Quantitative and Qualitative Empirical Analysis of Three Feature Modeling Tools

Juliana Alves Pereira[1(✉)], Kattiana Constantino[2], Eduardo Figueiredo[2], and Gunter Saake[1]

[1] Otto-von-Guericke-University Magdeburg (OvGU), Magdeburg, Germany
{juliana.alves-pereira,gunter.saake}@ovgu.de

[2] Federal University of Minas Gerais (UFMG), Belo Horizonte, Brazil
{kattiana,figueiredo}@dcc.ufmg.br

Abstract. During the last couple of decades, feature modeling tools have played a significant role in the improvement of software productivity and quality by assisting tasks in software product line (SPL). SPL decomposes a large-scale software system in terms of their functionalities. The goal of the decomposition is to create well-structured individual software systems that can meet different users' requirements. Thus, feature modeling tools provides means to manage the inter-dependencies among reusable common and variable functionalities, called features. There are several tools to support variability management by modeling features in SPL. The variety of tools in the current literature makes it difficult to understand what kinds of tasks are supported and how much effort can be reduced by using these tools. In this paper, we present the results of an empirical study aiming to support SPL engineers choosing the feature modeling tool that best fits their needs. This empirical study compares and analyzes three tools, namely `SPLOT`, `FeatureIDE`, and `pure::variants`. These tools are analyzed based on data from 119 participants. Each participant used one tool for typical feature modeling tasks, such as create a model, update a model, automated analysis of the model, and product configuration. Finally, analysis concerning the perceived ease of use, usefulness, effectiveness, and efficiency are presented.

Keywords: Software product lines · Variability management · Feature models · SPLOT · Featureide · Pure::variants

1 Introduction

The growing need for variability management in larger and complex software applications demands better support in benefiting from reusable software artifacts. *Software Product Line* (SPL) has proven to be an efficient software development practice by exploiting large-scale reuse and dealing with many challenges of today's software development, such as variability [26]. Experience already shows that SPL can allow companies to realize order-of-magnitude improvements in time to market, cost, productivity, quality, and flexibility [9]. Large industries, such as Hewlett-Packard, Nokia,

L.A. Maciaszek and J. Filipe (Eds.): ENASE 2016, CCIS 703, pp. 66–88, 2016.
DOI: 10.1007/978-3-319-56390-9_4

Motorola, and Dell have been investing significant effort incorporating software variability into their product line approaches [8, 28].

Variability is one of the key concepts in SPL. It allows the development of similar applications from a shared and interdependent set of software functionalities, called features [2]. Feature modeling is a way for representing variability in SPL [20]. A *feature model* provides a formal notation to represent and manage the interdependencies among reusable common and variable features. Interdependencies are employed to delimit the variability's space and to define the incompatibilities of infeasible combinations of features. The term feature model was proposed by Kang et al. [19] in 1990 as a part of the *Feature-Oriented Domain Analysis* (FODA) method. Since then, features models have been applied in a number of domains, including mobile phones [14, 16], telecom systems [17, 22], automotive industry [5, 13], template libraries [11], network protocols [3], and others.

Due the complex interdependencies among features, the adoption of SPL practices by industry depends on adequate tooling support. However, in the current literature there are several available tools to support variability management by modeling features in SPL [25]. The variety of tools makes it difficult to choose one that best meets the SPL development goals. Hence, most software development teams adopt new tools without establishing a formal evaluation. Thus, in order to contribute with relevant information to support software development teams choosing a feature modeling tool that best fits their needs, this paper presents a detailed empirical analysis of three tools, namely `SPLOT` [23], `FeatureIDE` [29], and `pure::variants` [7]. We choose to focus our analysis on these tools because they provide the key functionality of typical feature modeling tools, such as to edit (create and update) a feature model, to automatically analyze the feature model, and to configure a product from a model.

The empirical study presented in this paper involves 119 participants enrolled in *Software Engineering* courses. Each participant used only one tool: `SPLOT`, `FeatureIDE`, or `pure::variants`. We relied on a background questionnaire and a 1.5-hour training session to balance knowledge of the participants. The experimental tasks exercise different aspects of feature modeling. All participants answered a questionnaire about the functionalities they used in each tool. We focus on quantitative and qualitative analyses of four typical functionalities of feature modeling tools: *Feature Model Edition*, *Automated Feature Model Analysis*, *Product Configuration*, and *Feature Model Import & Export*. Based on this analysis, we uncover several interesting findings of the analyzed tools. For instance, we observed that `SPLOT` presented the best results for *Automated Feature Model Analysis* with twenty-five different operation of analysis mechanisms. The *Feature Model Editor* of `FeatureIDE` was considered the easiest and most intuitive one with many mechanisms available. Moreover, `FeatureIDE` also achieved the best results for the *Feature Model Import & Export* functionalities with a total of eight different possible either import or export formats. In general, the main issues we observed in the three analyzed tools are the lack of adequate mechanisms for managing the variability, such as visualization mechanisms to support the *Product Configuration* functionality.

The remainder of this paper is organized as follows. Section 2 describes the empirical study settings. Section 3 reports and analyzes the results. Section 4 points out the main issues to be addressed in the future. Section 5 discusses some threats to the study validity. In Sect. 6, some related works are discussed. Finally, Sect. 7 concludes this paper by summarizing its main contributions and directions for future work.

2 Study Settings

In this section, we present the study configuration aiming to evaluate and compare three alternative feature modeling tools, namely `SPLOT`, `FeatureIDE`, and `pure::variants`. Section 2.1 defines the study research questions. Section 2.2 introduces the three analyzed tools and explains the reasons for selecting them. Section 2.3 summarizes the background information of participants that took part in this study. Finally, Sect. 2.4 explains the training session, describes the target feature model used in the experiment, and the tasks assigned to each participant.

2.1 Research Questions

The goal of this study is to investigate how feature modeling tools are supporting variability management in SPL. We formulate three *Research Questions* (*RQ*) focusing on specific aspects of the evaluation. The answer to these questions may support researchers and practitioners, for instance, in selecting or developing new feature modeling tools. The research questions investigated in this study are as follows.

RQ1. What functionalities of feature modeling tools are hard and easy to use?
RQ2. Does the developer background impact on the use of feature modeling tools?
RQ3. What are the strengths and weaknesses of these feature modeling tools?

To address *RQ1*, we list a four-level ranking in relation to the degree of difficulty for each of the analyzed functionalities (see Sect. 3.1). With respect to *RQ2*, we are willing to investigate whether the developers background can impact on the results of this study (see Sect. 3.2). Finally, with respect to *RQ3*, we aim at highlighting the strengths of the analyzed tools and identifying weaknesses and missing mechanisms to be addressed by researchers and practitioners in the future (see Sect. 3.3).

2.2 Feature Modeling Tools

A previous systematic literature review [25] identified 41 tools for SPL development and feature modeling. Based on this review, we used the following three *Exclusion Criteria* (*EC*) in order to filter tools to be analyzed in this study.

EC1. (*Functionalities*) We excluded all tools that do not include the main functionalities required for variability management in SPL [12].
EC2. (*Prototype tools*) We excluded all prototype tools from our study because they are not applicable to industry, as they do not cover all relevant functionalities that we aim to evaluate, hindering some sorts of analysis.

