Personalized Recommender Systems for Software Product Line Configurations

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Dipl. Inf. Juliana Arriel (Geburtsname: Juliana Alves Pereira)

geb. am 22.08.1989 in Perdões, Brazil

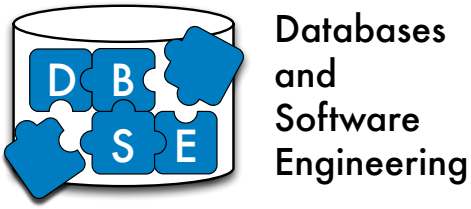
Gutachterinnen/Gutachter

Prof. Dr. Gunter Saake
Prof. Dr. Myra Spiliopoulou
Prof. Dr. Eduardo Figueiredo

Magdeburg, den 25.06.2018

University of Magdeburg

School of Computer Science



Dissertation

Personalized Recommender Systems for Software Product Line Configurations

Author:

Juliana Arriel

June 25, 2018

Reviewers:

Prof. Dr. Gunter Saake

University of Magdeburg, Germany

Prof. Dr. Myra Spiliopoulou

University of Magdeburg, Germany

Prof. Dr. Eduardo Figueiredo

Federal University of Minas Gerais, Brazil

Arriel, Juliana:
Personalized Recommender Systems for Software Product Line Configurations
Dissertation, University of Magdeburg, 2018.

Abstract

Software Product Lines (SPLs) have been employed in the industry as a mass customization process that reduces production costs and time-to-market. However, the inherent complexity and variability of SPLs lead to an exponentially growing amount of possible products. Thus, especially when dealing with large SPLs, scalability and performance concerns start to be an issue and specialized assistance becomes crucial to guide decision makers during product configuration. In this context, SPL configuration has been a hot research topic in the last years. In this thesis, we provide an overview on SPL configuration techniques and, based on our insights, employ recommendation techniques to enable an efficient SPL configuration process and provide accurate and scalable solutions to decision makers. To this end, we offer four main contributions. First, we adapt state-of-the-art collaborative-based recommender algorithms to the SPL configuration context. Second, we consider non-functional properties to improve the efficiency and quality of the recommendations. Third, we propose an advanced recommender system that relies also on contextual information to enable reconfiguration at runtime. Fourth, we provide visual support to guide decision makers through an easy and comprehensive configuration process by allowing them to focus on a limited set of valid and relevant parts of the configuration space. We empirically demonstrate the usability of the implemented algorithms and tool in different real-world scenarios from different domains.

Zusammenfassung

Software-Produktlinien (SPLs) wurden in der Industrie zur Massenproduktion individualisierter Produkte eingesetzt, um Produktionskosten und Time-to-Market zu reduzieren. Die inhärente Komplexität und Variabilität von SPLs führt jedoch zu einer exponentiell anwachsenden Menge an möglichen Produkten. Gerade bei großen SPLs sind daher Skalierbarkeit und Performance ein Thema, und spezialisierte Unterstützung ist entscheidend, um Entscheidungsträger bei der Produktkonfiguration zu unterstützen. In diesem Zusammenhang war die Konfiguration von SPLs in den letzten Jahren ein heißes Forschungsthema. In dieser Arbeit wird einen Überblick über die SPL-Konfigurationstechniken gegeben und auf Basis der Erkenntnisse mit Hilfe von Recommendersystemen ein effizienter SPL-Konfigurationsprozess ermöglicht, um Entscheidungsträgern genaue und skalierbare Lösungen anzubieten. Die Arbeit umfasst dabei vier wesentliche wissenschaftliche Beiträge. Erstens, es werden moderne kollaborative Empfehlungsalgorithmen an den SPL-Konfigurationskontext angepasst. Zweitens, es werden nicht-funktionale Eigenschaften betrachtet, um die Effizienz und Qualität der Empfehlungen zu verbessern. Drittens, es wird ein erweitertes Empfehlungssystem vorgeschlagen, das sich auch auf Kontextinformationen stützt, um Rekonfiguration zur Laufzeit zu ermöglichen. Viertens, es werden visuelle Unterstützung eingeführt, um Entscheidungsträger durch einen einfachen und umfassenden Konfigurationsprozess zu führen, indem ihnen erlaubt wird, sich auf eine begrenzte Anzahl von validen und relevanten Teilen des Konfigurationsraums zu konzentrieren. Wir demonstrieren empirisch die Anwendbarkeit der implementierten Algorithmen und Werkzeuge in verschiedenen realen Szenarien.

Acknowledgements

I would like to thank Gunter Saake and Myra Spiliopoulou for giving me the opportunity to pursue my Ph.D. under their supervision. Gunter provided me an excellent research environment and gave me the freedom to follow my own research direction. Even in busy times, he has always supported me unconditionally. Besides Gunter, Myra has been a perfect supervisor for my thesis. Myra's guidance resulted in the first paper for this thesis and their substantial feedback from her vast knowledge in recommender systems broadened my view on the Ph.D. topic. Since then, Myra has been a constant source of support and I highly value her feedback.

I would also like to thank Eduardo Figueiredo, who has guided me since my Master's thesis. I am extremely grateful for his collaboration and constructive feedback. He provided an incredibly in-depth review of the thesis. Evaluating a doctoral thesis is an arduous task, and I am very appreciative for his feedback.

Special thanks to Pawel Matuszyk, Sandro Schulze and Sebastian Krieter for their guidance and numerous fruitful discussions. During the last years, we had many brainstorming sessions that had a major impact on my research. Furthermore, I would like to thank many colleagues and students for their collaboration, discussions, and support regarding different aspects of this thesis. Special thanks to Ebrahim Bagheri, Jabier Martinez and Lina Ochoa. Also, I gratefully acknowledge the financial support of the *Brazilian National Council for Scientific and Technological Development* (CNPq) for the Ph.D. grant.

Finally, I would like to express my deep gratitude to my advisor Gunter Saake for giving me the opportunity to join the workgroup Databases and Software Engineering in Magdeburg. Coming to Magdeburg and being a Ph.D. student at his group was an incredible experience. I feel honored to have been a part of this workgroup. I have always felt welcome here. Thanks to everybody in the team. I am also thankful to the Institute of Computer Science (FIN) for facilitating my research studies by providing me a nice working space.

I couldn't have done this without all of you. Thank you!

Contents

List of Figures

List of Tables

Code Listings

1. Introduction

Today's competitive software market requires the industry to understand unique and particular needs of their customers. *Software Product Line* (SPL) has proven to be an efficient and effective strategy for mass customization by exploiting large-scale reuse. However, although customization has been extensively studied over the past decades, it remains a source of concerns. In the context of SPLs, additional constraints between features and non-functional properties (emerging from large-scale variability spaces) add one more layer of complexity to the customization process, *a.k.a.* configuration process. In this circumstances, to assist users in dealing with the increased complexity, an easy and comprehensive configuration process becomes crucial. Previous researchers have proposed several *interactive*, *semi-automatic*, and *automatic* SPL configuration approaches. Despite their maturity, the current configuration process is still not able to fully use the power of customization when dealing with variability of highly configurable SPLs.

With an *interactive* SPL configuration tool (known as configurator), users configure personalized products by consecutively selecting desired features based on their individual needs and SPL constraints. In this context, there are many approaches[1] that aim to guide users into a valid configuration, ensuring that any partially configured product is in accordance with the SPL constraints. However, when using those approaches, features of no importance to the stakeholders also need to be taken into account [Pereira et al., 2016b]. On this scenario, as most features are interdependent, users must understand the impact of their gradual selections in order to make valid decisions. However, the amount and complexity of options presented by *interactive* approaches may exceed the capability of a user to identify an appropriate configuration. Consequently, this may lead to delays due to users' exploration of choices at each step of the configuration process. Thus, especially when dealing with large SPLs with complex dependencies, it is a tedious task to find a valid feature combination as product configuration. Therefore, additional support is needed to guide the users through the configuration process and allow them to focus on valid and relevant parts of the configuration space.

[1] For a survey on these approaches, we refer to Pereira et al. [2015].

For the *semi-automatic* SPL configuration scenario, Galindo et al. [2015a] proposed a dynamic decision model with a set of questions and a defined set of possible answers. In a similar scenario, other authors [Asadi et al., 2014, Bagheri and Ensan, 2014a, Tan et al., 2014a] proposed a pair-wise based decision approach where users are constantly asked to compare a pair of features and identify their relevance in terms of satisfying given non-functional requirements. However, while question-based decisions may introduce some vague descriptions and even misleading information to questionnaires, pair-wise based decisions may introduce inconsistencies in the feature ranking. Moreover, if the features, or even the set of answers, are of equal (or no) interest to the user, no support is provided to guide the selection process. Furthermore, as one feature may contribute to many non-functional requirements, the amount and complexity of information presented to users can be overwhelming, and thus, may impede an appropriate choice.

Finally, for the *automatic* SPL configuration scenario, approaches[2] have used constraint programming and evolutionary algorithms to automatically derive a configuration in a single step that is in accordance with users product requirements. Although constraint programming approaches guarantee the optimality of the generated configuration, due to the NP-hard computational complexity of finding an optimal variant, exact approaches have inefficient exponential time. On this scenario, evolutionary algorithms have been deeper studied in order to manage large configuration spaces, deriving near optimal solutions in an efficient polynomial time. However, when using such approaches, the specification of multiple requirements may lead to conflicting solutions. For example, a mobile-phone system often achieves a high security only by raising its cost. Therefore, as these techniques provide a set of feasible solutions, users may not know which one would be the better choice. In this context, the current approaches neither guide the users in choosing a suitable solution nor offer further support for specification of stakeholders' preferences. Finally, improvements related to scalability and performance are still needed.

Based on the identified limitations, the envisioned contribution of my doctoral research is therefore to fill the gaps in the literature by proposing a more efficient configuration process for highly configurable SPLs. To achieve this goal, we propose for the first time a technique that adopts a collaborative-based recommender system that relies on past configurations from previous users to generate personalized recommendations for a current user. Moreover, this system provides visual support that allows users to focus on a reduced amount of information from valid and relevant parts of the configuration space. To this end, we aim at answering the following five research questions:

- *RQ1.* How do the current studies support the SPL configuration processes with specific attention on feature-model configuration?
- *RQ2.* How to use collaborative-based recommendation techniques to predict a suitable set of features from an SPL based on explicit information from users?
- *RQ3.* How can we handle the user's context as part of a personalized collaborative-based recommender system?

[2]For a survey on these approaches, we refer to Ochoa et al. [2018, 2017].

- *RQ4.* How can we handle the self-configuration of dynamic SPLs at runtime by using a collaborative-based recommender system?
- *RQ5.* How can a state-of-the-art SPL configurator be integrated with the previously proposed recommender systems?

To address RQ1–5, this thesis presents five major contributions:

- *RQ1:* In Chapter 3, we conduct a systematic literature review on SPL configuration and classify a corpus of 157 articles. Based on our insights, we define a set of activities and mechanisms related to the SPL configuration process. Moreover, we analyze open issues and missing support in SPL configuration to infer a research agenda to guide future research in this field.
- *RQ2:* In Chapter 4, we propose a preliminary adaptation of six state-of-the-art recommender algorithms to the SPL configuration context. We empirically demonstrate the accuracy results of the implemented algorithms in different domain scenarios, based on two real-world datasets of configurations. The results of our evaluation show that recommender algorithms, such as CF-shrinkage [Bell et al., 2007], CF-significance weighting [Herlocker et al., 1999], and BRISMF [Takács et al., 2009], can efficiently support the SPL configuration process.
- *RQ3:* In Chapter 5, we propose a context-aware recommender system for predicting feature selections in an extended SPL configuration scenario, *i.e.* taking non-functional properties of features into consideration. We present an empirical evaluation based on a large real-world dataset of configurations derived from industrial experience in the *Enterprise Resource Planning* domain. Our results indicate significant improvements in the predictive accuracy of our context-aware recommendation approach over the binary-based approach presented in Chapter 4.
- *RQ4:* In Chapter 6, we propose a tensor-based recommender [Karatzoglou et al., 2010] that allows an integration of contextual data by modeling an N-dimensional tensor User-Feature-Context instead of the traditional two-dimensional User-Feature matrix. In the proposed approach, different types of non-functional properties are considered as additional contextual dimensions. Moreover, we show how our approach can support the self-configuration of dynamic SPLs at runtime. We evaluate our approach by means of an empirical study using two datasets of configurations derived from medium-sized product lines. Our results reveal significant improvements in the predictive accuracy of the configuration over the non-contextual matrix factorization approach presented in Chapter 4 and the reduction-based approaches presented in Chapter 5.
- *RQ5:* In Chapter 7, we provide an interactive open-source configurator, called PROFilE. PROFilE is an Eclipse plug-in that provides several visualization mechanisms which encompass a *recommender system* that aids a user in the configuration process. It narrows the configuration space of possible features down to the permitted features and highlights selected features needed to finish a configuration. The remaining features are then scored with respect to their relevance

by our recommender system. Consequently, a user is provided with a limited set of permitted, necessary and relevant choices. We evaluate the usability of the proposed configurator from the user's point of view. We show that the use of a recommender system: *(i)* makes the configuration process easier, *(ii)* enhances the desirability of the end product, and *(iii)* reduces considerably the user's mental burden to a more manageable level. Overall, the proposed configuration environment compare favorably with existing ones in terms of usability, providing to the user a most efficient SPL configuration process.

