FERRAMENTAS DE LINHA DE PRODUTOS DE SOFTWARE: VISUALIZAÇÃO DE DADOS EXPERIMENTAIS

KATTIANA FERNANDES CONSTANTINO

FERRAMENTAS DE LINHA DE PRODUTOS DE SOFTWARE: VISUALIZAÇÃO DE DADOS EXPERIMENTAIS

Dissertação apresentada ao Programa de Pós-Graduação em Departamento de Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Departamento de Ciência da Computação.

Orientador: Eduardo Magno Lages Figueiredo
Coorientadora: Raquel Cardoso de Melo Minardi

Belo Horizonte

Setembro de 2015

KATTIANA FERNANDES CONSTANTINO

SOFTWARE PRODUCT LINE TOOLS:

VISUALIZATION OF EXPERIMENTAL DATA

Dissertation presented to the Graduate Program in Department of Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Department of Computer Science.

Advisor: Eduardo Magno Lages Figueiredo
Co-Advisor: Raquel Cardoso de Melo Minardi

Belo Horizonte
September 2015

© 2015, Kattiana Fernandes Constantino.
Todos os direitos reservados.

Constantino, Kattiana Fernandes

C758s Software Product Line Tools: Visualization of Experimental Data / Kattiana Fernandes Constantino. — Belo Horizonte, 2015
xxiii, 81 f. : il. ; 29cm

Dissertação (mestrado) — Federal University of Minas Gerais
Orientador: Eduardo Magno Lages Figueiredo

1. Computação — Teses. 2. Engenharia de software. 3. Visualização de Informação. 4. Engenharia de linha de produto de software. I. Orientador. II. Título.

CDU 519.6*32(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Software product line tools: visualization of experimental data

KATTIANA FERNANDES CONSTANTINO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Eduardo M Figueiredo

PROF. EDUARDO MAGNO LAGES FIGUEIREDO - Orientador
Departamento de Ciência da Computação - UFMG

PROFA. RAQUEL CARDOSO DE MELO MINARDI - Coorientadora
Departamento de Ciência da Computação - UFMG

PROF. GLAUCO DE FIGUEIREDO CARNEIRO
NUPERC - UNIFACS

PROF. MARCO TULIO DE OLIVEIRA VALENTE
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 10 de setembro de 2015.

Dedico aos meus futuros filhos que ainda virão

Acknowledgments

Agradeço ao meu Deus pela misericórdia renovada a cada manhã. Pelo amor incondicional. Pela graça divina. Pela cruz. Pela paz, pela força, pelo dia, pela noite, pelo alimento diário, pela saúde, pela família, pelos amigos, pelos professores, pela instituição, por tudo, tudo, tudo, tudo... Obrigada Deus, a Ti toda honra e toda glória para todo sempre.

Agradeço aos meus pais, Edgar e Cícera, pelo amor incondicional, pelo incentivo total nesta fase e por cada palavra motivadora. Agradeço aos meus irmãos Kellen, Kássio e Kelbe, ao meu cunhado Moacir Júnior e a minha cunhada Daliane pelo amor, apoio e pelas palavras sempre incentivadoras. Como essas palavras motivadoras e incentivadoras são preciosas! Agradeço as minhas amadas sobrinhas, Bruna e Fernanda, pelos momentos sempre alegres e importante para meu descanso mental. Obrigada meus queridos, amo vocês todos.

Agradeço a todos os tios e primos da minha família pelo apoio dispensado, especialmente as minhas primas Amanda e Marina por terem aberto as primeiras portas desta fase em Belo Horizonte. Obrigada família.

Saudades eternas da tia Elenice. Como as suas palavras de incentivo, ao me parabenizar por ter sido aceita no mestrado, ecoaram ao longo desses dois anos. Que ótima lembrança guardarei da nossa penúltima conversa. Quão precioso foi cada incentivo, cada palavra... saudades, muitas saudades. Saudades eternas também da tia Nair (minha vó por 21 anos), tio João Lima (meu vô por adoção), tia Zilda, tia Nery e Idalina. 2014, que ano marcante e difícil. Descansem em paz com o Eterno.

Agradeço aos meus amigos de curta, média e longa jornada que estão perto ou distantes em outro estado ou país, mas que estão sempre por perto orando, incentivando, torcendo. Obrigada amigos.

Agradeço aos meus amigos da pós-graduação, colegas e funcionários do departamento, pelos diversos momentos de conversas sobre o mestrado ou além dele, pelos incentivos, pelos conselhos e atenção nos momentos de angústias, pelos momentos de alegrias, pelos momentos de trabalho. Muitos contribuíram ou com uma sugestão de tema/trabalho, ou com uma dica mais técnica, ou com alguma motivação. Como eu sempre acreditei e disse, "mestrado não se faz sozinha!". Não mesmo!!! Os amigos/-colegas de trabalho/estudo são imprescindíveis para a saudável conclusão de uma etapa tão árdua como um mestrado (ou doutorado). Preciso agradecer explicitamente ao Luís Damilton e Júlio Reis, por alguns vários momentos que vocês contribuiram diretamente no meu trabalho além, dos momentos de desabafos. Aos colegas do Laboratório de Engenharia de Software (LabSoft) que também participaram e deram sugestões preciosas. Aos queridos colegas da Laboratório Pensi pelo convívio diário, sugestões constantes, momentos de "relax". Aos demais amigos da turma... Graças a Deus, são tantos os amigos. Obrigada meus queridos, cada um foi especial! O que seria de mim sem vocês?

Agradeço imensamente ao professor Eduardo Figueiredo, meu orientador, que além de me aceitar como orientanda, dedicou-se com sugestões e incentivos preciosos para a realização e conclusão desta etapa. Muito obrigada de coração. Também, agradeço a professora Raquel Minardi pela co-orientação e sugestões oportunas. Agradeço ainda, aos membros da banca, professor Glauco Carneiro e professor Marco Túlio Valente pela atenção dispensada na leitura e sugestões para a melhoria deste trabalho. Obrigada mestres.

Enfim, meu coração está repleto de alegria pela conclusão desta etapa tão almejada e batalhada. Também, estou muito agradecida, sobretudo, a todos que contribuiram, motivaram, incentivaram com palavras positivas ao longo destes dois anos. Como elas são preciosas e importantes para um estudante.

Obrigada Deus por mais essa vitória!

"Coração de estudante
Há que se cuidar da vida
Há que se cuidar do mundo
Tomar conta da amizade
Alegria e muito sonho
Espalhados no caminho
Verdes, planta e sentimento
Folhas, coração
Juventude e fé"
(Milton Nascimento e Wagner Tiso)

Abstract

Software Product Line (SPL) is becoming widely adopted in industry due to its capability of minimizing costs and improving quality of software systems through systematic reuse of software artifacts. An SPL is a set of software systems sharing a common, managed set of features that satisfies the specific needs of a particular market segment. There are several tools to support variability management by modeling features in SPL. However, it is hard for a developer to choose the most appropriate tool due to several options available. In order to support this research, we developed the Vi**SPL**atform. It is a visual platform developed using Data Driven Documents (D3) to present and to favor the understanding of empirical data about SPL tools. We used Vi**SPL**atform in two research studies. First, we present and discuss the findings from a Systematic Literature Review (SLR) of SPL management tools. Based on the results of the SLR, we later designed and executed an empirical study. This empirical study compares and analyzes three SPL management tools, namely SPLOT, FeatureIDE, and pure::variants, based on data from 124 participants that used the analyzed tools. In this study, we performed a four-dimension quantitative and qualitative analysis with respect to common functionalities provided by SPL tools: (i) Feature Model Edition, (ii) Automated Feature Model Analysis, (iii) Product Configuration, and (iv) Feature Model Import/Export. Our aim with the Vi**SPL**atform is to explore different data types of our results and to provide visualization support to empirical data of SPL tools.

Keywords: software product line, variability management tools, SPLOT, FeatureIDE, pure::variants, data visualization.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Software Product Line (SPL) is becoming widely adopted in industry due to its capability of minimizing costs and improving quality of software systems through systematic reuse of software artifacts. A SPL is a set of software systems sharing a common, managed set of features that satisfies the specific needs of a particular market segment or customer [Pohl et al., 2005]. The advantages with the adoption of SPL approach ensure competitiveness for organizations [Clements et al., 1999]. Large companies, such as Boeing, General Motors, Hewlett-Packard, Lucent, Nokia, Motorola, Philips, Siemens, and Toshiba have adopted SPL practices [SPL Hall of Fame, 2015]. To take full advantages of SPL practices, supporting tools are required. There are a variety of SPL tools to support variability management. However, it is hard for a stakeholder to choose the most appropriate tool due to several options available.

1.1 Motivation

The number of published experiments in software engineering has increased. A substantial number of empirical studies about SPL tools has been published [Pereira et al., 2013; Simmonds et al., 2011; Šmite et al., 2010; Unphon, 2008; Sjøberg et al., 2008; Dybå and Dingsøyr, 2008; Djebbi et al., 2007]. Systematic literature review (SLR) and experiments in software engineering are a type of empirical studies that can involve too many empirical data. A SLR is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest. Individual studies contributing to a systematic review are called primary studies; a systematic review is a form of secondary study [Keele, 2007]. SLR has gotten much attention lately in Software Engineering (SE) [Kitchenham et al., 2009a]. An experiment is a formal, rigorous, and controlled investigation. The empir-

ical data can be complex, involving a high number of variables and possible crossings what could generate a high number of analyses. All these factors aforementioned confirms that knowledge discovery is not a trivial task.

In this context, visualization techniques could be combined with other data analysis techniques in order to favor the understanding of the results. Charts are illustrative and give a data set overview [Wohlin et al., 2012a]. However, chart design must take into account several principles for perception and cognition, which are the information visualization field' goals. Information visualization is defined as the use of computer-supporting, interactive, and visual representations of abstract data to amplify cognition [Card et al., 1999]. The design process of an information visualization technique consists of transforming data into images that are better comprehensible by perception and cognitive system of human beings [Spence, 2014]. The primary goal of visualization is to convey information in an understandable, effective, easy-to-remember way [Diehl, 2007].

1.2 The Proposed Approach

In order to support researchers and practitioners to choose or compare SPL management tools, we developed Vi**SPL**atform. This visual platform is composed by five different visualization techniques (or visual metaphors). This set of visualizations and analytical interaction techniques were projected to analyze the quantitative data and make possible to represent and explore empirical SPL tool data presented in this dissertation in order to make analysis and reach conclusions about the tools.

In the first visual metaphor, we use a strip plot chart to present in the time line the SPL management tools cited by literature since 2000. In addition, data analysts can interact with the visualization to explore detailed information about each tool, such as name, year, brief description, developers, country(ies) involved, and reference. The second visual metaphor consists of a bubble chart used to represent each participant of the empirical study. In this case, the data analyst can interact with the bubbles to have information about backgrounds of a specific participant. He/She also can sort the participants by gender, used tool, skill level, work experience or university that each participant is enrolled.

In the third visual metaphor, we combine heatmap and donut chart to give an overview about the impact of participant' background in the result of tasks related to four functionalities (feature model edition, automated analysis feature model, product configuration, and import/export feature model) of three different SPL tools. We have

four different answers concerning the level of difficulty in performing the tasks. With a similar propose, we create the fourth visual metaphor to detail this influence / impact. For that, we use a combination of bar and donut chart. Each bar represents different skill levels for each knowledge and work experience. Each donut represents the result of a particular task for a tool. Thereby, in this visual metaphor, the data analyst can combine different skill levels and compare the results of the tasks for each one of three tools. Finally, in the fifth visual metaphor, we use divergent stacked chart to present and explore problems faced by participants during the experiment in addition to strengths and weaknesses of each tool pointed by them.

1.3 Goal and Contributions

The main goal of this dissertation is to use the information visualization techniques to explore data from two empirical strategies and to provide analytical support to empirical SPL tools data exploration. To this end, we propose a visual platform composed by five different visualizations techniques, called Vi**SPL**atform. This set of visual representations and analytical interaction techniques focused on empirical data analysis to help data analysts to comprehend and explore data from research methods in the SPL tool context. This kind of visual platform can help designers, data analysts, project managers, researchers, and other stakeholders to understand better their projects and make decisions.

This dissertation presents three main contributions. The first is Vi**SPL**atform which aims to help providing an understanding of data from research methods in the SPL tool context. The second is a systematic literature review focusing on SPL management tools. We give an overview about the SPL management tools available in the literature during a determined period of time, in order to find out how the available tools are providing support to SPL development. Finally, the third contribution of this dissertation is an empirical study of three SPL management tools, which identifies strengths and weaknesses of each tool.

The results presented in this dissertation are parts of the following papers:

- Constantino, K. ; Figueiredo, E. ; Carneiro, G. ; Melo-Minardi, R.. "Multiple View Interactive Environment to Analyze Software Product Line Tools". In: *Brazilian Symposium on Information Systems (SBSI)*, 2016, Florianópolis/Brazil. Proceedings of Brazilian Symposium on Information Systems (SBSI). Porto Alegre/Brazil: SBC, 2016. v. 1. p. 1-8.

- Constantino, K. ; Pereira, J. ; 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)*, 2016, Rome. Proceedings of International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE). Germany: Springer, 2016. v. 1. p. 1-8.

- Pereira, J. ; Constantino, K. ; Figueiredo, E.. "A Systematic Literature Review of Software Product Line Management Tools". In: *14th International Conference on Software Reuse (ICSR)*. Lecture Notes in Computer Science. 14ed.Miami: Springer International Publishing, 2015, v. 8919, p. 73-89.

1.4 Dissertation Outline

This dissertation is organized as follows. Chapter 2 introduces background concepts about information visualization, software product line engineering (SPLE), and software product line (SPL) tools. Chapter 3 presents Vi**SPL**atform, a visual platform developed to present and to favor the understanding of empirical data in SPLE. Chapter 3 also explains the visualization techniques used within the platform in details. Chapter 4 presents a systematic literature review (SRL) focusing on SPL management tools. The SLR allows us to understand which functionalities have been investigated in past research and, thus, to identify gaps and opportunities for future research. Based on the results of the SLR, Chapter 5 presents an empirical study involving 124 participants taking courses related to the Software Engineering area. Chapter 6 discusses relevant related work. Finally, Chapter 7 concludes this dissertation and outlines future research directions.

Chapter 2

Background

The process of designing an information visualization technique consists of transforming data into images that are better comprehensible by perceptive and cognitive system of human beings [Spence, 2014]. These visualization techniques can be used in the Software Engineering (SE) context in order to reveal and to better understand data, structures, and processes [Gotel et al., 2008]. In the SE domain, Software Product Line Engineering (SPLE) is a paradigm to develop a variety of similar software applications (software-intensive systems or software products) using a platform and mass customization [Pohl et al., 2005]. The adopting of SPLE by industry depends on adequate tool support.

In this chapter, we provide the background information necessary for reading this dissertation including information visualization, software product line engineering, and three SPL tools, namely SPLOT, FeatureIDE, and pure::variants. Section 2.1 defines information visualization and shows how it can help to explore and comprehend data from empirical research in SE. Section 2.2 introduces concepts about Software Product Line Engineering. Section 2.3 presents three tools to support SPL management. Finally, concluding remarks are discussed in Section 2.4.

2.1 Information Visualization

Visualization techniques can be combined with other data analysis techniques in order to favor the understanding of semantics of the data. Charts are much more illustrative than simple text and tables giving an important first overview of the data set opening up a starting point to further investigations [Wohlin et al., 2012a]. However, chart design must take into account several principles for perception and cognition that are goals of the information visualization field.

Information visualization is defined as the use of computer-supported, interactive, visual representations of abstract data to amplify cognition [Card et al., 1999]. Each of these characteristics is detailed below [Few, 2009].

- Computer-supported: the visualization is displayed by a computer.
- Interactive: the visualization can be manipulated simple and direct manner, including actions such as filtering the data to focus on details.
- Visual representations: the information is presented in visual form using attributes such as location, length, shape, color, and size of objects to form a image of the data. Thereby, we can see patterns, trends, and exceptions that might not otherwise be visible.
- Abstract data: Information such as quantitative data, processes, or relationships is considered abstract. Because abstract information has no natural physical form, visualizations must map the data to visual characteristics, such as shapes and colors that represent the data in perceptible and meaningful ways.
- Amplify cognition: Interacting with these visualizations extends our ability to reflect about information in ways that our brains can easily comprehend.

The process of designing a visualization technique consists of encoding data into images that are better comprehensible by perceptual and cognitive system of human beings [Spence, 2014]. Figure 2.1 presents the structure of the reference model for visualization proposed by Card et al. [1999]. In this reference model, visualization can be described as the mapping of the data relations into visual form that supports human interaction in a workplace for visual sense making.

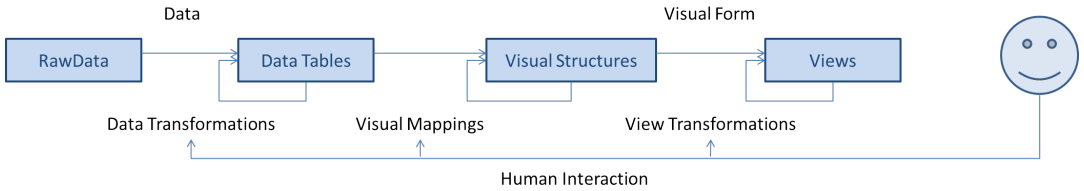


Figure 2.1: Visualization Reference Model [Card et al., 1999].

The primary goal of visualization is to convey information in an understandable, effective, and easy-to-remember way [Diehl, 2007]. The process of visual data exploration usually follows the information seeking mantra: "Overview first, zoom and filter, and then details-on-demand" [Shneiderman, 1996]. First, the data analyst needs to get an overview of the data. In this phase, the data analyst identifies interesting patterns and decides which are more promising for the next steps and which questions can be asked to go further in the desired analysis as well as the hypothesis that can be formulated. For analyzing more deeply the patterns, the data analyst needs to drill down and access more details [Keim et al., 2002]. An appropriate visualization technique can be used to support each of these three steps.

2.1.1 Perception and Cognition

According to Few [2009], visual perception works as follows:

- Our eyes perceive the reflected light from the surfaces of objects in the world.
- An object is built up in our brains as a composite of several visual properties, which are the building blocks of vision.
- Despite of human beings perceive this composite of properties as a whole object, they can still recognize the properties which compose the object.
- These individual properties include two-dimension (2-D) location, length, width, area, shape, color, and orientation, to name a few.

Moreover, there are a few facts about how we collect and process visual information [Few, 2009]. These facts can help us to create effective information visualization.

- *Fact 1*: Our visual perception is selective. Our attention is often drawn to contrasts to the something that is usual, typical, or standard. For instance, supposing a line chart with several lines of similar behavior. Suddenly, one of them has a peak. In this contrast focused our attention.

- *Fact 2*: We see what is familiar for us. Thereby, visualizations work best when they present information as patterns that are both common and easy to recognize. For instance, when we represent quantitative information in visual form, our ability to think about it is better than when we see same information in the table.

- *Fact 3*: Memory plays an important role in human cognition, but working memory is extremely limited. Thereby, information visualization must provide an external support to augment working memory. For instance, it is easier comparing two figures (or charts) when they are side by side, than when they are in different places (or on other page, or on other screen).

Software can only support information visualization effectively if the software operates on principles that respect how visual perception and cognition work.

2.1.2 Gestalt Theory

Gestalt theory was originally described in 1910 [Köhler, 1938; Wertheimer, 1938; Koffka, 1935; Ellis, 1999]. Gestalt theory tries to explain how humans arrange individual elements into groups and how humans perceive and recognize patterns. Of particular relevance are the created rules of perceptual organization, also known as Gestalt laws of grouping, which, by explaining how humans perceive a well-organized pattern, can easily translate into brief design principles.

Three of the Gestalt laws - similarity, proximity, and common fate are particularly important. The law of similarity asserts that elements which are similar, either in terms of color, shape, or size. They are perceived to be more associated than elements that are dissimilar. The law of proximity is related to elements which are close together are perceived as being more associated than elements that are farther apart. Finally, the law of common fate declares that elements that move simultaneously in the same direction and at the same speed are perceived as being more related than elements that are static or that move in different directions [Lima, 2011].