EC3. (*Material available*) We excluded all tools without enough examples available, tutorials, or user guides. This criterion was required in order to prepare the experimental material and training session.

EC4. (*Unavailable tools*) We excluded all tools unavailable for download and the commercial tool without an evaluation version.

After applying the exclusion criteria (*EC1–EC3*), we filter six feature modeling tools that might be used in our empirical study: `SPLOT`, `FeatureIDE`, `pure::variants`, `FAMA`, `VariAmos`, and `Odyssey`. From the six candidate tools, we used the following *Inclusion Criteria* (*IC*) in order to choose a set of three tools and make possible to conduct a deeper study.

IC1. (*Mature tools*) We include the three most mature tools, as the maturity has a great effect on software quality and productivity (e.g., less errors are likely to be introduced during the development and consequently less effort is required to correct errors). However, in order to verify how mature a feature modeling tool is for variability management, we analyze the most cited tools in the SPL literature. For that, we identify primary studies from three scientific database libraries, namely ACM Digital Library[1], IEEE Xplore[2], and ScienceDirect[3]. *IC1* relies on the following search string: *("splot" OR "featureide" OR "pure:variants" OR "fama" OR "variamos" OR "odyssey")*. The search was performed using the specific syntax of each specific database and considering only the title, abstract, and keywords. The search strings and results of each scientific database engine are provided in the Web supplementary material [1].

We found 256 primary studies for `Pure::Variants`, 251 for `SPLOT`, 96 for `FeatureIDE`, 74 for `Odissey`, 35 for `VariAmos`, and 3 for `FAMA`. Thus, we choose `pure::variants`[4], `SPLOT`[5], and `FeatureIDE`[6] as representative tools. These tools are actively used (by industry or academic researchers), and accessible tools in order to evaluate the state-of-the-art of feature modeling tools. Next, we present a brief overview of the selected tools.

`SPLOT.` `SPLOT` (Software Product Lines Online Tools) is an open source Web-based tool. It does not provide means for code generation or integration [23]. However, at the tool website, we can find a repository with more than four hundred feature models created by tool users for over 5 years. You can download the tool's code and also a Java library (SPLAR) created by the authors to perform the analysis of feature models. It also provides a standalone tool version that can be installed on a private machine. We used the online version of `SPLOT` for this empirical study.

[1] http://dl.acm.org/.
[2] http://ieeexplore.ieee.org/.
[3] http://link.springer.com/.
[4] http://www.pure-systems.com/pure_variants.49.0.html.
[5] http://www.splot-research.org.
[6] http://featureide.cs.ovgu.de.

FeatureIDE. FeatureIDE is an open-source Eclipse-based tool which widely covers the SPL development process [29]. Besides having feature model editor and product configurator, it is integrated with several programming and composition languages with a focus on development for reuse [4, 21]. FeatureIDE can be downloaded separately or in a package with all dependencies needed for implementation.

pure::variants. pure::variants is a commercial Eclipse-based tool developed by the Pure-Systems GmbH to support the development and deployment of SPL [7]. It supports all phases of SPL development from requirements specification to test cases and maintenance. Although it is a commercial tool, there is an evaluation version available in its web site (http://www.pure-systems.com/pure_variants.49.0.html). We used the evaluation version of pure::variants in this study.

2.3 Background of the Participants

Participants involved in this study are 119 young developers taking a Software Engineering course. They were organized as follows: 41 participants worked with SPLOT, 42 participants worked with FeatureIDE, and 36 participants worked with pure::variants. All participants are graduated or close to graduate since they are mostly post-graduated MSc and Ph.D students from four different Brazilian universities: UFLA[7], UFMG[8], UFJF[9], and PUC-Rio[10]. To avoid biasing the study results, each participant only took part in one study semester and only used one tool, either FeatureIDE or SPLOT or pure::variants. The participants were nicknamed as follows: (i) F1 to F42 worked with FeatureIDE, (ii) S1 to S41 worked with SPLOT and (iii) P1 to P36 worked with pure::variants. Our goal is to use these nicknames while keeping the anonymity of the participants separating them by the tool since we did not repeat participants in the experiments. Further details about the distribution of participants are available at the project website [1].

Before starting the experiment, we used a background questionnaire to acquire previous knowledge of each participant. Figure 1 summarizes knowledge that participants claimed to have in the background questionnaire with respect to *Object-Oriented Programming* (OOP), *Unified Modeling Language* (UML), and *Work Experience* (WE). The bars show the percentage of participants who claimed to have knowledge high, medium, low, or none in OOP and UML. For WE, the options were: more than 3 years, 1 to 3 years, up to 1 year, and never worked in software development industry. Answering the questionnaire is not compulsory, but only 2 participants did not answer the questionnaire about UML knowledge and 3 participants did not answer about WE. In summary, we observe that about 75% of participants have medium to high knowledge in OOP and 48% have medium to high knowledge in UML. In addition, about 52% have

[7] Federal University of Lavras.
[8] Federal University of Minas Gerais.
[9] Federal University of Juíz de Fora.
[10] Pontifical Catholic University of Rio de Janeiro.

more than 1 year of work experience in software development. Therefore, despite heterogeneous backgrounds, we can conclude that all participants have at least the basic knowledge in the technologies required to perform the experimental tasks.

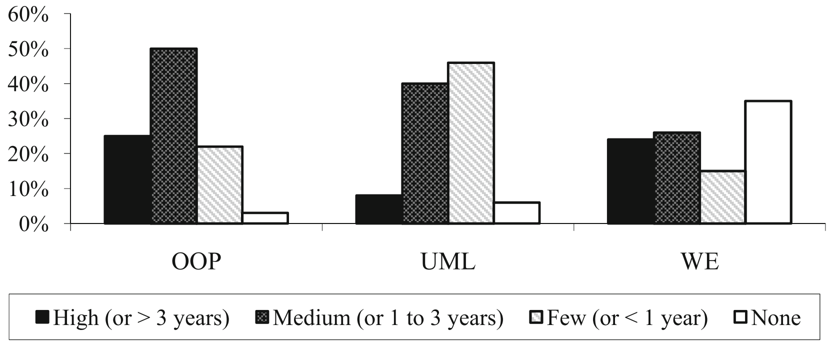

Fig. 1. Background of participants with respect to object-oriented programming (OOP), unified modeling Language (UML), and work experience (WE). Reproduced from [10].

2.4 Training Session and Tasks

In order to balance knowledge of participants, we conducted a 1.5-hour training session where we introduced participants to the basic concepts of SPL and the analyzed tools. The same training session by the same researcher to all participants was performed in all four institutions (Sect. 2.3). All material about the course was available for all participants. In addition, we have not restricted participants of accessing (e.g., via Web browsers) other information about the tools, such as tutorials and user guides.