Besides these five main chapters, we give a brief introduction to SPLs and collaborative-based recommender systems in Chapter 2. Finally, we conclude our thesis and summarizes opportunities for future research in Chapter 8.

2. Background

In this chapter, we present all the required information to make this thesis self-contained. Initially, we introduce *software product line engineering* with a brief overview of the domain and application engineering phases. Furthermore, we present the basic concepts about *collaborative-based recommender systems* and a brief overview of the algorithms used in this work.

2.1 Software Product Line Engineering

Software product line engineering is a paradigm within software engineering, used to define and derive sets of similar products from reusable assets [Kang et al., 2002]. The development life-cycle of software product line engineering encompasses two main phases: *domain engineering* and *application engineering* [Pohl et al., 2005]. While domain engineering focuses on establishing a reuse platform, application engineering is concerned with the effective reuse of assets across multiple products.

2.1.1 Domain Engineering

Variable software systems are essential to fulfill the individual requirements of several users. Such systems, known as *Software Product Line* (SPL), are commonly based on reusable but interdependent artifacts represented by a set of features that can be combined to form customized products [Pohl et al., 2005]. The *domain engineering* phase is responsible for capturing and documenting reusable assets from SPLs through variability models. Feature models are amongst the most widely used domain engineering variability models [Czarnecki et al., 2000]. They provide a standard notation to represent and manage the interdependencies among reusable common and variable assets of an SPL, called *features* [Lee et al., 2002]. The term feature model was proposed by Kang et al. in 1990 as a part of the *Feature-Oriented Domain Analysis* (FODA) method. FODA is a graphical representation of commonalities and variability of systems [Kang et al., 1990]. Over the past years, several variability modeling techniques have been developed in order to document and manage reusable software artifacts [Chen and Babar, 2011, Vale et al., 2016]. Since then,

feature models have been applied in a number of domains, including network protocols [Barbeau and Bordeleau, 2002], smart houses [Cetina et al., 2009], mobile phones [Czarnecki et al., 2012], telecom systems [Griss et al., 1998], the Linux kernel [Lotufo et al., 2010], and others.

A common visual representation for a feature model is a *feature diagram* [Kang et al., 1990]. A feature diagram is a tree-based structure, where each node represents a feature, and different edges illustrate the dependencies between two or more features [Kang et al., 1998]. As an example, consider a simplified *smart-home* feature model presented as a feature diagram in Figure 2.1 encompassing several functional features (*e.g.*, an `alarm` can be `silent`, `visual` or `siren`) and *non-functional properties* (NFPs) (*e.g.*, the `cost` of the features). The model represents a building equipped with a set of electrical sensors and actuators, to allow for intelligent sensing and controlling of building devices. Features are depicted as boxes (*i.e.*, nodes) with their feature name inside and interdependencies between features are shown as links (*i.e.*, edges). The feature diagram defines common features found in all products of the product line, known as *mandatory* features, such as `illumination`, and variable features that allow the distinction between products in the product line, referred to as *optional* and *alternative* features, such as `security` and the group `sensor`, respectively. Moreover, features can be classified as *primitive features* (*i.e.*, atomic features) which are features without any children situated at the leaf level of the feature model; otherwise they are classified as *compound features* (*i.e.*, non-atomic features) which are features decomposed into sub-features [Benavides et al., 2005]. As an example, the smart-home feature model presents sixteen primitive features and eleven compound features. Note that a child feature can only appear in a product configuration if its parent feature is selected.

In addition to regular edges, feature models often contain *additional composition rules* [Czarnecki et al., 2000]. Additional composition rules, also known as *cross-tree constraints* (CTCs), add further feature interdependencies, thus, restricting the selection of non-directly connected optional features [Pohl et al., 2005]. Benavides et al. [2010] classify cross-tree constraints into two forms:

- The inclusion of a set of features `a` in a configuration *requires* the inclusion of a set of features `b` (*i.e.*, $\texttt{a} \longrightarrow \texttt{b}$).
- The inclusion of a set of features `a` in a configuration *excludes* a set of features `b` (*i.e.*, $\texttt{a} \longleftrightarrow \texttt{b}$).

Moreover, `a` and `b` are usually written as logical expressions formed by the binary operators $\wedge$ (*and*), $\vee$ (*or*), and the unary operator $\neg$ (*negation*), and the *variables* that represent the features. As an example, the *requires* cross-tree constraint $\texttt{monitoring} \rightarrow \texttt{camera} \wedge \texttt{audio}$ in Figure 2.1 ensures that product configurations containing the `monitoring` feature must contain both features `camera` and `audio`.

Furthermore, dashed boxes extend feature models with extra information about features (*i.e.*, NFPs). This type of models where system's NFPs are included is called *Extended Feature Model* (EFM) [Benavides et al., 2010]. EFMs define mandatory and optional features, as well as their NFPs and relationships [Kang et al., 1990].

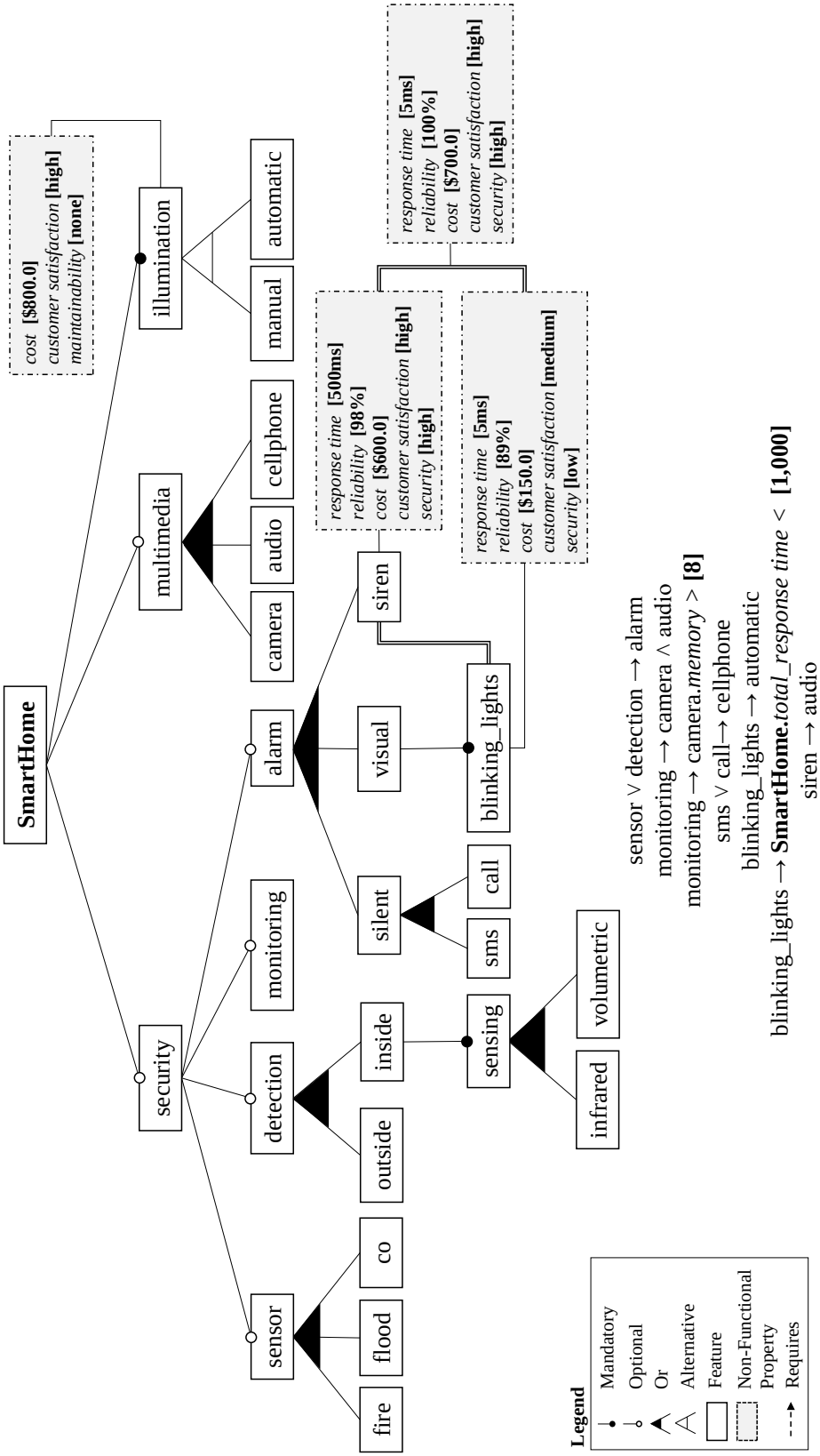


Figure 2.1: A simplified extended feature model for a *smart-home* product line (adapted from Cetina et al. [2009]).

In EFMs, NFPs can be either quantitative or qualitative, and multiple NFPs can be associated to the same feature. Quantitative NFPs are properties represented as a numeric value (*e.g.*, `response time` and `cost`), otherwise qualitative NFPs are represented using an ordinal scale, such as low, medium, and high (*e.g.*, `maintainability` and `security`). Moreover, EFMs can also include *hard cross-tree constraints*. Hard cross-tree constraints constitute features and NFP dependencies. They are represented using operators (*i.e.*, $\wedge$, $\vee$, $\neg$, $\rightarrow$, $<$, $>$, and $=$) and arithmetical expressions (*i.e.*, $+$, $-$, $*$, $/$, and $\%$) in combination with NFPs or contextual information. All these operators and expressions can be nested arbitrarily. As an example, the hard cross-tree constraint `monitoring` $\rightarrow$ `(camera.memory > 8)` is depicted for the smart-house feature model in Figure 2.1. It expresses that all houses with security monitoring must have a camera installed with at least 8GB of memory. Additionally, it is also possible to have hard cross-tree constraints only representing NFP dependencies.

EFMs ensure that a derived product variant forms a valid configuration. Once specified, (extended) feature models can be used for communicating with stakeholders and for configuring a product.

2.1.2 Application Engineering

The set of features and dependencies defined by a feature model in the domain engineering phase specifies the complete set of valid feature combinations of the SPL. Each combination refers to a different product instance, which is derived as part of the application engineering phase.

The application engineering phase is responsible for capturing the requirements of a product, defining a valid combination of features that fulfill these requirements, and deriving a product configuration from this selection of features [Pohl et al., 2005]. *Product configuration* is a decision-making process that involves selecting a concrete combination of features from a feature model. A *concrete configuration* defines a configuration that complies with the feature model constraints and covers the product requirements as much as possible. Interdependencies between features and CTCs in the model define feature model constraints (*i.e.*, how features can be combined to obtain a valid configuration). For example, a smart-home configuration cannot have both `manual` and `automatic illumination` (Figure 2.1). In addition, product requirements are calculated by aggregating the NFPs of all selected features by considering feature interactions. Interactions occur when features share a common component or require additional component(s), affecting NFP values [Siegmund et al., 2012b]. An example is shown in Figure 2.1. Configurations that include both `siren` and `blinking_lights` features have a global positive impact of 100% on `reliability`. Although none of them have a 100% reliability, the unexpected result is caused by the positive interaction of both features (*i.e.*, if the siren fails, the smart home can make the home lights blink as a replacement for the failed alarm).

In Chapter 3 two types of product requirements are identified: *product constraints* and *stakeholders' preferences*. *Product constraints* are decision rules with regards to product limitations, such as *budget=$550.00*. They complement the interdependencies expressed through the EFM, restricting the set of valid configurations. In

addition, *stakeholders' preferences* allow the specification of the relative importance of NFPs, such as minimize `cost` and `response time`, and maximize `security`. Thus, among all valid product configurations, stakeholders desire the optimal one that can meet multiple product constraints and stakeholder preferences.

