Software Variability Management

An Exploratory Study with Two Feature Modeling Tools

Juliana Alves Pereira, Carlos Souza, Eduardo Figueiredo
Computer Science Department,
Federal University of Minas Gerais (UFMG)
Belo Horizonte, Brazil
{juliana.pereira, carlosgsouza, figueiredo}@dcc.ufmg.br

Ramon Abilio, Gustavo Vale, Heitor Augustus Xavier Costa
Computer Science Department,
Federal University of Lavras (UFLA)
Lavras, Brazil
ramon@posgrad.ufla.br,
gustavo.a.vale@sistemas.ufla.br,
heitor@dcc.ufla.br

***Abstract*—Software Product Line (SPL) is becoming widely adopted in industry due to its capability of minimizing costs and improving quality of software systems through systematic reuse of software artifacts. An SPL is a set of software systems sharing a common, managed set of features that satisfies the specific needs of a particular market segment. A feature represents an increment in functionality relevant to some stakeholders. There are several tools to support variability management by modeling features in SPL. However, it is hard for a developer to choose the most appropriate feature modeling tool due to the several options available. This paper presents the results of an exploratory study aiming to support SPL engineers choosing the feature modeling tool that best fits their needs. This exploratory study compares and analyzes two feature modeling tools, namely FeatureIDE and SPLOT, based on data from 56 participants that used the analyzed tools. In this study, we performed a four-dimension qualitative analysis with respect to common functionalities provided by feature modeling tools: (i) Feature Model Editor, (ii) Automated Analysis of Feature Models, (iii) Product Configuration, and (iv) Tool Notation. The main issues we observed in SPLOT are related to its interface. FeatureIDE, on the other hand, revealed some constraints when creating feature models.**

Keywords—software product line, feature models, SPLOT, FeatureIDE.

I. INTRODUCTION

The growing need for developing larger and more complex software systems demands better support for reusable software artifacts [21] [28]. In order to address these demands, Software Product Line (SPL) has been increasingly adopted in software industry [1] [41]. "An SPL is a set of software intensive systems sharing a common, managed set of features that satisfies the specific needs of a particular market segment or mission [28]". It is developed from a common set of core assets and variable features [11]. "A feature represents an increment in functionality relevant to some stakeholders [22]".

The high degree of similarity among software systems in a specific domain favors the adoption of SPL [11]. This adoption is expected to bring significant improvements to the software development process [28]. Due to these expected benefits, large companies [41], such as Boeing, Hewlett Packard, Nokia, and Siemens, have moved forward towards adopting SPL practices.

Feature modeling tools are used to support the management of variability in an SPL. These tools support the representation and management of reusable artifacts instead of providing means for conventional development from scratch. There are many available options of feature modeling tools [17] [22] [23] [29] [26] [39]. Therefore, choosing one tool that best meets the SPL development goals is far from trivial.

After surveying several tools for variability management of SPL, this paper presents a detailed qualitative analysis of two feature modeling tools, namely SPLOT [26] and FeatureIDE [23]. We choose to focus our analysis on these tools because they provide the key functionality of typical feature modeling tools, such as to create and edit a feature model, to automatically analyze the feature model, and to configure a product. In addition, we decide to exclude early prototypes [17] [39] and proprietary tools [29] from our study because they could hinder some sorts of analyses. In addition, early prototype tools (i) do not cover all relevant functionalities we aim to evaluate and (ii) are not applicable to industry-strength SPL.

This exploratory qualitative study involved 56 young developers taking an advanced SE course. Each participant used only one tool, either SPLOT or FeatureIDE. We relied on a background questionnaire and a 1.5-hour training session to balance knowledge of the study participants. The experimental tasks included typical variability management functionalities. After that, participants answered a survey with open questions about the functionalities they used in each tool.

The study results may contribute with relevant information to support software engineers to choose a tool for the development and management of variability in SPL that best fits their needs. We focus our analysis on four functionalities available in typical feature modeling tools: Feature Model Editor, Automated

We would like to acknowledge CNPq, FAPEMIG and CAPES.

Analysis of Feature Models, Product Configuration, and the Feature Model Notation used by each tool. Based on this analysis, we uncover several interesting findings about the analyzed tools.

First, we observed that most participants like the user interface of both SPLOT and FeatureIDE. However, although shortcuts and automatic feature model organization work fine in FeatureIDE, they were considered an issue by many SPLOT users. With respect to automated analysis, both tools present statistical information, but SPLOT succeeds by presenting data in an easier way to understand. Participants seem to be happy with SPLOT because it provides steps to guide users during the product configuration. However, a configuration can be created in SPLOT, but it cannot be saved. FeatureIDE, on the other hand, allows us to create multiple product configurations, to save them, and to set the default one.

The remainder of this paper is organized as follows. Section 2 briefly reviews the feature modeling concepts. In Section 3, the study is set and some characteristics of SPLOT and FeatureIDE are described. Section 4 reports and analyzes the results of this exploratory qualitative study. In Section 5, some threats to the study validity are discussed. Section 6 concludes this paper by summarizing its main contributions and pointing out directions for future work.

II. Software Product Line

Large software companies have adopted SPL to develop their products [41]. SPL makes companies more competitive by providing large scale reuse with mass customization [28]. This section introduces SPL, focusing on feature model notations and tools.

A. Feature Modeling

Feature models are popular for representing variability in an SPL [13]. A feature model is a way to represent the space of possible configurations of all products in an SPL [2] [13]. It allows the visualization of hierarchical features and their relationships [14] [27]. Fig. 1 shows a feature model of a system, called MobileMedia [17]. Nodes in this figure represent features and edges show relationships between them. A single root node, MobileMedia, represents the concept of the domain being modeled.

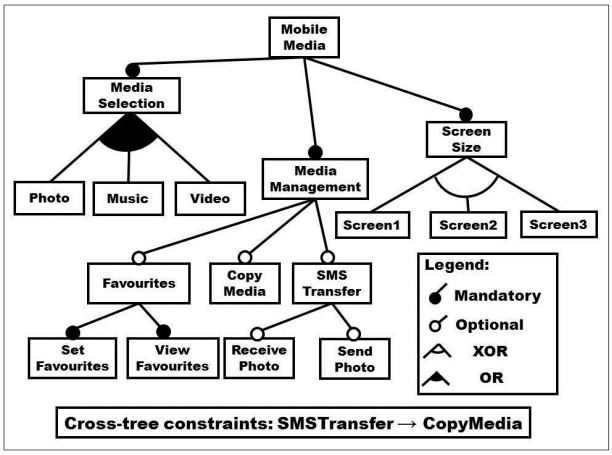

Fig. 1. Mobile media feature model

Features in a feature model can be classified as mandatory, optional, and alternative. Optional features are represented with an empty circle, such as Receive Photo in Fig. 1. They may or may not be part of a product. On the other hand, mandatory features, such as Media Management, are represented by filled circles and are part of all SPL products containing their parent feature. Alternative features may be exclusive (XOR) or not exclusive (OR). The former indicates that only one sub-feature can be selected from the alternatives. For example, Screen 1, Screen 2 and Screen 3 in Fig. 1 are alternative features for Screen Size. OR features, such as Photo and Music, allow the selection of more than one option for a product.

