An Empirical Study of Two Software Product Line Tools

Kattiana Constantino, Juliana Alves Pereira, Juliana Padilha, Priscilla Vasconcelos and Eduardo Figueiredo

Software Engineering Laboratory (LabSoft), Computer Science Department (DCC), Federal University of Minas Gerais (UFMG), Belo Horizonte, MG, Brazil

{kattiana, juliana.pereira, juliana.padilha, prisvasconcelos, figueiredo}@dcc.ufmg.br

Keywords: Software Product Lines, Variability Management, Feature Model, FeatureIDE, pure::variants.

Abstract: In the last decades, software product lines (SPL) have proven to be an efficient software development technique in industries due its capability to increase quality and productivity and decrease cost and time-to-market through extensive reuse of software artifacts. To achieve these benefits, tool support is fundamental to guide industries during the SPL development life-cycle. However, many different SPL tools are available nowadays and the adoption of the appropriate tool is a big challenge in industries. In order to support engineers choosing a tool that best fits their needs, this paper presents the results of a controlled empirical study to assess two Eclipse-based tools, namely FeatureIDE and pure::variants. This empirical study involved 84 students who used and evaluated both tools. The main weakness we observe in both tools are the lack adequate mechanisms for managing the variability, such as for product configuration. As a strength, we observe the automated analysis and the feature model editor.

1 INTRODUCTION

Software Product Line (SPL) is a set of software systems sharing a common, managed set of features that satisfies the specific needs of a particular market segment (Pohl et al., 2005). A feature represents an increment in functionality relevant to some stakeholders. It may refer to functional requirements (Jarzabek et al., 2003), architecture decisions (Bernardo et al., 2002), or design patterns (Prehofer, 2001). Feature models are used to represent the common and variable features in SPL (Czarneck and Eisenecker, 2000) (Kang et al., 1990). It provides us with an abstract, concise, and explicit representation of variability in software. Variability aims to provide support to the product derivation in an SPL (Metzger and Pohl, 2007). It refers to the ability of an artifact to be configured, customized, extended, or changed for use in a specific context.

The expected advantages in the adoption of SPL are: large-scale productivity, decreased time to market and product risk, and increased product quality (Clements et al., 2002). However, the adoption of SPL by industry depends of adequate tooling support. Existing tools for SPL support the representation and management of reusable artifacts. In fact, there are many available options of SPL tools (Pereira et al., 2015) (Simmonds et al., 2011) (Djebbi et al., 2007). These tools are diverse with different strengths and weaknesses. Therefore, choosing one tool that best meets the SPL development goals is far from trivial.

After a literature review of SPL tools (Pereira et al., 2015), this paper presents a comparative analysis of two Eclipse-based SPL tools, namely FeatureIDE (Thüm et al., 2014) and pure::variants (Beuche, 2003). We choose to focus our analysis on these tools because they are integrated to the same development environment, namely Eclipse, which makes the comparison easier. FeatureIDE and pure::variants also provide the key functionality of typical SPL tools, such as to edit (create and update) a feature model, to automatically analyze the feature model, to configure a product, and to import/export the feature model.

The empirical study of this paper (Section 2) involves 84 participants enrolled in Software Engineering courses. Each participant used only one tool: FeatureIDE or pure::variants. The experimental tasks exercise different aspects of SPL development. All participants also answered a questionnaire about the functionalities they used. We focus on quantitative and qualitative analyses of four typical functionalities of SPL tools: Feature Model Edition, Automated Feature Model Analysis, Product Configuration, and Feature Model Import/Export.

Based on the analysis (Section 3), we observed that the Feature Model Editor of FeatureIDE was considered the easiest and most intuitive one. In

ISBN: 978-989-758-189-2

comparison, pure::variants achieved the best results for the Import/Export functionalities. The overall findings are that both SPL tools have issues related to interfaces, lack of examples, tutorials, and limited user guide. Section 1 presents some threats to the study validity and Section 5 discusses related work. Finally, Section 6 concludes this paper.

2 STUDY SETTINGS

The goal of this study is to investigate how two Eclipse-based SPL tools, namely FeatureIDE and pure::variants. support SPL development and variability management.