Configuration processes are often incremental such that a user starts with an empty configuration and selects or deselects one feature at a time until the configuration is concrete. Due to the combinatorial possibilities of selecting variable features, the number of possible configurations grows exponentially with the number of features in an SPL. Consequently, the selection task of an optimal configuration requires the exploration of a huge search space of valid configurations to find the optimal ones that balance multiple product requirements simultaneously.

Although the configuration of a valid product arising from EFMs may reduce the configuration space through product requirement specifications, selecting the most appropriate set of features is still an overwhelming task due to the diversity of application scenarios and requirements [Zanardini et al., 2016]. In addition, EFMs tend to be inherently large and complex, with several types of variability relations, and constraints among features and NFPs. Furthermore, a wide variety of feature combinations may meet the product requirements. Therefore, even configuring an EFM with such a (relatively low) number of features is not a simple task.

Configurators usually support users by representing the feature model structure in an outlined tree hierarchy form [Bashroush et al., 2017, Pereira et al., 2015, 2016a] and the features with check boxes, where decision makers manually check the (un)required features in order to configure a product. Nevertheless, considering the large configuration space for even SPLs with a relatively low number of features, finding the best configuration for a given set of product requirements can still be a challenging task. According to Hubaux [2014], the current configurators are still not able to efficiently guide the interactive configuration process of large-scale SPLs due to the high information workload and the complexity of decision making. In this context, the interactive configuration process becomes a time-consuming, error-prone, and tedious task. A system that narrows the possible choices down to the relevant ones is indispensable. Therefore, our focus in this thesis is to ease the application engineering phase by employing techniques from recommender systems to support the product configuration processes. Recommender systems use learning algorithms to automatically find relevant features from a large set of options in a personalized way. In the next section, we introduce recommender systems.

2.2 Collaborative-Based Recommender Systems

Today's rapidly changing and technology-intense market environment has led to the development of recommender systems, from companies like Amazon, YouTube, Netflix, and eBay. A recommender system is a personalized information filtering technique used to suggest a set of n items that will be most likely of interest to a particular user [Ricci et al., 2011]. The suggestions provided are aimed at supporting the user in various decision-making processes, such as the SPL configuration process. Their goal is to alleviate the problem of information overload that occurs also in SPL

configuration, where the number of possible features and configurations is too high for a single user to handle.

A recommender system addresses this process by using information filtering algorithms to predict whether a particular user will like a particular feature. Our main motivation in applying recommender mechanisms in the SPL configuration domain is to better understand users' preferences, to increase their satisfaction with the configured products, and to improve the efficiency and quality of the configuration process. Therefore, recommender systems aim to alleviate the problem of information overload in SPL configuration. The most common classes of such algorithms are *content-based recommender systems* and *collaborative filtering algorithms*.

Content-based recommender systems analyze the content of items (*e.g.*, text of a book or audio file of a music piece). Then, given the history of a user, they search for items similar to the ones the user purchased before. For more information on those algorithms, we refer to the extensive survey by Lops et al. [2011]. This type of algorithms is applicable only when the content of items can be analyzed efficiently. In our application scenario, features are the items to be analyzed. Since features often do not have a detailed representation (*e.g.*, text descriptions) and can not be easily analyzed by automated processes, their analysis is not efficiently possible. While there are recommender systems that make use of additional content-related information, such as ontologies, this thesis focuses on a general case, where no existence of ontologies is required. Therefore, the class of content-based algorithms is not applicable to our scenario.

In contrast to content-based algorithms, *Collaborative Filtering* (CF) algorithms do not analyze the content of items. Instead, they build a model based on relevance feedback from users past behavior to predict future items of interest in a personalized way. The feedback has the form of ratings (*e.g.*, a 5-star rating for relevant items and a 1-star rating for non-relevant items) that are stored in a user-item-rating matrix. In this thesis, we focus on this class of algorithms.

CF algorithms can be divided into two categories: *neighbourhood-based* and *factor-based* algorithms. Since there is no single recommendation algorithm that performs best in all applications, in this thesis, we adapt four neighbourhood-based recommendation algorithms (*i.e.*, *Neighbourhood-Based CF*, *CF Significance Weighting*, *CF Shrinkage*, and *CF Hoeffding*) and two factor-based recommendation algorithms (*i.e.*, *Matrix Factorization* and *Tensor Factorization*) to investigate which of them performs better in the domain of SPL configuration. Next, we present a brief description of these algorithms.

Neighbourhood-Based Collaborative Filtering (kNN-CF)

The first of them is the *k-Nearest Neighbor Collaborative Filtering* (kNN-CF). Neighbourhood-based CF algorithms search for neighbours of the active user in the user-item-rating matrix. The concept of a neighborhood implies that we need to discover the k nearest neighbours (therefore, kNN-CF) of the active user in order to make recommendations [Desrosiers and Karypis, 2011]. To find the nearest neighbors, such neighborhood-based CF algorithms commonly make use of similarity metrics (*e.g.*, cosine similarity).

After determining the neighbourhood of the active user, predictions of ratings for items unknown to the active user are made by calculating weighted mean rating of all neighbours. Subsequently the top-n items with the highest predicted ratings are recommended. We adapt this algorithm to recommending features in SPL configuration. In our case, the user-item-rating matrix is a matrix with the state of previous configurations (*i.e.*, 1 if a feature was selected in a configuration, 0 if a feature was deselected, and -1 otherwise). A formalization of this algorithm and our adaptation of it to the SPL configuration scenario are shown in Section 4.2.3.1. For further details on kNN-CF, we refer to Desrosiers and Karypis [2011], Herlocker et al. [1999], Bell et al. [2007], and Matuszyk and Spiliopoulou [2014].

Collaborative Filtering Significance Weighting (CF-sig.)

In contrast to kNN-CF, the CF-significance weighting algorithm uses the principle that decisions made based on configurations with a small set of selected features are not representative and the amount of trust in these configurations should be limited. In this context, the CF-significance weighting algorithm builds a user's neighborhood by assigning lower weights to configurations that have too few selected features in common. The goal of this algorithms is to weight neighbors based on how likely they are to provide an accurate prediction. Thus, the fewer similar selected features between configurations exist, the less influence does the particular similarity value have on the predicted rating value (see Section 4.2.3.2 for details and formal definition). For further details on this algorithm, we refer to Herlocker et al. [1999].

Collaborative Filtering Shrinkage (CF-Shrinkage)

Similar to the CF-significance weighting algorithm, the CF-Shrinkage algorithm assumes that similarity based on many similar selected features is more informative. In this context, the CF-Shrinkage algorithm shrinks the similarities towards a null-value to an extent that is inversely proportional to the number of selected features. As the CF-significance weighting algorithm, the fewer similar selected features between configurations exist, the less influence does the particular similarity value have on the predicted rating value (see Section 4.2.3.3 for details and formal definition). For further details on this algorithm, we refer to Bell et al. [2007].

Collaborative Filtering Hoeffding (CF-Hoeffding)

In contrast to CF-significance weighting and CF-Shrinkage, the assumption of the CF-Hoeffding algorithm is that decisions made on a naive computation of configurations' similarity are unreliable. In this context, CF-Hoeffding builds a user's neighborhood by selecting only historical configurations that are *reliably similar* to the current partial configuration, independently of the number of selected features. *Reliable configurations* are computed through a significance-based solution derived from Hoeffding's Inequality [Hoeffding, 1963] to compare whether a given historical configuration is more similar to the current partial configuration than a default configuration (see Section 4.2.3.4 for details and formal definition). For a complete definition of this algorithm, we refer to Matuszyk and Spiliopoulou [2014].

Matrix Factorization (MF)

The second class of CF algorithms is factor-based algorithms. The first of them is the *Matrix Factorization* (MF) algorithm. MF algorithms decompose the user-item-rating matrix approximately into a product of two other latent matrices [Koren et al., 2009]. The decomposition is done by minimizing an error function using *e.g.*, stochastic gradient descent. The latent matrices are used then to make rating predictions of unknown items. MF algorithms have shown their great predictive power in numerous studies [Koren, 2009, Koren et al., 2009, Takács et al., 2009]. Especially, the work by Koren et al. [2009] has coined their application in recommender systems. Nowadays, MF is considered state-of-the-art in recommender systems. Therefore, in our work, we use a representative of this class of algorithms, the BRISMF (*Biased Regularized Incremental Simultaneous Matrix Factorization*) method by Takács et al. [2009], and we adapt it to our scenario of SPL configuration (see Section 4.2.3.6 for details and formal definition).

Tensor Factorization (TF)

Tensor Factorization (TF) is an N-dimensional extension of MF. While MF algorithms assume homogeneity of context, in TF additional contextual variables come into play, *e.g.*, in the SPL configuration scenario, products are personalized based on contextual data from features and previous users. In particular, both methods have proven to be accurate CF techniques in the literature of recommender systems [Karatzoglou et al., 2010, Koren and Bell, 2015]. However, the use of TF can provide recommendations based on multiple dimensions that go beyond the typical two dimensions (*i.e.*, users and features) used in MF [Karatzoglou et al., 2010] (see Section 6.2). TF takes advantage of most of the benefits of MF, such as fast prediction computations as well as simple and efficient optimization techniques. For a complete definition of this algorithm, we refer to Karatzoglou et al. [2010].

3. Current Research on Software Product Line Configuration

We presented initial surveys on product configuration approaches, software product line management tools, and feature model notations at VaMoS'17 [Ochoa et al., 2017], ICSR'15 [Pereira et al., 2015], and SCCC'16 [Vale et al., 2016], respectively. Moreover, we presented a systematic literature review on the semi-automatic configuration of extended product lines in the Journal of Systems and Software [Ochoa et al., 2018]. In this chapter, we present a complete systematic literature review on current research in Software Product Line Configuration.

According to Benavides et al. [2013], the *Software Product Line* (SPL) configuration field is an active area of research and has attracted both practitioners and researchers in the last years. Since the introduction of feature models in the 90's by Kang et al., a large number of publications have provided new approaches, techniques, tools, and algorithms to support decision makers in the SPL configuration process. However, these publications focus on very specific contributions and there is still no clear definition regarding the mechanisms required in this phase and which existing studies provide support for each mechanism. Thus, the literature lacks a systematic review to analyze the currently existing approaches and present a complete overview of the progress made in this domain. In this context, the objectives of this chapter are to summarize the current research trends in SPL configuration and increase the understanding of the fundamental research issues in this field. We address these objectives by providing the following four contributions:

1. We define the relation between *product derivation* and *product configuration*, through the categorization of the main activities supported by the *application engineering* phase.

2. We perform a structured *Systematic Literature Review* (SLR) by following Kitchenham and Charters [2007] guidelines. This review aims at identifying,

classifying, and assessing available research techniques in the SPL configuration domain.

3. We present a formal categorization of SPL *configuration mechanisms* addressed in the literature and how each work copes with the defined mechanisms.

4. We identify a set of *open issues* in SPL configuration to guide future research in this field.

To this end, we assessed 157 primary studies to answer the following four research questions:

- *RQ1.* What are the main SPL configuration activities?
- *RQ2.* Which mechanisms have been formally proposed in the literature to support the SPL configuration activities?
- *RQ3.* How are these mechanisms addressed in the literature?
- *RQ4.* What are the main limitations faced by the current approaches and open challenges that need attention in the future?

As a result of our SLR and corresponding research questions, we identified 112 different approaches, 5 product configuration activities, and 17 mechanisms. Mainly, we give an in-depth view of the techniques used by each mechanism, conducted evaluation process, tool support, and their main shortcomings.

This chapter is structured as follows. We describe our formal definition for *product derivation* and *product configuration* in Section 3.1. Section 3.2 describes the research methodology used to conduct the SLR. Section 3.3 and Section 3.4 present a set of configuration *activities* and *mechanisms* regarded as essential for SPL configuration. Moreover, Section 3.4 analyzes how the identified SPL configuration approaches support the defined mechanisms. In addition, we show examples to demonstrate how these mechanisms can be realized. Section 3.5 discusses the main findings and challenges to be faced in the future. Section 3.6 presents threats to the validity of this study. Section 3.7 discusses related work and compares our work with them. Finally, Section 3.8 summarizes the chapter.

3.1 Preliminaries

Application engineering is an interactive process that consists of eliciting product requirements and binding variability. In Figure 3.1, we identify and categorize a sequence of key activities supported by the application engineering phase. The application engineering phase consists of two main processes, namely *product derivation* and *product configuration*. In the literature, these terms are not clearly defined and they are easily mixed. In this section, we define *product configuration* as a subset of *product derivation*.