After the training session, we asked the participants to perform some tasks using either `FeatureIDE` or `SPLOT` or `pure::variants`. These tasks were based on the target feature model of *Mobile Media* [16]. Mobile Media is an SPL for applications with about 3 KLOC that manipulate photo, music, and video on mobile devices, such as mobile phones [16]. Second Eduardo et al. [16], Mobile Media was developed for a family of 4 brands of devices, namely Nokia, Motorola, Siemens, and RIM. As an example, consider the simplified view of the Mobile Media feature model presented in Fig. 2. The features are represented by boxes, and the interdependencies between the features are represented by edges [11]. In feature models, there are common features found in all products of the product line, known as mandatory features, such as *Media Management*, and variable features that allow the distinction among products in the product line, referred to as optional and alternative features, such as *Copy Media* and the group *Screen Size*, respectively. The optional and alternative features are configurable on selected devices depending on the provided API support. Notice that a child feature can only appear in a product configuration if its parent feature does. Thus, each of the primitive features (i.e., atomic features) is a decision option related to the given parent feature, resulting in eleven decision options.

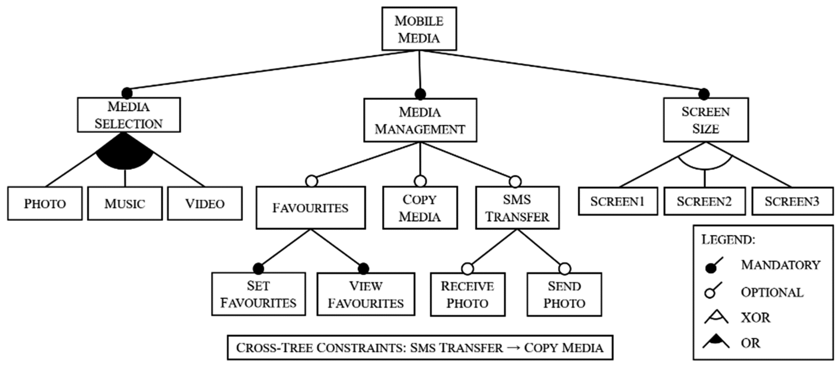

Fig. 2. A feature model for mobile media (adapted from Figueiredo et al. [16]).

In addition to features and their relationships, feature models often contain additional composition rules [11]. Composition rules refer to additional cross-tree constraints to restrict feature combinations that cannot be expressed by the feature tree. Cross-tree constraints are responsible for validating a combination of not directly connected features (i.e., they add new relations to the feature model not described in the feature tree). As an example, the cross-tree constraint "*SMS Transfer → Copy Media*" ensures that all product configurations containing the feature *SMS Transfer* must contain the feature *Copy Media*.

All tasks were based on the Mobile Media feature model to provide the same level of difficulty among the participants. We performed a four-dimension task analysis with respect to common functionalities provided by feature modeling tools as follows: *Feature Model Edition*, *Automated Feature Model Analysis*, *Product Configuration*, and *Feature Model Import & Export*. *Feature Model Edition* includes representing variability, such as creating, updating, and adding features and interdependencies in the feature model. Product' requirements are the main entry in this step. *Automated Feature Model Analysis* refers to extract information from the feature model. Based on Benavides et al. [6], we consider the following *Operations of Analysis* (OA):

OA1. (*Void Feature Model*) A feature model is void if it represents no products.
OA2. (*Valid Configuration*) A valid product configuration must not violate the feature model constraints (i.e., all features interdependencies must be considered).
OA3. (*Valid Partial Configuration*) A partial configuration requires additional features to be a complete configuration. A complete configuration has a defined selection state for each feature from the feature model.
OA4. (*Number of Configurations*) This operation returns the number of valid configurations represented by the feature model. As an example, the number of product configurations from the feature model presented in Fig. 2 is 252.
OA5. (*Dead Features Detection*) A feature is dead if it cannot appear in any of the products of the SPL. In addition, a feature is conditionally dead if it becomes dead under certain circumstances, e.g. when selecting another feature(s).

OA6. (*False Optional Features*) A feature is false optional if it is included in all the products of the product line despite not being modeled as mandatory.

OA7. (*Redundancies*) A feature model contains redundancies when the interdependencies among features are modeled in multiple ways.

OA8. (*Core Features*) This operation returns the set of features that are part of all the product configurations in the product line.

OA9. (*Variant Features*) Variant features are those that do not appear in all the products of the product line.

OA10. (*Dependency Analysis*) This operation returns all the feature dependencies from a defined partial configuration as a result of the propagation of constraints in the feature model.

In the *Product Configuration* task, a mobile phone should be configured by (de)selecting a set of features from the product line that forms a valid and concrete resultant configuration. A concrete configuration defines a set of (de)selected features from a feature model that covers as much as possible the product' requirements. Finally, the feature model should be *exported* and *imported* (e.g., using the formats XML and CSV) to a new project.

We ran seven rounds of this experiment, three of them for `SPLOT`, two for `FeatureIDE` and two for `pure::variants`. Each round of the experiment was performed in a computer laboratory with configured machines satisfying the minimum configuration required for each tool. While performing the tasks, all participants answered a questionnaire with open and closed questions. All answers are available in the project website [1].

3 Results and Discussion

This section reports and discusses data of this empirical study. Section 3.1 reports the degree of difficulty encountered by participants when performing the requested tasks. Section 3.2 focuses the discussion on whether the background of participants can impact on the use of each tool. Finally, Sect. 3.3 discusses the strengths and weaknesses of the analyzed tools.

3.1 Problems Faced by Developers

This section analyzes the level of problems that developers may have to carry out tasks in each analyzed tool. In other words, we aim to answer the following research question.

RQ1: What functionalities of feature modeling tools are hard and easy to use?

For this evaluation, we have identified interesting results extracted from the analysis of quantitative and qualitative data from the questionnaires answered by the participants after performing each task (see Sect. 2.4). The questionnaires are composed with open and closed questions. For closed questions, participants had the following options to answer (i) *I was unable to perform the task*, (ii) *I performed the task with a major* *problem*, (iii) *I performed the task with a minor problem*, and (iv) *I had no problem performing the task*. Note, in order to validate the closed questions, we look up for the opened questions to know whether the participants finished the task properly (i.e., for options (ii), (iii), or (iv)).

3.1.1 Hard and Easy Functionalities

In order to answer the research question *RQ1*, we first rely on data presented in Fig. 3. This figure summarizes the results grouped by functionality and tool. We defined a Y-axis to quantify the cumulated results, where the negative values mean *hard to use* and positive values mean *easy to use* the respective functionality.

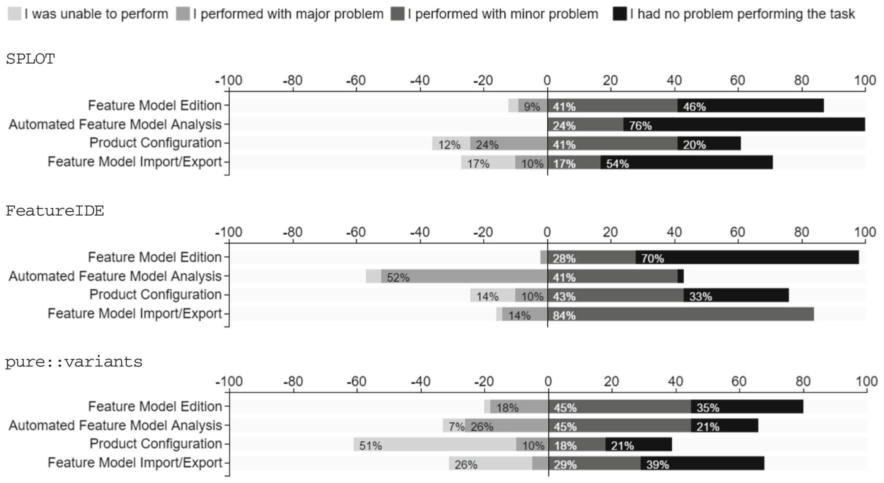

Fig. 3. Problems reported by participants to complete their tasks (reproduced from [10]).