In addition to features and their relationships, a feature model can also include composition rules. A composition rule refers to additional cross-tree constraints to restrict feature combinations [13]. It is responsible for validating a combination of unrelated features. Typical cross-tree constraints are inclusion or exclusion statements in the form "if feature F1 is included, then feature F2 must also be included (or excluded)". For example, the feature model in Fig. 1 shows an inclusion constraint between SMS Transfer and Copy Media. That is, in order to receive a photo via SMS, this photo has to be copied in an album.

B. Survey of Feature Model Notations

Several modeling notations can be found in the literature [7] [12] [15] [19] [22] [37]. A feature model is an artifact generated during the domain analysis and used to describe a set of features and their relationships into the domain. A method for domain analysis is called Feature-Oriented Domain Analysis (FODA) proposed by Kang [22] which uses a graphical representation (feature model) to show the identified features.

Some FODA's extensions can be found in the literature, such as, (i) Feature-Oriented Reuse Method (FORM) [22], (ii) Generative Programming Feature Tree (GPFT) [12], (iii) Van Gurp and Bosch Feature Diagram (VBFD) [37], (iv) Variability Feature Diagram (VFD) [7], (v) Product Line Use Case Modeling for System and Software engineering (PLUSS) [15], (vi) FeatuRSEB (combination of FODA and Reuse-Driven Software Engineering Business) [19], (vii) UML-Based Feature Models (UML-BFM) [14], and (viii) Integrating Feature Modeling into UML (IFM-UML) [38]. These extensions changed how the features and their relationships are depicted. For instance, features are represented with a rectangle around its name in FORM. An OR decomposition was added and the representation of XOR decomposition was modified in FeatuRSEB. UML-BFM and IFM-UML extensions use UML notation.

In addition to graphical representation, variability in SPL can be represented also by text-based models. These models use structured text to describe features and its relationships. Examples of the languages used to write text-based models are: (i) Textual Variability

Language (TVL) [8]; (ii) Clafer [4]; (iii) GUIDSL [3]; and iv) SXFM [26]. In Section V, we discuss the notations used in SPLOT and FeatureIDE.

C. Feature Modeling Tools

Since feature models are undergoing a rapid process of maturation, feature modeling tools are constantly being developed and adopted in practice. In order to assist modeling and management of SPL, several tools are already available, such as SPLOT [26], FeatureIDE [23], XFeature [39], FMP [18], and Pure::Variants [29]. We performed a survey of tools for variability management before choosing SPLOT [26] and FeatureIDE [23] as representatives. This section presents a brief overview of five tools for variability management in SPL.

SPLOT [26] is a Web-based tool for creating feature models and product configuration. SPLOT does not provide means for generation or integration of code. At the tool website, we can find a repository of more than 200 feature models created by tool users over 3 years. This is a free and open source project. You can download the tool's code and also SPLAR (a Java library created by the authors to perform the analysis of feature models). It also provides a standalone tool version that can be installed in a private machine. There is also an interesting feature, called workflow configuration, which defines a flow for product configuration. By using this feature, many people interact with each other in a collaborative way to configure an SPL product.

FeatureIDE [23] is a tool which widely covers the SPL development process. Besides having feature model editors and configuration of products, it is integrated with several programming and composition languages with a focus on development for reuse. FeatureIDE was developed to support both aspect-oriented [24] and feature oriented programming [3]. This tool is implemented as an Eclipse plugin and can be downloaded separately or in a package with all dependencies needed for implementation.

XFeature [39] is a modeling tool implemented as an Eclipse plugin whose main goal is to automate the modeling and configuration of reusable artifacts. Initially, the XFeature was created to assist the development of space applications. However, it currently supports the general development of SPL. Focused on creating models and meta-models, the XFeature is still in a proof of concept stage and is rarely used in the software industry.

Similar to XFeature, FMP[1] [18] is a tool implemented as an Eclipse plugin which focuses on the variability modeling of SPL. Developed at the University of Waterloo, it was supposed to be integrated with Rational Software Modeler (RSM) and Rational Software Architect (RSA). However, the project is currently discontinued.

Developed by Pure Systems, a company specializing in the reuse-based development of software, Pure::Variants [29] is a mature tool for the development of SPL. It can be used not only as an Eclipse plugin, but it can also be integrated into some commercial tools, such as Enterprise Architect and Rational Rhapsody from IBM.

[1] FMP stands for Feature Modeling Plugin

III. Study Settings

This section presents the study configuration aiming to evaluate two alternative feature modeling tools, namely SPLOT and FeatureIDE. Section III.A compares these tools and Section III.B summarizes the background information of participants that took part in this study. Section III.C explains the training session and tasks assigned to each participant.

A. Selection of the Analyzed Tools

SPLOT and FeatureIDE tools were selected for this study. We focus our analysis on both tools because these tools are mature and used in large software projects. Other tools, such as FMP and XFeature, are only academic prototypes and do not provide all functionalities available in professional tools. We also aimed to select mature, actively developed, and accessible tools in order to evaluate the state-of-the-art in feature modeling. Therefore, we also excluded proprietary tools, such as Pure::Variants, because proprietary tools could hinder some sorts of analyses. For instance, we do not have access to all the features of the tool.

Through using both tools, the key features mentioned by participants in the study are summarizes in Table I.