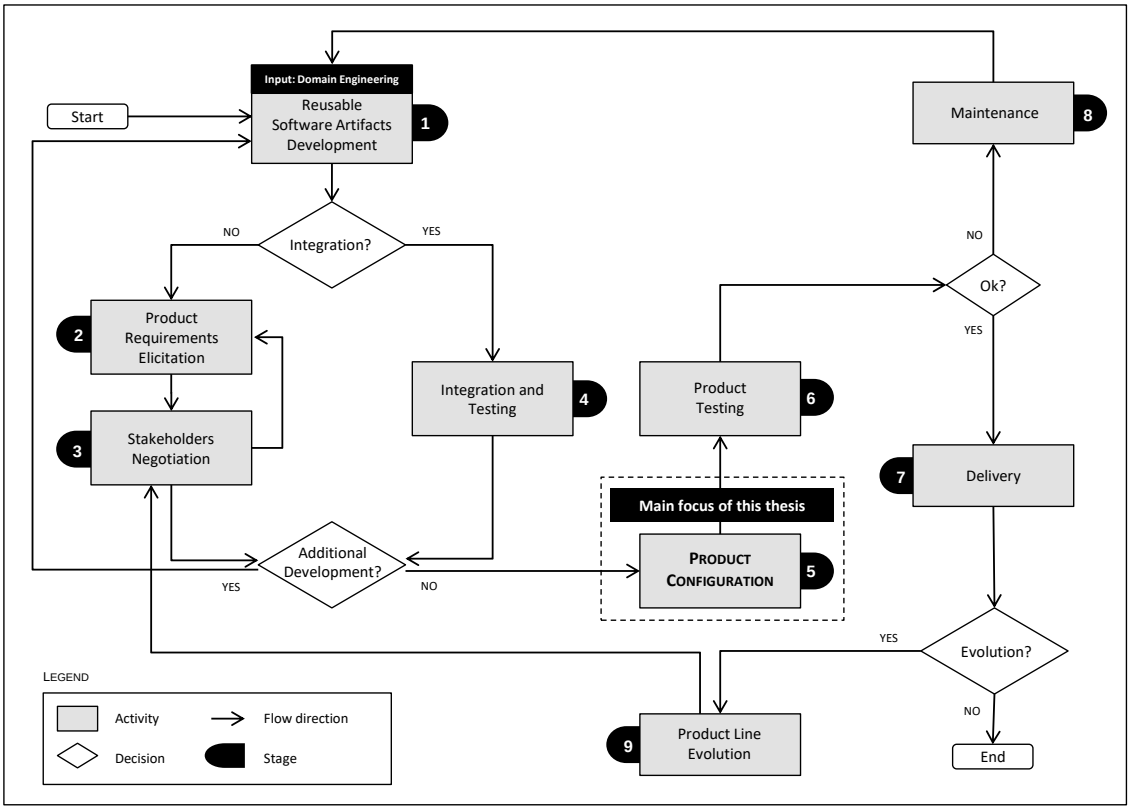


Figure 3.1: Activities supported by the application engineering phase.

Given an SPL, *product derivation* refers to the complete process of building individual products from reusable software artifacts (see Figure 3.1). It is performed as a process of *customization* (2-5, 9) and *generation* (6-8) of a specific product through *customer negotiation*. During this process, continuous iterations are performed until all *new requirements* (2-3) and *testing* (4) of the product have been satisfied. Then, the resulting updated SPL is used as input for the *product configuration* process (5). This process is performed interactively, mapping the *target application requirements* to features in the feature model. Finally, as a result of the product configuration process, if the *testing* (6) has been fulfilled, a concrete implementation of the derived product can be packed and *delivered* (7) to the customer. Notice that here *evolution* (9) of the SPL can happen in parallel. In any circumstance, if the *tests* have not been fulfilled, *maintenance* (8) is needed.

Based on both the state-of-the-art in application engineering and the expert knowledge from five SPL researchers, we subsequently describe the nine activities presented in Figure 3.1.

1. *Reusable Software Artifacts Development.* This activity includes the development of SPL assets (*e.g.*, implementation) from the variability model developed in the domain engineering phase.

2. *Product Requirements Elicitation.* This activity is concerned with collecting the non-functional product requirements of consumers and other stakeholders.

3. *Stakeholders Negotiation.* This activity aims at supporting the process of negotiating existing SPL assets with the stakeholders.

4. *Integration and Testing.* This activity is concerned with the integration of new assets (when requested) with existing ones, which requires new tests to verify and validate if the additional development performs correctly.

5. *Product Configuration.* This activity refers to the task of making decisions to select a set of valid combination of features from a feature model that fulfill the product requirements. A configuration task does not require any implementation effort and is often supported by automated tools, called *configurators*.

6. *Product Testing.* To guarantee that the product will work as expected, this activity verifies if the product meets both, functional and non-functional product requirements. This activity is different from activity 4, since it focuses on testing the derived product.

7. *Delivery.* This activity focuses on the deployment of the specific derived product (*i.e.*, configured features) in the customer environment.

8. *Maintenance.* Modification of SPL assets to correct newly discovered faults.

9. *Evolution.* Over time, an SPL will inevitably need to adapt (*i.e.*, introducing new features or interdependencies, and eliminate or customize features) to satisfy real-world changes in external conditions, such as, changes in technology, and improving design and performance.

In this thesis, we focus specifically on the *product configuration* activity (5) shown in the central part of Figure 3.1. The vast variety of work in the SPL configuration domain and the challenges faced in this field motivated us to carry out an SLR. This SLR investigates how and to what degree existing literature addresses the various aspects of the SPL configuration process. The main goal of this review is to help industries decide systematically when choosing new configuration techniques in their development environment. Moreover, we identify open issues and remaining challenges in this field, in order to point out areas that need attention and guide future research.

3.2 The Review Methodology

An SLR is a secondary study method that has received much attention in the software engineering community [Brereton et al., 2007, Dybå and Dingsøyr, 2008, Kitchenham et al., 2009]. It is defined as *a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest* [Kitchenham and Charters, 2007]. In this thesis, we have adapted and applied Kitchenham and Charters [2007] SLR guidelines, in order to identify, classify and assess relevant information related to approaches framed in the context of SPL configuration. In this section, we present the SLR methodology that consists of the following phases: *planning the review*, *conducting the review*, and *reporting the results*. The main activities related to each SLR phase and their location in the thesis are shown in Figure 3.2.

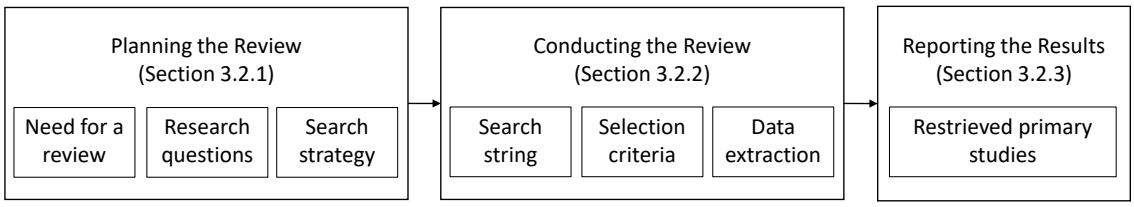


Figure 3.2: Systematic literature review phases.

3.2.1 Planning the Review

The *planning the review* phase aims at developing an explicit review protocol that defines the plan that the literature review will follow for identifying, assessing, and comparing evidence [Kitchenham and Charters, 2007]. This phase includes several actions: *(i)* identification of the need for a review; *(ii)* specification of the research questions; and *(iii)* development of a systematic search strategy that increases the rigor, transparency, and repeatability of the review, while establishing a means to reduce the risk of a biased study.

Identification of the need for a review. The need for an SLR originates from the increase in the number of SPL configuration studies made available in the recent years (see Section 3.2.3). Our literature review aims to complement an initial research by Benavides et al. [2010] and Benavides et al. [2013], identifying studies specifically in the *application engineering* phase, with focus on the *SPL configuration* process. Thus, the rationale of this SLR is *to get an overview about techniques suitable for an SPL configuration infrastructure*. The results contribute with relevant information to support practitioners in choosing appropriate techniques, and researchers identifying gaps in the existing techniques in order to indicate areas for further investigation. Therefore, we expect that both practitioners and researchers can benefit from the results of this review.

Specification of the research questions. We defined four main research questions with the help of *Population, Intervention, Comparison, Outcome, and Context (PICOC)* criteria [Higgins and Green, 2005]. Table 3.1 shows the PICOC structure. This SLR was conducted to analyze all available publications related to the research questions *RQ1—4* introduced in the beginning of this chapter. These questions are centered around three concerns: *(i)* SPL configuration activities and mechanisms; *(ii)* employed mechanisms; and *(iii)* open challenges. To address *RQ1*, we systematically identified the main SPL configuration activities using a literature review. Then, we show how these activities are supported by each study in the literature. Moreover, we investigate whether these studies are supported by tools and the evaluation process carried out by each of them (see Section 3.3). Next, to address *RQ2*, we developed an initial set of mechanisms regarded as essential (based on selected studies) for SPL configuration infrastructures. In addition, *RQ3* follows collecting evidence about existing studies and giving an in-depth view on how each study addresses each defined mechanism (see Section 3.4 for both *RQ2* and *RQ3*). Analyzing existing techniques allows identifying their limitations, such as which mechanisms are missing or have not been considered, addressed by *RQ4* (see Section 3.5).

PICOC	DESCRIPTION
Population	Literature in SPL configuration.
Intervention	Mechanisms, *i.e.* techniques, methods, strategies, tools, approaches that support SPL configuration.
Comparison	Product configuration mechanism(s) supported by each study.
Outcome	How existing studies support the product configuration process, and what the main open issues in this field are.
Context	Within the domain of the application engineering phase, with focus on the SPL configuration process.

Table 3.1: Research questions structured by the PICOC criteria.

Development of a systematic search strategy. The process of gathering and selecting relevant publications involved three steps. First, our initial set of candidate papers was identified from five scientific database libraries[1], namely ACM Digital Library, IEEE Xplore, ScienceDirect, Scopus and SpringerLink. These libraries were chosen because they are the most popular online scientific databases, since they index ACM, IEEE, Springer, and Elsevier publications that together cover many of the leading publications in the software engineering field [Travassos and Biolchini, 2007]. We conducted a search on these libraries for studies published in journals and conferences (including symposiums and workshops) proceedings from January 1^{st} 2012 up to December 31^{th} 2017. We selected the year of 2012 because Ochoa et al. [2017] observed an increasing tendency of research on the SPL configuration field in this year. For the second step, we conduct a screening process to exclude duplicate studies and studies that are not relevant for answering our research questions (see Section 3.2.2). For the third step, in addition to the search in digital libraries, we also follow analyzing the relevant cited reference lists from the retrieved and secondary studies (*i.e.*, related work) to find additional contributions outside the above mentioned subset (technique called *snowballing*) [Wohlin et al., 2000]. Finally, we merge all the results from scientific databases and the snowballing technique. The finally included papers were considered to be the *primary studies*.

3.2.2 Conducting the Review

Conducting the review means executing the protocol planned in the previous phase. This phase includes several actions: *(i)* definition of the terms used for systematic search; *(ii)* definition of the inclusion and exclusion criteria for selecting a relevant subset of publications; and *(iii)* definition of the data-extraction process for each retrieved study.

Search criteria. We use the strategy from Chen and Babar [2011] to construct the search string. We extracted the keywords used to search the primary study sources from our stated research questions and the analysis of selected primary studies by Benavides et al. [2010] and Benavides et al. [2013] within the SPL configuration domain. The search string is specified as follows:

[1] dl.acm.org; ieeexplore.ieee.org; sciencedirect.com; scopus.com; and link.springer.com.

(product line OR product family OR system family)
AND
(configuration OR configuration process OR product generation OR feature model configuration OR multi-step configuration)

We applied variants of the terms *product line* and *product configuration* to compose the search query. The terms *product line*, *product family*, or *system family* restrict the search to product line techniques. In addition, the terms *configuration*, *configuration process*, *product generation*, *feature model configuration*, or *multi-step configuration* refer to the product-configuration process. This string was subtly modified in each database according to the offered search capabilities. The search was performed considering only the *title*, *abstract*, and *keywords*. We developed scripts to partially automate this process. However, the screening of the queries' results was performed manually, as well the snowballing process. The search strings and the results of each search engine are provided in the Web supplementary material[2].