2.1 Research Questions

We formulate three Research Questions (RQ) focusing on aspects of the evaluation are as follows.

RQ1. *What functionalities of SPL tools are hard and easy to use?* We investigated four functionalities: (i) Feature Model Edition, (ii) Automated Feature Model Analysis, (iii) Product Configuration, and (iv) Feature Model Import/Export. We list a four-level ranking for the degree of functionality difficulty.

RQ2. *Does the background of developers impact on the use of the SPL tools?* With RQ2, we are willing to investigate whether the background of developers can impact on the results of this study.

RQ3. *What are the strengths and weaknesses of different SPL tools?* In RQ3, we analyzed the tools based on the study quantitative and qualitative data.

2.2 Software Product Line Tools

A previous systematic literature review (Pereira et al., 2015) identified 41 tools for SPL development and variability management. Based on this review, we used the following exclusion criteria in order to choose the two tools for this study. First, we excluded all tools without enough examples available, tutorials, or user guides to prepare the experimental material and training session. After that, we excluded all prototype tools from our study. In addition, we excluded all tools unavailable for download and the commercial tool without an evaluation version. Therefore, we have six tools candidates for our empirical study: SPLOT, FeatureIDE, pure::variants, FAMA, VariAmos, and Odyssey. From the six candidate tools, we picked up two tools which are seamlessly integrated with the same development environment (Eclipse).

FeatureIDE (Thüm et al., 2014) and pure::variants (Beuche, 2003) are mature, actively used (by industry or academic researches) and accessible tools. These tools are also well-known and cited in the SPL literature (Bagheri and Ensan, 2014) (Simmonds et al., 2011) (Djebbi et al., 2007). Furthermore, we decided to focus only on 2 tools in order to make it possible to conduct a deeper study even if we have limitation of time and human resources. FeatureIDE is an open-source tool integrated with several programming languages and supports both aspect oriented (Kiczales et al., 1997) and feature oriented programming (Batory et al., 2004). On the other hand, pure::variants is a commercial tool with an evaluation version available. We used this evaluation version.

2.3 Background of the Participants

Participants involved in this study are 84 young developers enrolled in courses related to Software Engineering. They were organized in two replications of this study, as follows: 42 participants worked with FeatureIDE and 42 participants worked with pure::variants. The participants were nicknamed as follows: (i) F1 to F42 worked with FeatureIDE and (iii) P1 to P42 worked with pure::variants. Each participant used only one tool in the experiment, either FeatureIDE or pure::variants. All participants are graduated (M.Sc. and Ph.D students) or close to graduate.

Before the experiment, we used a background questionnaire. 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 71% of participants have medium to high knowledge in OOP and 33% have medium to high knowledge in UML. In addition, about 40% have more than 1 year of work experience in software development. Therefore, despite heterogeneous backgrounds, participants have at least the basic knowledge required to perform the experimental tasks.

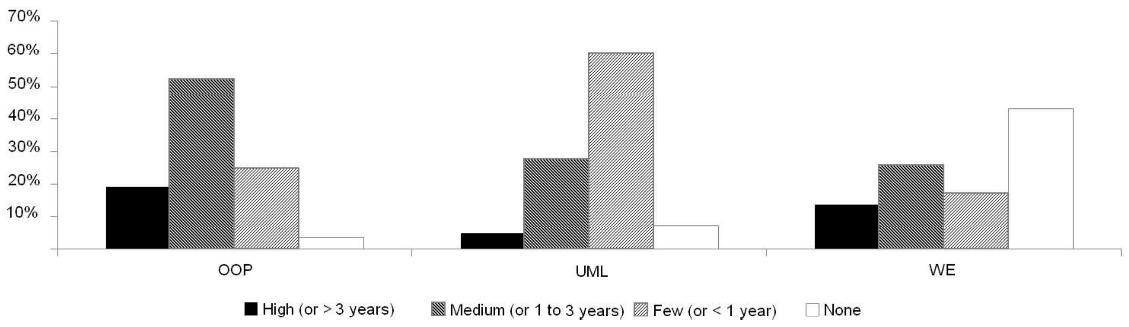

Figure 1: Background of participants in OOP, UML, and Work Experience (WE).