We first investigated the SPLOT tool. On the one hand, *Product Configuration* seems the most challenge functionality to use by the SPLOT participants. About 12% of them were unable, and 24% had major problems to perform the *Product Configuration* task. On the other hand, 24% participants of SPLOT had minor problems and 76% performed without problems the *Automated Feature Model Analysis* task. These results endorse one major goal of this tool, which is to support developers with automatic operations of analysis [23], such as depth of the feature tree and number of possible configurations. Moreover, SPLOT also focuses on critical debugging tasks, such as checking the consistency of feature models, and detecting the presence of dead and common features.

Unlike SPLOT, about 57% of the participants using FeatureIDE indicated that they failed and had major problems to perform the *Automated Feature Model Analysis* task. That is, 52% of participants had major problems and 5% were unable to perform this task. Thus, this functionality was considered the hardest one to be used by participants using FeatureIDE (see Fig. 3). The most of the participants concerned about the limited support to guide them into the functionality. Regarding *Feature Model Edition*, about 28% had minor problems and 70% had no problem to perform this task.

This seems a positive result for `FeatureIDE` because only 2% (1 participant of 42) reported a major problem to edit a feature model.

Finally, we investigated the `pure::variants` tool. On the one hand, the *Product Configuration* functionality presented the worst result for this tool with a total of 61% of participants unable and with major problems to perform this task. On the other hand, the tool succeeds for the *Feature Model Edition* functionality where 80% of the participants had minor or no problems performing the task. As both `pure::variants` and `FeatureIDE` are Eclipse plug-in, this fact could be the reason why participants had minor or no problems with this task.

The general observation is that participants had more difficulties to perform the *Product Configuration* task in `pure::variants`. We believe that this task was a challenge in `pure::variants` because the tool still lacks powerful-enough solutions for managing the variability, such as the resolution of valid feature models applying decision propagation mechanisms dynamically. Next, we have identified the ranking of negative and positive functionalities for each tool.

3.1.2 Ranking of Negative Functionalities

Table 1 summarizes the rank of the three analyzed tools with respect to two negative answers "*I was unable to perform the task*" and "*I performed the task with major problem*" given by all participants. The first column relates to the feature modeling tools and the other columns relate to the functionalities analyzed. The first (1st) in Table 1 means that the respective tool presented more negative answers compared to the other tools. For instance, `pure::variants` can be considered the worst tool with respect to *Feature Model Edition* and *Product Configuration*.

Table 1. The rank of the three tools by functionalities from problems faced by developers.

Tools	Functionalities			
	Feature model edition	Automated feature model analysis	Product configuration	Feature model import & export
`SPLOT`	2nd	3rd	2nd	1st
`FeatureIDE`	3rd	1st	3rd	3rd
`pure::variants`	1st	2nd	1st	2nd

According to the `SPLOT` users, the main issues in this tool are related to its interface. For instance, participants reported they had trouble in the task of renaming features in the model. They also complained about the lack of examples. Other problems mentioned freely by its participants were that the tool does not work in some browsers. Furthermore, they mentioned that some terms such as "CTRC" and "CTC" were confused and, so, they did not understand the terms used by this tool when they were trying to configure a product.

For `FeatureIDE` participants, although they manage to edit the feature model, the tool interface still was the target of complaints. Besides, the participants also claimed concerns about the confusing terms used by the tool, such as "primitive features", "compound feature", "abstract features", and "feature hidden". Another complaint was regarding the navigation to find the related menu for the *Automatic Feature Model Analysis* and *Product Configuration*. Thus, they consider that the tool is not intuitive.

With respect to `pure::variants`, the main issues pointed out by participants were difficult to add cross-tree constraints in the feature model and many problems to perform the tasks about *Product Configuration*. Moreover, some participants also had trouble with the *Automatic Feature Model Analysis*, such as finding the activity menu for this task and the dead features. Furthermore, like in `FeatureIDE`, they claimed about terms used. Lastly, they also had interpreting problems in the results analyzed.

As a general observation, we encourage researchers and developers of feature modeling tools to unify vocabulary or notation in order to work in better way. In our study, we are convinced that the current examples available, technical report, tutorial, and users' guide are not clear enough to help the software developers using the tools and, consequently, adopting SPL. In addition, our results indicate that the developers of SPL tools need to focus more on usability and in human-computer interaction to provide the better user experience for their users.

3.1.3 Ranking of Positive Functionalities

Tables 2 and 3 summarize the ranking of the three analyzed tools considering the answers "*I performed with minor problem*" and "*I had no problem performing the task*", given by participants with strong and weak backgrounds, respectively. The first column in these tables is the feature modeling tools and the second column relates the functionalities analyzed. The first (1st) means that the respective tool presented more positive answers compared to the other tools. For instance, `SPLOT` was considered the best tool with respect to the *Automated Feature Model Analysis* functionality by participants with strong and weak backgrounds.

Table 2. The rank of the three tools by functionalities from strong background participants.

Tools	Functionalities			
	Feature model edition	Automated feature model analysis	Product configuration	Feature model import & export
`SPLOT`	3rd	1st	2nd	3rd
`FeatureIDE`	1st	3rd	1st	1st
`pure::variants`	2nd	2nd	3rd	2nd

Table 3. The rank of the three tools by functionalities from weak background participants.

Tools	Functionalities			
	Feature model edition	Automated feature model analysis	Product configuration	Feature model import & export
`SPLOT`	2nd	1st	2nd	3rd
`FeatureIDE`	1st	2nd	1st	1st
`pure::variants`	3rd	3rd	3rd	2nd

It is interesting to note that developers with weak and strong backgrounds have different viewpoints about the analyzed tools. For instance, on the one hand, `pure::variants` can be considered the worst tool for developers with weak background regards to three functionalities (i.e., *Feature Model Edition*, *Automated Analysis*, and *Product Configuration*). On the other hand, this tool is only considered the worst option by highly skilled participants for *Product Configuration.* Therefore, this result suggests that `pure::variants` is more suitable for experienced developers than for novice ones.

3.2 Background Influence

This section analyzes whether the background of developers can impact on the use of the analyzed tools. In other words, we aim to answer the following research question.

RQ2. Does developer background impact on the use of the feature modeling tools?

In order to answer *RQ2*, we first classified the participants by their level of knowledge and work experience into two groups. Group 1 (Strong Experience) includes participants that claimed to have high and medium knowledge in OOP, UML, and more than 1 year of work experience. Group 2 (Weak Experience) includes participants that answered few and no knowledge in OOP, UML, and less than 1 year of work experience. In this analysis, we excluded participants that did not answer the experience questionnaire and participants with mixed experiences. For instance, a participant with good knowledge in OPP, but less than one year of work experience.