Selection of the primary studies. The basis for the selection of the primary studies is the *Inclusion Criteria (IC)* and *Exclusion Criteria (EC)* [Kitchenham and Charters, 2007]. We established a set of IC and EC with the goal of filtering only potential primary studies. The following three IC were used to include studies that are relevant to the research questions:

- *IC1.* (*English Papers*) We included only studies published in English. Moreover, articles have been published in journals, conferences, workshops, or symposiums. These are four of the most common types when publishing research results in the field of computer science and engineering.
- *IC2.* (*Within Scope*) We included only studies that deal directly with technique(s) to support the SPL configuration process, as opposed to considering the whole product derivation process.
- *IC3.* (*Extended Studies*) When different extensions of one paper were observed, *e.g.*, if a conference paper is extended into a journal version or a new functionality is created for a specific tool-support, we intentionally classified and evaluated them as separate primary studies for a more rigorous analysis. Note that in the presentation of results, we grouped these works with no major differences and we mention each different group of work as a different contribution in the field.

Initially, studies are only excluded if they meet at least one of the EC. To this end, the following four EC were used to exclude studies that we did not consider relevant to the research questions:

- *EC1.* (*Abstract*) We excluded introductions to special issues, workshops, tutorials, conferences, conference tracks, panels, poster sessions, as well as editorials.

[2] http://wwwiti.cs.uni-magdeburg.de/~jualves/SLR-SPLConfig

- *EC2.* (*Secondary Studies*) Secondary studies were not included in this review, such as literature reviews, comparative papers, articles presenting lessons learned, position or philosophical papers, with no technical contribution. However, the references of these studies were read in order to identify other relevant primary studies for inclusion through snowballing technique. Moreover, we consider secondary studies in the related work section.
- *EC3.* (*Completeness*) We excluded studies that contain unsupported claims, *i.e.* which do not provide, for each claim, proper references to existing work or results that prove the claim.
- *EC4.* (*Automated Analysis of Feature Models*) We excluded publications that focus on detecting the violation of the feature model constraints during the product line configuration process, or restrict themselves to identifying all valid products derivable from a model.

Data extraction. After applying the search string to each scientific database, the retrieved studies were first analyzed regarding the excluded criteria *EC1—EC4*. Afterwards, the remaining subset of studies were carefully checked for conformance to the inclusion criteria *IC1—IC3*. Then, for each retrieved study, the data were extracted and stored in a spreadsheet using a data extraction form. The form included the following data:

- Date of search, scientific database, and search string.
- Database, authors, title, publication type (*i.e.*, journal, conference, symposium, or workshop), and publication year.
- Exclusion criteria *EC1*, *EC2*, *EC3*, and *EC4* (yes or no)?
- Inclusion criteria *IC1*, *IC2*, *IC3*, and *IC3* (yes or no)?
- Selected (yes or no)? If not selected, justification regarding exclusion.

A set of three experts on the domain of SPL configuration had specific roles when performing this SLR. The author of this thesis performed the search for primary studies through the data extraction process. In this process, each retrieved publication was rated separately based on reading the titles and abstracts and if necessary checking the full text. When she decided that a paper was not relevant, she provided a short rationale why the paper should not be included in the study. In addition, another expert checked each included and excluded paper at this stage. Any discrepancies were resolved by calling upon a third expert. This step was done in order to check that all relevant papers were selected. Furthermore, all experts assisted during planning and reporting the review.

Once the list of primary studies was decided, each selected publication was then read in *full detail* and the content data for each selected paper was captured and extracted in a second form. The data extraction aimed to summarize the data from the selected primary studies for further analysis and for increasing confidence regarding their

relevance. All available documentation from studies served as data sources, such as dissertations/thesis, websites, tool support, as well as the communication with authors (*e.g.*, emails exchanged). Here, we grouped related studies as an unique contribution. The data extracted from each contribution were:

- Short summary about the study (*i.e.*, what contributions and novelty they constitute to), main reference(s), tooling support and website (if available).
- Evaluation type supported by the study (*i.e.*, industrial case studies, academic case studies, industrial examples, academic examples, randomized examples, observations and (or) experiences, and expert opinions), number of used case studies, and efficiency (*i.e.*, maximum size of the feature models used in performance evaluations) and effectiveness (*i.e.*, the optimality degree to which the proposed techniques is successful in producing the desired result) where applicable.
- SPL configuration activities and mechanisms supported by each retrieved study.
- Description of the main challenges and open issues in the SPL configuration domain that is raised by the authors of each selected study (if any). We captured challenges and open issues from the future work, conclusion, or threat to validity sections of a primary study. In the end, gaps and open challenges were identified for each activity (see Section 3.5).

Additionally, to validate the defined activities and mechanisms in the SPL configuration field, we invited 15 experts to participate in a survey. Our questionnaire contained two questions:

- *1)* How important do you regard the following activities and mechanisms in SPL configuration? (irrelevant [-2], unimportant[-1], important [1], very important [2]).
- *2)* What other activities or (and) mechanisms do you regard as important?

3.2.3 Reporting the Results

The goal of the reporting phase is to make it clear to others how the search was performed and how the study can be replicated. To select primary studies, we followed the three steps shown in Figure 3.3. First, the *retrieved* publications were found by applying the search terms to the defined sources thus revealing an initial list of *relevant studies*. We obtained a set of 320 papers, illustrated as *Step 1* in Figure 3.3. As we considered studies retrieved from different scientific databases, 64 papers were excluded because they were found in more than one search engine. Duplicate papers were automatically identified and discarded with *EndNote*. In addition, we performed a screening process over the retrieved studies and excluded 155 papers that were not identified to be relevant based on the selection criteria (see Section 3.2.2). At the end of the screening process, we identified 101 potentially relevant papers from the *automated search*, illustrated as *Step 2* in Figure 3.3.

From the 155 excluded papers during the screening process, 13 are secondary studies (*i.e.*, exclusion criteria *EC2* presented in Section 3.2.2). Given that these studies

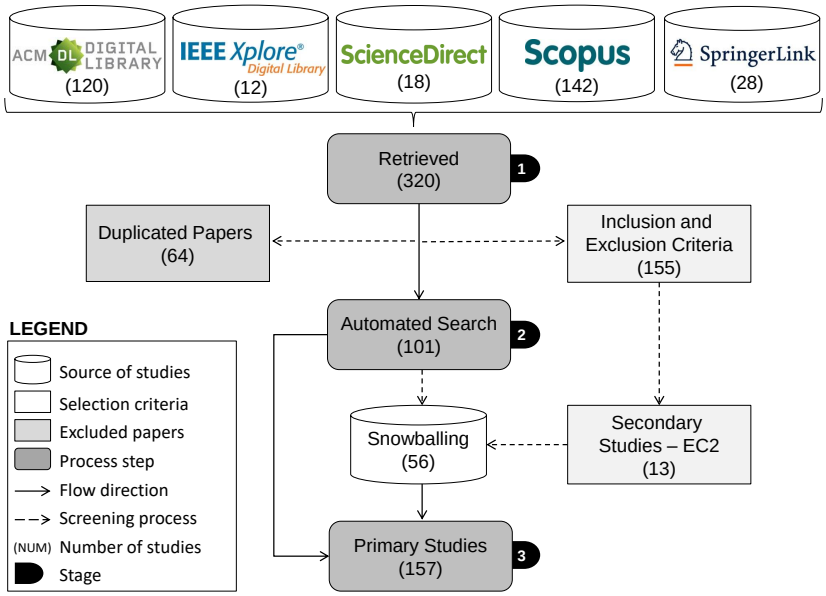


Figure 3.3: Selection procedure of primary studies.

followed a systematic process to select relevant papers in the *application engineering* phase, we acknowledge that they might also include relevant primary studies in their references. Therefore, we manually analyzed the reference list of each secondary study to identify missing relevant papers aiming at increasing the coverage and quality of our SLR. Moreover, we also followed the reference list from the 101 relevant retrieved papers obtained from the automated search. This technique retrieved an additional 56 papers, shown in Figure 3.3 as the *snowballing technique*. This technique ensured a significant increase of the number of relevant publications.

Finally, we conducted an update on the list of primary studies, through the merging process of the results from the *snowballing technique* and the *automated search*. We identified a total of 157 selected *primary studies* that were in the scope of this review for further analysis (*i.e.*, full-text reading) illustrated as *Step 3* in Figure 3.3.

Figure 3.4 shows the temporal distribution of primary studies per year by publication venues. We observe a high number of relevant publications in the SPL configuration community since 2012. In 2012, the total number of publications reaches its maximum of 34 publications. Then, from 2013 to 2016 there is a significant decrease in the number of publications (*i.e.*, 24-29 retrieved publications). Notice that data from 2017 cannot be incorporated in the tendency analysis because the snowballing technique was not able to retrieve publications in 2017, since hardly authors cited 2017 papers. Otherwise, the snowballing technique retrieved a considerable amount of important papers from 2012 to 2016 (67% in average).

In total, note that only 33 out of 157 studies, *i.e.* 21% (see Figure 3.4), were found in workshops (17) and symposiums (16), whereas most (*i.e.*, 79%) were published in conferences (72) and journals (52). As expected, most of the selected publications appeared in the proceedings of the *International Systems and Software Product Line Conference* (SPLC) considered the most representative conference for the SPL engineering community, followed by the *International Workshop on Variability Mod-*

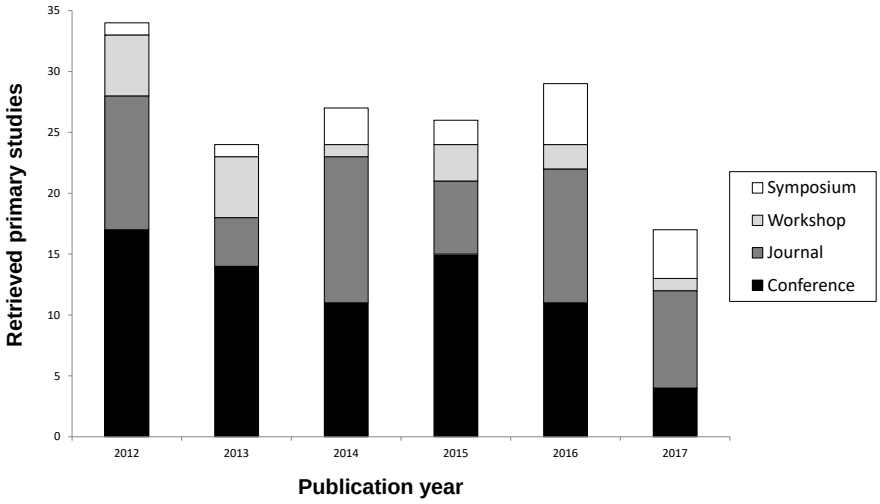


Figure 3.4: Temporal distribution of primary studies in six years of research on SPL configuration retrieved in the SLR by year from each venue: conferences, journals, workshops, and symposiums.

eling of Software-Intensive Systems (VaMoS), a workshop also dedicated to the SPL community, and the *Journal of Information and Software Technology* (IST). They contributed with 26, 9, and 7 primary studies respectively. In total, the source from the 157 selected primary studies were gathered from 89 different venues: 36 journals, 36 conferences, 10 symposiums, and 7 workshops (the complete list of paper venues is found in Section A.1). Such a distribution gives us the initial impression that there are many relevant studies in this field, since journals and conference contribute significantly to higher quality works. However, there is still no clear picture regarding the mechanisms needed for supporting the SPL configuration process. Next, we developed an initial definition of the activities (Section 3.3) and mechanisms (Section 3.4) supporting the SPL configuration process by systematically analyzing the set of 157 retrieved primary studies.

3.3 Product Configuration Activities

We extracted 112 different contributions in the SPL configuration literature from 157 retrieved primary studies (*i.e.*, 45 studies are extensions of other studies). In this section, we analyze these contributions, in order to answer *RQ1*. We performed a complete reading of all selected primary studies and we collected the SPL configuration concern addressed by each contribution. After having extracted the main concern from each study, we discussed their central proposal with experts and grouped close contributions. We identified five groups of activities used to support the SPL configuration process (A1—A5). Figure 3.5 gives an overview of the activities and defines the most appropriate order to configure a product.

- *A1) Mapping Non-Functional Properties.* This activity maps *Non-Functional Properties* (NFPs) to their respective implied features (or configurations) in the feature model.