2.4 Target SPL Exemplar

In this empirical study, we used the same software product line exemplar, called MobileMedia (Figueiredo et al., 2008), in both replications. MobileMedia is a SPL for applications with about 3 KLOC that manipulate photo, music, and video on mobile devices, such as mobile phones. We also used the same feature model to provide the same level of difficulty in the carrying on tasks.

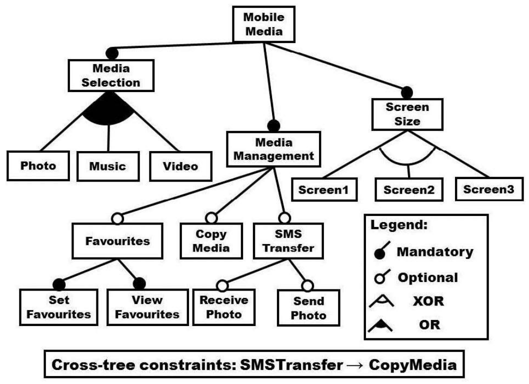

Figure 2: MobileMedia Feature Model.

Figure 2 presents a simplified view of the MobileMedia feature model. This feature model represents all possible product configurations in the MobileMedia SPL. In Figure 2, there are mandatory features, such as "Media Management", and variable features that allow the distinction between products in the SPL, such as "Copy Media" (optional) and "Screen Size" (alternative). In addition to features and their relationships, feature models often contain composition rules, known as cross-tree constraints, such as "SMSTransfer -> CopyMedia" (it means that the first feature requires the second one).

2.5 Training Session and Tasks

In order to balance knowledge of participants, we conducted 1.5-hour training session to introduce participants to the basic concepts of feature modeling, SPL, and the analyzed tools. The same training session by the same researcher to all participants. 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 guide.

After the training session, we asked the participants to perform some tasks using either FeatureIDE or pure::variants. We performed a four-functionality quantitative and qualitative analysis with respect to common functionalities provided by SPL tools as follows: Feature Model Edition, Automated Feature Model Analysis, Product Configuration, and Feature Model Import/Export. *Feature Model Edition* includes creating, updating, and adding constraints in the feature model representation. *Automated Feature Model Analysis* refers to counting the number of features, valid configurations, etc. In the *Product Configuration* task**,** a product should be specified by selecting or deselecting features. Finally**,** the feature model should be exported, as XML or CSV, and imported to a new project (Feature Model Import/Export). After performing the tasks, all participants answered a questionnaire with open and closed questions. The questionnaire is available in the project Web site (http://homepages.dcc.ufmg.br/~kattiana/spl2tools/).

3 RESULTS AND ANALYSIS

This section reports and discusses data of this empirical study.

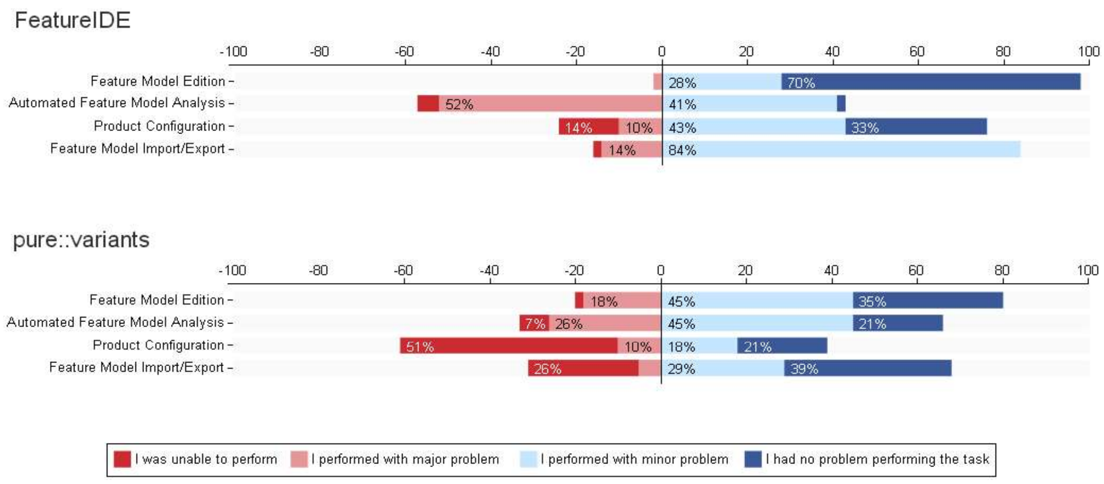

Figure 3: Percentage of problems reported by participants to complete their tasks.