2.1.3 Overcoming the Limits of Memory

Memory is divided into two fundamental type: long-term memory and short-term memory. Long-term memory is where information is stored permanently. By contrast, short-term memory store information for a brief period of time. The short-term memory (or working memory) can be divided into different sets of memory storage for verbal

information and visual information. When you work with an information visualization, you use the visual working memory. However, this memory is very limited. Research has found that visual working memory can only handle about three chunks of information at a time [Few, 2009]. A chunk can be a mental representation of an object, or a plan, or a group of objects, or a method for achieving some goal, or anything. [Ware, 2012].

For instance, most of the human beings might only be able to hold three to five numbers [Cowan, 2012]. If this same information is presented in a line chart, however, each line could be stored as a single chunk in visual memory. Therefore, this is one of the great advantages of visualization for exploring and analyzing data. When quantitative values are displayed as visual images, more information is chunked together in these images, thus we can deal with more information simultaneously than if we were relying only a tables of numbers. This greatly multiplies the number and complexity of insights that can appear [Few, 2009].

2.1.4 Choosing the Visualization Techniques

Different taxonomies and classification models have been proposed to describe data visualization techniques and possible data analyst interactions [Roth and Mattis, 1990; Zhou and Feiner, 1996; Morse et al., 2000; Fujishiro et al., 2000; Valiati et al., 2006; Nazemi et al., 2011]. Nonetheless, different models focus on distinct aspects according to the goal or context in which the visualization is being used. According to Barros [2015] which studied and proposed a taxonomy aimed to unify the main existing taxonomies, most of the data visualization techniques could be characterized according to three main dimensions: *Data*, *Analytical Tasks*, and *Visual Attributes*. Each dimension can be then subdivided into others. For instance, one of the sub-dimensions of *Data* is the *Data Type*, that classifies the data as *Quantitative* or *Qualitative* [Roth and Mattis, 1990]. Knowing the type of data and the capabilities of visualization techniques is essential do encode data visual in an appropriate way.

The *Analytical Tasks* dimension, in its turn, describes actions that can be performed by data analysts when interacting with an interactive data visualization as part of an analytical process. For example, *User Interaction* and *Data Analysis* [Fujishiro et al., 2000]. *User Interaction* tasks describe a set of high-level tasks performed by data analysts during their interaction with a data visualization system (overview, zoom, filter, details-on-demand, relate, history and extract) [Shneiderman, 1996]. For instance, *Filter* is the act of reducing the set of data. In other words, we select the information which are important at moment from view. By contrast, we can restore the data

previously selected. Both of them are task of filtering. *Data Analysis* describes a set of low-level tasks performed by data analysts during the data analysis. Knowing the possible analytical interaction techniques, with what visualization techniques they can be implemented and the type of questions they can answer is also very important to design appropriate interactive visualizations.

Finally, the *Visual Attributes* dimension represents a set of visual and normally pre-attentive features used as primitives in the construction of data visualization techniques. One of its sub-dimension is the *Retinal Variable* which are the visual variables related to depth perception defined by the experimental psychology (Size, Texture, Color Orientation, Form and Value) [Nazemi et al., 2011; Bertin, 1973]. This are pre-attentive attributes and knowing how to use them according to they are perceived by human beings is of great importance in visualization design. For instance, not every visual attribute can be precisely used to encode quantities. Color intensity for example can be used to encode quantitative values but are not as precise as the size of an object. That is why bar charts are so popular.

2.1.5 Visual Support Tool

The following functionalities are recommended for a tool in supporting data analysis [Few, 2015]:

- Effective charts. Analysts need a limited set of useful and well designed charts.
- Effective data interactions. Analysts need a limited set of useful, well designed, and efficient interaction methods (filtering, sorting, annotating, etc.).
- Useful statistics. Proven statistical methods should be available and well elaborated to produce reliable results.
- Effective human-computer interfaces. The interface should be easy for analysts to efficiently interact with data without distraction.

The power of a visualization implies in the possibility to have a far more complex concept structure represented by a visual display than can be held in visual and verbal working memories. People with cognitive tools are far more effective thinkers than people without cognitive tools and computer-based tools with visual interfaces may be the most powerful and flexible cognitive systems. Combining a computer-based information system with flexible human cognitive capabilities, such as pattern finding, and using a visualization as the interface between the two is far more powerful than an without human cognitive process [Ware, 2005].

In this work we join the well-known and accepted principles about human perception and cognition together with knowledge about data types and the appropriate visualization techniques to each of them playing with visual attributes and finally exploring how analytical interaction techniques could help us to depict, analyze and find conclusions about our data from empirical studies.

2.2 Software Product Line Engineering

The growing need for developing larger and more complex software systems demands better support for reusable software artifacts [Kang et al., 1998; Mendonça et al., 2009]. When artifacts are reused in several kinds of systems, this reuse implies a cost reduction for each system. In order to address these demands, Software Product Line Engineering (SPLE) has been increasingly adopted in software industry [SPL Hall of Fame, 2015]. SPLE is a paradigm to develop a variety of similar software applications (software-intensive systems or software products) using a platform and mass customization [Pohl et al., 2005].

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 or a system property relevant to some stakeholders [Batory, 2005b; Kang et al., 1990]. It may refer to functional requirements [Jarzabek et al., 2003], architecture decisions [Bernardo et al., 2002], or design patterns [Prehofer, 2001]. An important concept of an SPL is the feature model. Feature models are used to represent the common and variable features in SPL [Czarnecki and Eisenecker, 2000; Kang et al., 1990].

In practice, developing an SPL involves modeling features that represent different viewpoints, sub-systems, or concerns of the software system [Batory, 2005b; Beuche et al., 2004]. Figure 2.2 depicts an example of feature model. In feature models, nodes represent features and edges show relationships between parent and child features [Batory, 2005b; Czarnecki and Wasowski, 2007]. These relationships define how the features can be combined. There are common features found in all products of the product line (known as mandatory features) and variable features that allow distinguishing between products in a product line. Variable features define points of variation and their role is to permit the instantiation of different products by enabling or disabling specific SPL functionality. Generally, these variable features are represented by optional (“OR”) or alternative (“XOR”) features.

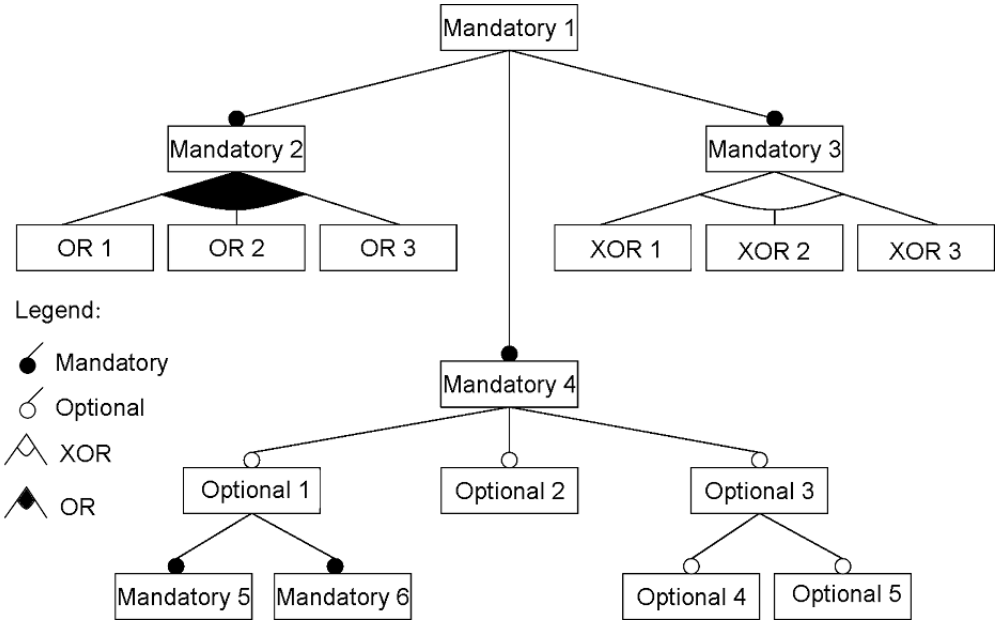


Figure 2.2: Example of a feature model.

The expected advantages in the adoption of SPL are: large-scale productivity, decreased time to market and product risk, ability to effect mass customization, increased product quality, more efficient use of human resources, ability to maintain market presence, and increased customer confidence and satisfaction. These advantages ensure competitiveness for organizations [Clements et al., 1999]. Experiences already show that SPL can allow companies to realize order-of-magnitude these improvements [Clements and Northrop, 2001]. Large companies, such as Boeing, General Motors, Hewlett-Packard, Lucent, Nokia, Motorola, Philips, Siemens, and Toshiba have adopted SPL practices [SPL Hall of Fame, 2015].

The adoption of SPL by industry depends of adequate tool support. SPL tools are mainly used to support variability management. Variability management is the ability to change or customize a software product in an SPL [Unphon, 2008]. These tools support the representation and management of reusable artifacts instead of providing means for conventional development from scratch. However, there are many available options of SPL tools [Djebbi et al., 2007; Pereira et al., 2015; Simmonds et al., 2011; Unphon, 2008]. In addition, these tools are diverse with different strengths and weaknesses. The following section presents three heterogeneous SPL tools selected for our empirical study (Chapter 5).

2.3 Software Product Line Tools

In this section, we present a brief overview of main functionalities of three heterogeneous SPL tools, relevant for this dissertation, namely SPLOT, FeatureIDE, and pure::variants. The criteria and reasons to select these three tools are presented in Chapter 5. Section 2.3.1 introduces SPLOT, a Web-based reasoning and configuration system for SPLs. Section 2.3.2 presents FeatureIDE, an extensible framework for feature-oriented software development. Finally, Section 2.3.3 outlines pure::variants, a tool for variant management that supports developers of SPLs throughout the entire product lifecycle.

2.3.1 SPLOT

Software Product Lines Online Tools (SPLOT)[1] [Mendonça et al., 2009] is a Web-based tool for creating and sharing feature models and product configuration. It is a free and open source project. Figure 2.3 shows the view for creating, editing and analyzing a feature model in the SPLOT tool. First, this figure shows a directory-like tree format of feature model provided by SPLOT. Some information about the feature model is essential to better document, identify and, eventually, change the model. In addition, the figure also presents some statistics about the feature model, such as, the amount of features, feature mandatory, optional, alternative, constraints, and others. SPLOT also checks the consistency and validates the feature model.

Furthermore, SPLOT supports the notion of feature-based interactive configuration in which developers make configuration decisions selecting or deselecting a feature of the model. For each decision, the configuration engine automatically checks the consistency. SPLOT uses a SAT Solver to support interactive configuration operations, such as configuring, resetting, undoing, toggling, and auto-completing. SPLOT does not provide means for code generation or integration with source. On the other hand, at the tool website, we can find a repository with hundreds of feature models created by the tool stakeholders since 2009. For that, the model must (i) be consistent, (ii) have at least 10 features, (iii) not have dead feature and (iv) have the information required by fields in the table.

[1] `http://www.splot-research.org`

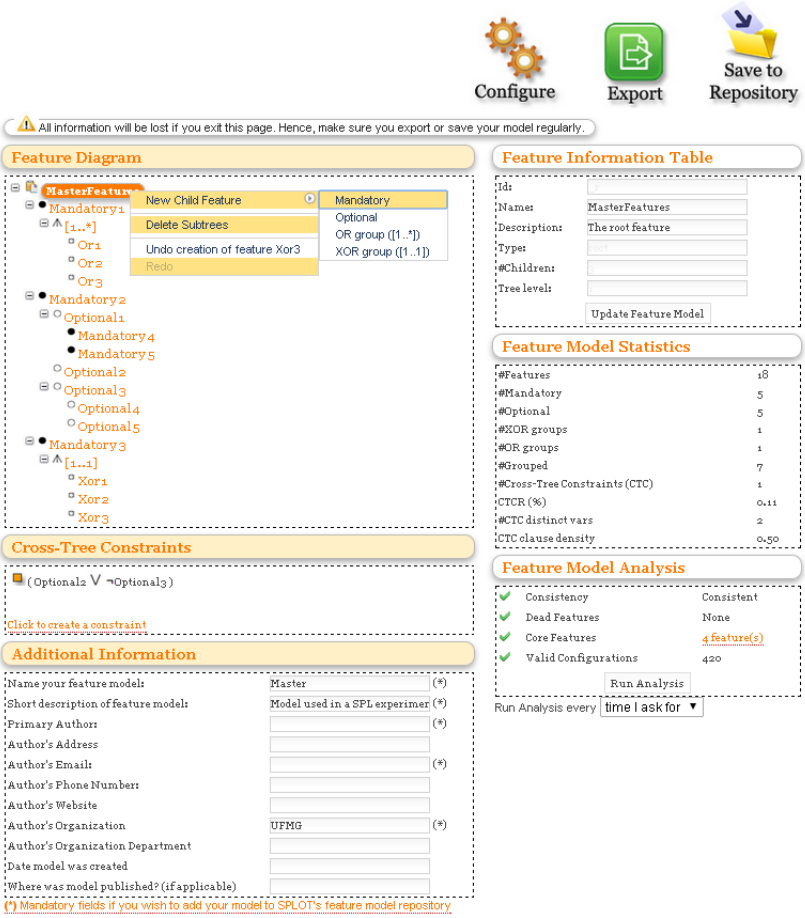


Figure 2.3: SPLOT interface view for creating and analyzing a feature model

2.3.2 FeatureIDE

FeatureIDE[2] [Thüm et al., 2014] is an open source framework of an IDE for software product line engineering based on Feature-Oriented Software Development (FOSD). It supports all phases of feature-oriented software development for the SPL development, starting with domain analysis and feature modeling [Czarnecki and Eisenecker, 2000], to covering requirements analysis, implementation, and code generation. Besides having feature model editors and configuration of products, it is integrated with several programming and composition languages with a focus on development for reuse.

FeatureIDE was developed to support both aspect-oriented [Kiczales et al., 1997] and feature-oriented programming (FOP) [Batory et al., 2004; Prehofer, 1997] in order to provide traceability between a feature model and the code base. This tool is implemented as an Eclipse plug-in and can be downloaded separately or in a package

[2] `http://featureide.cs.ovgu.de`

with all dependencies needed for implementation. Figure 2.4 presents a feature model in FeatureIDE. This feature model editor allows different views of the feature model and the creation of cross-tree constraints [Thüm et al., 2014].

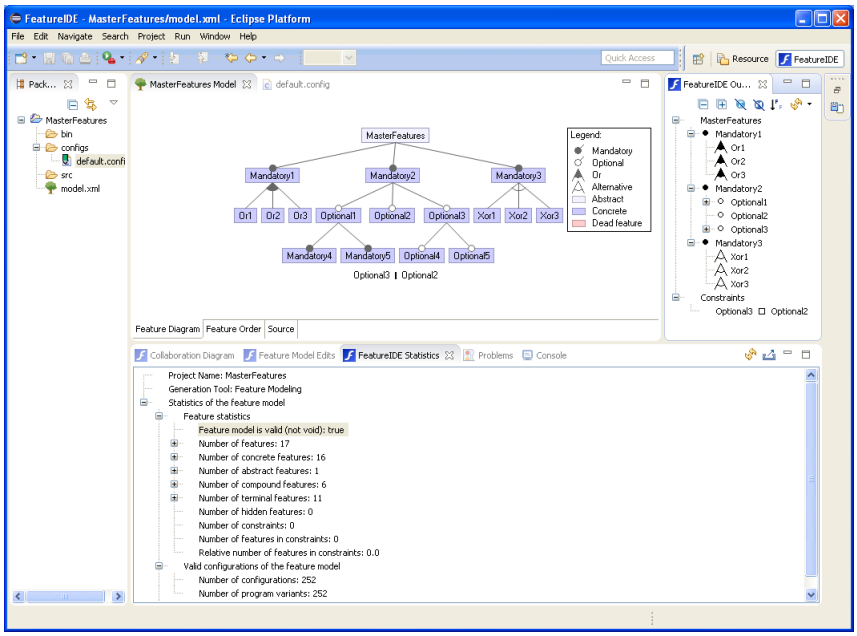


Figure 2.4: Feature model in FeatureIDE tool

For product configuration functionality, FeatureIDE receives as input the feature model and provides configuration options. Developers can select or deselect the features in the model and, automatically, a new product is configured and saved in a configuration file. The product configuration editor checks the new product configured to indicate whether it is a valid setting. FeatureIDE also provides export and import functionality in order to integrates with other feature modeling tools, for example, GUIDSL [Batory, 2005a], SPLOT, and others. The feature model created with FeatureIDE is stored in an XML format. Furthermore, the feature models can also be stored in several graphic formats or printed to a PDF file [Thüm et al., 2014].

2.3.3 pure::variants

pure::variants[3] [Beuche, 2012] is a tool for variability management. It is developed by the pure-Systems GmbH, a company specialized in the development of reuse-based software. This tool supports SPL developers throughout the entire product lifecycle [Beuche, 2003] since requirements specification and test cases until the maintenance phase. Its user interface is based on Eclipse plug-in.

[3] `http://www.pure-systems.com/pure_variants.49.0.html`

The pure::variants architecture follows the Feature-Oriented Domain Analysis (FODA) notation [Kang et al., 1998]. It has three different models: feature model, family model, and variant model. Feature model shows common features and variability of the product line. Family model describes the solution family in terms of software architectural elements/components. Finally, variant model defines an instance of the product line [Beuche, 2003]. Figure 2.5 presents requirements expressed in the form of feature models, which depict the individual products in the SPL. The design solution of the feature model is performed and included in the family model which can be automatically processed by pure::variants. In Figure 2.5, each element with the icon "F" is a feature. This icon refers to its variation type, which describes dependencies between features. Four different variation types exist in pure::variants feature models: mandatory feature (represented by "!"), optional features (represented by "?"), alternative feature "xor" (represented by "<->"), and feature "or" (represented by "X").

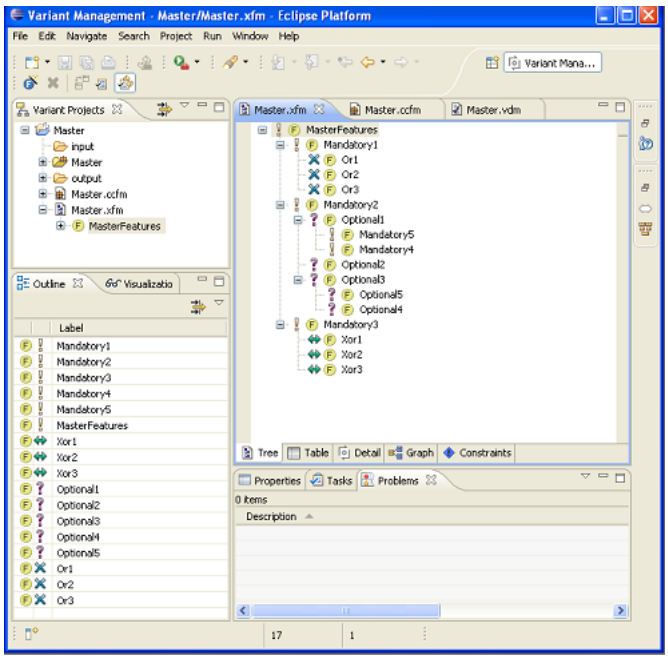


Figure 2.5: pure::variants interface

To derive variants, developers need to select or deselect the feature which should be or not be included in a variant, respectively. In addition, pure::variants can be integrated with other tools (such as, IBM Rational DOORS, IBM Rational Rhapsody or Sparx Systems Enterprise Architect) for requirements engineering, designing the software architecture, implementation, or test management.