TABLE I. Functionalities of SPLOT and FeatureIDE

Functionality	SPLOT	FeatureIDE
Feature model editor	✓	✓
Automated feature model analysis	✓	✓
Interactive product configuration	✓	✓
Feature model notation	tree	tree and diagram
Integration with code		✓
Available online	✓	
Repository features models	✓	
Configuration workflow	✓	

B. Participants

Participants involved in this study are 56 young developers (between 20 to 32 years) taking an advanced Software Engineering course spanning four consecutive semesters from 2011-1 to 2012-2. All participants are graduated or close to graduate since the course targets post-graduated MSc and PhD students. To avoid biasing the study results, each participant only took part in one study semester and only used one tool, either SPLOT or FeatureIDE. That is, only one tool was used in each semester as indicated in Table II. FeatureIDE was used by 27 participants being 6 in the first and 21 in the second semester. Additionally, SPLOT was used by 29 participants being 15 in the first and 14 in the second semester. Each participant worked individually to accomplish the study tasks (Section 3.3). Participants in

TABLE II. BACKGROUND OF PARTICIPANTS

	T1 (2011-1)	T2 (2011-2)	T3 (2012-1)	T4 (2012-2)	No Answer
	FeatureIDE		SPLOT		
Work Experience	S01, S03, S04, S06	S02, S04, S05, S07, S14, S18	S04, S07, S09, S10, S14, S15	S01-S03, S05-S12	T1: S02
UML Design	S01, S03, S04 - S06	S02, S04, S05, S08, S14, S18, S20	S03, S04, S07-S10, S12, S14, S15	S02, S04, S05, S08-S12	T2: S13, S16
Java Programming	S01, S03, S04 - S06	S01, S02, S04-S06, S08-S12, S14, S15, S17, S18, S20, S21	S03, S04, S07-S10, S12, S14, S15	S01-S03, S05-S12	T3: S01, S02, S05, S06, S11, S13
# of Participants	6	21	15	14	

the 1st semester are named T1-S01 to T1-S06; in the 2nd semester, T2-S01 to T2-S21; in the 3rd semester, T3-S01 to T3-S15; and in the 4th semester, T4-S01 to T4-S14.

Before starting the experiment, we used a background questionnaire to acquire previous knowledge about the participants. Table II summarizes knowledge that participants claimed to have in the background questionnaire with respect to work experience, UML design, and Java Programming. Second, third, fourth and fifth columns in this table show the participants who claimed to have knowledge medium or high in a particular skill. Answering the questionnaire is not compulsory and participants who have not answered it are annotated in the last column (No Answer). However, although some cases participants who chose not to answer the questionnaire, we observe in Table II that, in general, all participants have at least basic knowledge in software development and technology.

C. Training Session and Tasks

We conducted a 1.5 hour training session where we introduced participants not only to the analyzed tools but also to the basic concepts of feature modeling and SPL. The same training session (with the same content and instructor) was performed in all four groups (2011-1, 2011-2, 2012-1 and 2012-2). After the training session, we asked participants to perform some tasks using either SPLOT or FeatureIDE (see Table II). The tasks include using functionalities (i) to create and edit a feature model, (ii) to automatically analyze the feature model created and observe its statistics, and (iii) to configure a product relying on the tool product configuration functionality. Finally, we ask participants to answer a questionnaire with two simple questions about functionalities of the tool that they like and dislike. Questionnaire with open questions was used as an initial study to list the main functionality provided by the tools analyzed (Table I).We focus this paper on the most interesting results, but the questionnaire and all answers are available in the project website [16]. The whole study was performed in a computer laboratory with 25 equally configured equipments.

IV. RESULTS AND ANALYSIS

This section reports and discusses data of this qualitative study. Section 4.1 reports the general quantitative analysis based on answers of the participants. Section 4.2 focuses the discussion on the Feature Model Editor of each tool. Section 4.3 discusses the automated analysis of feature models and Section 4.4 presents the results for the product configuration. Finally, Section 4.5 analyzes the impact of the different notations adopted by each tool.

A. General Analysis

Figures 2, 3, 4, and 5 show the most recurrent answers of participants grouped by category. Both FeatureIDE and SPLOT users like the respective interface of the used tool and, in both cases, in average 63% of participants made some positive comment about it. As indicated by Fig. 2, FeatureIDE also received positive comments about its presentation and usability, such as feature editor usability, reorganization of features, automatic organization, and the keyboard shortcut functionality. What draws our attention mostly, however, is that most participants said that contact with FeatureIDE was very short to proper evaluate it (Fig. 3). About 41% of participants said they could not write any negative comments about this tool. For instance, Participant T2-S19 states: “I believe that with a little more practice I would better understand the tool in order to criticize it”. Similarly, Participant T1-S106 states: “The tool has many features, but some take time to learn to handle. It can be confusing to programmers beginners”. Once the exposure time was the same for both FeatureIDE and SPLOT, we conclude that the FeatureIDE is a more complex tool for users at first glance. Consequently, it requires a longer learning time.

The SPLOT users have been widely praised in relation to the automated analysis and display of statistics about feature models (Fig. 4). In fact, a major goal of SPLOT is to provide statistical information quickly during model creation and product configuration. On the other hand, several participants indicated usability-specific issues in this tool, such as inability to rearrange the features inside an existing model and difficulty in finding buttons (for instance) to save the model. These observations show that the tool is on the right path, but there are still many opportunities to improve its usability. Other issues, such as problems in setting up the model, confusing nomenclature, lacking of integration with others tools, lacking of privacy are also cited by some participants (Fig. 5).

B. Feature Model Editor

Both FeatureIDE and SPLOT have a feature model editor (Figures 6 and 7). Interestingly, participants cited the feature model editor as a functionality they like in