3.1 Problems Faced by Developers

Our goal in this section is to analyze 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 SPL Tools are Hard and Easy to Use?*

For this evaluation, participants answered a questionnaire with the following options for each task: (i) *I was unable to perform*, (ii) *I performed with major problem*, (iii) *I performed with minor problem*, and (iv) *I had no problem*. Data presented in Figure 3 summarize 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. The general observation is that participants had minor problems or no problem to perform most tasks. Conclusions below were also supported by qualitative feedback from the study participants.

FeatureIDE. About 57% of the participants 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. Therefore, this functionality was considered the hardest one to be used by participants in FeatureIDE, as we see in Figure 3. On the other hand, with respect to Feature Model Edition, about 28% had minor problems and 70% had no problem to perform this task using FeatureIDE. It seems a positive result for this tool because only 2% (1 participant of 42) reported a major problem to edit a feature model.

pure::variants. If in the one hand, the Product Configuration functionality presented the worst result for this tool. On the other side, the tool succeeds in the other three functionalities (Feature Model Edition, Automated Feature Model Analysis, and Feature Model Import/Export). Both pure::variants and FeatureIDE are plug-in of Eclipse and this fact could be the reason why people had minor problems with these tasks. Interestingly, however, participants found it very hard to configure a product using pure::variants. That is, Figure 3 shows the negative ratio of Product Configuration in pure::variants is bigger than in FeatureIDE, meaning that the participants had more difficulties to perform this task using pure::variants.

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 the Background of Developers Impact on the Use of the SPL Tools?*

In order to answer RQ2, we apply a 2^k full factorial design (Jain et al., 2010). 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

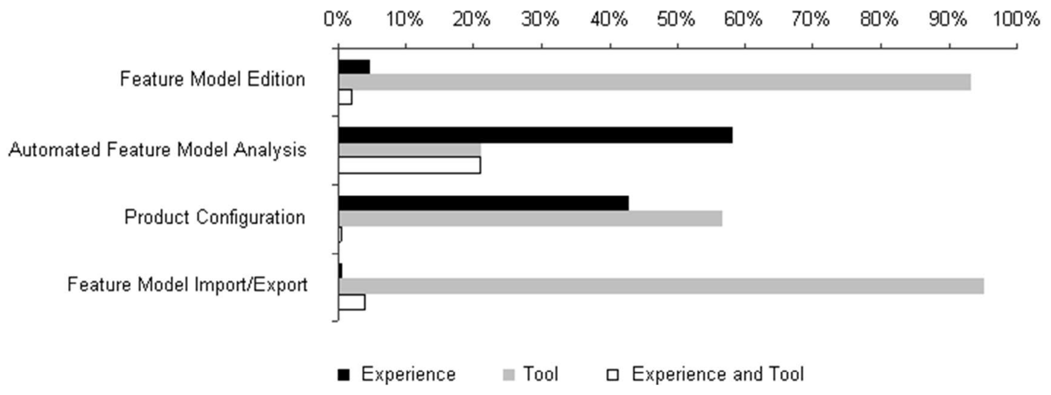

Figure 4: Background Influence by Factorial Design test.

explained by a factor, the more important it is to the response variable (Jain et al., 2010).

We classified the participants by their level of knowledge and work experience into two groups. Group 1 (Strong Experience) includes 48% of the 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 26% of the 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, was removed from this analysis.

In general, results show that the type of tool tends to have a higher influence on the effectiveness. Figure 4 outlines that for three out of the four functionalities namely, 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.

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 subjects who use the FeatureIDE tool achieved the better results for these tasks. One possible explanation is the complexity of pure::variants. Additionally, even subjects who have no experience tend to obtain a higher effectiveness when they use 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 subjects with strong experience achieved the better results. One possible explanation is the complexity of the terms used during the analysis task, which require more knowledge from subjects.