3.2.1 Data Summary

Figure 4 shows pie charts summarizing the results. Similarly to Fig. 3, this figure depicts the percentage of participants who (i) *were unable to perform the task*, (ii) *performed the task with major problem*, (iii) *performed the task with minor problem*, and (iv) *had no problem to perform the tasks*. Charts on top indicate results for participants in the highly skilled group and charts on the bottom indicate participants with weak background. Besides, each pie chart summarizes the result of one task in one specific tool. The legend in the center of each pie is to identify the matching tool. That is, `S` means `SPLOT`, `F` means `FeatureIDE` and `P` means `pure::variants`. Each set of three pie charts relates to one of the four functionalities analyzed in this empirical study.

Based on the results of Fig. 4, we compared these two groups for each dimension. For *Feature Model Edition*, for instance, we realized that `SPLOT` (`S`) and `pure::variants` (`P`) showed some differences between these two groups. In the case of `SPLOT`, about 10% of participants with the weak background (Group 2) reported they were unable to conclude their task, while 99% of the participants with the highly skilled background (Group 1) completed their tasks. In addition, the total percentage of participants who had minor problems and had no problem did not change from Group 1 to Group 2. The reason for this result may be due to the Web interface of `SPLOT` and participants seem familiar with it. For `pure::variants`, the difference between Group 1 and Group 2 was even clearer. Approximately 92% of participants in Group 1 performed the *Feature Model Edition* task with minor or no problem. In Group 2, this percentage decreased to 60%. Therefore, we noticed the percentage of success is related to the skill level of participants in these cases. Good knowledge in OOP and UML may have contributed positively to the success of participants in this task because the task of editing a feature model involves creating an abstract representation and relationships, similarly to UML software modeling.

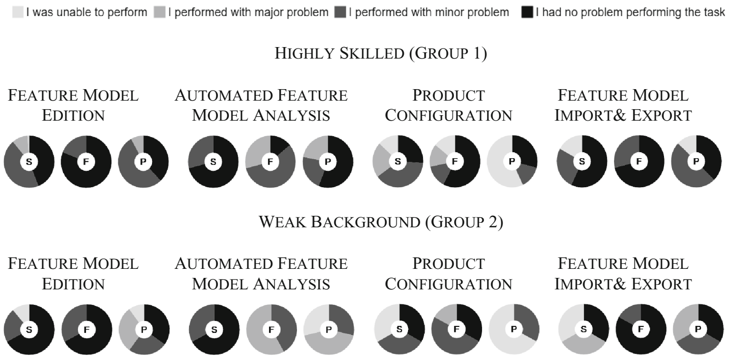

Fig. 4. Comparative results of participants with high skilled and with weak background.

For the *Automated Feature Model Analysis* functionality, the main difference between the groups occurred using the `FeatureIDE` and `pure::variants` tools. While 14% and 56% of the `FeatureIDE` and `pure::variants` participants in Group 1 had no problem performing the tasks, all participants in Group 2 failed or had some problem performing the tasks.

With respect to *Product Configuration*, while in `SPLOT` the Group 2 had 33% failures and the Group 1 only 13%, in `FeatureIDE` the Group 2 had no failures and the Group 1 had 13% failures. It shows that for `FeatureIDE` participants, the background did not influence the task performance. For `pure::variants` tool, both groups had a high percentage of failures. Although this seems a simple task, through (de)selecting features based on product requirements, `pure::variants` does not support the dynamic resolution of valid configurations. Thus, further knowledge about the feature model is also important, such as comprehension of the notations, and the relationships between features and constraints. Therefore, we realized that in this tool, this task is not trivial for either beginners or experienced SPL developers.

For the *Feature Model Import & Export* functionality, participants who used `SPLOT` presented a big difference in the results when comparing both groups, while for the other tools both groups had similar performance. For `SPLOT`, the percentage of failures increased from 17% in Group 1 to 33% in Group 2. Although the repository of the model is an interesting functionality of this tool, the participants of this study seem not familiar with it. Thus, it was difficult for participants with the weak background to perform this task in `SPLOT`. However, this task is easier for experienced software developers in Group 1.

Finally, based on the discussions described earlier, our analysis suggests that, in general, participants who have knowledge in OOP, UML, and high work experience have less trouble using the tools analyzed in this study. Therefore, as expected, the background of the participants has an impact on the use of the analyzed tools.

3.2.2 Statistical Analysis

To prove statistically the preliminary analysis, we apply a 2k full factorial design [18]. For this experiment, we have considered two factors (k = 2), namely the participants experience and the tool used. To quantify the relative impact of each factor on the participant effectiveness, we compute the percentage of variation in the measured effectiveness to each factor in isolation, as well as to the interaction of both factors. The higher the percentage of variation explained by a factor, the more important it is to the response variable [18].

In general, results show that the type of tool tends to have a higher influence on the effectiveness. Figure 5 outlines that for three out of the four functionalities (i.e., *Feature Model Edition*, *Product Configuration*, and *Feature Model Import & Export*), the type of tool used by the participants has the highest influence on the effectiveness. For the *Feature Model Edition* task, 96% of the total variation can be attributed to the type of used tool, whereas only 5% is due to participants' experience and 2% can be attributed to the interaction of these two factors. For *Product Configuration*, 57% is attributed to the type of tool, and 43% is due to participants' experience. Finally, for *Feature Model Import & Export*, 95% is attributed to the type of tool, whereas only 1% is due to participants' experience and 4% is attributed to the interaction of these two factors.

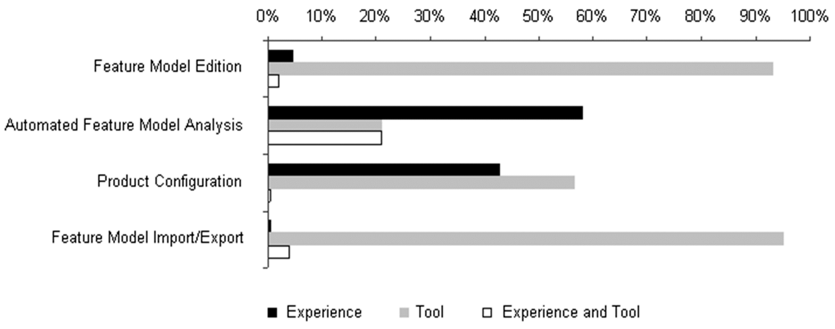

Fig. 5. Background Influence reported by factorial design test (reproduced from [10]).

Therefore, for the *Feature Model Edition* and *Feature Model Import & Export* tasks, both the participants experience factor and the interaction seem of little importance to the results. Indeed, the results clearly show that the participants who used the `SPLOT` and `FeatureIDE` tools achieved the better results for these tasks. One possible explanation is the complexity of `pure::variants`. Additionally, even participants who have no experience tend to obtain a higher effectiveness when they use `SPLOT` and `FeatureIDE` in these two tasks.

For *Automated Feature Model Analysis*, the participants experience factor was more significant. 58% of the total variation is attributed to the participants' experience factor, and whereas only 21% is due to the type of tool used and to the interaction of these two factors. Therefore, the results for this task clearly show that the participants with strong experience achieved the better results. One possible explanation is the complexity of the terms used during the analysis task, which requires more knowledge from participants.

3.3 Strengths and Weaknesses in Feature Modeling Tools

This section investigates some of the strengths and weaknesses of SPLOT, FeatureIDE, and pure::variants tools. We aim to answer the following research question.

RQ3. What are the strengths and weaknesses of the feature modeling tools?