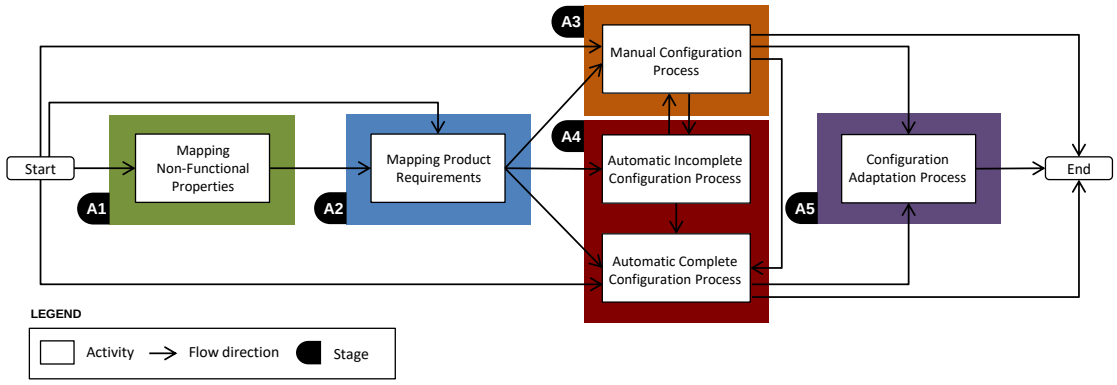


Figure 3.5: Overview of the activities supported by the SPL configuration process.

- *A2) Mapping Product Requirements.* This activity focuses on the translation of the product requirements into concrete specifications, in order to automatically configure a product that best matches them.

- *A3) Manual Configuration Process.* This activity takes as input consecutive feature selections and deselections from the feature model. Manual configuration inevitably introduces human errors and therefore ensures that each decision conforms with pre-defined constraints is a very important functionality of this activity. It is helpful to begin configuring a product or to change automatic configuration results.

- *A4) Automatic Configuration Process.* Feature models tend to be inherently large and complex, with several types of variability relations and constraints. Moreover, the diversity of stakeholders' requirements leads to *heterogeneous* optimization objectives, making the manual configuration process inappropriate or even infeasible. Therefore, it indicates the need for faster and more automatic mechanisms. *Automatic configuration mechanisms* can be classified into two groups:

  a. *Automatic Complete Configuration.* A configuration is complete if the automatic method applied over the feature model returns a single valid configuration with a defined selection for each feature from the feature model.

  b. *Automatic Incomplete Configuration.* A configuration is incomplete if the automatic method applied over the feature model returns a set of valid configurations that satisfy the product non-functional requirements.

- *A5) Configuration Adaptation Process.* This activity focuses on the adaptation from the manual or automatic configuration considering events, such as change in the production environment which consequently implies changes in A1 and A2.

Table 3.2 shows the configuration activities addressed by each contribution. The first column identifies the study *reference*. For simplification, we present one citation per contribution (*i.e.*, the most recent study). For a general overview of all studies addressing each contribution, we refer to Table A.4 in Section A.2. The *Activities* column shows which activities are supported by each different contribution (A1—A5). We use ● to contributions in the literature that offer full support for the

activity, ◐ to contributions that offer partial support, and ○ to contributions that offer no support. The # and *Evaluation* columns are about the *number of instances evaluated* to support the paper claims and the *evaluation type* performed to prove the study validity. The following evaluation types were considered: *industrial case studies* (ICS), *academic case studies* (ACS), *industrial examples* (IEX), *academic examples* (AEX), *randomized examples* (REX), *observations and experiences* (OBE), and *expert opinions* (EOP). We use NEV for studies covering *no evaluation*. Finally, the *Tool-Support* column identifies if there is tool-support and their respectively related tool when mentioned in the paper.

Table 3.2: Literature supporting the SPL configuration process.

Study	Activities					#	Evaluation	Tool-Support
	A1	A2	A3	A4	A5			
Siegmund et al.	●	◐	◐	◐	○	9	IEX, AEX	SPL Conqueror
Zanardini et al.	◐	●	◐	◐	○	1	IEX	FeatureIDE
Mazo et al.	◐	◐	◐	◐	◐	50	AEX	VariaMos
Kifetew et al.	◐	◐	◐	◐	○	1	AEX	Yes
Noorian et al.	◐	◐	◐	◐	○	80	REX	No
Pereira et al.	◐	◐	◐	◐	○	22	AEX, REX	SPLConfig
Bąk et al.	◐	◐	◐	◐	○	9	IEX, AEX	ClaferMoo
Noir et al.	◐	◐	◐	◐	○	1	ICS, OBE	pure::variants
Ochoa et al.	◐	◐	◐	◐	○	1	AEX	FeatureIDE
Asadi et al.	◐	◐	◐	◐	○	12	REX	fmp
Bagheri et al.	◐	◐	◐	◐	○	1	AEX, OBE	No
Ostrosi et al.	◐	◐	◐	◐	○	1	ACS	Yes
Schroeter et al.	◐	◐	◐	○	◐	0	NEV	PuMA
Sion et al.	◐	◐	◐	○	○	1	IEX	FeatureIDE
Myllärniemi et al.	◐	◐	◐	○	○	1	IEX	Kumbang
Bashari et al.	◐	◐	◐	○	○	-	OBE	AUFM tool
Tan et al.	◐	◐	◐	○	○	1	AEX	No
Zhang et al.	◐	◐	◐	○	○	1	IEX	QAPCTool
Lettner et al.	◐	◐	◐	○	○	15	IEX	DOPLER
Guedes et al.	◐	◐	○	◐	◐	2	AEX	VariaMos
Ruiz et al.	◐	◐	○	◐	◐	1	ICS	No
Leite et al.	◐	◐	○	◐	◐	2	IEX	Dohko
Pascual et al.	◐	◐	○	◐	◐	12	AEX	FamWare
Sánchez et al.	◐	◐	○	◐	◐	1	IEX	No
Parra et al.	◐	◐	○	◐	◐	1	IEX	No
Wittern et al.	◐	◐	○	◐	◐	3	IEX	Yes
Horcas et al.	◐	◐	○	◐	○	4	IEX	HADAS
Oh et al.	◐	◐	○	◐	○	4	AEX	No
Umpfenbach et al.	◐	◐	○	◐	○	1	ICS	No
Hierons et al.	◐	◐	○	◐	○	7	IEX, AEX	No
dos Santos Neto et al.	◐	◐	○	◐	○	1	AEX	Yes
Benali et al.	◐	◐	○	◐	○	1	ICS	No
Henard et al.	◐	◐	○	◐	○	5	IEX	Yes
Lian and Zhang	◐	◐	○	◐	○	2	AEX	No
Rezapour et al.	◐	◐	○	◐	○	1	IEX	No
Guo et al.	◐	◐	○	◐	○	3	IEX, AEX	Yes
Olaechea et al.	◐	◐	○	◐	○	5	AEX	No
Wang and Pang	◐	◐	○	◐	○	600	REX	No
Sayyad et al.	◐	◐	○	◐	○	7	IEX, AEX	Yes
Roos-Frantz et al.	◐	◐	○	◐	○	2	IEX	FaMa-OVM

Table 3.2: Literature supporting the SPL configuration process.

Study	Activities A1	A2	A3	A4	A5	#	Evaluation	Tool-Support
Mussbacher et al.	◐	◐	○	◐	○	1	ACS	jUCMNav
Ognjanovic et al.	◐	◐	○	◐	○	1	AEX	ConfBPFM
Ter Beek et al.	◐	◐	○	○	◐	2	IEX	MultiVeStA
Mauro et al.	◐	◐	○	○	◐	1	ICS	HyVarRec
Santos et al.	◐	◐	○	○	◐	2	ACS	No
Wang et al.	◐	◐	○	○	○	1	IEX	No
Camacho et al.	◐	◐	○	○	○	1	IEX	SPLAcris
Triando et al.	◐	◐	○	○	○	1	ICS	Yes
Winkelmann et al.	◐	◐	○	○	○	1	ACS	No
Xue et al.	◐	◐	○	○	○	6	IEX, AEX	No
Qin and Wei	◐	◐	○	○	○	0	NEV	No
Khoshnevis and Shams	◐	○	◐	◐	○	10	ICS, OBE	Yes
Pleuss and Botterweck	◐	○	◐	○	◐	1	ACS	S2T2 Configurator
Mazo et al.	◐	○	◐	○	○	1	IEX, OBE	VariaMos
Zhao et al.	◐	○	◐	○	○	1	ICS	No
Soares et al.	◐	○	○	◐	○	-	OBE	No
Murwantara et al.	◐	○	○	◐	○	1	IEX	No
Bürdek et al.	◐	○	○	○	◐	1	IEX	Yes
Valov et al.	◐	○	○	○	○	6	IEX	SPL Conqueror
Kolesnikov et al.	◐	○	○	○	○	0	NEV	No
Nieke et al.	○	◐	◐	○	◐	1	ICS	DarwinSPL
Zheng et al.	○	◐	◐	○	◐	1	AEX	Yes
Eichelberger et al.	○	◐	◐	○	◐	0	NEV	EASy-Producer
Mazo et al.	○	◐	◐	○	◐	0	NEV	VariaMos
Murguzur et al.	○	◐	◐	○	◐	0	NEV	LateVa toolkit
Acher et al.	○	◐	◐	○	◐	5	ICS, ACS	FAMILIAR
Martinez et al.	○	◐	◐	○	○	1	IEX	No
Foster et al.	○	◐	○	◐	○	2	IEX	No
White et al.	○	◐	○	◐	○	23	AEX	FaMa
Karimpour and Ruhe	○	◐	○	◐	○	2	AEX	No
Adjoyan and Seriai	○	◐	○	○	◐	0	NEV	No
Zheng et al.	○	◐	○	○	◐	1	ICS	No
Ayala et al.	○	◐	○	○	◐	2	ICS	No
Sharifloo et al.	○	◐	○	○	◐	0	NEV	No
Cubo et al.	○	◐	○	○	◐	-	OBE	DAMASCo
Kramer et al.	○	◐	○	○	◐	1	IEX	FeatureIDE
Saller et al.	○	◐	○	○	◐	1	AEX	No
Wang and Ng	○	◐	○	○	◐	1	ACS	No
Bajaras and Agard	○	◐	○	○	○	1	AEX	Yes
Zdravkovic et al.	○	◐	○	○	○	1	IEX, AEX	Yes
Chen et al.	○	◐	○	○	○	1	IEX	No
Ge et al.	○	◐	○	○	○	1	AEX	No
Safdar et al.	○	○	◐	○	◐	2	ICS	No
Rabiser et al.	○	○	◐	○	◐	2	ICS	FORCE
Brink et al.	○	○	◐	○	◐	0	NEV	No
Galindo et al.	○	○	◐	○	◐	6	REX	Invar
Chavarriaga et al.	○	○	◐	○	◐	0	NEV	No
Urli et al.	○	○	◐	○	◐	1	IEX	SpineFM
Klambauer et al.	○	○	◐	○	◐	-	IEX, EOP	DOPLER
Heider et al.	○	○	◐	○	◐	1	IEX	VaMoRT
Hajri et al.	○	○	◐	○	○	1	ICS, OBE	PUMConf

Table 3.2: Literature supporting the SPL configuration process.

Study	Activities					#	Evaluation	Tool-Support
	A1	A2	A3	A4	A5			
Lu et al.	○	○	◐	○	○	12	IEX	Zen-Configurator
Pereira et al.	○	○	◐	○	○	0	NEV	FeatureIDE
Pereira et al.	○	○	◐	○	○	2	IEX	PROFilE
Schwäger and Westfechtel	○	○	◐	○	○	0	NEV	SuperMod
Zhang and Becker	○	○	◐	○	○	0	NEV	EXConfig
Fang et al.	○	○	◐	○	○	-	EOP	MagicDraw
Behjati et al.	○	○	◐	○	○	1	IEX	Yes
Martinez et al.	○	○	◐	○	○	1	IEX, OBE	FeatureIDE
Tan et al.	○	○	◐	○	○	-	OBE	Yes
Nöhrer et al.	○	○	◐	○	○	7	IEX, AEX	No
Thurimella and Bruegge	○	○	◐	○	○	-	OBE	Sysiphus
Lin and Kremer	○	○	○	◐	○	1	IEX	No
Gençay et al.	○	○	○	○	◐	1	ICS	No
Nieke et al.	○	○	○	○	◐	1	ACS	No
Tanhaei et al.	○	○	○	○	◐	2	ICS, ACS	No
Bures et al.	○	○	○	○	◐	0	NEV	No
Gamez and Fuentes	○	○	○	○	◐	4	AEX	Hydra
Jannach and Zanker	○	○	○	○	◐	1	IEX	No
Lee	○	○	○	○	◐	0	NEV	No
Mitchell	○	○	○	○	◐	1	ICS	No
Neves et al.	○	○	○	○	◐	2	ACS	No

Table 3.2 shows that activities A1 and A2 are only fully supported by Siegmund et al. [2015] and Zanardini et al. [2016], respectively. Besides, despite the importance of all five activities working together in a same approach, there is only one approach that offers at least partial support to all five activities [Mazo et al., 2012c]. In 53 out of the 112 approaches (46.4%) at least three configuration activities are partially supported. Only 28 approaches (25%) offer support for just one activity. Although it indicates that the research community has realized the benefits of proposing approaches which encompass different configuration activities, there is still need for approaches that would fully support at least two of the defined activities (Section 3.4 presents the set of mechanisms supported by each activity).