3.3 Strengths and Weaknesses

Our goal in this section is to investigate some of the strengths and weaknesses of FeatureIDE and pure::variants. In other words, we aim to answer the following research question.

RQ3. *What are the Strengths and Weaknesses of Different SPL Tools?*

Figures 5 and 6 show diverging stacked bar chart of the strengths and weaknesses of FeatureIDE and pure::variants, respectively. In particular, we ask the participants about the following items: (i) automatic organization, (ii) automatic analysis, (iii) editor, (iv) examples available, (v) hot keys, (vi) integration with other tools, (vii) interface, (viii) persistence models, (ix) product configuration, and (x) tutorials and users 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. These items are sorted in alphabetical order in both figures. Participants could also freely express about other

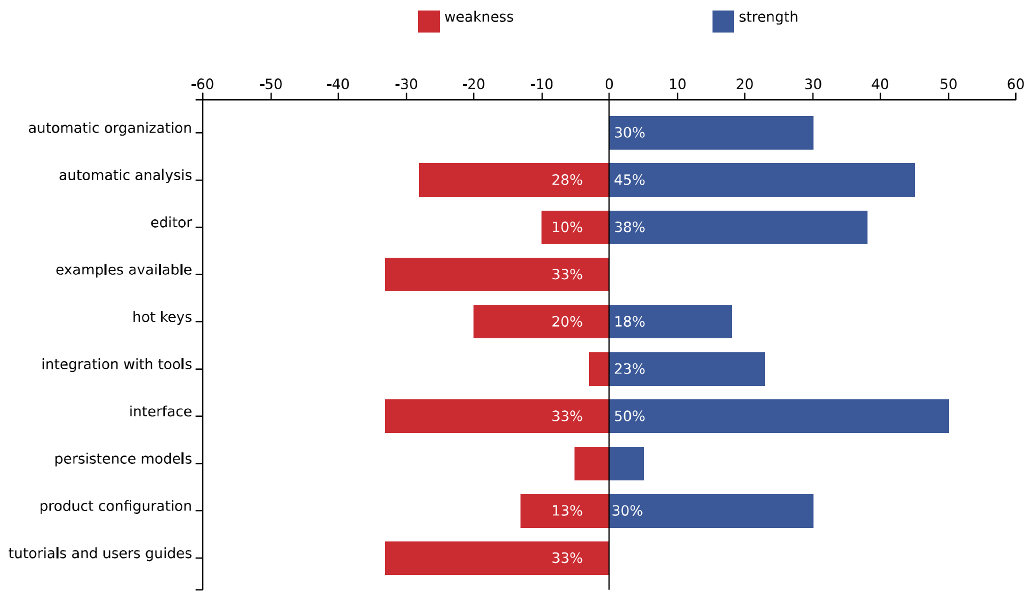

Figure 5: Strengths and weaknesses faced by participants during the tasks with FeatureIDE.

strengths or weaknesses they encountered during the tasks.

FeatureIDE. Figure 5 shows that about 33% of participants voted as weaknesses of FeatureIDE: the examples available, interface, and tutorials and user guides. Besides, 28% found the automatic analysis as hard to use in FeatureIDE. However, the main problem regards the navigation to find the related menu for automatic analysis of the model. On the other side, 50% of its participants found the interface easy and intuitive, 45% found automatic analysis of the models, and 38% editor of model as strengths of FeatureIDE. Another strength pointed freely by participants is the creating of constraints.

pure::variants. Figure 6 shows that for 49% of its participants, the interface was voted as the biggest weakness. Furthermore, about 46% and 38% of its participants pointed examples available and, tutorial and users guide as weaknesses, respectively. In opposition, for 59% of its participants the automatic analysis was considered as the biggest strength. About 59% pointed editor as strength. Automatic organization had 46% of the votes and, product configuration was other positive points (35%).

Overall Results. Considering all participants, 40% of them found the interface as the biggest weakness of both tools. Further, 39% of participants indicated the lack of examples available and 35% indicated the lack of tutorial and user guide as weaknesses of both tools. Note that, the interface may be impact on negative results of relatively simple tasks, such as Product Configuration, which the participant would select or deselect the features of a feature model based on their preferences. That is, about 14% of participants using FeatureIDE and 51% of participants using pure::variants failed to perform Product Configuration task (see Figure 3). As a result, it is recommended that SPL developers take into consideration some aspects related to user experience in order to improve the SPL tools. On the other hand, 52% of participants found the automatic analysis as the biggest strengths and, 48% indicated the editor as strengths of all two tools.