Figures 6, 7, and 8 show diverging stacked bar chart of the strengths and weaknesses of SPLOT, FeatureIDE and pure::variants, respectively. In particular, we ask the participants about the following terms (i) tool interface, (ii) feature model editor, (iii) cross-tree constraints, (iv) automatic analysis, (v) product configuration, (vi) integration with code, (vii) hotkey mechanisms, (viii) online tool, (ix) feature model repository, (x) eclipse plug-in, and (xi) examples and user guides. The percentages of participants who considered the items as strengths are shown to the right of the zero line. The percentages who considered the items as weaknesses are shown to the left of the zero line. These items are sorted in alphabetical order in all figures. Participants could also freely express about other strengths or weaknesses they encountered during the tasks.

For SPLOT participants (see Fig. 6), the three most voted strengths were: the automatic analysis of the models (76%), the fact being an online tool (63%), and the feature model editor (41%). We believe that the automatic analysis of SPLOT was pointed out as the biggest strengths, because it presents the most basic required operations while compared with other tools. However, although 41% of participants have considered the editor as a strength of this tool, 44% of them pointed the editor as a weakness. The participants claimed mainly about the shape size. Second the participants, each feature

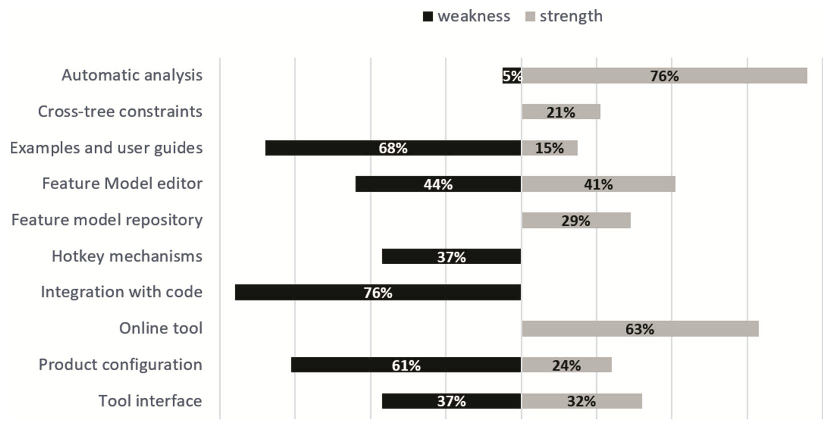

Fig. 6. Strengths and weaknesses reported by participants using SPLOT.

should be presented with sufficient size to be readable. Moreover, 68% of them pointed out the lack of examples available as a problem to understand the tool, and 76% indicated integration with source code as a missing mechanism. Lastly, the product configuration was one of the main concerns with 61% of votes. The participants claimed mainly regards the missing functionalities, such as to set multiple configurations and to save them. `SPLOT` does not allow users to create multiple configurations and keep the specified ones. In this tool, only the feature model can be exported or (and) kept in the repository.

Analyzing the `FeatureIDE` tool (see Fig. 7), the three most voted strengths were: the fact being an Eclipse plug-in (64%), and the feature model (62%) and cross-tree constraints (57%) editors. Although, the feature model editor is similar mechanisms in all tools, `FeatureIDE` editor presents many additional functionalities when compared with the other tools (e.g., zoom, filter, hotkey, and layout organization mechanisms). Moreover, when creating cross-tree constraints, it is possible to have immediate feedback regards dead features, redundant constraints, and false-optional features. As a main weakness, 64% of `FeatureIDE` users voted in the interface. In accordance with the qualitative data, the main problem is regards to the navigation to find the related menu for automatic analysis of the model and product configuration. Moreover, as in `SPLOT`, the product configuration for large feature models is challenging. For both tools, when the automatic validation is applied the immediate changes in the visual representation generate unnecessary surprises and confusion to the users. In this context, interactive mechanisms (e.g., animations, color hue, and highlighting) can be used to support users navigate in the tree, (de)select the features, and understand the interdependencies among them.

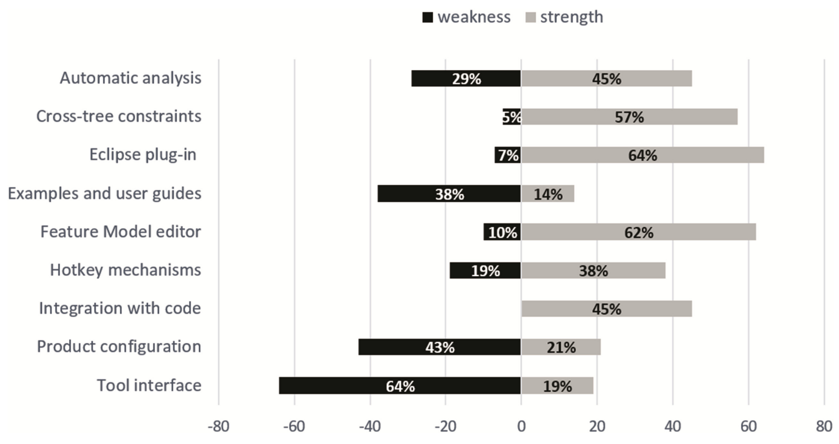

Fig. 7. Strengths and weaknesses reported by participants using `FeatureIDE`.

Finally, the `pure::variants` tool was analyzed (see Fig. 8). the three most voted strengths were: the fact being an Eclipse plug-in (78%), automatic analysis of the models (58%) and the feature model editor (56%). As weakness, 67% of its users voted in the product configuration functionality. The `pure::variants` tool configurator does not support the automatic validation of cross-tree constraints. Moreover, as in the other tools, it represents them only textually in the feature model editor screen. Thus, no cross-tree constraints visualizations are provided to the users in the configurator screen.

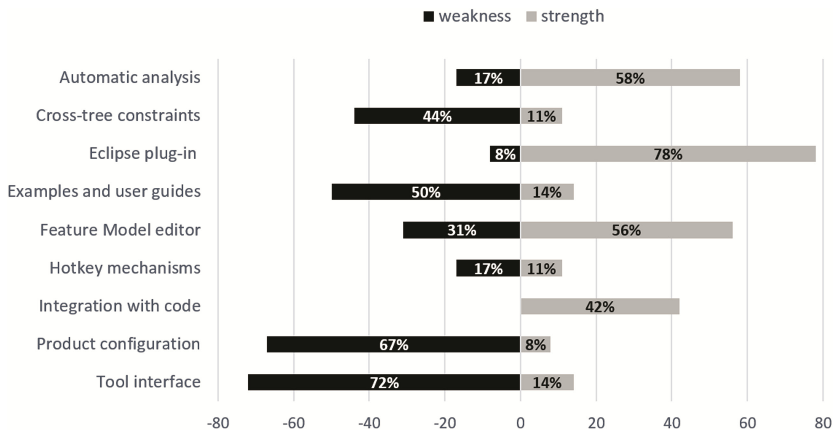

Fig. 8. Strengths and weaknesses reported by participants using `pure::variants`.