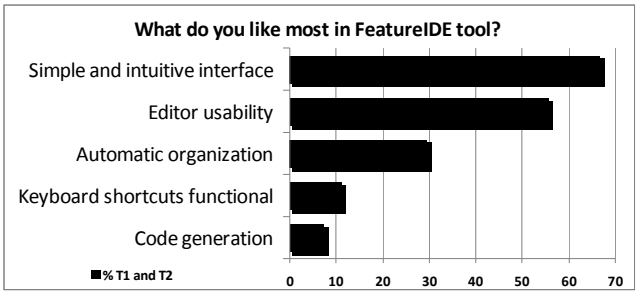

Fig. 2. Positive comments in FeatureIDE

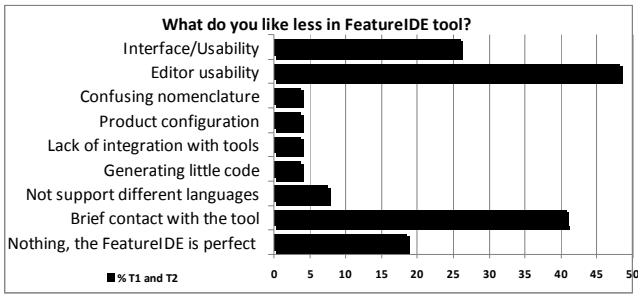

Fig.3. Negative comments in FeatureIDE

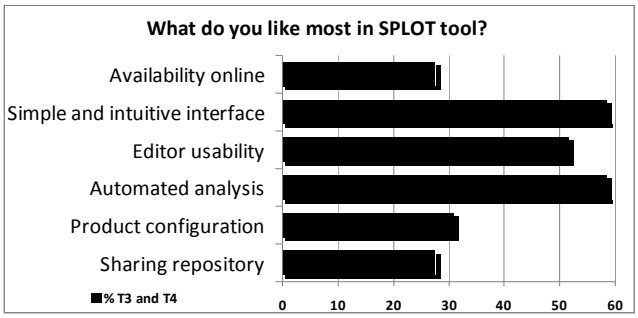

Fig. 4. Positive comments in SPLOT

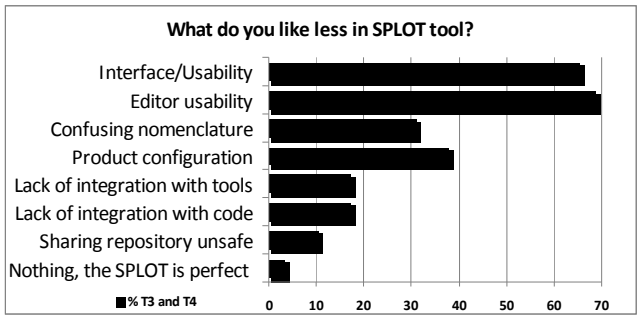

Fig. 5. Negative comments in SPLOT

both tools (Figures 2 and 4). Focusing on FeatureIDE, its editor allows users to create and delete features and constraints. Constraints are listed immediately below the feature model in this tool. Even less experienced participants think the editor interface of FeatureIDE is simple and easy to use. For instance, T2-S17 said that "Even without seeing the tutorial, I used it and performed all tasks right from the first time". Similar comments were made by other participants, such as T2-S21 who stated that "The graphical representation of the feature model is organized and facilitates visualizing the configuration space".

About 57% of participants made positive comments about editor usability. Two positive functionalities were cited: namely automatic organization of features (30% commented on it) and shortcuts for the mostly used functions (12% commented on it). For instance, with respect to the automatic organization of features, Participant T1-S01 stated that "the tool allows a nice view of the feature model because when we insert new features it automatically adjusts spaces between boxes to keep everything on screen". Participant T1-S01 also observed that "shortcuts make it easy to switch between mandatory, optional, and alternative features". Following the same trend, Participant T2-S6 concludes that "shortcuts helped to speed up the feature model creation".

Although participants claimed that the interfaces of both tools are simple and easy to use, some of them pointed out issues with specific functionalities in the feature model editors. For instance, Participant T1-S02 complained about FeatureIDE that "the automatic organization of features in the feature model editor does not allow specific adjustments for better visualization". In fact, Participant T1-S2 contradicts Participant T1-S01

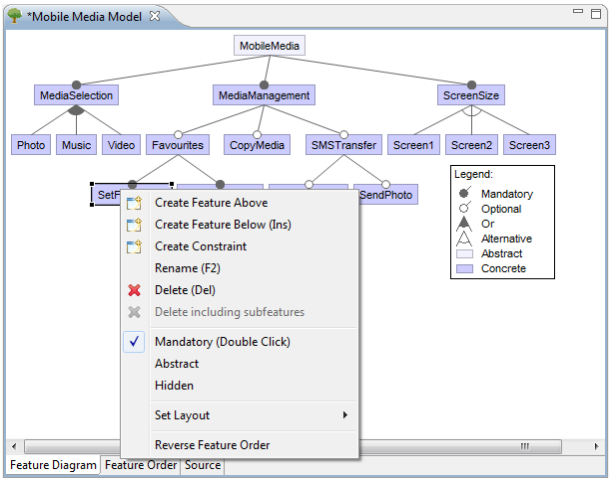

Fig. 6. Feature model editor in FeatureIDE

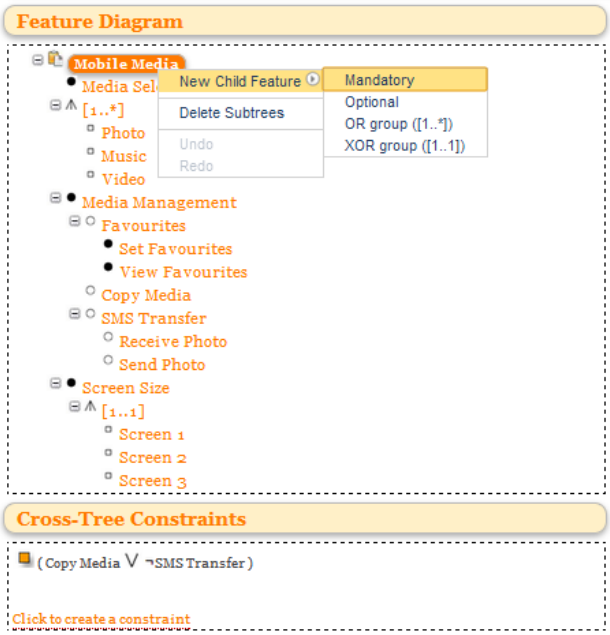

Fig. 7. Feature model editor in SPLOT

(above) about automatic organization of features. However, we observed that, in fact, FeatureIDE users like this functionality, but they also expect more control to disable it when necessary. Another typical complaint was that "an accidental double click in one feature causes it to change, for instance, from optional to mandatory feature" as observed by Participant T2-S7. Again, to solve this problem, users of FeatureIDE expect a way to enable and disable this shortcut for changing the feature type.

With respect to SPLOT, almost 60% of participants like the automatic feature model analysis functionality (Section 4.3). However, participants recurrently complained about the lack of support to restructure a feature model. For instance, Participant T3-S12 said that "there is no way to manually reorder features of a tree by dragging and dropping them". A similar observation is made by Participant T3-S02: "one cannot create an OR or XOR group based on pre-existing features".

The Web interface of SPLOT also led some usability-related issues. One of these issues was raised by Participant T3-S3 saying that "The ENTER button does not work to save changes in a feature; you need to click outside a feature box to validate your changes". Besides the Enter button, another participant also spots an issue with the Delete button. Participant T3-S3 stated that "a feature is not removed by pressing the delete button". In other words, participants of this study observed that, in general, shortcuts work fine in FeatureIDE, but it is a weak aspect in SPLOT.

C. *Automated Feature Model Analysis*

It is notable that features models are continually increasing in size and complexity. Therefore, it is required automated support for product configuration and verification of model consistency [6] [15] [27]. Both FeatureIDE and SPLOT offer different types of statistics as presented in Figures 8 and 9.