4 THREATS TO VALIDITY

A key issue when performing this kind of experiment is the validity of the results. In this section, threats to the validity are analyzed. We discuss the study validity with respect to the four categories of validity threats (Wohlin et al., 2012): construct validity, internal validity, external validity, and conclusion validity.

Construct validity 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 different institutions. With respect to *internal validity*, a limitation of this study concerns the absence of balancing the participants in groups

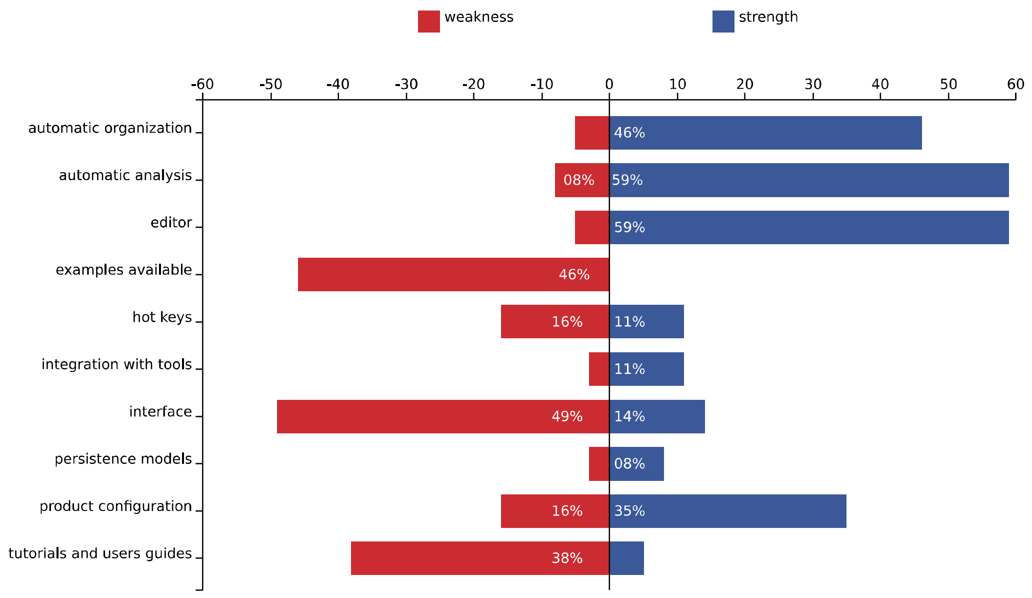

Figure 6: Strengths and weaknesses faced by participants during the tasks with pure::variants.

according to their knowledge. To minimize this threat, we provide at least 1.5 hour training session to introduce participants to the required knowledge.

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. Participants may not reflect the state of the practice developers. In addition, the results could be another if they were analyzed by other researchers (*conclusion validity*). To minimize this threat, we discuss the results data to make more reliable conclusions.

5 RELATED WORK

This section presents some previous empirical studies about SPL tools. An evaluate study of some SPL management tools (XFeature, pure::variants, and RequiLine) was performed by Djebbi et al.,(2007) in collaboration with a group of industries. In this evaluation, pure::variant and RequiLine were the tools that best satisfied the defined criteria. Simmonds et al., (2011) also investigated several tools, such as Clafer, EPF Composer, FaMa-OVM, fmp, Hydra, SPLOT, VEdit, and XFeature. The tools were evaluated based on the process they support. However, their results focus more on the techniques than on the tool support, while our empirical study is based on experimental data. In addition, our empirical study was conducted with two different tools (FeatureIDE and pure::variants).

Pereira et al., (2013) 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. Our empirical study involved other 84 new participants (no participant 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 expand the data set of participants, it includes one tool, pure::variants, in the set of analyzed SPL tools. Moreover, the 84 new participants performed different tasks to exercise other aspects of SPL development.

