented in Table 1.

Table 1. The List of Duplicated Features and New features

Feature	*Variation Point*	*Variant 1*	*Variant 2*	*Variant 3*
The new application must support stores management. The stores can belong to the company itself or to another chain.	DOM.Store	DOM.Exclusive	DOM. NonExclusive	
The sales of the company include both sales on the company's website and stores' sales.	APP.Sales	APP.OnLineSales	DOM. OffLineSales	
The visits to stores must be managed by allowing the sector header to produce visits reports.	DOM. VisitManagement	DOM.ReportMgmt		
The sector header is also able to manage his planning of visits and meetings	APP.Planning	APP.Meeting	DOM.StoreVisit	
The customers can be contacted by telephone or email, or by setting up an appointment.	APP. ContactCustomers	APP.Tel	APP.Email	DOM.RDV

Activities of customers' retention must be taken into account	APP.Marketing	NEW. CustomerRetention		
Regarding Customer support, the system has to support the new activity of pre-sales service	APP. CustomerSupport	NEW. PreSalesService		
The addition of a module that deals with internal and external training	NEW.Training	NEW. InternalTraining	NEW. ExternalTraining	
The reporting module will be responsible for the extraction of summaries to Excel and BO	NEW.Reporting	NEW.ExcelReport	NEW.BOReport	

The table cells that are blue correspond to the duplicated features against the domain model. The orange cells consist of the duplications against the application model. The green cells represent the new features.

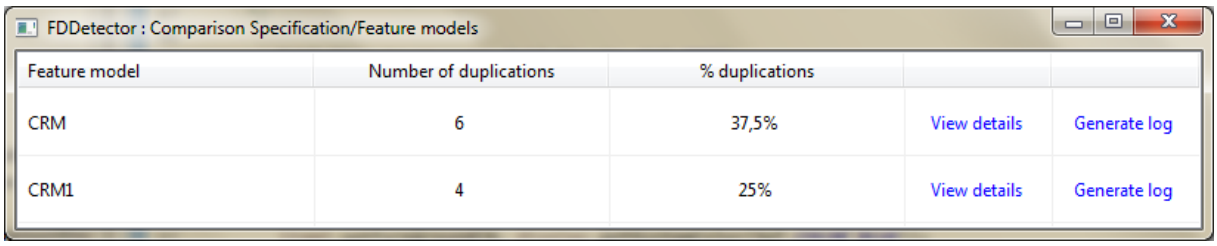

Fig. 9. The results of external duplication detection.

The table of Fig. 9 is displayed by the tool and shows the test results. In our case, six duplications were detected against the domain model and four duplications against the application model. For the first category, the features can be implemented by deriving them directly from the domain model. For the second category, features should not be implemented because they already exist in the application in question. The table presents also the duplication percentage for each model. These percentages are calculated as follows:

- $\%DomainDuplication = \frac{Number\ of\ duplicate\ variants\ between\ the\ specification\ and\ the\ domain}{Number\ of\ all\ the\ variants\ detected\ in\ the\ specification} \times 100$
- $\%ApplicationDuplication = \frac{Number\ of\ duplicate\ variants\ between\ the\ specification\ and\ the\ application}{Number\ of\ all\ the\ variants\ detected\ in\ the\ specification} \times 100$

The log of duplications can be generated via the link « Generate log ». The comparison table and the generated log are used by the analyst to take the right decisions concerning the duplicate features, namely deleting the new features, replacing the existing features by new ones, or the modification of the new features.

4.2.4. Evaluation

In order to evaluate the efficiency and precision of FDDetector, we use three metrics: the recall, the precision [31], and the F-measure [32].

- **Recall**: The ratio of the number of relevant duplications detected by the tool to the number of all the duplications that really exist and that must be detected by the tool. Thus, the recall is high if the tool is capable of detecting a large number of relevant duplications comparing to what it must detect.

$$Recall = \frac{Number\ of\ relevant\ duplications\ detected}{Number\ of\ detectable\ relevant\ duplications}$$

- **Precision:** The ratio of the number of relevant duplications detected by the tool to the number of all the duplications detected by the tool (relevant and irrelevant). Thus, the precision is high if a large number of duplications detected by the tool are relevant.

$$Precision = \frac{Number\ of\ relevant\ duplications\ detected}{Number\ of\ all\ detected\ duplications}$$

- **F-Mesure:** The harmonic mean of the recall and the precision. The best value of this measure is 1 (achieved when the precision and the recall are both perfect). The worst value is 0 (achieved when both the precision and the recall are poor).

$$F - Measure = 2 \times \left(\frac{Precision \times Recall}{Precision + Recall}\right)$$

The three metrics can be calculated for the two types of duplications (internal and external) and for duplications against domain and application models.