2.4 Concluding Remarks

In this chapter, we introduced the primary goal of visualization that is to convey information in an understandable, effective, and easy-to-remember way [Diehl, 2007]. Thereby, information visualization is defined as the use of computer-support, interactive, visual representations of abstract data to amplify cognition [Card et al., 1999]. In addition, we defined that Software Product Line Engineering as a paradigm to develop a variety of similar software applications (software-intensive systems or software products) using platform and mass customization [Pohl et al., 2005]. Finally, we presented a brief overview of main functionalities of three SPL tools, relevant for this dissertation, namely SPLOT, FeatureIDE, and pure::variants. The next chapter presents the Web-based visual platform, called Vi**SPL**atform that is composed by visualization techniques to support and to analyze data from two software engineering research methods discussed in Chapters 4 and 5.

Chapter 3

The Visualization Platform

This chapter presents a Web-based visual platform, called Vi**SPL**atform[1]. It is a tool developed to analyze the quantitative data and make possible to explore and to find visual patterns, trends and anomalies in empirical data related to SPL tools. The visual platform supports several analyzes of empirical SPL data presented in this dissertation. The rest of this chapter depicts five visualizations presented in Vi**SPL**atform. It is organized as follows. Section 3.1 shows the architecture of Vi**SPL**atform and technologies used. Section 3.2 presents the first visual technique which uses circles to represent quantitative data in a time series. Section 3.3 depicts the visual technique which uses bubbles to represent the participants and their background. Section 3.4 shows the third one which is a heatmap linked to a donut chart to illustrate the influence of a specific skill level in task execution success. Section 3.5 presents a visualization that combines different skills to evidence the impact of the combination in task execution. This visualization is a combination of bar and donut charts. Section 3.6 presents a diverging stacked bar chart that illustrates some problems faced by participants and the strengths and weaknesses of each tool pointed out by participants. Finally, Section 3.7 concludes and presents final remarks.

[1] `http://homepages.dcc.ufmg.br/~kattiana/visplatform/`

3.1 ViSPLatform Architecture

This section presents the architecture of Vi**SPL**atform and technologies used in its implementation. Figure 3.1 presents the packages view of Vi**SPL**atform. The APP module manages visual models for one or more data sets, separating visual attributes (location, size, color, etc) from the abstract data. One or more views provide a graphical display of the visualization, while control modules process user interaction in the Vi**SPL**atform.

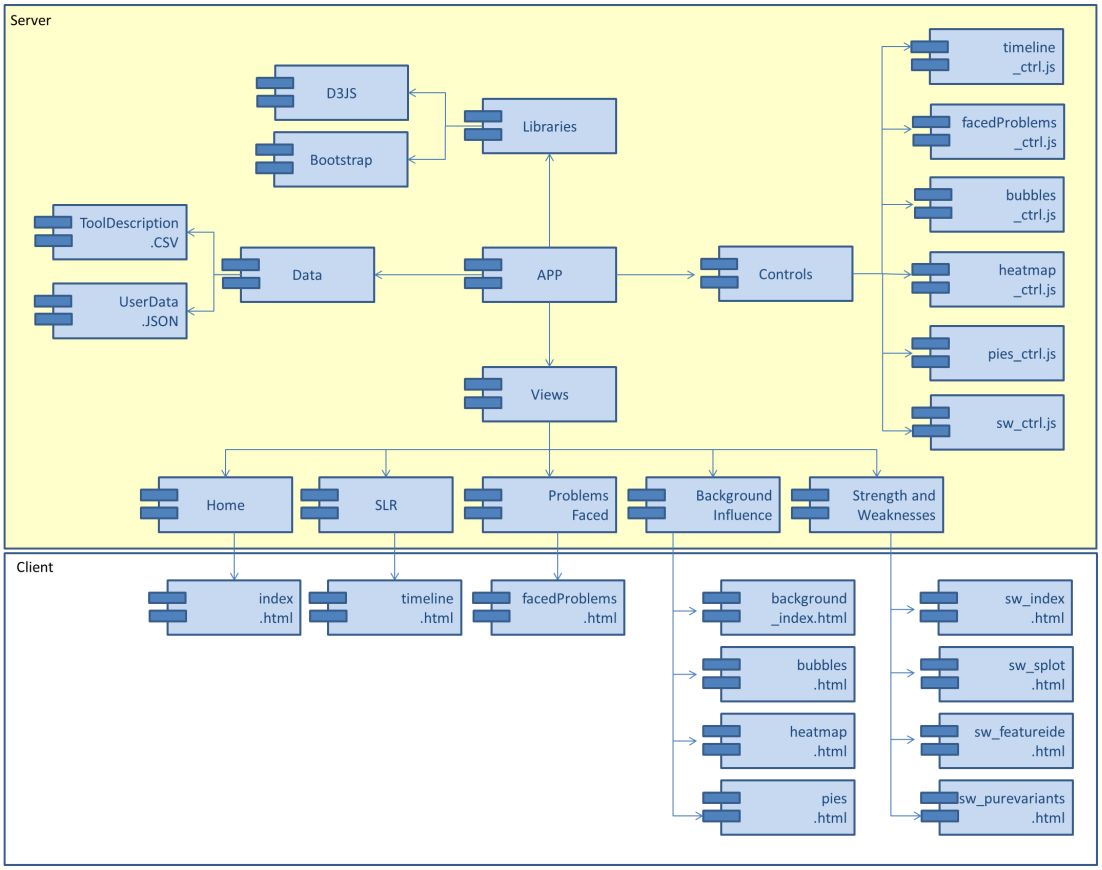


Figure 3.1: Packages view of Vi**SPL**atform

We have designed and implemented a dynamic, analytical data visualization tool using *Data-Driven Documents*[2] (D3) which is a JavaScript[3] library for manipulating documents based on data. This library is a freely available extension of JavaScript and the basic idea behind D3 is to provide a way to join data with elements on a web page and then manipulate the elements based on that data. D3 library enables the developer to directly set the attributes of graphical elements in Scalable Vector Graphic (SVG),

[2] `http://d3js.org/`
[3] `http://www.w3schools.com/js/`

which is a XML-based vector image format for two-dimensional graphics with support for interactivity and animation, according to data. Moreover, we used Cascading Style Sheets[4] (CSS) that is a style sheet language used for describing the presentation of a document written in a markup language. Finally, we built our Web-based visual platform using some components provided by the Bootstrap Framework[5], which comes several stylesheets and jQuery plugins for establishing interactive web sites or application user interfaces. All these technologies were employed for realizing a dynamic exploration and visualization experience.

3.2 Strip Plots

Figure 3.2 presents the visualization schema of Strip Plots. Our goal with this visualization is to show how many and what are the tools proposed during a determined time period. Strip plots are charts that depict the quantitative relationship distribution. In our implementation the X axis represents the time and each point is a particular publication or a SPL tool. Notice that, all points are drawn in the same size and color as they represent objects of the same type. The vertical position of the points has no meaning.

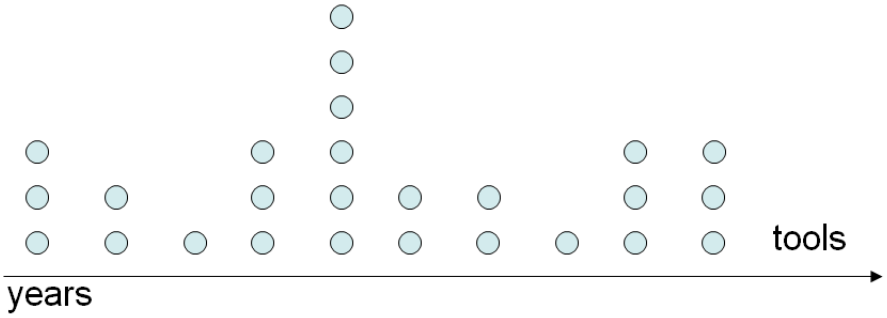


Figure 3.2: Tools cited in the literature on a period of time

The chart, in Figure 3.2, first gives an overview of the tools and when (year) each of them was presented. However, data analysts can demand further details about the data points in the visualization by interacting through a mouse over event. Furthermore, to make the visualization more pleasurable, we used transformations as the chart is composed of two phases: when it is being plotted on screen and when the data analyst interacts with a particular point. In the first phase, the radius of the circles starts at 1 and grows to 7 in second fractions; the opacity starts at 0 and

[4] http://www.w3.org/Style/CSS/#specs
[5] http://getbootstrap.com/

goes to 1; circles appear in a chronological order in an interval of 500ms. The second phase starts with data analyst interacting with a specific circle by mouse over event. Figure 3.3 shows the data analyst interaction: the radius size changes to 12 and the opacity of all other circles change to 0.8 while the opacity of the current circle does not change to highlight the selected point. Moreover, a legend appears with the name, year and the country(ies) where the selected tool was developed. Besides, a brief description about the tool and other information such as name, year, university and/or industry and country(ies) involved appear below the chart. When the data analyst leaves the circle (mouse out event), the legend and all information about the tool disappear and size and opacity return to standard values as showed in Figure 3.2. We believe that this visualization is a visual index that helps the reader of the SLR to explore and to be involved with the results arousing his curiosity about the data and its specificities.

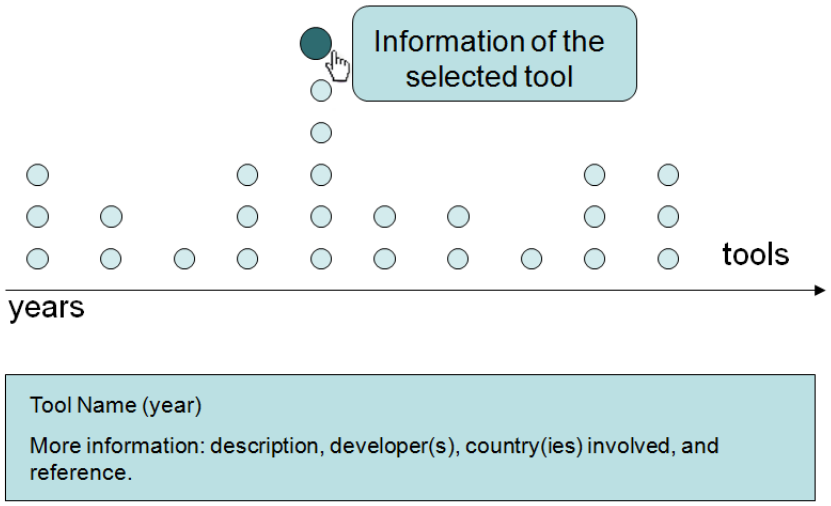


Figure 3.3: Details on demand: data analyst selects a particular tool to get more information

3.3 Bubble Chart

Knowing the profile of the participants of an experiment is a very important information in empirical studies. Before any analysis and conclusions, it is essential to have an overview of the characteristics of the group of people involved. In our empirical study, we mapped the "*background of participants*" (more details in Chapter 5). We collected

data about this background through a questionnaire. The information about each participant are: id, gender, tool used, level of different skills, and university that each participant is enrolled.

We have represented several possible background information of the different groups of people (genders, universities, etc) through a Bubble chart. Each bubble represent a participant and the areas of the circles encode levels of experience. Bubbles can then be grouped using different possible categorical attributes that segment the data. It is based on the Gestalt principle of proximity [Bertin, 1967] and what we want to verify is if the background of the participants is similar in each group. For instance, Figure 3.4 shows the participants grouped by tools. In our study each color encodes a tool. Each participant used only one tool and the size of the bubble encodes the work experience (WE) in software development industry. The radius are 5, 4, 3, 2 and 1 for more than 3 years of WE, 1 to 3 years, until 1 year, none WE and, no answers, respectively.

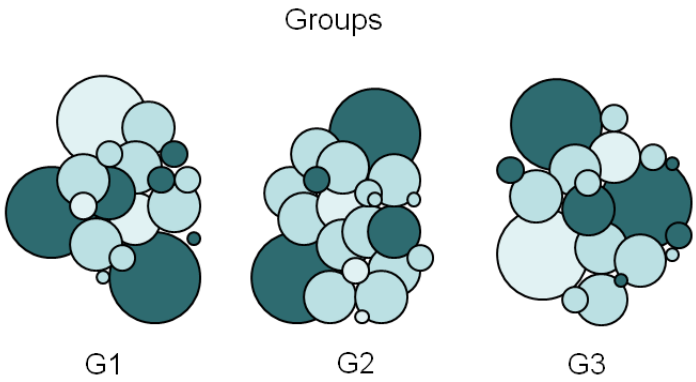


Figure 3.4: Participants grouped by tools.

The visualization is interactive and data analysts can choose different attributes to group data. For each new selection, the bubble diagram suffers a transition. data analysts have two ways to interact with the visualization. Figure 3.5 depicts one of the interactions for data analysts, when they run mouse over a bubble. In this case, a legend appears with extra information about the specific participant, such as id, gender, used tool, English level, skills level, work experience, and university of origin. Second, data analysts can rearrange the bubbles grouping by tool, university, specific skill level, English level, or work experience. In addition, when data analysts choose one of these options, the bubbles are separated and grouped into different regions of the screen according to the selection.

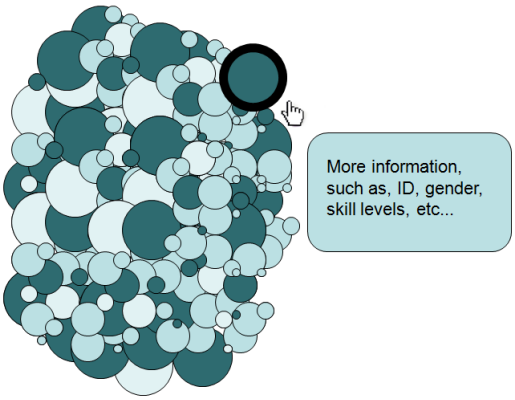


Figure 3.5: Background of participants

3.4 Heatmap

A remarkable aspect of the empirical data is the possible correlations that can be established between the background of participants of an study and the quality of the tasks done, the level of difficulty faced or the number of mistakes. This aspect is one of the most interesting and, for this reason, we propose two different visualizations interactively linked. Figure 3.6 shows an example of the heatmap which is a matrix-based display that simultaneously reveals correlations between the values in a first attribute (encoded as rows) and the second (depicted as columns). The quantitative values of the attributes are encoded as colors in different intensities. In other words, it consists of a set of cells, each colored based on a quantitative scale corresponding to attributes of the matrix.

In a usual heatmap, cells are colored based on a real number scale or can be discretized if necessary. Instead, in our heatmap, we used a divergent color scale going from red to blue to encode failure and success in task completion. We also decided to combine the colors of the cells for the whole set of participants related to the specific cell to depict a summary of the results per cell. Notice that, for each cell we have a number of participants and four different possible levels of difficulty. For this reason, we linked a donut chart presenting the part-to-whole quantitative relationship of this variable. Hence, data analysts can not only have more information about the results represented by each cell but also have a general view about the results presented in all the combinations of tasks and background in the matrix.

Figure 3.6 shows the map of each level of knowledge versus results of the tasks performed by participants of an experimental study. Each cell is colored with the combination of colors by percentage of results. For example, if we want to evaluate the impact for “tool 1” of participants with “skill 5” knowledge and their performance on

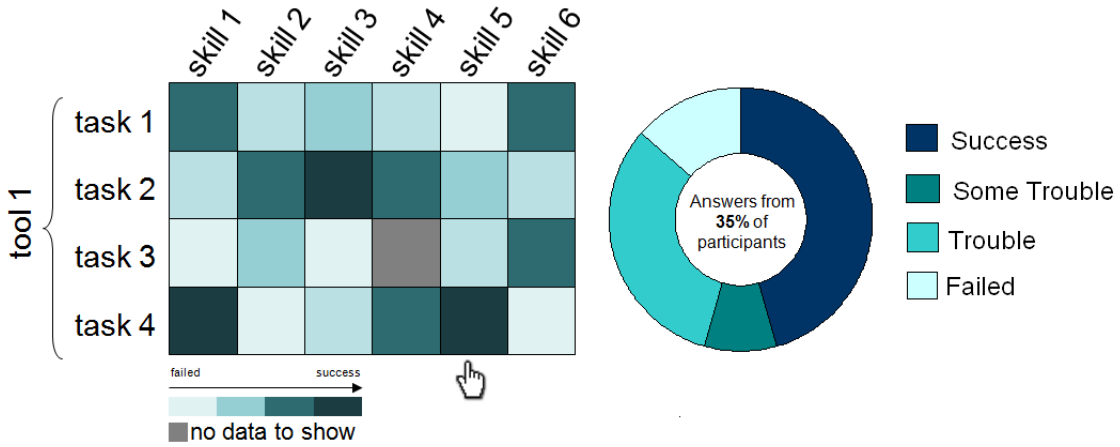


Figure 3.6: The map of the levels of knowledge versus result of the tasks for participants of an experimental study

the "task 4", data analysts can move the mouse over a specific cell and a donut chart with the result task of these participants appears showing details about the proportions of each result. In addition, the center of the donut shows the percentage of answers of the participants that have this knowledge and perform this task using "tool 1".

3.5 Bar and Donut Charts

The heatmap and donut charts provide a general view of the data set and the results obtained by an empirical study. However, a possible limitation is that they do not combine skills or tasks in an *ad hoc* manner and see how these variable can impact on the results obtained by that participant in the performed test. With this objective, we proposed an interactive combination of bar and donut charts to show at a first glance both the distributions of the different backgrounds and the proportions of the results obtained in each task. Taking into account the human perceptual system and specially the memory limitations when charts to be compared are distant, we presented the whole set of charts organised in a dashboard where data analysts can compare and contrast results in a straightforward manner.

Figure 3.7 shows the overview of this dashboard that depicts the background versus task results. This figure shows the task results, knowledge, tasks and tools for all participants. Our goal is to analyze whether the background of participants impact on the use of the analyzed tools and, as the dashboard is highly interactive, different combination between the skills can be done in order to analyze the results of tasks

in each SPL tool. For instance, supposing that we want to evaluate only the results obtained by highly skilled participants, we can select this level in all possible categories (backgrounds) and analyze the results obtained by only this group on the respective donut charts.

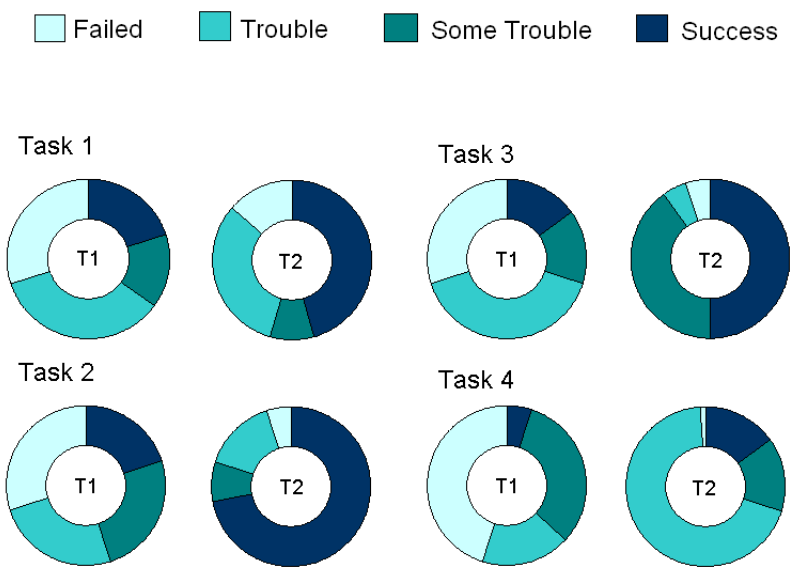


Figure 3.7: Background of participants impact on the use of each tools

In Figure 3.8(a), each bar is intended to show the percentage of participants who claimed to have high, medium, low, or none knowledge in Object-Oriented Programming (OOP), Unified Modeling Language (UML), and Software Product Line (SPL). For Work Experience (WE), the qualitative scale is mapped to: more than 3 years, 1 to 3 years, up to 1 year, and never worked in software development industry. We used bars to encode the respective values (their distribution) and visually reinforce the independent nature of these knowledge and their levels. Figure 3.8(b) shows that the data analyst can interact by selecting and deselecting any bar representing each possible level of each of the possible types of knowledge.