6 FINAL REMARKS

SPL focuses on systematic reuse based on the composition of artifacts and domain modeling. FeatureIDE and pure::variants are tools to support SPL variability management. This paper presents a quantitatively and qualitatively analysis of these tools. 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 SPL tools to improve them based on the issues reported.

Our conclusions indicate that the main issues observed in the two SPL tools are related to their interfaces, lack of examples available, tutorials, and limited user guide. On the other hand, the most mentioned strengths were automated analysis and feature model editor. Our study does not aim to reveal "the best tool" in all functionality. On the contrary, the two analyzed tools have advantages and drawbacks. As future work, this study can be extended in further experiment replications. For instance, other tools can be analyzed and compared using the same (or similar) experiment design in order to contribute to improve body of knowledge about SPL tools.

ACKNOWLEDGEMENTS

This work was partially supported by CAPES, CNPq (grant 485907/2013-5), and FAPEMIG (grant PPM-00382-14).

REFERENCES

Bagheri, E., and Ensan, F. (2014). Dynamic decision models for staged software product line configuration. *Requirements Engineering*, 19(2), 187-212.

Batory, D., Sarvela, J. N., and Rauschmayer, A. (2004). Scaling step-wise refinement. IEEE Transactions on *Software Engineering*, 30(6), 355-371.

Bernardo, M., Ciancarini, P., and Donatiello, L. (2002). Architecting families of software systems with process algebras. *ACM Transactions on Software Engineering and Methodology (TOSEM*), 11(4), 386-426.

Beuche, D. (2003). Variant management with pure::variants. *pure-systems GmbH.*

Clements, P., and Northrop, L. (2002). *Software product lines: Practices and patterns*. Addison-Wesley.

Czarnecki, K., and Eisenecker, U. (2000). Generative programming: methods, tools, and applications.

Djebbi, O., Salinesi, C., and Fanmuy, G. (2007). Industry survey of product lines management tools: Requirements, qualities and open issues. In *Int'l Requirements Engineering Conference (RE),* 301-306.

Figueiredo, E. *et al*. (2008). Evolving software product lines with aspects. In *International Conference on Software Engineering (ICSE),* 261-270.

Jain, R. *et al*. (2010). *The Art of Computer Systems Performance Analysis*. John Wiley & Sons.

Jarzabek, S., Ong, W. C., and Zhang, H. (2003). Handling variant requirements in domain modeling. *Journal of Systems and Software*, 68(3), 171-182.

Kang, K. C. *et al*. (1990), Feature-Oriented Domain Analysis (FODA) feasibility study. Carnegie-Mellon University, Software Engineering Institute.

Kiczales, G. *et al*. (1997). *Aspect-oriented programming*, 220-242. European Conf. on OO Program. (ECOOP).

Machado, L., Pereira, J., Garcia, L., and Figueiredo, E. (2014). SPLConfig: Product configuration in software product line. In *Brazilian Conference on Software (CBSoft), Tools Session*, 1-8.

Metzger, A., & Pohl, K. (2007). Variability management in software product line engineering. In *International Conference on Software Engineering (ICSE)*, 186-187.

Pereira, J. et al. (2013). Software variability management: An exploratory study with two feature modeling tools. In *Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS),* 20-29.

Pereira, J. A., Constantino, K., and Figueiredo, E. (2015). A systematic literature review of software product line management tools. In *International Conference on Software Reuse (ICSR)*, 73-89.

Pohl, K., Böckle, G., and van der Linden, F. J. (2005). *Software product line engineering: foundations, principles and techniques*. Springer.

Prehofer, C. (2001). Feature-oriented programming: A new way of object composition. *Concurrency and Computation Practice and Experience*, 13(6), 465-501.

Simmonds, J., Bastarrica, M., Silvestre, L., and Quispe, A. (2011). Analyzing methodologies and tools for specifying variability in software processes. Universidad de Chile, Santiago, Chile.

Thüm, T. et al. (2014). FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming*, 79, 70-85.

Vale, G.; Albuquerque, D.; Figueiredo, E.; and Garcia, A. (2015) Defining Metric Thresholds for Software Product Lines: A Comparative Study. In *International Software Product Line Conference (SPLC)*, pp. 176-185.

Wohlin, C. et al. (2012). *Experimentation in software engineering*. Springer.