4.2.5. Example: Internal duplication in a specification

If the specification contains 5 duplications and the tool detects 3 duplications (two relevant duplications and one irrelevant duplication), then:

$$Recall = \frac{2}{5} = 0{,}4$$

$$Precision = \frac{2}{3} = 0{,}67$$

$$F - Measure = 2 \times \left(\frac{\frac{2}{3} \times \frac{2}{5}}{\frac{2}{3} + \frac{2}{5}}\right) = 0{,}5$$

In this case, the tool is said to have an accuracy of 50%.

4.2.6. Application on the CRM case study

Concerning our case study, we detected 3 duplications in the specification, 4 duplications between the specification and the application model, and 6 duplications between the specification and the domain model. The test has been performed in several iterations. In each iteration, we considered new annotations of the product line general specifications, we added more information and descriptions to the domain features, and we filled the dictionary with new entries. This helped us detect all the duplications introduced by the new evolution and to achieve a high degree of accuracy (F-Measure=1).

Through this use case, we could prove that our solution is capable of detecting internal duplication inside specifications and external duplication between the specification and the models. However, the CRM adopted in this test does not contain a large number of features and the number of duplications introduced by the evolution is not very significant. Therefore, to evaluate the tool scalability, we need a large scale CRM with high number of features and big evolutions. We can also consider a quantitative study to measure the gain in time and cost related to the use of FDDetector.

5. Conclusion

Among the several model defects that may occur in software product lines, we focus in our work on feature duplication. In this vein, we proposed in previous papers a conceptual framework that describes the different processes to follow to detect and correct duplications in product lines. We have also developed a tool support called FDDetector based on this framework for the same purpose. To test and evaluate our tool, we have opted for a case study from the CRM field. In this paper, we explained the reasons behind choosing a CRM product line, we detailed the different steps performed in our test and we provided the results achieved. We also presented the metrics used to evaluate the tool accuracy. As the adopted CRM contains limited number of features, we intend, in future work, to apply our solution on a large-scale product line to prove the effectiveness and scalability of our tool. We also plan to develop other versions of the tool that support new functionality.

References

[1] Elfaki, A. O. (2016). A rule-based approach to detect and prevent inconsistency in the domain-engineering process. *Expert Systems*, *33(1)*, 3-13.

[2] Alférez, M., Lopez-Herrejon, R. E., Moreira, A., Amaral, V., & Egyed, A. (2014). Consistency checking in early software product line specifications-the VCC approach.

[3] Quinton, C., Pleuss, A., Berre, D. L., Duchien, L., & Botterweck, G. (2014, September). Consistency checking for the evolution of cardinality-based feature models. *Proceedings of the 18th International Software Product Line Conference-Volume 1* (pp. 122-131).

[4] Stephenson, Z., Attwood, K., & McDermid, J. (2011). Product-line models to address requirements uncertainty, volatility and risk. *Relating Software Requirements and Architectures*, Springer, Berlin, Heidelberg.

[5] Neves, L., Borba, P., Alves, V., Turnes, L., Teixeira, L., Sena, D., & Kulesza, U. (2015). Safe evolution templates for software product lines. *Journal of Systems and Software*, *106*, 42-58.

[6] Groher, I., Reder, A., & Egyed, A. (2010, March). Incremental consistency checking of dynamic constraints. *Proceedings of the International Conference on Fundamental Approaches to Software Engineering* (pp. 203-217). Springer, Berlin, Heidelberg.

[7] Khtira, A., Benlarabi, A., & El Asri, B. (2018). Model defects in evolving software product lines: A review of literature. *American Scientific research Journal for Engineering, Technology, and Sciences*, *45(1)*, 20-41.

[8] Sajnani, H. (2016). Large-Scale code clone detection (Doctoral dissertation, UC Irvine).

[9] Dang, Y., Zhang, D., Ge, S., Huang, R., Chu, C., & Xie, T. (2017, May). Transferring code-clone detection and analysis to practice. *Proceedings of 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track*.

[10] Svajlenko, J. T. (2018). Large-scale clone detection and benchmarking (Doctoral dissertation, University of Saskatchewan).

[11] Khtira, A., Benlarabi, A., & Asri, B. (2015). Duplication detection when evolving feature models of software product lines. *Information*, *6(4)*, 592-612.

[12] Khtira, A., Benlarabi, A., & El Asri, B. (2018). Modelling and correcting duplication in evolving software product lines. *International Journal of Computer Science Issues* (IJCSI), *15(4)*, 29-39.