Figure 3.9 shows donut charts which are later used to summarize the task results. This figure depicts the percentage of participants who *were unable to perform*, *performed with major problem*, *performed with minor problem*, and *had no problem to perform their tasks*. Besides, each donut chart is intended to summarize the results of one task in one specific tool. The legend in the center of each donut is to identify the matching tool. For instance, *"T1"* means "Tool 1" and *"T2"* means "Tool 2".

We used donut charts to represent the results of each task by tool. We know that

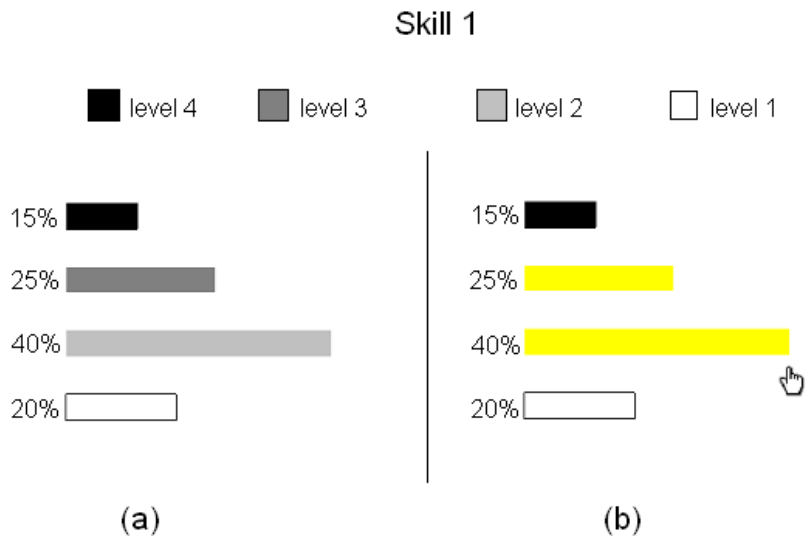


Figure 3.8: Background of participants with respect to one skill (a). Data analyst makes a combination of knowledge, selecting or deselecting some skill levels (b)

donuts are as less accurate than bar chats [Tufte and Graves-Morris, 1983]. Besides, we did not show the percentage in each slice to avoid visual pollution and too many information. Nevertheless, Figure 3.9 shows that the data analyst can interact with specific slices by mouse overing events to see tooltips with the result and its percentage.

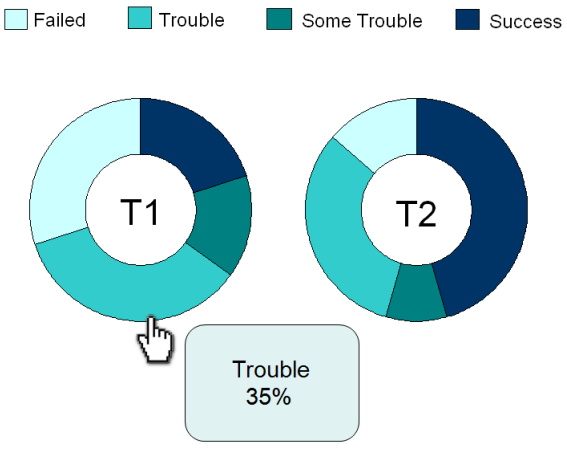


Figure 3.9: The results for participants with some knowledge selected.

3.6 Diverging Stacked Bar Chart

Figure 3.10 illustrates a diverging stacked bar chart. This chart provides an effective way to communicate summaries of data collected with Likert and other rating scales [Robbins and Heiberger, 2011; Heiberger and Robbins, 2014]. In our study, we used this visualization to illustrate if two different scales correlate in our empirical study. First, we used the diverging stacked bar chart to present the results of tasks when we asked about the problems faced during the tasks performed by participants. For each task, the participant had four options as answers: (i) *I was unable to perform*, (ii) *I performed with major problem*, (iii) *I performed with minor problem*, (iv) *I had no problem performing the task*. We considered (i) and (ii) as more difficult to perform the task (negative percentage) and they are shown on the left of the zero line. We considered (iii) and (iv) as easier to perform the task (positive percentage) and they are shown on the right of the zero line.

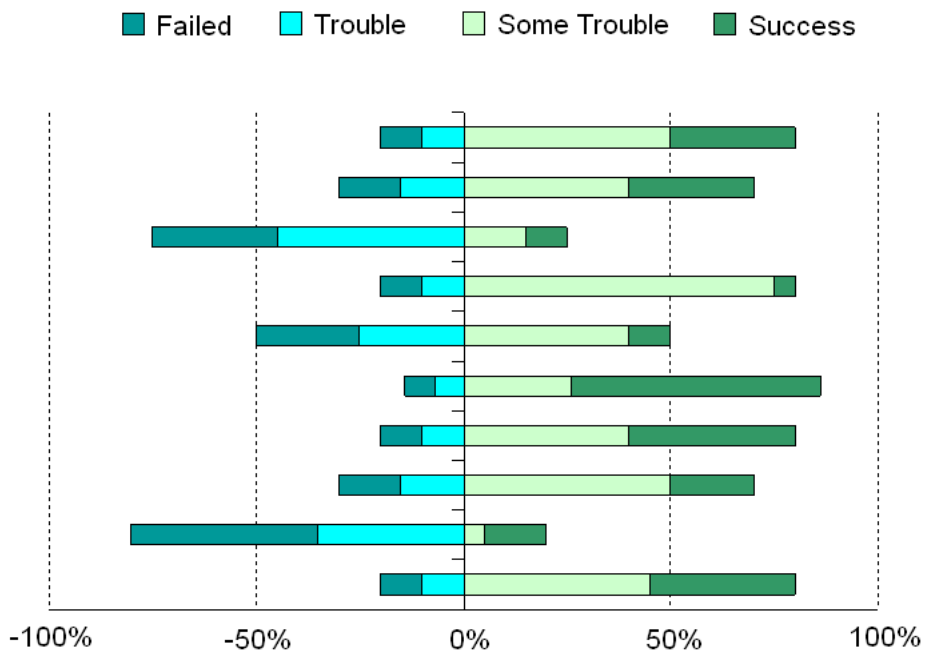


Figure 3.10: Diverging Stacked Bar Chart for presenting results of rating scales

Second, we used the diverging stacked bar chart to present the participants' answers about strengths and weaknesses of each tool. In our study the points were: (i) *automatic organization*, (ii) *automatic analysis*, (iii) *editor*, (iv) *examples available*, (v) *hot keys*, (vi) *integration with tools*, (vii) *interface*, (viii) *persistence models*, (ix) *product configuration*, (x) *tutorials and users guides*, and others. We considered strengths as positives and weaknesses as negatives points that the participants faced during the tasks with the SPL tools. The percentages of participants who considered the items as strengths are shown on the right of the zero line. The percentages who considered the items as weaknesses are shown on the left.

3.7 Concluding Remarks

In this chapter, we presented the Vi**SPL**atform. It is a visual platform to present and to favor the understanding of empirical data in the SPL tool context. In addition, this chapter explained the architecture and each visual representation available in the platform and it helped us to communicate and to analyze empirical data. The next two chapters present the two research methods in the field of software engineering where the data set was collected and presented using these visualizations. The next chapter outlines an systematic literature review (SLR) about SPL management tools. The goal of this SLR is to get an overview of existing research on SPL management tools, focusing since the SPL conception to its development, maintenance, and evolution.

Chapter 4

Visualizing Systematic Literature Review Data

This chapter uses visualizations of Vi**SPL**atform to analyze data of a Systematic Literature Review (SLR). SLR is one study method that has obtained much attention lately in software engineering [Kitchenham et al., 2009a]. Our SLR represents a step forward in the state-of-the-art by examining many relevant tools for managing software product lines. The general propose of this study is to give a visual summary, by categorizing existing SPL tools. We search in journals and conference proceedings since 2000. Therefore, we have used a systematic and rigorous method to accomplish this study, identifying and selecting the reviewed primary studies. The next sections are organized as follows. Section 4.1 presents design details of the review. Section 4.2 reports and analyzes the results obtained, and contributes specifically with relevant information of the existing tools. Section 4.3 presents the threats to validity related for this SLR and how they were addressed prior of the study to minimize their impact. Finally, Section 4.4 concludes this chapter with some final remarks.

4.1 Study Settings

This section gives a detailed description of the review design, the search string used, and the research questions. Our systematic review was performed following guidelines proposed by Kitchenham et al. [2009a], which is divided in three main phases: planning the review, conducting the review, and reporting the review.

4.1.1 Planning the review

This phase has the goal of producing a review protocol and defining the basic review procedures to identify, assess, and collect evidences. The *Planning* step includes several actions presented below.

Identification of the need for a review. Prior to undertaking an SLR, researchers should ensure that a systematic review is necessary [Kitchenham et al., 2009a]. In this study, the need for a systematic review originates from increase in the number of SPL management tools made available. In this context, the choice of one tool that best fits practitioner needs in a specific context of SPL is far from trivial. Therefore, this systematic review aims to give a complete, comprehensive, and valid picture of the tools available in the literature, in order to find out how the available tools are providing support to manage variability.

Specifying the research questions. The goal of this study is to get an overview of existing research on SPL management tools, focusing since the SPL conception, through out its development, maintenance, and evolution. The overall goal is to answer the main Research Question (RQ): *How do the available tools for SPL support the management of variability?* In order to answer this question, we defined two research sub-questions:

RQ1. *How many SPL management tools have been cited in the literature between 2000 and 2014?* We want to identify the largest number of tools cited in the literature since 2000. Further, we want to know which of them are discontinued and currently active.

RQ2. *What are the main functionalities of the SPL management tools?* We concerned with how the tools support each stage of the development process, considering the phases of SPL conception, development, and maintenance. In particular, what SPL topics, contributions, and novelty they aggregate to. Hence, it is possible to map how tools are supporting the SPL management process and if the process is not fully supported, i.e., if there are gaps in the existing tools, or if there is a need of developing functionalities that are not available in existing tools.

Developing a review protocol. We conducted a SLR in journals and conference proceedings published from January 1st 2000 to December 2014[1], by the fact that visibility provided by SPL in recent years has produced a higher concentration of

[1]The SLR was first performed until Dec. 2013 and later extended to Dec. 2014.

research [Lisboa et al., 2010]. Three researchers, with experience in software product line engineering, were involved in this process and all of them continuously discussed and refined the research questions, search strings, inclusion, and exclusion criteria.

4.1.2 Conducting the review

This phase is responsible for executing the protocol planned in the previous phase. It includes several actions, as discussed below.

Identification of research. Based on the research questions, some keywords were extracted and used to search the primary study sources. The search string used was constructed using the strategy by Chen and Babar [2011]. Following this strategy, the search string addressed by this study is:

("management tool") AND
("feature modeling" OR "product configuration" OR "variability management") AND
("product line" OR "product family" OR "system family")

The search for primary studies was based on the following digital libraries: ACM Digital Library[2], IEEE Xplore[3], and ScienceDirect[4]. These libraries were chosen because they are some of the most relevant sources in software engineering [Keele, 2007]. The search was performed using the specific syntax of each database and considering only the title, keywords, and abstract. In addition, we used the technique called "snowballing" which refers to using the reference list of a paper or the citations to the paper to identify additional papers [Wohlin, 2014].

Selection of primary studies. The basis for the selection of primary studies is the inclusion and exclusion criteria [Kitchenham et al., 2009b]. We defined two inclusion criteria (IC):

- *IC1.* The publications should be "journal" or "conference" and written in English.
- *IC2.* Primary studies about tools that manage one or more phases of the development cycle and maintenance of SPLs. Therefore, the abstract should explicitly mention that the focus of the paper contributes with tools to support SPL variability management.

[2]http://dl.acm.org/
[3]http://ieeexplore.ieee.org/
[4]http://www.sciencedirect.com/

We defined the following exclusion criteria (EC). The first three exclusion criteria were applied during the reading of titles and abstracts:

- *EC1.* Technical reports that present lessons learned, theses/dissertations, and duplicate papers. If a primary study is published in more than one paper (for example, if a conference paper is extended to a journal version), only one instance should be counted as a primary study. Mostly, the journal version is preferred, as it is most complete.
- *EC2.* Studies that describe events, studies that are indexes or programming.
- *EC3.* Papers that present approaches, methods, and techniques, but do not focus on SPL management tools.

After that, other exclusion criteria were applied for the set of papers selected:

- *EC4.* Tools that do not address the phases of development and maintenance of SPLs.
- *EC5.* Tools without executable or documentation describing its functionalities available. Moreover, the tools with written documentation that does not have usable description about its functionalities were excluded, because it is not possible to describe how the tools work.

Each paper was included or rejected by three experts in software engineering: two M.Sc. students and a researcher. All they were responsible to applied each criterion and check the results for the inclusion and exclusion of all papers. This step was done in order to check that all relevant papers were selected.

Data extraction and monitoring. First, the researchers read the abstracts and look for keywords and concepts that reflect the contribution of the paper. While doing so, the researchers also identified the context of the research. When abstracts are not enough, researchers also read the introduction and conclusion sections. When the researchers concluded that a paper is not relevant, they provided a short rationale why the paper should not be included in the study.

Once the list of primary studies is decided, the data from the tools cited by the papers are extracted. The phase of data extraction aims to summarize the data from the selected studies for further analysis. All tool documentation served as data sources for the data extraction; i.e, tutorials, technical reports, theses, websites, as well as

the communication with authors. However, during the data extraction, the exclusion criteria (*EC4* and *EC5*) were verified. The data extraction is designed based on the research questions. The data extracted from each selected tool were: (i) date of data extraction, (ii) primary studies (reference(s)), (iii) tool name, (iv) main reference, (iii) release year, (iv) tool website (if available), (v) main characteristics, (vii) where the tool was developed (academia or industry), and (vi) main functionalities each tool supports. The data extraction and monitoring are detailed in Section 4.2.

4.1.3 Reporting the review

This final phase is responsible for writing up the results of the review and circulating the results to community. Thereby, we published the detailed results in the project website[5]. The website also provides more detailed information about the results of the search protocol (with complete lists of included and excluded primary studies) and the chosen tools. Therefore, it is clear to others how the search was conducted, and how they can find the same documents.

4.2 Results and Analysis

This section presents and reports the results of this SLR to answer our research questions. Section 4.2.1 aims to identify the tools cited in literature between 2000 and 2014 and Section 4.2.2 describes the functionalities available or not available in each existing tool.

4.2.1 Selected SPL Management Tools

Our goal in this section is to identify the tools cited in the literature since 2000. In other words, we want to answer the following research question:

RQ1. *How many SPL management tools have been proposed in the literature between 2000 and 2014?*

For this analysis, we applied the inclusion and exclusion criteria to identify the relevant papers about the tools. In the first stage, after applied inclusion and exclusion criteria, 35 papers were included and 110 papers were excluded. In the second stage, we included 4 papers [Asikainen et al., 2004; Dhungana et al., 2007; Simmonds et al.,

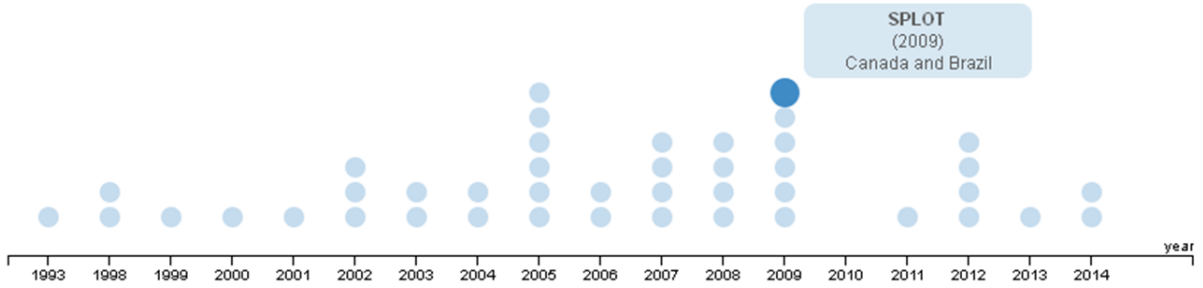
[5]http://homepages.dcc.ufmg.br/~kattiana/SLR/

2011; Unphon, 2008] that also matched our inclusion criteria. These papers were mined from references of included papers (technique called "snowballing"). This technique was necessary in order to have a more complete set of information and references about tools. At the end of the search, 39 papers were included for extracting and analyzing the data. Table 4.1 summarizes the number of papers of our systematic review per year of publication.

Table 4.1: Number Searched for Years 2000-2014

Year	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	All
Total	1	1	4	2	1	2	7	7	7	10	16	16	15	16	40	145
Total Selected	0	0	0	1	0	1	2	2	2	5	6	6	4	4	2	35
Total Snowballing	0	0	0	0	1	0	0	1	1	0	0	1	0	0	0	4
Total Included	0	0	0	1	1	1	2	3	3	5	6	7	4	4	2	39

After the papers inclusion process, 61 potentially relevant tools were selected for extracting and analyzing the data. Another search was performed on Web engines with the particular information of every tool cited by the papers in order to find more documentation about these tools. During data extraction, 18 tools were excluded, as a result of applying the detailed exclusion criteria *EC4* and *EC5*. Figure 4.1 presents the 43 tools included and their brief description can be visualized in Vi**SPL**atform.



SPLOT (2009)

Software Product Lines On-line Tool is a Web-based tool for creating feature models and product configuration.

Developed by: University of Waterloo

Country(ies) involved: Canada and Brazil

Reference: Mendonca, M. et al.: S.P.L.O.T.: Software Product Lines Online Tools. In: 24th Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), pp. 761-762 (2009).

Figure 4.1: More information detailed about the SPL tool selected

Our goal with Figure 4.1 is to show how many and what are the SPL tools presented during a determined time period. Each circle appears in a chronological order, contributing to the realistic look of the time line. In addition, when the data analyst interacts with a specific circle a legend appears with the name, year of the tool and, the country(ies) where it was developed. Besides, a brief description about the tool and other information such as, name, year, university and/or industry and country(ies) involved appear below the chart.

Note that, the papers included also mentioned tools developed before 2000. Further, some paper presented the year of the first release of the tool. When this information was not clear, we decided to use the year of the conference/journal paper as the year of the presentation of the tool to academic/industry community. With this visualization, it is also possible to see that most tools were developed between 2005 and 2009 (22 tools). Although in our SLR there was no tool developed in 2010, the development of new tools is still a research subject after 2009, as showed in Figure 4.1.

4.2.2 Main Functionalities of the Tools

Our goal in this section is to map how tools are supporting the SPL management process and if there are gaps in the existing tools, or if there is a need of developing functionalities that are not available in existing tools. In other words, we aim to answer the following research question:

RQ2. *What are the main functionalities of the SPL management tools?*

In order to answer RQ2, our analysis extended an existing classification of research approaches [Lisboa et al., 2010], summarized in Table 4.2. These classification schema present a practical approach for analysis based on a well-defined set of guidelines and metrics. We extend the previous classification [Lisboa et al., 2010] by new functionalities, such as *source code generation* and *support for integration*.

Based on Table 4.2, Table 4.3 shows which functionalities each tool supports. The numbers refer to the functionalities described in Table 4.2 and the column separate each group of functionalities: *Planning*, *Modeling*, *Validation*, and *Product Configuration and Integration* (PCI), respectively. The analysis of the results focuses on presenting the frequencies of publications for each functionality. This result facilitates the identification of which functionalities should be focused on future research. In addition, it can help to discover which tool best satisfies the need of a stakeholder in a given context.