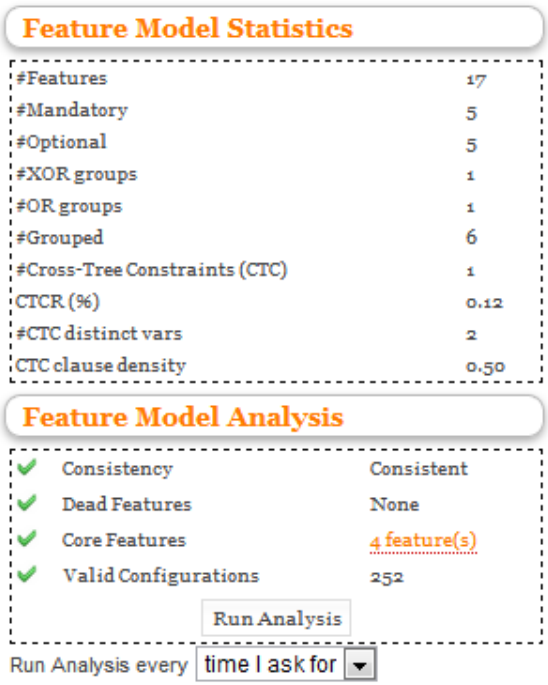

Fig. 8. Statistical analysis in SPLOT

Statistical information can be useful for software engineers who are modeling an SPL or configuring a product. For instance, the number of possible configurations is a valid indicator of the customization power of an SPL. Participants mention a great benefit that the tool brings by allowing automatic analysis of valid configurations. For instance, according to Participant T3-S05 It "allows you to view all products and validate a feature model". This participant also mentions that "we can hardly see this statistics in the middle of several lines of code".

Fig. 8 shows a SPLOT screenshot with the feature model statistics. Among other information, SPLOT shows the number of dead features, if the model is consistent, and the number of valid configurations of the analyzed feature model. SPLOT also requires the user to save the feature model on its repository and use an URL to analyze the model. Participant T3-S13 points out this fact as a drawback by saying "if user does not generate the URL for the feature model, s/he cannot use the automated analysis and product configuration options". SPLOT and FeatureIDE rely on BDD [9] and on a SAT [20] solver for statistical analysis.

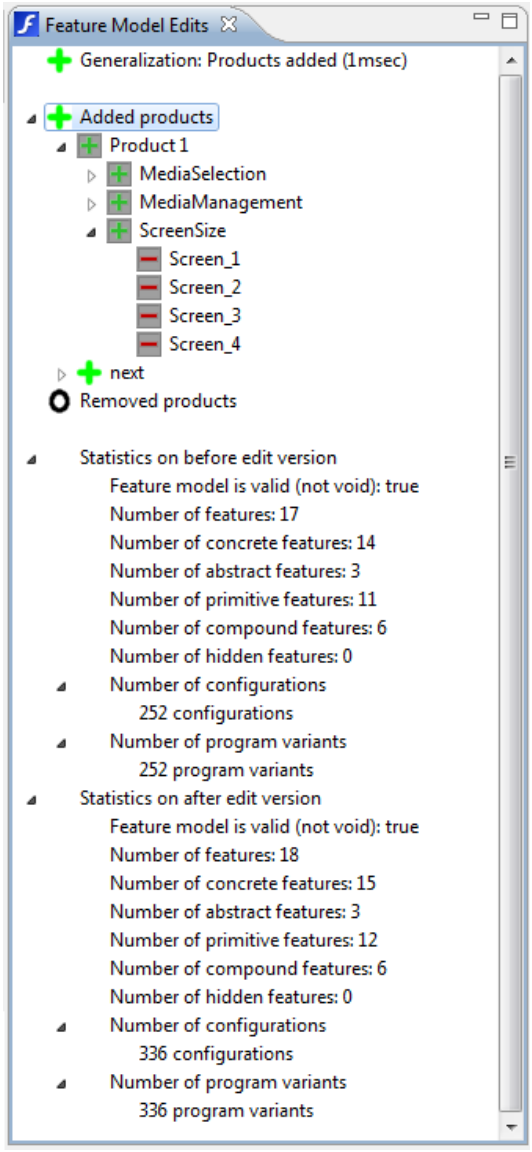

Fig. 9. Statistical analysis in FeatureIDE

Fig. 9 presents statistical information in FeatureIDE. FeatureIDE displays, for instance, (i) the number of features added and removed after changes in the product configuration, and (ii) the number of concrete, abstract, primitive, compound and hidden features. However, although both tools are mature enough, many of the participants who have used FeatureIDE reported great difficulty in finding some functions in a tool. One of the participants says "The interface FeatureIDE is not intuitive. The tool has many features, but some of them take time to learn how to use, which can be confusing for novice programmers".

D. *Product Configuration*

After creating a feature model, we asked participants to configure a product. Product configuration was performed in both tools by means of the "auto complete" functionality as shown in Figures 10 and 11, respectively. This functionality only allows instantiation of valid products. SPLOT allows a single product configuration to be created and it does not allow saving it in the repository (Fig. 10). On the other hand, FeatureIDE allows to create multiple configurations and to set the default one (Fig. 11). Although SPLOT has this disadvantage, it provides a table that contains the steps users should take to configure a product. Through this table, it is possible to have enhanced control over the feature choices. Participant T3-S04 stated that in SPLOT "the screen to setup a product is well developed with information to configure products generated on the fly".

E. *Feature Model Notation*

Two different types of features can be modeled in FeatureIDE [35]: (i) concrete (non-abstract); and (ii) abstract, but only concrete features are represented in SPLOT. The difference between both features is the first one to be mapped to at least one implementation artifact. In other words, both features can be mandatory or optional, but concrete features represent implementable

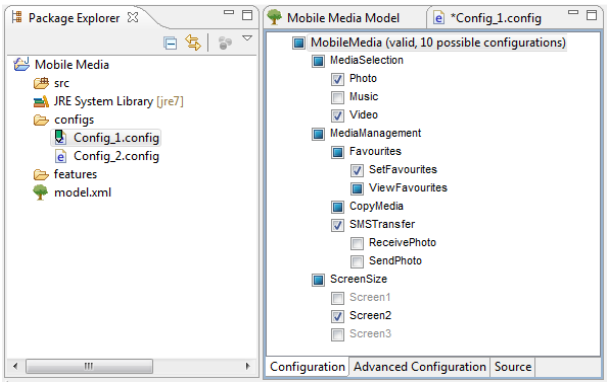

Fig.11. Product configuration in FeatureIDE

ones and abstract features are used to structure a feature model, and they do not have any impact at implementation level. The abstract features can be used (i) to represent complex functions that cannot be implemented without first be factored out or (ii) to represent features not implementable. FeatureIDE also has the concept of hidden features. The hidden features are concrete or abstract ones that for some reason were hidden, but are present in the diagram. Hidden features should not be related to cross-tree constraints, because they will not be considered in the model validation.