[13] Khtira, A., Benlarabi, A., & El Asri, B. (2015). A tool support for automatic detection of duplicate features during software product lines evolution. *International Journal of Computer Science Issues* (IJCSI), *12(4)*, 1.

[14] Khtira, A., Benlarabi, A., & El Asri, B. (2019). FDDetector: A tool for deduplicating features in software product lines. *American Scientific Research Journal for Engineering, Technology, and Sciences* (ASRJETS),

62(1), 192-209.
[15] Andrew, H., & David, T. (2000). The Pragmatic Programmer: From Journeyman to Master.
[16] Tonscheidt, K. (2015). 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*, 4-16.
[17] Koschke, R. (2008). Identifying and removing software clones. In *Software Evolution* (pp. 15-36). Springer, Berlin, Heidelberg.
[18] Koschke, R. (2007). Survey of research on software clones. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
[19] Juergens, E., Deissenboeck, F., Hummel, B., & Wagner, S. (2009, May). Do code clones matter?. *Proceedings of 2009 IEEE 31st International Conference on Software Engineering* (pp. 485-495). IEEE.
[20] Schubanz, M., Pleuss, A., Botterweck, G., & Lewerentz, C. (2012, January). Modeling rationale over time to support product line evolution planning. *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems* (pp. 193-199). ACM.
[21] Van der Linden, F. J., Schmid, K., & Rommes, E. (2007). *Software product lines in action: the best industrial practice in product line engineering*. Springer Science & Business Media.
[22] Edwards, D. (2014, September). "Enabling DevOps Success with Shift Left Continuous Testing". Retrieved January, 2020, from the IBM website https://www.ibm.com/developerworks/community/blogs/invisiblethread/entry/enabling_devops_success_with_shift_left_continuous_testing?lang=en
[23] Kasse, T. (2008). *Practical insight into CMMI*. Artech House.
[24] Kastner, C., Thum, T., Saake, G., Feigenspan, J., Leich, T., Wielgorz, F., & Apel, S. (2009, May). FeatureIDE: A tool framework for feature-oriented software development. In Proceedings of the 31st International Conference on Software Engineering (pp. 611-614). IEEE Computer Society.
[25] Khtira, A., Benlarabi, A., & El Asri, B. (2015, April). Detecting feature duplication in natural language specifications when evolving software product lines. In 2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE) (pp. 257-262). IEEE.
[26] The Eclipse Foundation. (2020). "SWT: The Standard Widget Toolkit". Retrieved January, 2020, from the Eclipse website: www.eclipse.org/swt/
[27] The Apache software foundation. (2020). "OpenNLP". Retrieved January, 2020, from the OpenNLP website: opennlp.apache.org
[28] Prefuse. (2020). "The prefuse visualization toolkit". Retrieved January, 2020, from the Prefuse website: https://web.archive.org/web/20181226190156/http://prefuse.org/
[29] MongoDB. (2020). "What is MongoDB?". Retrieved January, 2020, from the MongoDB website: https://www.mongodb.com/what-is-mongodb
[30] MongoDB, (2020). "Java MongoDB Driver". Retrieved January, 2020, from the MongoDB website: https://docs.mongodb.com/ecosystem/drivers/java/
[31] Carterette, B. (2009). *Precision and Recall. Encyclopedia of Database Systems*, 2126-2127.
[32] Zhang, E., & Zhang, Y. (2009). *F-measure. Encyclopedia of Database Systems*, 1147-1147.

Amal Khtira received in 2008 a degree in software engineering from National High School of Computer Science and System Analysis (ENSIAS), Mohamed V University, Rabat. She has been working in the IMS (Models and Systems Engineering) Team of ADMIR Laboratory at ENSIAS since 2012 and obtained a PhD degree in computer science in 2019. Her research interests include software product line engineering,

requirements engineering, feature modeling, software evolution, machine learning and natural language processing.

Anissa Benlarabi received in 2010 a degree in software engineering from National High School of Computer Science and System Analysis (ENSIAS), Mohamed V University, Rabat. She obtained a PhD degree in software product line evolution issues in 2018 from ENSIAS. She has been working with the IMS Team of ADMIR Laboratory at ENSIAS on many challenges related to software product lines.

Bouchra El Asri is a professor of higher education in the Software Engineering Department at ENSIAS, Mohamed V University, Rabat. She is also a member of the IMS Team of ADMIR Laboratory in the same school. Her research interests include service-oriented computing, model-driven engineering, cloud computing, component-based systems, software product line engineering and big data.