Table 4.2: Explanation of functionalities evaluated

Functionalities	Explanation
Planning	*It is responsible for collecting the data needed to define domain scope*
(1) Pre-analysis documentation	Stores and retrieves the information
(2) Matrix of the domain	It is represented using rows and columns (features and applications).
(3) Scope definition	Identifies the features that should be part of the reuse infrastructure
Modeling	*It represents the domain scope (commonality, variability and constraint)*
(4) Domain representation	Represents the defined scope
(5) Variability	Represents the variability a feature can have (optional, alternative and or)
(6) Mandatory features	Represent the features that will always be in the products
(7) Composition rule	Create restrictions for representing and relating the features
(8) Relationship types	Provides different types of relationships between the features
(9) Feature attributes	Permits the inclusion of specific information for each feature
(10) Source code generator	Responsible for generating source code based on model
Validation	*This group refers to functionalities responsible to validate the domain*
(11) Domain documentation	Provides documentation about the domain
(12) Manage requirements	Provides support for inclusion of requirements or use cases in the tool
(13) Relationship (features and requirements)	Relates the existing features of a domain to the requirements
(14) Reports	Generates reports about the information available in the domain
(15) Consistency check	Verifies if the generated domain follows the composition rules created
Product Configuration and Integration (PCI)	*Product configuration is built from a common set of reusable asset, and Integration allows the interoperability between other applications.*
(16) Product derivation	Identifies the features that belong to a product
(17) Import/export	Provides the function of Import/export from/to other applications

In Table 4.3, there is an evident lack of tools to support all stages of SPL development and evolution. In this classification, the majority of the analyzed tools have similar functionalities. Most of the tools available both commercially and freely support the first four functionalities in group Modeling, the last functionality in the Validation group, and the two functionalities in the Product Configuration and Integration group. Therefore, this result identified that the largest gap in the analyzed tools is support to the Planning group. In the Planning group, the identified functionalities were available only in twelve tools, and only five tools implement all functionalities in this group.

The results of this systematic review reveal that 72% of the selected tools stress the need for planning support. A much smaller number of tools (28%) provides concrete support. This seems to indicate that, despite increasing interest and importance, planning is not yet the main focus of the product line research community. Particularly, in Modeling group, thirty-seven tools (86%) support at least four of the functionalities of this group. Regarding the Validation group, only one tool (YaM) does not support any functionality. Note that eight-teen tools (41%) support at least three of the functionalities of this group.

Table 4.3: Functionalities of SPL management tools

Tool	Planning			Modeling							Validation					PCI	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
SPLICE [Vale et al., 2014]	•	•	•	•	•	•	•	•	•		•	•	•	•		•	
EASy-Producer [Eichelberger et al., 2014]					•		•	•		•					•	•	•
FeatureIDE [Thüm et al., 2014]				•	•	•	•	•		•				•	•	•	•
LISA toolkit [Groher and Weinreich, 2013]	•		•	•	•	•	•	•	•	•	•	•	•				•
Invar [Dhungana et al., 2013]											•				•	•	•
pure::variants [Beuche, 2012]				•	•	•	•	•	•	•	•			•	•	•	•
ISMT4SPL [Park et al., 2012]	•		•	•	•	•		•		•		•	•		•	•	
Sysiphus-IVMM [Thurimella and Bruegge, 2012]	•	•	•	•	•	•	•					•	•		•	•	
VariaMos [Mazo et al., 2012]				•	•	•	•							•	•	•	•
VULCAN [Lee et al., 2012]				•	•	•	•	•	•	•					•	•	•
DOPLER [Dhungana et al., 2011]	•		•	•	•	•	•		•		•	•	•		•	•	•
Pacogen [Hervieu et al., 2011]														•	•		•
Metadoc FM [Thurimella and Janzen, 2011]				•	•	•	•				•	•	•		•	•	•
AORA [Varela et al., 2011]	•	•	•	•	•	•	•				•	•	•		•	•	•
FMT [Laguna and Hernández, 2010]				•	•	•	•		•						•	•	•
Hephaestus [Bonifácio et al., 2009]											•	•	•		•	•	•
Hydra [Salazar, 2009]				•	•	•	•		•						•	•	•
SPLOT [Mendonça et al., 2009]				•	•	•	•		•		•			•	•	•	•
S2T2 [Botterweck et al., 2009]				•	•	•	•							•	•	•	
FeatureMapper [Heidenreich et al., 2008]				•	•	•	•		•		•	•	•		•	•	•
MoSPL [Thao et al., 2008]				•	•	•	•								•	•	•
VISIT-FC [Cawley et al., 2008]				•	•	•	•								•	•	
GenArch [Cirilo et al., 2008]				•	•	•	•		•		•				•	•	•
DECIMAL [Dehlinger et al., 2007]				•	•	•	•								•	•	
GEARS [Krueger, 2007]				•	•	•	•	•		•				•	•	•	•
DecisionKing [Dhungana et al., 2007]	•		•	•	•	•	•		•		•	•	•		•		•
FaMa [Benavides et al., 2007]				•	•	•	•	•	•					•	•	•	•
Kumbang [Myllärniemi et al., 2007]				•	•	•	•		•						•	•	
REMAP-tool [Schmid et al., 2006]	•		•	•	•	•	•				•	•	•		•	•	•
ASADAL [Kim et al., 2006]				•	•	•	•	•		•					•	•	•
YaM [Jain and Biesiadecki, 2006]			•													•	
DOORS Extension [Buhne et al., 2005]				•	•	•	•					•	•		•		•
PLUSS toolkit [Eriksson et al., 2005]				•	•	•	•						•	•	•	•	
XFeature [Ondrej and Alessandro, 2005]				•	•	•	•	•	•	•	•				•	•	•
COVAMOF-VS [Sinnema et al., 2004]				•	•	•	•								•	•	•
DREAM [Park et al., 2004]		•	•	•	•	•		•					•	•			•
WeCoTim [Asikainen et al., 2004]				•	•	•	•		•						•	•	•
VARMOD [Pohl, 2003]				•	•	•	•		•		•	•	•		•		•
Captain Feature [Bednasch et al., 2003]				•	•	•	•								•	•	
HOLMES [Succi et al., 2001]	•	•	•	•	•	•	•				•				•	•	•
Odyssey [Braga et al., 1999]				•	•	•					•	•	•				
DARE [Frakes et al., 1997]	•	•	•	•	•	•					•						
001 [Krut, 1993]				•	•	•	•	•		•	•			•	•	•	

In the Product Configuration and Integration group, thirty-five tools (81%) support the product derivation functionality, and the import/export from/to other applications functionality is exploited for thirty tools (70%). Integration is a desirable functionality since it allows the interoperability between other applications. The lack of this functionality could hinder the adoption of the tool, as it hampers their integration with other existing tools. Therefore, although the results highlight the lack of tools to fully support SPL management; e.g., much of the analyzed tools do not support the Planning group. On the other hand, we found that the tools offer interoperability between other applications. This fact maximizes the reuse within the solution itself and externally, for instance, allowing stakeholders to migrate from on technology (e.g., conditional compilation [Figueiredo et al., 2008]) to another (e.g., feature oriented programming [Gaia et al., 2014]).

4.3 Threats to Validity

This section discusses the different validity threats related to the literature review and how they were addressed prior to the study to minimize the likelihood of their realization and impact. We discuss the study validity with respect to the four categories of validity threats [Wohlin et al., 2012b]: external validity, internal validity, construct validity, and conclusion validity.

External validity concerns the ability to generalize the results to other environments, such as to industry settings [Wohlin et al., 2012b]. A major external validity threat to this study was during the identified primary studies. The search for tools was conducted in several digital libraries in order to capture as much as possible the available tools and to avoid all sorts of bias. However, the quality of search engines could have influenced the completeness of the identified primary studies. That means our search may have missed those studies whose authors would have used other terms to specify the SPL tool or would not have used the keywords that we used for searching in the title, abstract, and keywords of their papers.

Internal validity concerns the question whether the effect is caused by the independent variables or by other factors [Wohlin et al., 2012b]. In this sense, a limitation of this study concerns the reliability. The reliability has been addressed as far as possible by involving three researchers, and by having a protocol which was piloted and hence evaluated. If the study is replicated by another set of researchers, it is possible that some studies that were removed in this review could be included and other studies

could be excluded. However, in general, we believe that the internal validity of the literature review is high given the use of a very systematic procedure, consultation with the researchers in the field, involvement, and discussion between three researchers.

Construct validity reflects to what extent the operational measures that are studied really represent what the researcher has in mind [Wohlin et al., 2012b]. The three reviewers of this study are researchers in the software engineering field, focused in SPL, and none of the tools was developed by us. Therefore, we are not aware of any bias we may have introduced during the analysis, but it might be possible that the conclusions have been affected by our personal interest and opinions. From the reviewers perspective, another construct validity threat could be biased judgment. The decision of which studies to include or to exclude and how to categorize the studies could be biased and thus pose a threat. A possible threat in such review is to exclude some relevant tool. To minimize this threat, both the processes of inclusion and exclusion were piloted by the three reviewers. Furthermore, potentially relevant studies that were excluded were documented. Therefore, we believe that we have not omitted any tool.

Conclusion validity is related with issues that effect the ability to draw the correct conclusions from the study [Wohlin et al., 2012b]. From the reviewers perspective, a potential threat to conclusion validity is the reliability of the data extraction categories from the tools, since not all the information was obvious to answer the research question and some data had to be interpreted. Therefore, in order to ensure the validity, multiple sources of data were analyzed, i.e, papers, prototypes, technical reports, manuals, and the tool execution. Furthermore, in the event of a disagreement between the two primary reviewers, the third reviewer acted as an arbitrator to ensure agreement was reached.

4.4 Concluding Remarks

In this chapter, we employed visualizations and interaction techniques, supported by Vi**SPL**atform, in order to illustrate and to explore the data of the SPL management tools. Besides, this chapter presented and discussed the findings of a Systematic Literature Review (SLR) of SPL management tools. In this step, our research method aimed at analyzing the available literature on SPL management tools. This review provides insights (a) to support companies interested to choose a tool for SPL variability management that best fits their needs; (b) to point out attributes and requirements relevant to those interested in developing new tools; and (c) to help the improvement

of tools already available. In the next chapter, we present visualizations to support an empirical study. The goal of this empirical study is to help SPL engineers choosing the SPL tool. This empirical study compares and analyzes three tools, namely SPLOT, FeatureIDE, and pure::variants, based on data from 124 participants that used the analyzed tools.

Chapter 5

Visualizing Empirical Study Data

This chapter presents the empirical study to evaluate three alternative SPL tools, namely SPLOT, FeatureIDE, and pure::variants. It relies on three integrated visualizations supported by Vi**SPL**atform to analyze the experimental data. Our empirical study involves 124 participants enrolled in courses related to the Software Engineering area. Each participant used only one tool, either SPLOT or FeatureIDE or pure::variants. We relied on a background questionnaire and a 1.5-hour training session to balance knowledge of the study participants. The experimental tasks focus on different aspects of SPL development. After all participants performed their tasks, they answered a questionnaire with close and open questions about the functionalities they used in each tool. This chapter is organized as follows. Section 5.1 describes the study configuration and some functionalities of SPLOT, FeatureIDE, and pure::variants. Moreover, this section presents the criteria and reasons to select these three tools. Section 5.2 reports and analyzes the results of this empirical study. In Section 5.3, some threats to the study validity are discussed. Finally, we end this chapter with some final remarks (Section 5.4).

5.1 Study Settings

This section presents the study configuration aiming to evaluate the three SPL tools, namely SPLOT, FeatureIDE, and pure::variants. Section 5.1.1 defines the study research questions. Section 5.1.2 introduces the three analyzed tools and explains the reasons for selecting them. Section 5.1.3 summarizes the background information of participants that took part in this study. Finally, Section 5.1.4 explains the training session and tasks assigned to each participant.

5.1.1 Research Questions

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

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

RQ2. *Does the background of developers impact on the use of the SPL tools?* With this RQ, we want 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?* Finally, with respect to RQ3, we aim at identifying what are the strengths and weaknesses of each tool analyzed based on quantitative data of the study participants.

5.1.2 Selected Software Product Line Tools

We found 43 tools for SPL variability management after performing a systematic literature review (Chapter 4). Based on this review, we used following exclusion criteria in order to choosing three heterogeneous 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 because (i) they do not cover all relevant functionalities we aim to evaluate, hindering some sorts of analysis; and (ii) they are not applicable to industry-strength SPL development. In addition, we excluded all tools unavailable for download and the commercial tool without an evaluation version. We also excluded tools discontinued.

Therefore, we have six tools candidates for our empirical study, they are SPLOT, FeatureIDE, pure::variants, FAMA, VariAmos, Odyssey. Other tools could be candidate, if they have an evaluation version available, such as, Gears and Feature Modeling Tool (this last is a plug-in to Visual Studio which is not free). Finally, from these six candidate tools left, we picked up the most well-known and cited tools in the SPL literature.

We chose SPLOT, FeatureIDE, and pure::variants as representative tools because these tools are mature, actively used (by industry or academic researches) and accessible tools in order to evaluate the state-of-the-art of SPL tools. In addition, they have been constantly extended/cited for relevant studies in the literature [Machado et al., 2014; Pereira et al., 2013; Simmonds et al., 2011; Unphon, 2008; Djebbi et al., 2007]. Furthermore, we chose only 3 tools in order to make it possible to conduct a deeper study (we have limitation of time and human resources to evaluate more tools). Table 5.1 presents a brief overview of these three SPL tools. The first column lists the functionalities and the other columns are the three SPL tools.

Table 5.1: Functionalities of SPLOT, FeatureIDE and pure::variants Tools

Functionality	SPLOT	FeatureIDE	pure::variants
Available	online/standalone	standalone	standalone
Feature model editor	yes	yes	yes
Feature model notation	tree	tree/diagram	tree/diagram
Automated feature model analysis	yes	yes	yes
Product configuration	yes	yes	yes
Integration with code		yes	yes
Repository of features models	yes		

A cell marked with "*yes*" in Table 5.1 means that the functionality is supported by the respective tool. As shown in this table, the analyzed tools support the main functionalities required to the development of SPL [Czarnecki et al., 2005]. SPLOT provides a standalone tool version that can be installed in a private machine. However, we used the online version of SPLOT in this study. pure::variants is a commercial tool, but there is an evaluation version available in its website. We used the evaluation version of pure::variants in this study.

5.1.3 Background of the Participants

Participants involved in this study are 124 young developers enrolled in courses related to the Software Engineering area. They were organized as follows: 41 participants worked with SPLOT, 42 participants worked with FeatureIDE, and 41 participants worked with pure::variants. Figure 5.1 presents the Bubble Chart visualization of Vi**SPL**atform. This visualization shows the participants represented by bubbles. Colors are used to identify the different SPL tools evaluated.

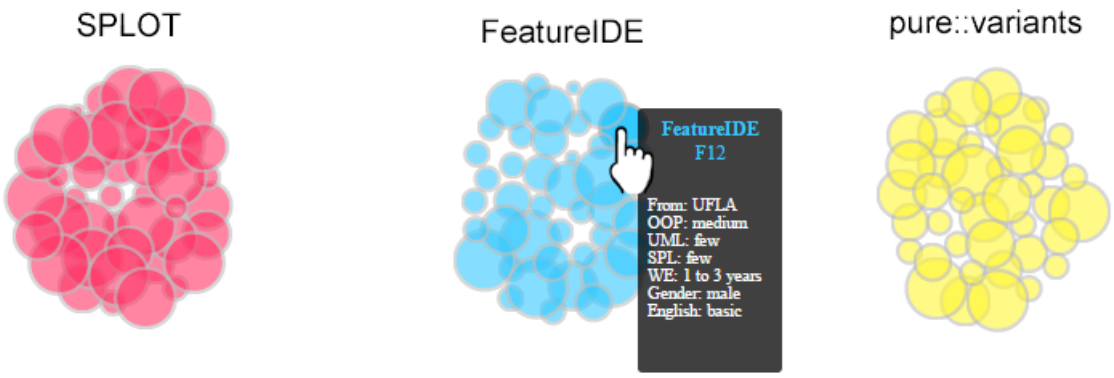


Figure 5.1: Each participant used only one tool, either SPLOT (pink) or FeatureIDE (blue) or pure::variants (yellow) tool

The participants were nicknamed as follows: (i) S1 to S41 worked with SPLOT, (ii) F1 to F42 worked with FeatureIDE and (iii) P1 to P41 worked with pure::variants. Our goal is to use these nicknames while keeping the anonymity of the participants separating them by tool, since we did not repeat participants in the experiments. All participants are graduated or close to graduate since they are mostly post-graduated M.Sc. and Ph.D. students from four different Brazilian institutions: UFLA[1], UFMG[2], UFJF[3], and PUC-Rio[4]. To avoid biasing the study results, each participant only took part in one study semester and only used one tool, either SPLOT or FeatureIDE or pure::variants.

Before starting the experiment, we used a background questionnaire to acquire previous knowledge of each participant. Figure 5.2 summarizes knowledge that participants claimed to have in the background questionnaire with respect to Object-Oriented Programming (OOP), Unified Modeling Language (UML), Software Product Line (SPL), and Work Experience (WE). The bars show the percentage of participants

[1]Universidade Federal de Lavras
[2]Universidade Federal de Minas Gerais
[3]Universidade Federal de Juiz de Fora
[4]Pontifícia Universidade Católica do Rio de Janeiro

who claimed to have knowledge high, medium, low, or none in OOP, UML, and SPL. For WE, the options were: more than 3 years, 1 to 3 years, up to 1 year, and never worked in software development industry.

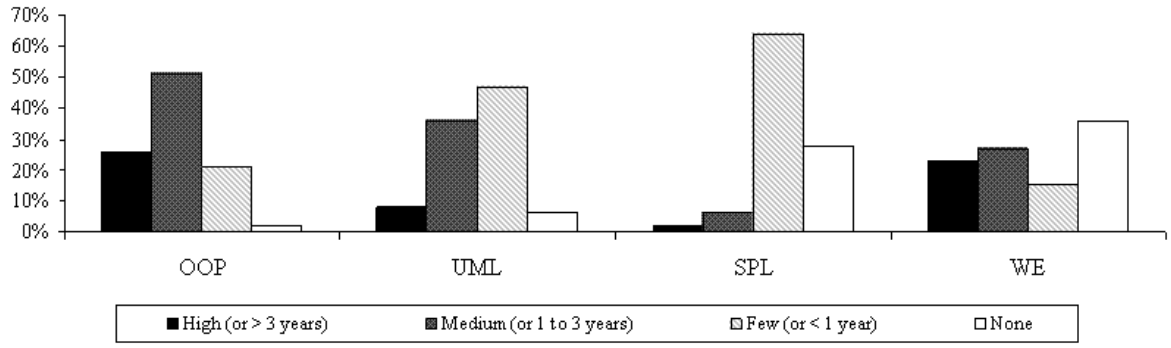


Figure 5.2: Background of participants with respect to Object-Oriented Programming (OOP), Unified Modeling Language (UML), SPL, and Work Experience (WE)

Answering the questionnaire is not compulsory, but only 2 participants did not answer the questionnaire. Thereby, we observe that about 77% of participants have medium to high knowledge in OOP, 44% in UML, and 8% in SPL. Finally, about 50% have more than 1 year of work experience in software development industry.

Although 28% of all participants claimed to have no knowledge in SPL, we conducted training session (Section 5.1.4). Basili et al. [1999] and Kitchenham et al. [2002] argue that even less experienced participants can help researchers to obtain preliminary, but still important evidence that should be further investigated in later controlled experiments. In general, we can conclude that all participants have at least basic knowledge in software development and technology required to perform the experimental tasks.

Further details about the distribution of participants, background questionnaire, and additional information about the participants (id of each participant, gender, the SPL tools used, and University that each participant is enrolled) can be observed by using Vi**SPL**atform. In our tool, bubbles can be grouped using different possible categorical attributes that segment the data (genders, universities, skills, etc). Figure 5.3, for instance, shows the participants grouped by work experience and the areas of the circles encode levels of work experience in software development industry.