Dead features can be generated by a cross-tree constraint validation in FeatureIDE and SPLOT. Fig. 12 shows a dead feature in FeatureIDE. A dead feature is any mandatory, optional or alternative feature that is never part of any valid product in the SPL because a constraint in the feature model "removes" the feature from any product configuration (typically by means of constraint propagation). For example, since Video is always included in any product that constraint would force Music to be excluded from all products and therefore feature Music is dead. In a consistent feature model since the cross-tree constraint was removed, the root feature is always mandatory in both SPLOT and FeatureIDE. However, the root is an abstract feature in

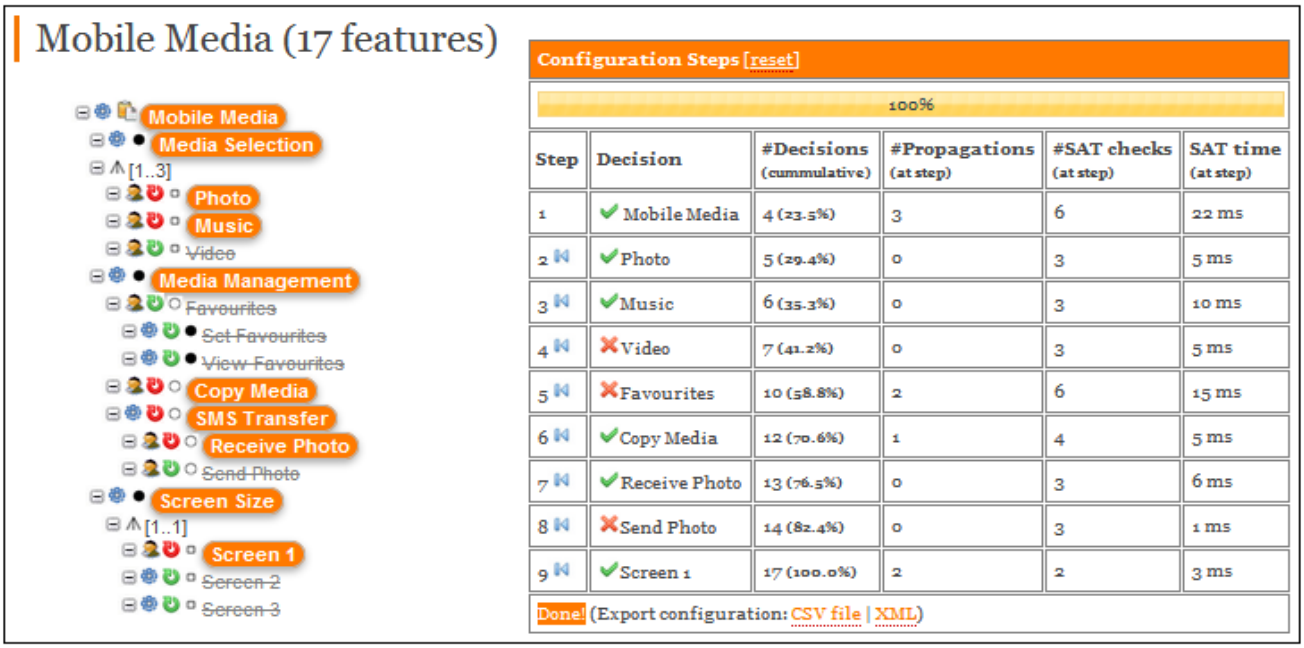

Fig. 10. Product configuration in SPLOT

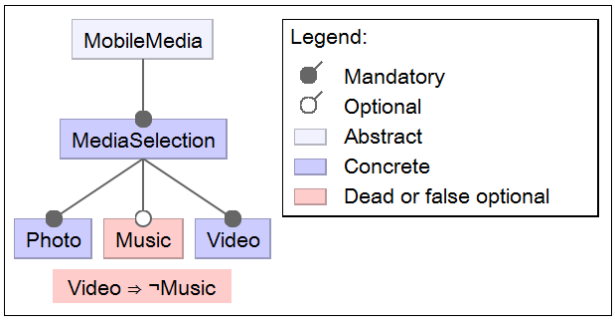

Fig. 12. Dead feature in FeatureIDE

FeatureIDE, because it is seen as an abstraction, something complex that treats the domain being modeled.

The feature model can be visualized as a directory-like tree of features in both tools. In addition to the standard visualization, FeatureIDE also allows the visualization of feature models as a graph diagram. FeatureIDE can import/export models in GUIDSL or SXFM [10] format while SPLOT relies only on SXFM as the standard file format. Both tools also allow feature modeling with the graphical editor. The model elements available and how they are depicted in SPLOT and FeatureIDE are different since they use different representations of feature diagrams. SPLOT shows the diagram in tree format and FeatureIDE has two representation ways: (i) diagram; and (ii) tree format. FeatureIDE and SPLOT also perform diagram validation/consistency analysis and show the number of (possible) valid configurations.

The graphical representation of the main elements of a feature model available in SPLOT and FeatureIDE is presented in Table III. Note that, OR and XOR decompositions are explicitly represented in SPLOT and FeatureIDE, but using different strategies. The cardinality is used to represent the OR and XOR decompositions in SPLOT, while these decompositions are represented by an arc in FeatureIDE. A similar relation to the AND decomposition is represented, when a feature have two mandatory sub-features. One may argue that cardinality should not be used when arcs have the same semantics. On the other hand, when greater precision is desired, cardinality can be useful. In fact, the cardinality increases expressiveness of the used notation [13].

In general, SPLOT and FeatureIDE can be fairly used to represent feature models since they have the main model elements required to model the domain. However, as observed by Participant T3-S4, SPLOT has a minor advantage "for allowing interactive configuration of a product using a unified and intuitive notation". SPLOT has certain insecurity "due to the public availability of the model" said T3-S4 because "anyone can change the repository content (and harm feature models of others)" pinpointed T3-S2. It also has some limitations to generate code and communicate with other tools (no import/export function). FeatureIDE by presenting some additional notations (abstract and hidden feature for example) may seem a little confused for novice users. Participants T4-S4, S5, S8, S9, S10 and S14 express large difficulties with the notation. Participant T4-S8 reports that "the terms used by this tool are sometimes very confused".

V. Related Work

The study of Turnes et al. [36] conducted a comparative analysis of variability management techniques for SPL tool development in the context of the SPL Hephaestus tool. Hephaestus is developed in Haskell and originally aimed at managing variability in requirements, but which has evolved to handle variability in different kinds of artifacts. Unlike our study, this study was particularly suitable to use in functional languages.