Regarding the evaluation type, it should be noted that the most commonly employed evaluations use industrial (36.6%) and academic (25%) examples. In 50 out of the 112 different contributions (44.6%), only one test instance (*i.e.*product line) is used to demonstrate the approach feasibility and 19.6% of them do not use any instance (*i.e.*, 64.2% in total). Moreover, only a small number (16.1%) of the analyzed contributions applied more than one evaluation method. Furthermore, although an empirical user evaluation process (through users observations and experiences) is desirable to increase the external validity of the proposed approach, only 9.8% of the selected contributions conducted this type of evaluation. While 59.8% of the contributions provide tool support, there is still a big lack in the literature regarding the empirical evaluation of these tools.

Given the complexity of the SPL configuration process, tool support seems to be an essential aspect that can assist decision makers. Most tools extend the well known state-of-the-art tool FeatureIDE [Martinez et al., 2014, Pereira et al., 2016b, 2017, 2016c, Rabiser et al., 2016, Sion et al., 2016, Zanardini et al., 2016]. However, only 34 studies (30.4%) make the reader aware of how the tool could be obtained. The remaining studies do not provide readers with detailed information about the tool neither make clear whether the tool was extended or built for the particular purpose of the proposed approach. Hence, we argue that the actual research community might not be aware of the importance of those information for researchers when extending their work. Moreover, it limits the research community to empirically compare and assess these tools, as well as for practitioners when using such proposed techniques. Furthermore, only 13.4% of these contributions made the evaluation material (including the detailed results) publicly available for the purpose of reproducibility of the study by other researchers. Thus, without the public resources (*i.e.*, tools or algorithms) it is hard to perform replication studies.

In addition to Table 3.2, Figure 3.6 shows the number of contributions in the literature partially (or fully) addressing each SPL configuration activity. Moreover, in the same figure, we illustrate contributions either *with* or *without* tool support. Although all five activities have drawn interest, the figure shows that activities A1 and A2 are supported by most of the approaches (*i.e.*, 60 and 73 respectively out of the 112 total contributions). Despite the observed trend to both activities, the percentage of contributions that implement tool-support remains constant between 53% and 62% for activities A1, A2, A4 and A5. It is important to notice that although only 50 approaches implement the activity A3, 82% of these contributions implement tool-support.

Figure 3.7 shows the percentage of studies per activity that employ any of the aforementioned evaluation techniques. In summary, our data reveals that from our retrieved contributions, an increasing attention was paid to the use of industrial examples as a means to prove the validity of the proposed activities (23-36%). Regarding activity A3, a higher interest was paid to the use of *observations and experiences* (OBE). This might be due to the fact that this activity offers the highest percentage of tool support (see Figure 3.6). Moreover, regarding activity A5, a large portion of the studies did not present any kind of evaluation to demonstrate their feasibility. Furthermore, there is still a portion of the contributions in the literature that use case studies (*i.e.*, ICS and ACS) as the main evaluation. Since most case study are just a running example used to explain the proposed technique, it could be suggested that further experiments (*e.g.*, performance analysis) could be conducted over these approaches as an additional contribution in this field. Next, Section 3.4 presents detailed information about the set of mechanisms implemented by each activity.

3.4 Product Configuration Mechanisms

In this section, we systematically investigate how the retrieved primary studies from our SLR address the SPL configuration activities. We aim at understanding different contributions and classifying possible future directions, by answering *RQ2* and *RQ3*. First, based on our insights from Section 3.3, we performed a categorization

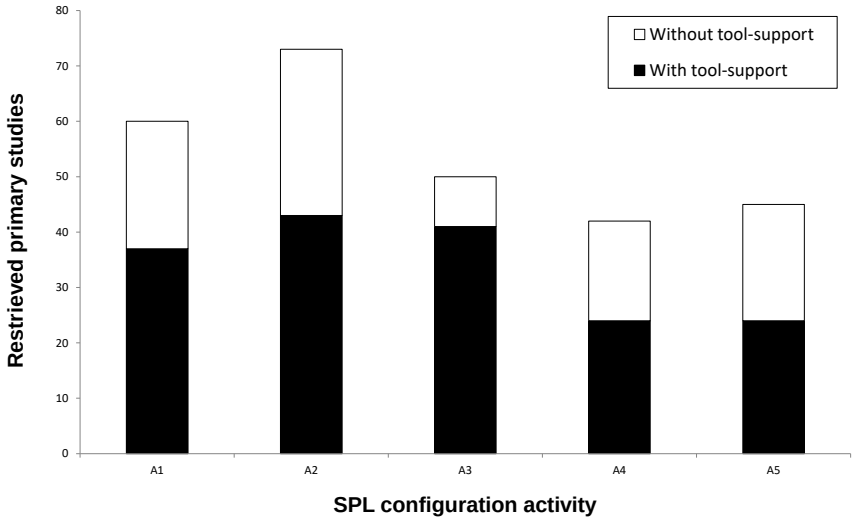


Figure 3.6: Retrieved primary studies that offer support for each activity and their tool-support distribution.

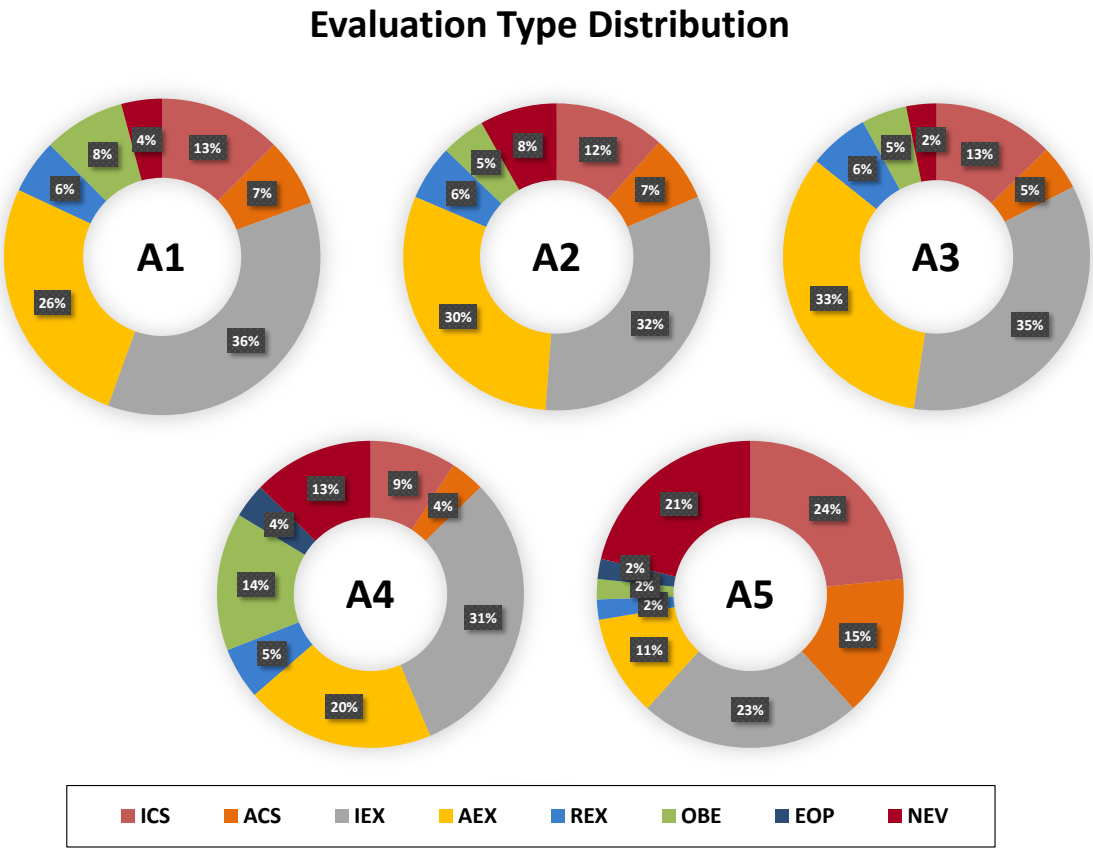


Figure 3.7: Evaluation type supported by each activity (ICS: industrial case studies, ACS: academic case studies, IEX: industrial examples, AEX: academic examples, REX: randomized examples, OBE: observations and experiences, EOP: expert opinions, and NEV: no evaluation).

of the mechanisms addressed by each SPL configuration activity. Then, to ensure completeness, we involved experts in this field to assess the relevance of these mechanisms (see Section 3.2.2). This resulted in the set of seventeen mechanisms described in Figure 3.8. We do not claim that this set is complete. However, it is simply used to classify the current literature in SPL configuration. In the following, we present possible practical applications to make these mechanisms easily understandable to the readers. Moreover, we describe a brief overview about each mechanism and discuss how the studies support them.

3.4.1 Mapping Non-Functional Properties

This section presents three mechanisms supported by the *Mapping Non-Functional Properties* activity: *(i)* Non-Functional Properties Specification (A1a); *(ii)* Non-Functional Properties Measurement (A1b); and *(iii)* Reuse of Non-Functional Property Measurements (A1c). Table 3.3 sketches which studies support these mechanisms. The first column identifies the study reference and the other three columns are about the mechanisms. We use ◐ for studies providing support only for *quantitative NFPs*. We use ◑ for studies providing support only for *qualitative NFPs*. We use ● for studies that support both, *qualitative* and *quantitative NFPs*. Finally, we use ○ for studies without any support.

3.4.1.1 Non-Functional Properties Specification

This mechanism takes as input a feature model and allows annotating primitive and compound features, or even valid and complete products with NFPs. NFPs can be defined as an extension of a feature inside a feature model (see Section 2.1). In Figure 2.1 in Chapter 2, we show an example of an *Extended Feature Model* (EFM) using the notation inspired by Benavides et al. [2005]. In addition, NFPs can be directly associated to valid and complete configurations due to feature interactions (see Section 3.4.1.2). There are a large number of quantitative and qualitative NFPs reported in the literature. For a survey on these properties we refer to [Boehm et al., 1978, Commission et al., 2001, McCall et al., 1977, Soares et al., 2014].

As far as we are aware, Benavides et al. [2005] were the first authors to introduce EFMs. However, there is still no consensus on a notation to define NFPs. Most proposals agree that NFPs should be modeled and defined for primitive as well as compound features, and they consist at least of a *name*, a *domain*, a *value*, and a *unit*. Some authors assume that only primitive features in a feature model have concrete implementation and consequently are annotated with NFPs. In this case, NFPs related to compound features are represented as the aggregation of feature NFPs of other features [Benavides et al., 2005]. As illustrated in Figure 3.9, NFPs are used to specify quantitative non-functional information (*i.e.*, *reliability* = 98%, *response time* = 500 milliseconds, and *cost* = $600.0) and qualitative non-functional information (*i.e.*, high *customer satisfaction* and high *security*) about the feature. The same property can vary for each domain, application scenario, environment, and stakeholder. For example, the property *response time* may be quantitatively or qualitatively specified. The same stands for a valid and complete configuration.