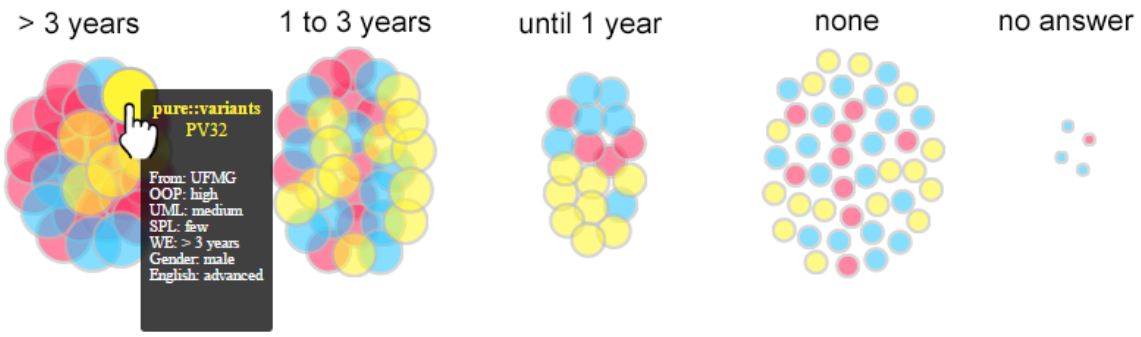


Figure 5.3: Background of participants grouped by work experience in software development industry

5.1.4 Training Session and Tasks

In this study, we conducted 1.5 hour training session where we introduced participants to the basic concepts of feature modeling, SPL, and a brief overview about the analyzed tools. The same training session was performed in all four institutions (Section 5.1.3) by the same researcher. All material about the course was available for all participants. After the training session, we asked the participants to perform some tasks using either SPLOT or FeatureIDE or pure::variants. We performed a four-dimension quantitative and qualitative analysis with respect to common functionalities provided by SPL tools as follows:

i **Feature Model Edition.** Edit a representation of all products of the SPL in terms of features. It includes creating, updating, and adding constraints in the feature model representation.

ii **Automated Feature Model Analysis.** It can be defined as the computer-aided extraction of information from feature models.For instance, counting the number of features, or dead features, or the number of valid configurations, etc.

iii **Product Configuration.** A product is declaratively specified by selecting or deselecting features according to stakeholder's preferences. After that, the task include some questions about the product configured, for instance, the number of configuration steps needs, or selecting or deselecting a constraints.

iv **Feature Model Import/Export.** It is how the Feature Model can be imported or exported as XML, CSV or other files and how to integrate the Feature Model in a new project.

The tasks include (i) to edit (create and update) a feature model, (ii) to automatically analyze the feature model created and observe its statistics, (iii) to configure a product relying on the tool product configuration functionality, and (iv) to import/-export a feature model. After performing the tasks, all participants answered a questionnaire with open and closed questions. The study was performed in the computer laboratory of each institution with at least the minimal configuration necessary for the experiment.

5.2 Results and Analysis

This section reports and discusses data of this empirical study by using the visualizations provided by Vi**SPL**atform. Section 5.2.1 reports the problems encountered by participants when performing the requested tasks. Section 5.2.2 focuses the discussion on whether the background of participants can impact on the use of each tool. Finally, Section 5.2.3 discusses the strengths and weaknesses of the analyzed tools: SPLOT, FeatureIDE, and pure::variants.

5.2.1 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 closed questions (Section 5.1.4) asking for the completion of the tasks. After each task of the experiment, participants had the following options to answer (i) *I was unable to perform*, (ii) *I performed with major problem,* (iii) *I performed with minor problem*, (iv) *I had no problem performing the task.* In order to answer the research question RQ1, we rely on data presented in Figure 5.4. This visualization 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. Thereby, the general observation is that participants had minor problems or no problem to perform the tasks. We also analyzed other characteristics of these results in this section. Some conclusions were also supported by qualitative feedback from the study participants.

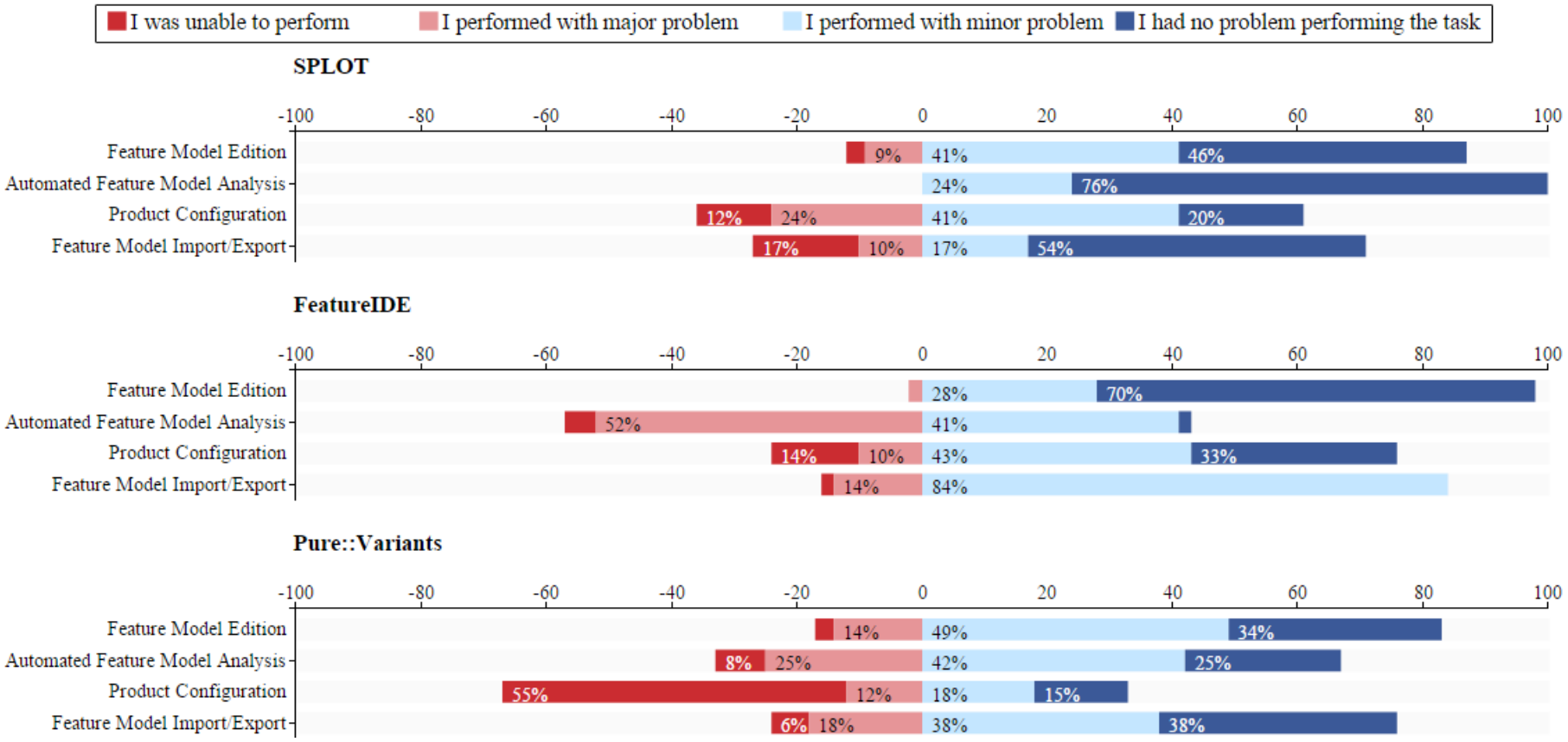


Figure 5.4: Percentage of problems reported by participants to complete their tasks

We first investigated the SPLOT tool. Product Configuration seems the most challenge functionality to use by the SPLOT participants. The visualization support in Figure 5.4 shows that 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 statistic computation [Mendonça et al., 2009], such as depth of the feature tree, number of possible configurations, and others. 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. Therefore, this dimension was considered the hardest one to use by participants, as we see in the visualization of Figure 5.4. 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. This seems a positive result for this tool because only 2% (1 participant of 42) reported a major problem to edit a feature model.

Last, in the light of the visualization platform, we investigated the pure::variants tool to answer the RQ1. If in the one hand, the Product Configuration dimension 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 5.4 shows the negative ratio of Product Configuration in pure::variants is bigger than in SPLOT and FeatureIDE, meaning that the participants had more difficulties to perform this task using pure::variants.

5.2.2 Background Influence

Our goal in this section is to analyze 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, Vi**SPL**atform presents two different visualizations interactively linked. Figure 5.5 shows the map of each level of knowledge versus results of the tasks performed by participants of our experimental study. Each line is one of the tasks: *Feature Model Edition*, *Automated Feature Model Analysis*, *Product Configuration*, and *Feature Model Import/Export*. In addition, each line corresponds one task followed by legend about the SPL tools. That is, *"(S)"* means SPLOT, *"(F)"* means FeatureIDE, and *"(P)"* means pure::variants. Each column is the knowledge level claimed by participants, such as high, medium, low, or no knowledge in Object-Oriented Programming (OOP), Unified Modeling Language (UML), and Software Product Line (SPL). For Work Experience (WE), the qualitative scale is mapped to: more than 3 years, 1 to 3 years, up to 1 year, and never worked in software development industry. Finally, each cell is colored with the combination of colors by percentage of results. We used a divergent color scale going from red to blue to encode failure and success in task completion.

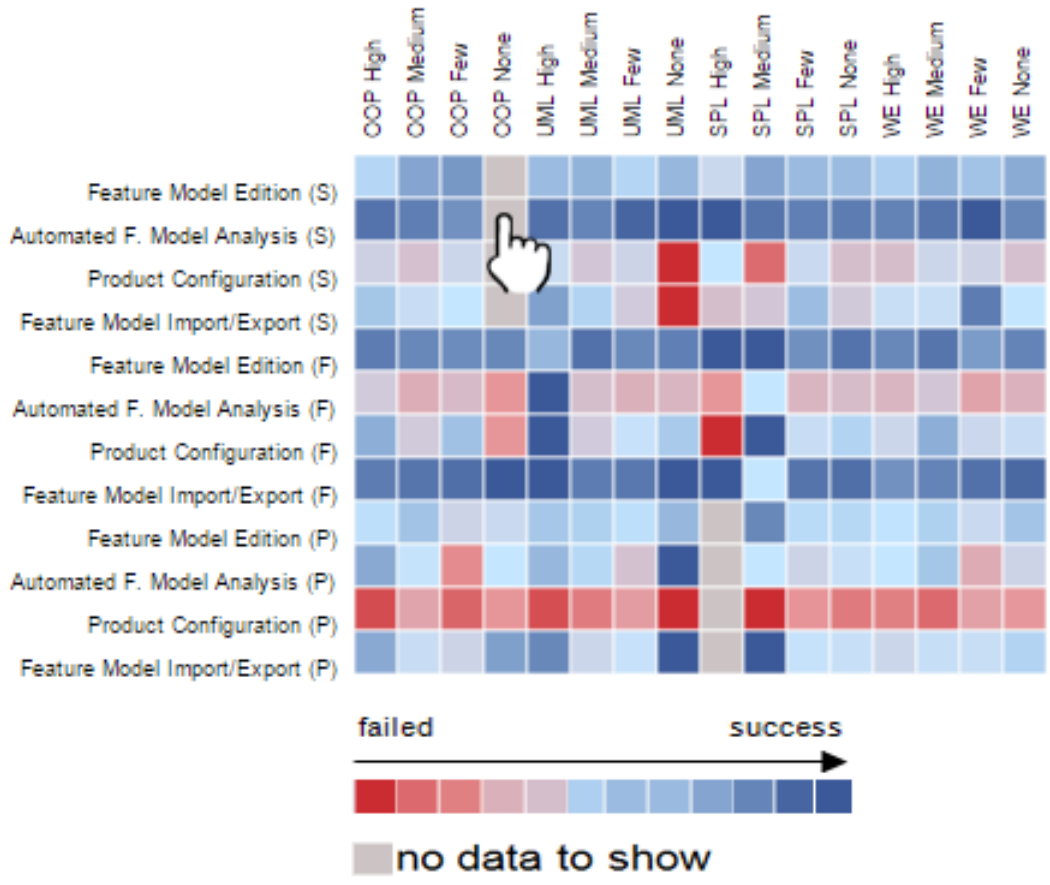


Figure 5.5: Overview of each level of knowledge versus results of the tasks performed by participants

In this visualization, data analysts can not only have more information about the results represented by each cell but also have a general view about the results presented in all tasks and skill levels in the matrix. Analogously, the visualization about the problems faced by participants (Figure 5.4), the data analyst can see with more details the task results of each level skill. Thereby, there are a great number of success or some troubles (blue intensity) for the *Automated Feature Model Analysis* task for SPLOT. Similarly, for FeatureIDE, the blue intensity is predominant for tasks *Feature Model Edition* and *Feature Model Import/Export*. By contrast, in all skill levels of the *Product Configuration* task for pure::variants participants have some difficult to perform it. Finally, note that, there is no participants who claimed to have "no" knowledge in OOP for SPLOT and "high" knowledge in SPL for pure::variants. In this manner, this insight was not easily visible in the other visualizations techniques. In these two cases, when data analysts interact with any one of these cells (gray cell) only a message "no data to show" appear for them. The limitation of this visualization is that the data analyst can not combine different skills level to analyze the background influence on the tasks. Thereby, we proposed another highly interactive visualization presented in Vi**SPL**atform, with different combination between the skills, in order to analyze the results of tasks of each SPL tool (RQ2).

First, we classified the participants by their level of knowledge and work experience. Group 1 (Strong Background) includes participants that claimed to have high and medium knowledge in OOP, UML, and more than 1 year of work experience. Group 2 (Weak Background) includes participants that answered few and no knowledge in OOP, UML, and less than 1 year of work experience. We did not include the SPL skill because most participants have little knowledge in SPL (see Figure 5.2) and we considered it as part of the experiment context. We believe that the training we gave to participants ensures a balanced knowledge with respect to SPL.

Figure 5.6 shows donut charts summarizing the results. Similarly to Figure 5.4, this figure depicts the percentage of participants who (i) *were unable to perform*, (ii) *performed with major problem*, (iii) *performed with minor problem*, or (iv) *had no problem to perform their tasks*. Charts on top indicate results for participants in the strong background group (32%) and charts on the bottom of Figure 5.6 indicate participants with weak background (18%). Besides, each donut chart summarizes the result of one task in one specific tool. The legend in the center of each donut is to identify the matching tool. That is, *"S"* means SPLOT, *"F"* means FeatureIDE, and *"P"* means pure::variants. Each set of three donut charts relates to one of the four functionalities analyzed in this study.

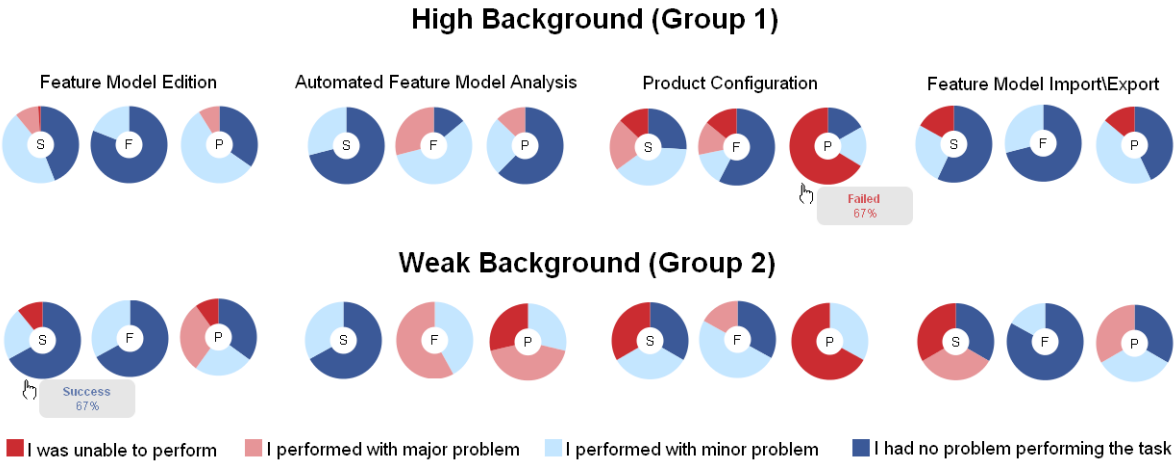


Figure 5.6: Strong and weak background of participants

Based on the visualizations of Figure 5.6, 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 11% of participants with weak background (Group 2) reported they were unable to conclude their tasks while 99% of strong background participants complete their tasks. In addition, the percentage of participants who had few problems or 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 for these participants of both groups do not seemed familiar with it. For pure::variants, the difference between Group 1 and Group 2 was even more clear. 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 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.

With respect to Product Configuration, in spite of SPLOT has been developed with focus on this task [Mendonça et al., 2009], both groups had similar performance. However, the Group 2 had 33% failures, while the Group 1 had only 13%. For the pure::variants tool, the two groups had equivalent results, where both exhibit a high percentage (67%) of failures. For all tools, the configuration of a product seems a simple activity. At first glance, this task seems to require only selecting or deselecting features of the model. However, other knowledge about the SPL model is also important, such

as comprehension of the notations, the relationships between features, and constraints. Thereby, we realized that this task is not trivial for beginners SPL developers of the pure::variants tool.

As shown in Figure 5.6, for Feature Model Import/Export, participants who used SPLOT presented big difference in the results when comparing both groups. The percentage of failures increased from 17% in Group 1 to 33% in Group 2. Although the repository of model is an interesting functionality of this tool, the participants of this study seem not familiar with it. Hence, it was difficult for participants with weak background to perform this task in SPLOT. However, this task is very common for experienced software developers, what may have contributed positively to the task in Group 1.

5.2.3 Observed Strengths and Weaknesses in SPL Tools

Our goal in this section is to investigate some of the strengths and weaknesses of SPLOT, 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 5.7, 5.8, and 5.9 show diverging stacked bar chart of the strengths and weaknesses of SPLOT, FeatureIDE and pure::variants, respectively. This visualization is supported by Vi**SPL**atform (Chapter 3). We considered strengths as positives and weaknesses as negatives points that the participants faced during the tasks with the SPL tools. 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. 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*, (x) *tutorials and users guides*. These items are sorted in alphabetical order in all three figures. Participants could also freely express about other strengths or weaknesses they encountered during the tasks.

For the SPLOT tool, 47% of its participants pointed the interface as the biggest weakness. 37% indicated the hot-key as a problem. In addition, 24% of them pointed product configuration and examples available as problems to understand the SPLOT tool, as shown in Figure 5.7. Although of SPLOT is not development tool, the integration with source code was another weakness pointed freely by its participants. On

the other hand, the two most voted strengths were: automatic analysis of the models (76%) and editor of model (61%). For other participants, 32% indicated that the tool interface was easy and intuitive and 24% indicated the product configuration as other positive points. Other strengths pointed freely by participants are the online tool and repository for sharing the models.

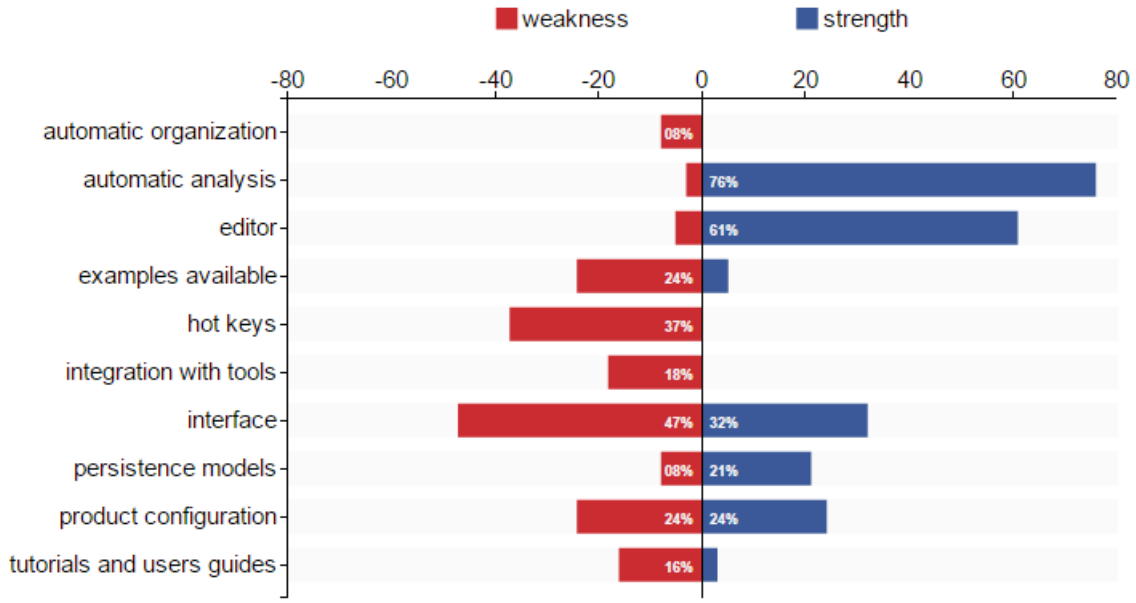


Figure 5.7: Strengths and weaknesses faced by participants during the tasks with SPLOT

After SPLOT, the FeatureIDE tool was analyzed. Figure 5.8 shows that about 33% of its participants voted in the examples available, interface (e.g., because it has been very difficult to find the functionalities) and, tutorials and users guides as weaknesses of this tool. Besides, 28% found the automatic analysis as deficiency of the FeatureIDE tool. However, the main problem regards the navigation to find the related menu for automatic analysis of the model. In a different manner, 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 the FeatureIDE tool. Other strengths pointed freely by participants are the creating of constraints, integration with source, and automatic creating of variants of the product by the tool.