The work by Saratxag et al. [31] explores the lack of tool support to control the evolution and design of product families, based on an exhaustive variant analysis. In this paper, a product line tool chain is presented based on the analysis of current SPL tools and approaches. The main goal is to show the benefits of a combination of SPL tools in an industrial scenario. In

TABLE III. Modeling Elements of Feature Diagrams

Element\Tool	SPLOT	FeatureIDE
Mandatory Feature	Feature	Feature
Optional Feature	Feature	Feature
Abstract Feature	-	Feature
Dead Feature	Feature Dead Feature	Feature
Hidden Feature	-	Feature
Root (concept/ domain)	Feature	Feature
Cross-Tree Constraint (requires)	(¬FeatureA ∨ FeatureB)	FeatureA ⇒ FeatureB
Cross-Tree Constraint (excludes)	(¬FeatureA ∨ ¬FeatureB)	FeatureA ⇔ FeatureB
OR Decomposition	[1..*]	
XOR Decomposition	[1..1]	

contrast, our studies evaluate only SPL tools individually.

A related study was conducted by Simmonds et al. [33]. This study investigates the appropriateness of different approaches for modeling process variability. The aims to evaluate the supporting tools that can be incorporated within a tool chain for automating software process tailoring. For this, summarizes some of the main features of eight tools that support process variability modeling. A more recent study [32] demonstrates an industrial case study with the SPLOT tool. It shows how it is used to specify and analyze software process models that include variability. Although both studies assess the degree of usability SPLOT, they are not based on concrete experimental data.

VI. Threats to Validity

The purpose of this study exploratory is to support SPL engineers choosing the feature modeling tool that best fits their needs. A key issue when performing this kind of experiment is the validity of the results. Questions one may seek to answer include: was the study designed and performed in a sound and controlled manner? To which domain can the results generalize? In this section, threats are analyzed. We discuss the study validity with respect to the four groups of common validity threats [40]: internal validity, external validity, constructs validity, and conclusion validity.

External validity concerns the ability to generalize the results to other environments, such as to industry practices [40]. A major external validity can be the selected tools and participants. We choose two tools, among many available ones, and we cannot guarantee that our observations can be generalized to other tools. Similarly, we select data of all participants in each of the four groups evaluated were used and, therefore, we do not take into consideration the knowledge of the participants. For instance, previous experiences with SPL or with some of the tools used not were taken into consideration. Thus, we cannot generalize the results because these experiences could positively influence the results obtained. We are currently running additional rounds of the experiment, with greater control over the study, in order to increase our data set, as well as the external validity of the results.

Internal validity of the experiment concerns the question whether the effect is caused by the independent variables (e.g. course period and level of knowledge) or by other factors [40]. In this sense, a limitation of this study concerns the absence of balancing the participants in groups according to their knowledge. It can be argued that the level of knowledge of some participant may not reflect the state of practice. To minimize this threat, we provide a 1.5 hour training session to introduce participants to the basic required knowledge and a questionnaire for help the better characterize the sample as a whole. Additionally, 1.5 hour training session may not have been enough for subjects that begin without much knowledge and being exposed for the first time to these tools. However, Basili [5] and Kitchenham [25] argue that even less experienced participants can help researchers to obtain preliminary, but still important evidence that should be further investigated in later controlled experiments.

Construct validity reflects to what extent the operational measures that are studied really represent what the researcher have in mind and what is investigated according to the research questions [30]. Conclusion validity, on the other hand, concerns the relation between the treatments and the outcome of the experiment [34] [40]. These threats may have occurred in the formulation of the questionnaire or during the interpretation of the results by the researchers since our study is mainly qualitative. Due to this qualitative nature, data are not suitable for quantitative analysis and statistical tests. Questionnaire with open questions was used as a qualitative initial study to list the main functionality provided by the tools, and futurelly conduct quantitative studies and statistical tests. As far as we are concerned, this is the first experiment conducted to analyze and compare these tools. To minimize this treat, we cross-discuss all the experimental procedures. Basili [5] and Kitchenham [25] argue that qualitative studies play an important role in experimentation in software engineering.

VII. Conclusion

SPL focuses on systematic reuse based on the composition of artifacts and domain modeling. SPLOT and FeatureIDE are tools to support SPL variability management. In this paper, these tools were qualitatively analyzed and some interesting results were presented and discussed. This analysis was based on an exploratory study, in which we investigate the strengths and weaknesses of each tool grounded by surveying 56 young developers. The results reported in this paper should support software engineers to choose one of these tools for feature modeling. Additionally, this study can also be used by developers and maintainers of SPLOT and FeatureIDE - and other feature modeling tools - to improve them based on the issues reported.

After carrying out the exploratory study, we analyze the data and discuss our main results. In fact, SPLOT and FeatureIDE are two very different tools, although both tools provide support for feature modeling. SPLOT is a tool to be readily accessible to those interested only on SPL modeling and analysis (before development) and FeatureIDE is a tool focused on integration with the development process. Models created by SPLOT are automatically stored into the tool repository available to all tool users. On the other hand, FeatureIDE supports different languages for SPL implementation, such as, AspectJ [24] and AHEAD [3]. However, when choosing one of the tools the need and purpose of use is one of the main factors to be taken into consideration.

In future work, results of this study can be used and extended in controlled experiment replications. Quantitative data analysis is planned to be further performed in controlled experiments. Participants can answer additional and specific questions about SPL and feature modeling or perform tasks that exercise interesting aspects of SPL development, such integration and testing of feature code. In addition, other feature modeling tools can be analyzed and compared to SPLOT and FeatureIDE.

ACKNOWLEDGMENT

We would like to acknowledge CNPq: grants 312140/2012-6 and 485235/2011-0; and FAPEMIG: grants APQ-02376-11 and APQ-02532-12. Juliana is sponsored by FAPEMIG and Ramon is sponsored by CAPES.

REFERENCES

[1] M. Alferez, J. Santos, A. Moreira, A. Garcia, U. Kulesza, J. Araujo and V. Amaral, "Multi-view composition language for software product line requirements". In proceedings of the 2nd International Conference on Software Language Engineering (SLE), 2009, pp. 103-122.

[2] E. Bagheri, T. D. Noia, A. Ragone and D. Gasevic, "Configuring software product line feature models based on stakeholders' soft and hard requirements". In proceed. of the Internat. Soft. Prod. Line Conference (SPLC), 2010, pp. 16-30.

[3] D. S. Batory, "Feature models, grammars, and propositional formulas". In proceedings of the 9th International Conference on Software Product Lines, 2005, pp. 7-20.

[4] K. Bak, K. Czarnecki, A. Wasowski, "Feature and meta-models in clafer: mixed, specialized, and coupled". In proceedings of the 3rd International Conference on Software Language Engineering, 2010, pp. 102-122.