As in `pure::variants`, when considering all participants and tools, the most voted weaknesses were the tool interface (64%) and the product configuration mechanism (61%). The main drawback pointed out by participants is regards the information visualization when configuring a product. The product configuration layout in those tools results in a lot of unused screen space. Thus, the main challenge is to improve its layout taking into account a large amount of data and making use of the whole screen space while still providing a sufficient degree of usability (e.g., using multi-product lines representation). Furthermore, 50% indicated the lack of examples available and user guide. Note that, the interface and the lack of guidance may impact on negative results of relatively simple tasks, such as *Product Configuration*. That is why about 46% of participants failed to perform this task. As a result, it is recommended that SPL developers take into consideration the aspects related to user experience in order to improve the feature modeling tools.

4 Variability Management Main Issues

When analyzing the qualitative and quantitative data from the participants, the main issues we observed in the three analyzed tools are the lack adequate mechanisms for managing the variability, such as visualization mechanisms to support the product configuration task. Based on the expert knowledge from authors of this paper, we extract three main issues to be addressed in the future.

Issue 1: Current tools offer limited support for advanced visualization mechanisms (i.e., fish-eye views, filters, zooming, focus and context, cross-tree constraints, and others) making variability harder to manage.

Issue 2: When the products to be configured are highly customized, the users are usually unable to find satisfactory configurations. This happen because the amount and complexity of options presented by the configurator lead users to get lost with so much information and make poor decisions due complex and hard to reasoning dependencies. Moreover, the feature model may present many subjective features that cannot be matched with the product' requirements. In this context, none of the analyzed tools present additional information about features and variability to guide users in an easier configuration process.

Issue 3: Cross-tree constraints often create a nightmare for users because they crosscut feature models, and the resolution of valid product configuration becomes computationally complex. In `SPLOT` and `FeatureIDE`, the cross-tree constraints used to delimit the scope of allowed products are managed by SAT solvers that can automatically resolve the variability model's consistency and validity during the product configuration. Each time the user (de)selects a particular feature decision propagation strategies are applied to automatically validate feature models, which result in a non-conflicting configuration. However, such views add confusion to the users. Thus, they need additional visualization mechanisms to show which feature implied in a (de)selection of other feature(s). Moreover, the decision propagation mechanisms by themselves are not enough to support users getting a valid configuration (i.e., decision propagation can only benefit to configure partial configurations). In this case, when the user has selected all features of their choice, their configuration might still be invalid due to unsatisfied feature dependencies. Consequently, it may be very difficult to the users specifying a valid configuration since features of no interest to them also need to be (de)selected in order to fulfill the feature model's interdependencies. In this context, the analyzed tools lack appropriated mechanisms to show the users which features should be (de)selected to guide them into a valid final configuration.

In summary, the product configuration process can be challenging, as users regularly do not know every feature and their interdependencies, particularly for large product lines. Thus, in order to ease the configuration process, we believe that a successful product configuration functionality would need to be able to present the following characteristics:

- Guide the users over each step of the product configuration process through a restricted and detailed view of the configuration space and features.
- Guide the product configuration process by delivering capabilities to effectively communicate with the users and understand their needs and preferences.

5 Threats to Validity

A key issue when performing this kind of experiment is the validity of the results. The results should be valid for the population of which the set of participants were involved. It is also interesting to generalize the results to a broader population. The results have adequate validity if they are valid for the population, which they intend to be generalized. In this section, threats to the validity are analyzed. We discuss the study validity with respect to the four categories of validity threats [31]: *constructs validity*, *internal validity*, *external validity*, and *conclusion validity*.

Construct validity reflects what extent the operational measures that are studied really represent, what the researcher has in mind, and what is investigated according to the research questions [31]. The most common threats to this type of validity are related to experiment design: in general, poor definition of the theoretical basis or the definition of the testing process. For example, participants can base their behavior on the research hypotheses or they may be involved in other experiments. This type of threat can occur in formulating the questionnaire in our experiment, although we have discussed several times the experiment design. To minimize social threats, we performed the experiment in four different institutions.

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 [31]. 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 participants may not reflect the state of practice (e.g., most of the participants have only minor knowledge of SPL). 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. However, 1.5-hour training session may not have been enough for the participants with the weak background.

External validity concerns the ability to generalize the results to other environments, such as to industry practices [31]. A major external validity can be the selected tools and participants. We choose three tools, among many available ones, and we cannot guarantee that our observations can be generalized to other tools. Moreover, in Brazil there are not many SPL developers, then this group may not reflect the state of the practice. We tried to minimize this threat by working with both new and experienced developers. These participants are graduated or close to graduate since the course targets post-graduated MSc and Ph.D. students.

Conclusion validity concerns the relation between the treatments and the outcome of the experiment [31]. This involves the correct analysis of the results of the experiment, and the measurement reliability of the implementation of the treatments. Then, the conclusion of the analyzed made by us could be another if it were done by other researchers. To minimize this threat, we discuss the results data with experienced researchers to make a more reliable conclusion.

6 Related Work

This section presents some previous studies about tools for feature modeling and variability management in SPL. Djebbi et al. [15] perform an evaluate study of three SPL management tools (i.e., `XFeature`, `pure::variants`, and `RequiLine`) in collaboration with a group of industries. The purpose of this study was to understand the salient characteristics of SPL management tools and to evaluate the ability of those tools to satisfy industrial needs. In this evaluation, `pure::variants` and `RequiLine` were the tools that best satisfied the defined criteria.

Simmonds et al. [27] also investigated several tools (i.e., `Clafer`, `EPF Composer`, `FaMa-OVM`, `fmp`, `Hydra`, `SPLOT`, `VEdit`, and `XFeature`). The authors conduct an analysis based on the expressiveness of each notation for dealing with the required variability, as well as the understandability of the specification, adherence to standard formats, and the availability of tool support. Specifically, the tools were evaluated based on supported formats, underlying formalism, supported analyses, interface, availability, and usability. As in our study, the purpose of this study is to facilitate tool selection in the context of SPL.

In another study [30], ten variability modeling tools were compared (i.e., `AHEAD`, `FAMA`, `Feature Modeling Plug-in`, `Gears`, `Kumbang Tools`, `MetaEdit +`, `Product Modeler`, `Pure::Variants`, `RequiLine`, and `XFeature`). The authors categorize the comparisons into general information, technical infrastructure, operating systems support, rendering of modeling, format of input/output models support, modeling and configuration functionalities, and development functionalities. However, their results focus more on the implemented mechanisms than on the tool support, while our empirical study is based on experimental data.

In a previous preliminary work [24], we performed a preliminary and exploratory study that compares and analyzes two feature modeling tools, namely `FeatureIDE` and `SPLOT`, based on data from 56 participants that used these two tools. This empirical study involved other 84 new participants (i.e., none of the participant of this current empirical study was the same of the previous one). Therefore, this current study expanded and deepened the previous one in several ways. For instance, in addition to expanding the data set of participants, it includes one more tool, `pure::variants`, in the set of analyzed feature modeling tools. Moreover, the 84 new participants performed different tasks to exercise other aspects of SPL development. As a similarity, both studies aim to compare feature modeling tools and to support engineers in the hard task of choosing the tool that best fits their needs.

First, we extend the previous short paper with the empirical analysis of one more state-of-the-art SPL tool SPLOT. Second, we have significantly expanded the discussion of our results by analyzing the three state-of-the-art SPL tools and by presenting additional content, figures, and tables. Third, we extend our results pointing out a list of variability management issues faced by those tools to be addressed in future research. Finally, this empirical study presents a substantial extension of our preliminary short paper [10].