In the third and last case, the pure::variants tool was analyzed. Figure 5.9 shows that for 50% of its participants, the interface was voted as the biggest weakness. Furthermore, about 47% and 39% of its participants pointed examples available and,

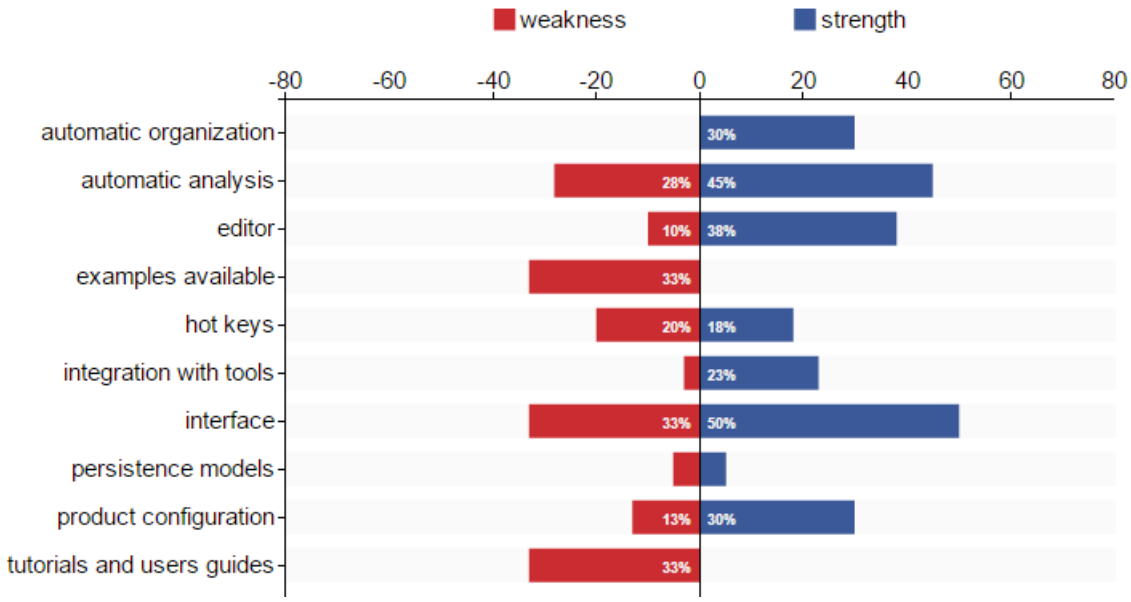


Figure 5.8: Strengths and weaknesses faced by participants during the tasks with FeatureIDE

tutorial and users guide as weaknesses, respectively. In opposition, for 61% of its participants the automatic analysis was considered as the biggest strength. About 58% of them pointed editor as strength. Automatic organization had 47% of the votes and, product configuration was other positive points with 33% votes.

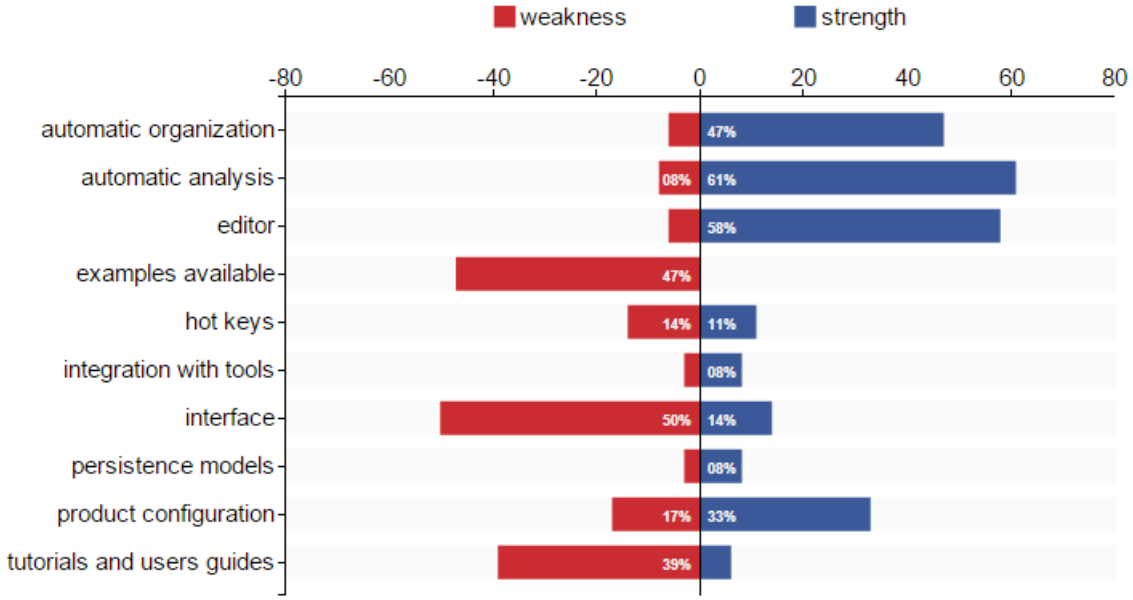


Figure 5.9: Strengths and weaknesses faced by participants during the tasks with pure::variants

Considering all participants, 43% of them found the interface as the biggest weakness of all three tools. Further, 34% indicated the lack of examples available, 29% indicated the lack of tutorial and user guide as weaknesses of all three tools. On the other hand, 61% of them found the automatic analysis as the biggest strengths and, 52% indicated the editor as strengths of all three tools.

5.3 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 from which the set of participants were involved. It is also interesting to generalize the results for 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 [Wohlin et al., 2012b]: constructs validity, internal validity, external validity, and conclusion validity.

Construct validity reflects to what extent the operational measures that are studied really represent what the researcher have in mind and what is investigated according to the research questions [Wohlin et al., 2012b]. The most common threats to this type of validity are related to experiment design and social threats. Threats of experiment design relate to poor definition of the theoretical basis or the definition of the testing process. Social threats, 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 [Wohlin et al., 2012b]. 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. To minimize this threat, we provide at least 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.

External validity concerns the ability to generalize the results to other environments, such as to industry practices [Wohlin et al., 2012b]. A major external validity can be the selected tools and participants. We choose three tools, among many avail-

able ones, and we cannot guarantee that our observations can be generalized to other tools. Because in Brazil there are not many SPL developers, then participants may not reflect the state of the practice in other countries. About participants, we tried to minimize this threat by working with random participants from various groups. We worked with both new and experienced developers. They are graduated or close to graduate since the course targets post-graduated M.Sc. and Ph.D. students.

Conclusion validity, on the other hand, concerns the relation between the treatments and the outcome of the experiment [Wohlin et al., 2012b]. This involves the correct analysis of the results of the experiment, measurement reliability and reliability of the implementation of the treatments. The conclusion of the analyzed we performed, could be another if it was done by other researchers or, if it was done by just one of us. To minimize this threat, we discuss the results data to make a more reliable conclusion.

5.4 Concluding Remarks

In this chapter, we presented the visualizations supported by Vi**SPL**atform to present and explore an empirical study that evaluate and analyzed three SPL tools, namely SPLOT, FeatureIDE, and pure::variants, based on data from 124 participants. In this study, we performed a four-dimension quantitative and qualitative analysis with respect to common functionalities provided by SPL tools: (i) Feature Model Edition, (ii) Automated Feature Model Analysis, (iii) Product Configuration, and (iv) Feature Model Import/Export. Our conclusion after analyzing the data in this chapter showed that the main issues observed in the three SPL tools are related to their interfaces, lack of examples available, tutorials, and limited users guide; 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 three analyzed tools have advantages and drawbacks. In the next chapter we investigate the state-of-the-art about the use of visualization information in the field of software product line engineering. It also discusses related work to our systematic literature review and empirical study of SPL management tools.

Chapter 6

Related Work

Section 6.1 presents some previous studies about information visualization in the Software Product Line context. In addition, Section 6.2 discusses related work to the systematic literature review and Section 6.3 outlines related work to the empirical study with the SPL tools presented in this dissertation.

6.1 Information Visualization in SPL

According to Diehl [2007], visualization techniques have been used in software engineering as a viable alternative to the difficult task of understanding, maintaining and evolving software systems. Thereby, some previous studies recognize the importance and utility of information visualization. Besides, they propose means to structure a body of knowledge in terms of techniques and algorithms. Although these studies are not related with information visualization applied to research methods, specially in SPL, they seek to comprise a set of techniques related to visualizations in Software Engineering . In this sense, several tools have been proposed in the literature that use visualization information to support developers in understanding and exploring individual characteristics by combining several visualization techniques. This approach is called software visualization which is a specialization of information visualization. We discuss some of these tools below.

Visual and Interactive Tool for Feature Configuration (*Visit-FC*) [Botterweck et al., 2008] is a visual and interactive tool for product configuration. It was based on a meta-model which specifies the main entities such as decisions, characteristics, components and the relationships between them. Visit-FC adds interactive functionality to SPL engineers, allowing clear operation and manipulation of data. It provides a compact and interactive representation of feature hierarchies, allowing configuration with

the spread of automatic restrictions, and providing tips for configuration issues and open decisions. The implementation of Visit-FC was based on the Eclipse Modeling Framework (EMF), which allows integration with other tools.

Colored Integrated Development Environment (*CIDE*) [Feigenspan et al., 2010] is an Eclipse plug-in to support the SPL development. It is based on filters and views on source code in order to visualize and trace features in source code to support especially the variability management in SPL. This tool maintains a direct mapping between features and implementation, based on colors. That is, the portion which code belongs and a given feature are of the same color. When a piece of code belongs to more than one feature, this stretch will be a combination of feature colors which it belongs. In addition, CIDE is a tool to work with SPL with fine granularity, as its mechanisms for separating and defining characteristics increase the organization, readability and understanding of the code.

There are other proposed annotation algorithm upon the CIDE tool. For instance, the CIDE+ tool [Valente et al., 2012] which automates the annotation of optional features of existing systems. In addition, CIDE+ supports the semi-automatic association of background colors to the lines of code that implement an optional feature. The authors pursued as goals of finding feature code at fine granularity in a single code base.

SourceMiner [Carneiro and Mendonça Neto, 2014] is an Eclipse plug-in to provide visual resources to enhance software comprehension activities. This tool allows the collection of data directly from the source code. In addition, this environment is able to log data from primitive operations executed by programmers on the visual scenarios. The plug-in is based on a set of principles aimed at building multiple visualizations and mapping it to software comprehension principles.

There is a lack of research work regarding information visualization on empirical data from SPL tools, focusing on experiments. Our work explores visualization techniques in this context. Thereby, it can help data analyst in interpreting semantic information about experimental process and the empirical data collected. Vi**SPL**atform was created to present and to help the analysis our empirical SPL tools data.

6.2 SLR of SPL tools

This section presents some previous studies about SRL of SPL tools. Analysis of existing SPL management tools has been performed in previous studies [Capilla et al., 2007; Lisboa et al., 2010; Pereira et al., 2015]. The purpose of these studies in general

terms is to facilitate tools selection in the context of SPL. First, our systematic review (Chapter 4) differs from other studies because it includes a large number of tools, characteristics, and functionalities not addressed by previous studies, totalizing 43 tools.

Capilla et al. [2007] and Lisboa et al. [2010] aim at finding out how the available tools offer support to the domain engineering process in SPL. They defined the current limits of tool support for the processes and they highlighted the need of these tools to incorporate additional functionalities. Capilla et al. [2007] analyze the current state of practice of five SPL management tools, and Lisboa et al. [2010] present an systematic literature review of nineteen SPL management tools. They involved the identification of external characteristics about the tools and of the functionalities they support. On the other hand, our systematic literature review Pereira et al. [2015] includes not only tools for domain engineering analysis, but also tools for the application engineering. Further, for this dissertation, we extended our SLR adding two new SPL tools, complementing these studies with the assessment of new tools and additional characteristics.

We complemented our systematic review with a detailed empirical study between three tools. Our empirical study involved 124 developers taking courses related to the Software Engineering area. To the best of our knowledge, we do not know any studies that perform this kind of analysis of tools by concrete experimental data.

6.3 Empirical Studies of SPL tools

A study about evaluation of some SPL management tools (XFeature, pure::variants, and RequiLine) was undertaken in collaboration with a group of industries [Djebbi et al., 2007]. The purpose of this study was twofold: to understand the salient characteristics of SPL management tools, and to evaluate the ability of existing tools to satisfy the expectations of industries. After the evaluation, pure::variant and RequiLine were the tools that best satisfied the defined criteria. On the other hand, our empirical study was conducted with three different tools (SPLOT, FeatureIDE and pure::variants). We could reveal strengths and weaknesses of each tool.

In another study [Unphon, 2008], ten variability modeling and configuration tools: AHEAD, FAMA, Feature Modeling Plug-in, Gears, Kumbang Tools, MetaEdit+, Product Modeler, pure::variants, RequiLine, and XFeature were compared. This study analyzes information (creator, first public release date/year, latest stable version, cost, and software license), technical infrastructure, operating systems support, rendering of modeling, format of input/output models support, modeling and configuration func-

tionalities, and development functionalities. Our systematic literature review complements this study with the assessment of new tools, and different characteristics and functionalities. In addition, our empirical study focuses on practical experience with three SPL tools. As in our empirical study, the purpose of this work is to facilitate tool selection in the context of SPL.

Simmonds et al. [2011] investigated relevant kinds of variability for processes and the appropriateness of different approaches for modeling variability. They make 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. The tools analyzed were: Clafer, EPF Composer, FaMa-OVM, fmp, Hydra, SPLOT, VEdit, and XFeature. The tools were evaluated based on the process they support: supported formats, underlying formalism, supported analyzes, interface, availability, and usability. However, their results focus more on the techniques than on the tool support, while our empirical study is based on experimental data.

Pereira et al. [2013] performed an 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 124 participants (no participant was the same of the previous one). That is, together, both studies involved 180 different participants. 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 124 new participants (41 for SPLOT, 42 for FeatureIDE, and 41 for pure::variants) answered another different (and more detailed) questionnaire, with additional and specific questions about SPL and feature modeling. They also performed different tasks to exercise other aspects of SPL development. As similarity, both studies aim to compare SPL tools and support engineers in the hard task of choosing the SPL tool that best fits their needs.

6.4 Concluding Remarks

In this chapter, we provided related work about information visualization in SPL context and about SPL tools. The human creative ability to transform data into meaningful information, supported by computing resources, amplifies the cognition by enabling data to be collected, store, organized, and accessed through interactive visual representations to visualize concepts and abstraction. The visualization has been widely

used in software engineering and has proven useful to amplify human cognition in data intensive application Gansner and North [2000]. However, adopting this technique in SPLE can also help stakeholders in supporting essential tasks and in enhancing their understanding of large and complex product lines.

Thereby, as shown in this chapter, the lack of publications about the use of information visualization in data from research methods in Software Engineering, specially in SPL and, the evaluation of SPL tool by practical experimental was a motivation for us to make this work happen. In the next chapter, we conclude this dissertation, discuss some limitations, and propose future research directions.

Chapter 7

Conclusion and Future Work

The process of designing an information visualization technique consists of transforming data into images that are better comprehensible by perception and cognitive system of human beings [Spence, 2014]. These visualization techniques can be used in the Software Engineering (SE) context in order to reveal and to better understand data, structures, and processes [Gotel et al., 2008]. The main goal of this dissertation was to use information visualization to explore different data types (multiple dimensional data and tree data) of our results and to provide visualization support to empirical data of SPL tools. To this end, we proposed a visual platform called Vi**SPL**atform. This platform is composed by five different visualization techniques. This set of visualization and analytical interaction techniques focused on empirical data visualization helped us to comprehend and explore data from research methods in the SPL context.

In the first visualization, we used a strip plot chart to present in the time line the SPL management tools cited by literature between 2000 and 2014. In addition, the data analysts could interact with this visualization to explore detailed information about each tool, such as name, year, brief description, developers, country(ies) involved, and reference. In second visualization, we used bubble to represent each participant of the empirical study. In this case, the data analyst could interact with the bubble to have information and background of a specific participant. data analyst also could sorted the participants by origin, gender, tool used, skill level or work experience. In the third visualization, we combined heatmap and donut chart to have an overview about the impact of participant' background in the result of tasks related to four functionalities (feature model edition, automated analysis feature model, product configuration and import/export feature model). With similar propose, we created the fourth visualization to detail this influence/impact. For that, we used a combination of bar and donut chart. In this visualization, data analyst could combined different

skill level and compare the results of each task for each tool. Finally, we used divergent stacked chart to present and explore problems faced by participants during the experiment in addition to strengths and weaknesses of each tool pointed by them.

Furthermore, this dissertation has directed contributions in the Software Product Line Engineering field. In industry, the development of large software systems requires an enormous effort for modeling and configuring multiple products. To face this problem, several tools have been proposed and used for representing and managing the variations of a system. There are many available options of variability management tools. Therefore, this dissertation aims to identify and evaluate existing tools in the literature to support the SPL management. To this end, we conducted a SLR following a guide of systematic review proposed by Kitchenham Kitchenham et al. [2009a].

Analysis of existing SPL management tools has been performed in previous studies [Beuche et al., 2004; Capilla et al., 2007; Djebbi et al., 2007; Unphon, 2008; Pereira et al., 2013]. The purpose of these studies in a general way was to facilitate tool selection in the context of SPL. However, they were aimed at studying only tools very specifics or a small group of tools. Our study differs from others by the fact of being preceded by an SLR. In addition, with regard to SLR presented by Lisboa et al. Lisboa et al. [2010], our study analyzes a relatively higher number of tools.

Our SLR identified 43 tools existing in the literature (Chapter 4). It provides an overview of tools where it is possible to see which characteristics and functionalities each tool implemented. The contribution of this research covers both the academic and the professional segments. In the academic environment, this research helps to highlight the lack of complete tools in this area, identifying gaps in current research to provide a background for new research activities. In the professional context, this research helps to find possible tools that will assist SPL developers or even help them manage existing SPL.

We intended to reveal the strengths and weaknesses of three SPL tools (SPLOT, FeatureIDE and pure::variants) in order to encourage provide insights to the improvement/extension of existing or new tools. To this end, these tools were quantitatively and qualitatively analyzed and some interesting results were presented and discussed (Chapter 5). This analysis was based on an empirical study of these three tools. The results reported in this dissertation aim to support software engineers to choose one of these tools for variability management. For instance, through the results obtained, it is clear that the little documentation available and the complexity and/or the unavailability of existing support tools are some of the reasons that may end up discouraging the adoption and wide use of these tools in organizations and academia.