[5] V. Basili, F. Shull and F. Lanubile, "Building knowledge through families of experiments". Transactions on Software Engineering. 25(4), 1999, pp. 456–473.

[6] D. Benavides, P. Trinidad and A. Ruiz-Cortes. "Automated reasoning on feature models". In proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE), 2005, pp. 491-503.

[7] Y. Bontemps, P. Heymans, P. Schobbens and J. Trigaux, "Semantics of FODA feature diagrams". In proceedings of the Workshop on Software Variability Management for Product Derivation (Towards Tool Support), 2004, pp. 48-58.

[8] Q. Boucher, A. Classen, P. Faber and P. Heymans, "Introducing TVL, a text-based feature modelling language". In proceedings of the 4th International Workshop on Variability Modelling of Software-intensive Systems, 2010, pp. 159-162.

[9] R. Bryant, "Graph-based algorithms for boolean function manipulation". IEEE Transact. on Comp., 1986, pp. 677–691.

[10] W. Chae and T. Hinrichs, "SMARTFORM: a web-based feature configuration tool". In proceedings of the Fourth International Workshop on Variability Modelling of Software-intensive Systems (VAMOS), Linz, Austria, 2010.

[11] P. Clements and L. Northrop, "Software product lines: practices and patterns". Addison-Wesley Professional, 2002.

[12] K. Czarnecki and U. Eisenecker, "Generat. program. methods, tools, and applications". Addison-Wesley, 2000, 864p.

[13] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing cardinality-based feature models and their specialization". Soft. Proc.: Improv. and Pract., vol. 10, issue 1, 2005, pp. 7-29.

[14] P. Dolog and W. Nejdl, "Using UML-based feature models and UML collaboration diagrams to information modelling for web-based applications". In proceedings of the International Conference of UML, 2004, pp. 425-439.

[15] M. Eriksson, J. Borstler, K. Borg, "The PLUSS approach - domain modeling with features, use cases and use case realizations". In proceedings of the International Software Product Line Conference (SPLC), 2005, pp. 33-44.

[16] Exploratory Study Data: http://www.dcc.ufmg.br/~juliana.pereira/spl_study. 15/01/2013.

[17] E. Figueiredo, et al. "Evolving software product lines with aspects: an empirical study". In proceedings of the Int'l Conf. on Software Engineering (ICSE).2008, pp. 261-270.

[18] Franklin Machine Products: http://www.fmponline.com/. 20/06/2012.

[19] M. L. Griss, J. Favaro and M. d'Alessandro,"Integrating feature modeling with the RSEB". In proceedings of the 5th International Conference on Software Reuse, 1998, pp. 76-85.

[20] M. Janota, "Do SAT solvers make good configurators?" In proceedings of the Workshop on Analyses of Software Product Lines (ASPL 2008) at SPLC, 2008, pp. 191-195.

[21] K. Kang, K. Sajoong, L. Jaejoon, K. Kijoo, S. Euiseob and H. Moonhang, "FORM: a feature-oriented reuse method with domain-specific reference architectures". Annals of Software Engineering, 1998, pp. 143-168.

[22] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature oriented domain analysis (FODA) feasibility study". Technical Report CMU/SEI-90-TR-021. Software Engineering Institute. 155p. Accessed: August, 1990, 2012. Available: http://www.sei.cmu.edu/reports/90tr021.pdf.

[23] C. Kastner, T. Thum, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz and S. Apel, "FeatureIDE: a tool framework for feature-oriented software development". In proceedings of the Int'l Conf. on Software Engineering (ICSE), 2009, pp. 611-614.

[24] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm and W. G. Griswold, "An overview of AspectJ". In proceedings of the European Conference on Object-Oriented Programming (ECOOP), 2001, pp. 327-354.

[25] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering". Transactions on Software Engineering, 28, 8, 2002, pp. 721-734.

[26] M. Mendonça, M. Branco and D. Cowan, "SPLOT - software product lines online tools". In proceedings of the International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA), 2009, pp. 761-762.

[27] M. Mendonça, A. Wasowski and K. Czarnecki, "SAT-based analysis of feature models is easy". In proceedings of the Int'l Software Product Line Conference (SPLC), 2009, pp. 231-240.

[28] K. Pohl, G. Bockle and F. Linden, "Software product line engineering: foundations, principles and techniques". Springer, 2005, 494p.

[29] Pure.Systems: http://www.pure-systems.com/Home.142.0.html. 20/06/2012.

[30] P. Runeson and M. Host, "Guidelines for conducting and reporting case study research in software engineering". Empirical Software Eng., 2009, pp. 131–164.

[31] C. L. Saratxaga, C. Alonso-Montes, O. Haugerr, C. Ekelirr and A. Mitschke, "Product line tool-chain: variability in critical systems". Product Line Approaches in Soft. Engineering (PLEASE), 3rd International Workshop on. IEEE, 2012.

[32] J. Simmonds and M. C. Bastarrica, "Modeling variability in software process models". Computer Science Department, Universidad de Chile, Santiago, Chile. 2012.

[33] J. Simmonds, M. C. Bastarrica, L. Silvestre and A. Quispe, "Analyzing methodologies and tools for specifying variability in software processes". Computer Science Depart., Universidad de Chile, Santiago, Chile. 2011.

[34] T. Thelin, P. Runeson, C. Wohlin, T. Olsson and C. Andersson "Evaluation of usage-based reading - conclusions after three experiments". Empirical Soft. Engineering, 2004, pp. 77-110.

[35] T. Thum, C. Kastner,, S. Erdweg and N. Siegmund, "Abstract features in feature modeling". In proceedings of the Int'l Software Product Line Conference (SPLC), 2011, pp. 191-200.

[36] L. Turnes, R. Bonifacio, V. Alves and R. Lammel, "Techniques for developing a product line of product line tools: a comparative study". Software Components, Architectures and Reuse (SBCARS), Fifth Brazilian Symposium on. IEEE, 2011.

[37] J. Van Gurp, J. Bosch and M. Svahnberg, "On the notion of variability in software product lines". In proceedings of the Working IEEE/IFIP Conf. on Soft. Architecture. 2001, pp. 45.

[38] V. Vranic and J. Snirc, "Integrating feature modeling into UML". In: NODe/GSEM, 2006, pp. 3-15.

[39] XFeature: http://www.pnp-software.com/ XFeature/. 20/06/2012.

[40] Wohlin C.; Runeson P.; Hst M.; Ohlsson M. C.; Regnell B.; Wessln As. "Experimentation in software engineering", 2012, Springer.

[41] Product Line Hall of Fame: http://splc.net/fame.html. 20/06/2012.