7 Conclusion and Future Work

SPL focuses on systematic reuse based on the composition of features and domain modeling. `SPLOT`, `FeatureIDE`, and `pure::variants` are tools used to support feature modeling in SPL. In this paper, these tools were quantitatively and qualitatively empirical analyzed and some interesting results were presented and discussed. The results reported in this paper aim to support software engineers to choose one of these tools for variability management. Additionally, this study can also be used by developers and maintainers of `SPLOT`, `FeatureIDE`, `pure::variants` - and other feature modeling tools - to improve them based on the issues reported. Besides, when choosing one of the tools, the needed and purpose of use is one of the main factors to be taken into consideration.

Our conclusions indicate that the main issues observed in the three feature modeling tools are related to the *Product Configuration* functionality. Our study does not aim to reveal "the best tool" in all functionality. On the contrary, the three analyzed tools have strength and weakness. For instance:

- `SPLOT` has as main strengths its *Automated Feature Model Analysis* functionality and the fact to be an online tool and as drawbacks, the interface and hotkeys.
- The main strength of `FeatureIDE` is the *Feature Model Editor* functionality. Its drawbacks include a limited user guide and no intuitive interface (e.g., no guide to support users finding the *Product Configuration* and *Automated Feature Model Analysis* functionalities).
- The main strengths of `pure::variants` are the *Feature Model Editor* and the *Automated Feature Model Analysis* functionalities. Its main drawbacks include the lack of examples and the *Product Configuration* functionality.

Today research on variability tools in academia and industry is attempting to solve the variability management problem. However, when hundreds of variants must be captured, visualized, and modified, the variability management still becomes challenging for companies. As future work, developers can provide a more adequate and advanced support in this context. Moreover, this study can be extended in further experiment replications. For instance, other tools can be analyzed and compared using similar experiment design in order to contribute to improving the body of knowledge about feature modeling tools. We hope that with the ongoing studies, as the one provided in this paper, feature modeling tools will become more mature and established, such that there will be more use of such tools in real practical scenarios.

Acknowledgements. This work was partially supported by CNPq (grant 202368/2014-9). We are grateful to the reviewers who contributed significantly to the improvement of the paper.

References

1. Data of the Experiment: http://homepages.dcc.ufmg.br/~kattiana/visplatform
2. Bachmann, F., Clements, P.C.: Variability in software product lines. Software Engineering Institute, CMU/SEI Report Number: CMU/SEI-2005-TR-012 (2005)
3. Barbeau, M., Bordeleau, F.: A protocol stack development tool using generative programming. In: Batory, D., Consel, C., Taha, W. (eds.) GPCE 2002. LNCS, vol. 2487, pp. 93–109. Springer, Heidelberg (2002). doi:10.1007/3-540-45821-2_6
4. Batory, D., Sarvela, J., Rauschmayer, A.: Scaling step-wise refinement. IEEE Trans. Softw. Eng. **30**(6), 355–371 (2004)
5. Benavides, D., Ruiz–Cortés, A., Trinidad, P., Segura, S.: A survey on the automated analyses of feature models. In: JISBD, Barcelona (2006)

6. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: a literature review. Inf. Syst. **35**(6), 615–636 (2010)
7. Beuche, D.: Modeling and building software product lines with pure::variants. In: International Software Product Line Conference (SPLC), p. 255 (2012)
8. Bosch, J., Capilla, R., Hilliard, R.: Trends in systems and software variability. IEEE Softw. **32**(3), 44–51 (2015)
9. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Reading (2001)
10. Constantino, K., Pereira, J.A., Padilha, J., Vasconcelos, P., Figueiredo, E.: An empirical study of two software product line tools. In: International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), pp. 164–171 (2016)
11. Czarnecki, K., Eisenecker, U.W.: Generative Programming: Principles, Techniques and Tools. Addison-Wesley, Reading (2000)
12. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. In: Software Process: Improvement and Practice, pp. 7–29 (2005)
13. Czarnecki, K., Wasowski, A.: Feature models and logics: there and back again. In: International Software Product Line Conference (SPLC), pp. 23–34 (2007)
14. Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., Wąsowski, A.: Cool features and tough decisions: a comparison of variability modeling approaches. In: Workshop on Variability Modeling of Software-intensive System (VaMoS), pp. 173–182 (2012)
15. Djebbi, O., Salinesi, C., Fanmuy, G.: Industry survey of product lines management tools: requirements, qualities and open issues. In: IEEE International Requirements Engineering Conference (RE), pp. 301–306 (2007)
16. Figueiredo, E., Cacho, N., Sant'Anna, C., Monteiro, M., Kulesza, U., Garcia, A., Soares, S., Ferrari, F., Khan, S., Filho, F.C., Dantas, F.: Evolving software product lines with aspects: an empirical study. In: International Conference on Software Engineering (ICSE), pp. 261–270 (2008)
17. Griss, M., Favaroand, J., d'Alessandro, M.: Integrating Feature Modeling with the RSEB. In: International Conference on Software Reuse (ICSR), pp. 76–85 (1998)
18. Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley, New York (1990)
19. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature oriented domain analysis (FODA) feasibility study. Software Engineering Institute, CMU/SEI Report Number: CMU/SEI-90-TR-021 (1990)
20. Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: a feature-oriented reuse method with domain-specific reference architectures. Softw. Eng. **5**(1), 143–168 (1999)
21. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: Akşit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997). doi:10.1007/BFb0053381
22. Lee, K., Kang, Kyo C., Lee, J.: Concepts and guidelines of feature modeling for product line software engineering. In: Gacek, C. (ed.) ICSR 2002. LNCS, vol. 2319, pp. 62–77. Springer, Heidelberg (2002). doi:10.1007/3-540-46020-9_5
23. Mendonça, M., Branco, M., Cowan, D.: SPLOT - software product lines online tools. In: Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA), pp. 761–762 (2009)
24. Pereira, J.A., Souza, C., Figueiredo, E., Abilio, R., Vale, G., Costa, H.A.: Software variability management: an exploratory study with two feature modeling tools. In: Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), pp. 20–29 (2013)

25. Pereira, J.A., Constantino, K., Figueiredo, E.: A systematic literature review of software product line management tools. In: Schaefer, I., Stamelos, I. (eds.) ICSR 2015. LNCS, vol. 8919, pp. 73–89. Springer, Cham (2014). doi:10.1007/978-3-319-14130-5_6
26. Pohl, K., Metzger, A.: Variability management in software product line engineering. In International Conference on Software Engineering (ICSE), pp. 1049–1050 (2006)
27. Simmons, J., Bastarrica, M.C., Silvestre, L., Quispe, A.: Analyzing methodologies and tools for specifying variability in software processes. Computer Science Department, Universidad de Chile, Santiago. http://swp.dcc.uchile.cl/TR/2011/TR_DCC-20111104-012.pdf
28. Software product line hall of fame. http://www.splc.net/fame.html. Accessed 14 May 2015
29. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: FeatureIDE: an extensible framework for feature-oriented software development. Sci. Comput. Program. **79**, 70–85 (2014)
30. Uphon, H.: A comparison of variability modeling and configuration tools for product line architecture. IT University of Copenhagen (2008)
31. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Heidelberg (2012)