Besides, when choosing one of the tools, the need and purpose of use is one of the main factors to be taken into consideration. Therefore, our conclusion after analyzing the data using Vi**SPL**atform in this dissertation showed that the main issues observed in the three SPL tools are related to their interfaces, lack of examples available, tutorials, and limited users guide; 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 three tools analyzed in Chapter 5 have advantages and drawbacks. For instance:

- SPLOT has as main advantages its automated analysis and the fact to be an online tool. As drawbacks: the interface and hotkeys.
- The main advantages of FeatureIDE are its interface and feature model editor. Its drawbacks include limited user guide and confusing product configuration functionality.
- For pure::variants, the main advantages are feature model export/import functionality and its editor. Lack of examples and interface were pointed as its main drawbacks.

We believe that this set of visualization supported by Vi**SPL**atform and the respective interaction techniques can be used in other empirical data helping to answer research question in different studies that shared similar characteristics or configuration explained and presented in this dissertation. As future work, we plan to evaluate the Vi**SPL**atform with developers in other settings. The evaluation should focus on usability and the effectiveness of the tool when used to support experiments in software engineering. In addition, exploring other visualization techniques for different empirical strategies and systematic literature review. Furthermore, as future work, our empirical study can be extended in further experiment replications. For instance, other tools different from the ones used in Chapter 5 can be analyzed and evaluated using the same (or similar) experiment design in order to contribute to improve the body of knowledge about SPL tools.

Bibliography

Asikainen, T., Männistö, T., and Soininen, T. (2004). Using a configurator for modelling and configuring software product lines based on feature models. In *Workshop on Software Variability Management for Product Derivation, Software Product Line Conference (SPLC)*, pages 24–35.

Barros, D. (2015). *UTIL: A Unified Taxonomy for Information Visualization*. PhD thesis, MsC Thesis, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais (UFMG), Brasi. Available at: `http://homepages.dcc.ufmg.br/\mathaccent"707E\relax{~}diego.barros/research/util/index.html` [Online; accessed August-2015].

Basili, V., Shull, F., and Lanubile, F. (1999). Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4):456–473.

Batory, D. (2005a). *Feature models, grammars, and propositional formulas*. Springer.

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

Batory, D. S. (2005b). Feature models, grammars and propositional formulas. In *9th International Software Product Lines Conference (SPLC)*, pages 7–20.

Bednasch, T., Endler, C., and Lang, M. (2003). Captain feature tool. Tool available on SourceForge at: `https://sourceforge.net/projects/captainfeature/` [Online; 01-April-2013].

Benavides, D., Segura, S., Trinidad, P., and Ruiz-cortés, A. (2007). Fama: Tooling a framework for the automated analysis of feature models. In *1th International Workshop on Variability Modelling of Software Intensive Systems (VAMOS)*, pages 129–134.

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

Bertin, J. (1967). Sémiologie graphique. *Paris: Mouton, 1966, 432p.*

Bertin, J. (1973). Sémiologie graphique: Les diagrammes-les réseaux-les cartes.

Beuche, D. (2003). Variant management with pure::variants. technical report, pure-systems gmbh. Available at: `http://www.pure-systems.com/`.

Beuche, D. (2012). Modeling and building software product lines with pure:: variants. In *Proceedings of the 16th International Software Product Line Conference-Volume 2*, pages 255--255. ACM.

Beuche, D., Papajewski, H., and Schröder-Preikschat, W. (2004). Variability management with feature models. *Journal Science of Computer Programming*, 53(3):333–352.

Bonifácio, R., Teixeira, L., and Borba, P. (2009). Hephaestus: A tool for managing spl variabilities. In *Brazilian Symposium on Components, Architectures and Reuse Software (SBCARS)*, pages 26–34.

Botterweck, G., Janota, M., and Schneeweiss, D. (2009). A design of a configurable feature model configurator. In *3th International Workshop on Variability Modelling of Software Intensive Systems (VaMoS)*, pages 165–168.

Botterweck, G., Thiel, S., Cawley, C., Nestor, D., and Preußner, A. (2008). Visual configuration in automotive software product lines. In *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*, pages 1070--1075. IEEE.

Braga, R., Werner, C., and Mattoso, M. (1999). Odyssey: A reuse environment based on domain models. In *IEEE Symposium on Application-Specific Systems and Software Engineering and Technology (ASSET)*, pages 50–57.

Buhne, S., Lauenroth, K., and Pohl, K. (2005). Modelling requirements variability across product lines. In *13th IEEE International Conference on Requirements Engineering*, pages 41–50.

Capilla, R., Sánchez, A., and Dueñas, J. (2007). An analysis of variability modeling and management tools for product line development. In *Software and Service Variability Management Workshop - Concepts, Models, and Tools*, pages 32–47.

Card, S. K., Mackinlay, J. D., and Shneiderman, B. (1999). *Readings in information visualization: using vision to think*. Morgan Kaufmann.

Carneiro, G. d. F. and Mendonça Neto, M. G. d. (2014). Sourceminer: Towards an extensible multi-perspective software visualization environment. In *Enterprise Information Systems*, pages 242--263. Springer.

Cawley, C., Nestor, D., Preussner, A., Botterweck, G., and Thiel, S. (2008). Interactive visualisation to support product configuration in software product lines. In *2th International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS)*, pages 7–16.

Chen, L. and Babar, M. A. (2011). A systematic review of evaluation of variability management approaches in software product lines. *Journal Information and Software Technology*, 53(4):344–362.

Cirilo, E., Kulesza, U., and Lucena, C. J. P. (2008). A product derivation tool based on model-driven techniques and annotations. *Journal of Universal Computer Science*, 14(8):1344–1367.

Clements, P. and Northrop, L. (2001). *Software product lines: Practices and patterns*. Addison-Wesley.

Clements, P., Northrop, L., et al. (1999). A framework for software product line practice. *SEI interactive*, 2(3).

Cowan, N. (2012). *Working memory capacity*. Psychology press.

Czarnecki, K. and Eisenecker, U. W. (2000). *Generative programming: Methods, tools, and applications*. Addison-Wesley.

Czarnecki, K., Helsen, S., and Eisenecker, U. (2005). Formalizing cardinality-based feature models and their specialization. *Software process: Improvement and practice*, 10(1):7--29.

Czarnecki, K. and Wasowski, A. (2007). Feature diagrams and logics: There and back again. In *11th International Software Product Lines Conference (SPLC)*, pages 23–34.

Dehlinger, J., Humphrey, M., Suvorov, L., Padmanabhan, P., and Lutz, R. (2007). Decimal and plfaultcat: From product-line requirements to product-line member software fault trees. In *29th International Conference on Software Engineering (ICSE)*, pages 49–50.

Dhungana, D., Grünbacher, P., and Rabiser, R. (2007). Decisionking: A flexible and extensible tool for integrated variability modeling. In *1st International Workshop on Variability Modelling of Software-intensive Systems*, pages 119–128.

Dhungana, D., Grünbacher, P., and Rabiser, R. (2011). The dopler meta-tool for decision-oriented variability modeling: A multiple case study. *Journal Automated Software Engineering*, 18(1):77–114.

Dhungana, D., Seichter, D., Botterweck, G., Rabiser, R., Grünbacher, P., Benavides, D., and Galindo, J. A. (2013). Integrating heterogeneous variability modeling approaches with invar. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, page 8. ACM.

Diehl, S. (2007). *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media.

Djebbi, O., Salinesi, C., and Fanmuy, G. (2007). Industry survey of product lines management tools: Requirements, qualities and open issues. In *15th IEEE International Requirements Engineering Conference (IREC)*, pages 301–306.

Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9):833--859.

Eichelberger, H., El-Sharkawy, S., Kröher, C., and Schmid, K. (2014). Easy-producer: product line development for variant-rich ecosystems. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2*, pages 133--137. ACM.

Ellis, W. D. (1999). *A source book of Gestalt psychology*, volume 2. Psychology Press.

Eriksson, M., Morast, H., Börstler, J., and Borg, K. (2005). The pluss toolkit: Extending telelogic doors and ibm-rational rose to support product line use case modeling. In *20th International Conference on Automated Software Engineering (ASE)*, pages 300–304.

Feigenspan, J., Kästner, C., Frisch, M., Dachselt, R., and Apel, S. (2010). Visual support for understanding product lines. In *2010 IEEE 18th International Conference on Program Comprehension*, pages 34--35. IEEE.

Few, S. (2009). *Now you see it: simple visualization techniques for quantitative analysis*. Analytics Press.

Few, S. (2015). What do data analysts most need from their tools? Available at: `http://www.perceptualedge.com/articles/visual_business_intelligence/what_do_data_analysts_need_from_tools.pdf`.

Figueiredo, E., Cacho, N., Sant'Anna, C., Monteiro, M., Kulesza, U., Garcia, A., Soares, S., Ferrari, F., Khan, S., Filho, F. C., and Dantas, F. (2008). Evolving software product lines with aspects: An empirical study. In *30th International Conference on Software Engineering (ICSE)*, pages 261–270.

Frakes, W. B., Prieto-Diaz, R., and Fox, C. (1997). Dare-cots: A domain analysis support tool. In *17th International Conference of the Chilean Computer Science Society*, pages 73–77.

Fujishiro, I., Ichikawa, Y., Furuhata, R., and Takeshima, Y. (2000). Gadget/iv: a taxonomic approach to semi-automatic design of information visualization applications using modular visualization environment. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, pages 77--83. IEEE.

Gaia, F. N., Ferreira, G. C. S., Figueiredo, E., and de Almeida Maia, M. (2014). A quantitative and qualitative assessment of aspectual feature modules for evolving software product lines. *Science of Computer Programming*, 96:230--253.

Gansner, E. R. and North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Software Practice and Experience*, 30(11):1203--1233.

Gotel, O. C., Marchese, F. T., and Morris, S. J. (2008). The potential for synergy between information visualization and software engineering visualization. In *Information Visualisation, 2008. IV'08. 12th International Conference*, pages 547--552. IEEE.

Groher, I. and Weinreich, R. (2013). Supporting variability management in architecture design and implementation. In *46th Hawaii International Conference on System Sciences (HICSS)*, pages 4995–5004.

Heiberger, R. M. and Robbins, N. B. (2014). Design of diverging stacked bar charts for likert scales and other applications. *Journal of Statistical Software*, 57(5):1--32.

Heidenreich, F., Kopcsek, J., and Wende, C. (2008). Featuremapper: Mapping features to models. In *30th International Conference on Software Engineering (ICSE)*, pages 943–944.

Hervieu, A., Baudry, B., and Gotlieb, A. (2011). Pacogen: Automatic generation of pairwise test configurations from feature models. In *22nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 120–129.

Jain, A. and Biesiadecki, J. (2006). Yam: A framework for rapid software development. In *Second IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pages 182–194.

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., Cohen, S., Hess, J., Novak, W., and Peterson, S. (1990). Feature-oriented domain analysis (foda) feasibility study. Available at: `http://www.sei.cmu.edu/reports/90tr021.pdf/`.

Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., and Huh, M. (1998). Form: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5(1):143–168.

Keele, S. (2007). Guidelines for performing systematic literature reviews in software engineering. In *Technical report, Ver. 2.3 EBSE Technical Report. EBSE.*

Keim, D. et al. (2002). Information visualization and visual data mining. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):1--8.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. (1997). Aspect-oriented programming. In *ECOOP'97Object-oriented programming*, pages 220--242. Springer.

Kim, K., Kim, H., Ahn, M., Seo, M., Chang, Y., and Kang, K. C. (2006). Asadal: A tool system for co-development of software and test environment based on product line engineering. In *28th international conference on Software engineering (ICSE)*, pages 783–786.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., and Linkman, S. (2009a). Systematic literature reviews in software engineering: a systematic literature review. *Information and software technology*, 51(1):7--15.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., and Linkman, S. (2009b). Systematic literature reviews in software engineering: A systematic literature review. *Journal Information and Software Technology*, 51(1):7–15.

Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K., and Rosenberg, J. (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734.

Koffka, K. (1935). Principles of gestalt psychology.

Köhler, W. (1938). Physical gestalten.

Krueger, C. (2007). Biglever software gears and the 3-tiered spl methodology. In *22th Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 844–845.

Krut, J. R. W. (1993). Integrating 001 tool support in the feature-oriented domain analysis methodology. technical report, software engineering institute (sei). Paper available at: `http://repository.cmu.edu/cgi/viewcontent.cgi?article=1166&context=sei`. [Online; 03-June-2013].

Laguna, M. and Hernández, C. (2010). A software product line approach for e-commerce systems. In *7th International Conference on e-Business Engineering (ICEBE)*, pages 230–235.

Lee, H., Yang, J.-s., and Kang, K. C. (2012). Vulcan: architecture-model-based workbench for product line engineering. In *Proceedings of the 16th International Software Product Line Conference-Volume 2*, pages 260--264. ACM.

Lima, M. (2011). *Visual Complexity: Mapping Patterns of Information*, volume 1. Princeton Architectural Press, New York, USA.

Lisboa, L. B., Garcia, V. C.and Lucrédio, D., Almeida, E. S., Meira, S. R. L., and Fortes, R. P. M. (2010). A systematic review of domain analysis tools. *Journal Information and Software Technology*, 52(1):1–13.

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

Mazo, R., Salinesi, C., and Diaz, D. (2012). Variamos: A tool for product line driven systems engineering with a constraint based approach. In *24th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 1–8.

Mendonça, M., Branco, M., and Cowan, D. (2009). S.p.l.o.t.: Software product lines online tools. In *24th Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 761–762.

Morse, E., Lewis, M., and Olsen, K. A. (2000). Evaluating visualizations: using a taxonomic guide. *International Journal of Human-Computer Studies*, 53(5):637--662.

Myllärniemi, V., Raatikainen, M., and Männistö, T. (2007). T.: Kumbang tools. In *11th Software Product Line Conference (SPLC)*, pages 135–136.

Nazemi, K., Breyer, M., and Kuijper, A. (2011). User-oriented graph visualization taxonomy: A data-oriented examination of visual features. In *Human Centered Design*, pages 576--585. Springer.

Ondrej, R. and Alessandro, P. (2005). Xfeature modeling tool. Available at: `http://www.pnp-software.com/XFeature/`. Automatic Control Laboratory, ETH Zürich [Online; accessed 01-April-2013].

Park, J., Moon, M., and Yeom, K. (2004). Dream: Domain requirement asset manager in product lines. In *International Symposium on Future Software Technology (ISFST)*.

Park, K., Ryu, D., and Baik, J. (2012). An integrated software management tool for adopting software product lines. In *11th International Conference on Computer and Information Science (ICIS)*, pages 553–558.

Pereira, J., Souza, C. G., Figueiredo, E., Abilio, R., Vale, G., and Costa, H. (2013). Software variability management: An exploratory study with two feature modeling tools. In *Brazilian Symposium on Components, Architectures and Reuse Software (SBCARS)*, pages 20–29.

Pereira, J. A., Constantino, K., and Figueiredo, E. (2015). A systematic literature review of software product line management tools. In *Software Reuse for Dynamic Systems in the Cloud and Beyond*, pages 73--89. Springer.

Pohl, K. (2003). Varmod-prime tool environment. Available at: `http://www.sse.uni-due.de/en/component/content/article/87-forschung/abgeschlossene-projekte/148-varmod-prime-tool-environment`. [Online; accessed 01-April-2013].

Pohl, K., Böckle, G., and Van der Linden, F. J. (2005). *Software product line engineering: Foundations, principles and techniques.* Springer-Verlag.

Prehofer, C. (1997). Feature-oriented programming: A fresh look at objects. In *ECOOP'97Object-Oriented Programming*, pages 419--443. Springer.

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

Robbins, N. B. and Heiberger, R. M. (2011). Plotting likert and other rating scales. In *Proceedings of the 2011 Joint Statistical Meeting*.

Roth, S. F. and Mattis, J. (1990). Data characterization for intelligent graphics presentation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 193--200. ACM.

Salazar, J. R. (2009). *Herramienta para el modelado y configuración de modelos de características*. PhD thesis, MsC Thesis, Dpto. Lenguajes y Ciencias de la Computación, Universidad de Málaga. Available at: `http://caosd.lcc.uma.es/spl/hydra/documents/PFC_JRSalazar.pdf` [Online; accessed 03-January-2014].

Schmid, K., Krennrich, K., and Eisenbarth, M. (2006). Requirements management for product lines: Extending professional tools. In *10th International Software Product Line Conference (SPLC)*, pages 113–122.

Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336--343. IEEE.

Simmonds, J., Bastarrica, M., Silvestre, L., and Quispe, A. (2011). Analyzing methodologies and tools for specifying variability in software processes. computer science department, universidad de chile, santiago, chile. Paper available at: `http://www.dcc.uchile.cl/TR/2011/TR_DCC-20111104-012.pdf`. [Online; 03-January-2014].

Sinnema, M., Deelstra, S., Nijhuis, J., and Bosch, J. (2004). Covamof: A framework for modeling variability in software product families. In *3th International Software Product Line Conference (SPLC)*, pages 197–213.

Sjøberg, D. I., Dybå, T., Anda, B. C., and Hannay, J. E. (2008). Building theories in software engineering. In *Guide to advanced empirical software engineering*, pages 312--336. Springer.

Šmite, D., Wohlin, C., Gorschek, T., and Feldt, R. (2010). Empirical evidence in global software engineering: a systematic review. *Empirical software engineering*, 15(1):91--118.

Spence, R. (2014). *Information Visualization: An Introduction*. Springer.

SPL Hall of Fame, S. (2015). Software product line hall of fame. `http://www.splc.net/fame.html`. Accessed: 2015-05-15.

Succi, G., Yip, J., and Pedrycz, W. (2001). Holmes: an intelligent system to support software product line development. In *23rd International Conference on Software Engineering (ICSE)*, pages 829–830.

Thao, C., Munson, E. V., and Nguyen, T. N. (2008). Software configuration management for product derivation in software product families. In *15th International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, pages 265–274.

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., and Leich, T. (2014). Featureide: An extensible framework for feature-oriented software development. *Journal Science of Computer Programming*, 79(0):70–85.

Thurimella, A. K. and Bruegge, B. (2012). Issue-based variability management information and software technology. *Journal Information and Software Technology*, 54(9):933–950.

Thurimella, A. K. and Janzen, D. (2011). Metadoc feature modeler: A plug-in for ibm rational doors. In *15th International Software Product Line Conference (SPLC)*, pages 313–322.

Tufte, E. R. and Graves-Morris, P. (1983). *The visual display of quantitative information*, volume 2. Graphics press Cheshire, CT.

Unphon, H. (2008). A comparison of variability modeling and configuration tools for product line architecture. Paper available at: `http://www.itu.dk/people/unphon/technical_notes/CVC_v2008-06-30.pdf`. [Online; accessed 15-January-2015].

Vale, T., Cabral, B., Alvim, L., Soares, L., Santos, A., Machado, I., Souza, I., Freitas, I., and Almeida, E. (2014). Splice: A lightweight software product line development process for small and medium size projects. In *Software Components, Architectures and Reuse (SBCARS), 2014 Eighth Brazilian Symposium on*, pages 42--52. IEEE.

Valente, M. T., Borges, V., and Passos, L. (2012). A semi-automatic approach for extracting software product lines. *Software Engineering, IEEE Transactions on*, 38(4):737--754.

Valiati, E. R., Pimenta, M. S., and Freitas, C. M. (2006). A taxonomy of tasks for guiding the evaluation of multidimensional visualizations. In *Proceedings of the 2006*

AVI workshop on Beyond time and errors: novel evaluation methods for information visualization, pages 1--6. ACM.

Varela, P., Araujo, J., Brito, I., and Moreira, A. (2011). Aspect-oriented analysis for software product lines requirements engineering. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 667--674. ACM.

Ware, C. (2005). Visual queries: The foundation of visual thinking. In *Knowledge and information visualization*, pages 27--35. Springer.

Ware, C. (2012). *Information visualization: perception for design*. Elsevier.

Wertheimer, M. (1938). Gestalt theory.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 38. ACM.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012a). *Experimentation in software engineering*. Springer Science & Business Media.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012b). *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers.

Zhou, M. X. and Feiner, S. K. (1996). Data characterization for automatically visualizing heterogeneous information. In *Information Visualization'96, Proceedings IEEE Symposium on*, pages 13--20. IEEE.