On the one hand, there are authors who propose the use of functional metrics for the quantifiable measurement of NFPs (see Section 3.4.1.2). On the other hand, there

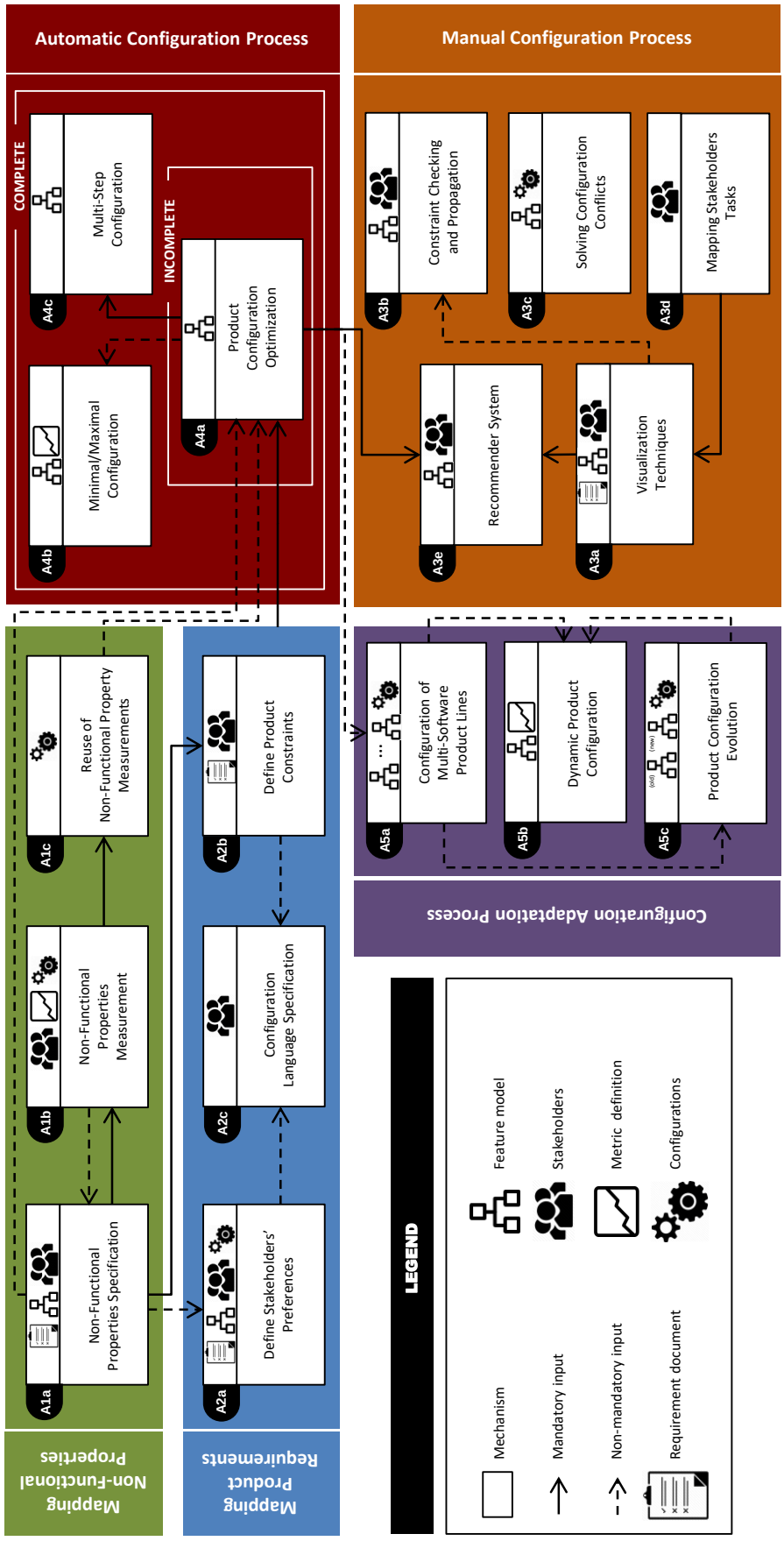


Figure 3.8: Overview of the mechanisms supported by each SPL configuration activity.

Study	A1a	A1b	A1c	Study	A1a	A1b	A1c
dos Santos Neto et al.	●	●	○	Murwantara et al.	◐	◐	○
Siegmund et al.	●	◐	◐	Kolesnikov et al.	◐	◐	○
Horcas et al.	●	○	●	Lettner et al.	◐	◐	○
Soares et al.	●	○	●	Kifetew et al.	◐	○	○
Guedes et al.	●	○	○	Oh et al.	◐	○	○
Noorian et al.	●	○	○	Ter Beek et al.	◐	○	○
Pereira et al.	●	○	○	Bąk et al.	◐	○	○
Umpfenbach et al.	●	○	○	Ruiz et al.	◐	○	○
Wang et al.	●	○	○	Winkelmann et al.	◐	○	○
Hierons et al.	●	○	○	Benali et al.	◐	○	○
Mauro et al.	●	○	○	Leite et al.	◐	○	○
Noir et al.	●	○	○	Ochoa et al.	◐	○	○
Santos et al.	●	○	○	Pascual et al.	◐	○	○
Sion et al.	●	○	○	Rezapour et al.	◐	○	○
Triando et al.	●	○	○	Bürdek et al.	◐	○	○
Xue et al.	●	○	○	Guo et al.	◐	○	○
Henard et al.	●	○	○	Mazo et al.	◐	○	○
Lian and Zhang	●	○	○	Olaechea et al.	◐	○	○
Asadi et al.	●	○	○	Wang and Pang	◐	○	○
Tan et al.	●	○	○	Sayyad et al.	◐	○	○
Zhang et al.	●	○	○	Roos-Frantz et al.	◐	○	○
Bagheri et al.	●	○	○	Mazo et al.	◐	○	○
Mussbacher et al.	●	○	○	Ostrosi et al.	◐	○	○
Schroeter et al.	●	○	○	Pleuss and Botterweck	◐	○	○
Zhao et al.	●	○	○	Qin and Wei	◐	○	○
Khoshnevis and Shams	◐	●	○	Wittern et al.	◐	○	○
Sánchez et al.	◐	●	○	Myllärniemi et al.	◑	○	○
Camacho et al.	◐	◐	○	Bashari et al.	◑	○	○
Zanardini et al.	◐	◐	○	Ognjanovic et al.	◑	○	○
Valov et al.	◐	◐	○	Parra et al.	◑	○	○

Table 3.3: Literature supporting the activity: *Mapping NFPs* (A1a: *Non-Functional Properties Specification*, A1b: *Non-Functional Properties Measurement*, A1c: *Reuse of Non-Functional Property Measurements*)

are authors who propose the use of domain expert judgments to assign qualitative or quantitative NFP values, which depend on the availability of domain experts, who must engage themselves in a time-consuming and error-prone activity.

In addition to the cross-tree constraints represented by features and binary operators, EFMs can also include *hard cross-tree constraints* (see Section 2.1.1). Although the specification of hard cross-tree constraints are useful to the use of decision propagation techniques (Section 3.4.3) and a more automatic configuration support (Section 3.4.4), there are only 11 approaches which provide support for hard cross-tree constraints [Bürdek et al., 2014, Lian and Zhang, 2015a, Mauro et al., 2016, Myllärniemi et al., 2015, Ognjanovic et al., 2012, Santos et al., 2016, Sayyad et al., 2013b, Ter Beek et al., 2016, Wang et al., 2017, Winkelmann et al., 2016, Zanardini et al., 2016]. In addition, we are not aware of any available contribution in the literature to guide domain experts specifying hard cross-tree constraints.

In Table 3.3, we observe 60 studies supporting NFP specification, which represents 53.6% out of the total amount of contributions. 50% of them allow users to specify

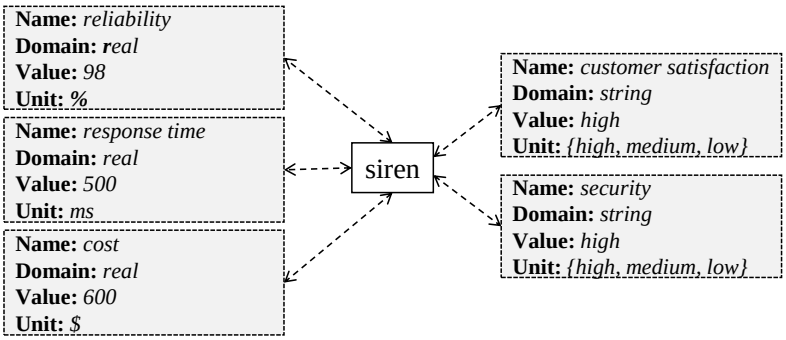


Figure 3.9: A sample of non-functional properties attributed to the feature siren from the *smart-home* product line illustrated in Figure 2.1.

NFPs for both primitive and compound features. Moreover, 25 studies (*i.e.*, 41.7%) provide support to both quantitative and qualitative NFPs, 31 studies (*i.e.*, 51.7%) provide only support to quantitative NFP, and 4 studies (*i.e.*, 6.7%) provide only support to qualitative NFP.

Although 60 contributions are expected to be a relevant amount of studies, 38 approaches (*i.e.*, 63.3%) work with a limited set of maximal four NFPs and 11 of them (*i.e.*, 20%) do not clearly specify which NFPs are supported by the respective approach (see Table A.5 in Section A.3 for a complete overview of the set of supported NFPs). Moreover, the few approaches providing support to qualitative NFPs map linguistic terms (*e.g.*, very negative, negative, neutral, positive, and very positive) onto real values to be handled as quantitative properties, except for Myllärniemi et al. [2015].

Myllärniemi et al. [2015] rely on descriptive countermeasures to capture the impact of the NFP security (*e.g.*, encryption and protection level) on features and other NFPs. In their approach, they define countermeasure variability through composition, attributes, inheritance, and constraints. Thus, countermeasures are organized into hierarchies and represented with the cardinality of the form $[n...m]$; with n and $m \in N$. For example, the NFP security in Figure 3.9 may be further represented by the countermeasure *encryption* and *protection level*. Encryption defines two possible types, *i.e. yes* or *no*, and protection level defines four possible types, *i.e. nothing*, *medium*, *high*, and *custom*. Therefore, countermeasures and its types can be represented as in cardinality-based feature modeling [Vale et al., 2016]. The authors use cardinality [1...1] to represent mandatory countermeasures, cardinality [0...1] for optional countermeasures, [1...1] for alternative exclusive and $[1...m]$ for alternative non-exclusive countermeasures.

EFM is also referred in the literature to as *extended product line*, *annotated feature model*, *annotated product line*, *annotation of feature models*, *attributed feature model*, and *attributed product line*. Moreover, NFPs can be also referred to as *feature attributes*, *non-functional interdependencies*, *non-functional requirements*, *attribute instances*, *quality attributes*, *non-functional concerns*, and *abstract concerns*. Finally, hard cross-tree constraints can be referred to as *cross-non-functional constraints*, *integrity non-functional constraints*, and *dependencies between features and non-functional properties*.

3.4.1.2 Non-Functional Properties Measurement

This mechanism takes as input the EFM and suitable metrics defined for each NFP, and automatically measure NFPs in terms of their real impact over features, and (or) valid configurations. In Table 3.3, we observe 10 approaches supporting this mechanism. This category uses NFPs that can be measured on a metric scale, such as *response time* and *cost*, among others.

NFPs can be measured using *dynamic* and *static* analysis. *Dynamic analysis* consists of executing the configurations and monitoring the measuring of NFPs at runtime, otherwise *static analysis* infers the NFP values only by examining the code, model, or documentation. Thus, static analysis gives just qualitative statements about configurations. For example, it cannot predict the accurate amount of *response time* required by smart-home configurations, but it can predict which configurations have a low *response time* compared to the other configurations. Although static measurement cannot make accurate predictions of NFP values, it is much faster than collecting data dynamically from a potentially exponential number of products. Moreover, when using static analysis, partial configurations can be measured.

There are several functional metrics defined in the literature to measure NFPs [Boehm et al., 1976, Chhabra and Gupta, 2010, Cleland-Huang et al., 2007]. These metrics refer to many factors depending on the context and scenario of the product line, product requirements, and related features. For example, on the one hand, in the software context, the *cost* NFP can be measured as the required effort to add a feature to a product under construction by analyzing the SPL cycle evolution, such as, the number of lines of code, the development time, or other functional size metrics. On the other hand, in the smart-home context, the *cost* NFP can be measured as the effort, in terms of human hours, to build and install the respective features. Additionally, it may refer to the accumulative sum of several metrics. However, features cannot be measured in isolation due to feature interactions.

Interactions occur when combinations among features share a common component or require additional component(s). As an example, consider the smart-home product line illustrated in Figure 2.1. Product variants that include both features `siren` and `blinking_lights` have a positive impact of 100% in the NFP *reliability* (*i.e.*, if the siren fails, the smart home can make the home lights blink as a replacement for the failed alarm). Therefore, these features interact positively in terms of *reliability*, although none of them have 100% of *reliability*. The unexpected observed results are caused by feature interactions of both `blinking_lights` and `siren` features.

Siegmund et al. [2015] have developed a technique called *SPL Conqueror* which supports the definition and dynamic measurement of NFPs. The authors approach takes feature interactions into consideration by having a model that defines known interactions and measures their influence using specific sampling heuristics that meet different feature-coverage criteria. However, in case of lack of domain knowledge to know in advance which features interact with each other, they assume the existence of feature interactions between each pair of features. Considering the fact that n optional features can generate 2^n different configurations, measuring all variants for large SPLs is impractical due to the very high number of possible combinations.

From their experimental results, they demonstrated that measuring *footprint* required the most amount of time for SPLs with either a large number of features (*e.g.*, 48 hous for the SPL SQLite with 85 features or 24 hours for the SPL Violet with 100 features) or a large code base (*e.g.*, 4 days in the case of the Linux kernel SPL with just a small set of 25 features).

In a similar context, other approaches such as Valov et al. [2015] and Murwantara et al. [2014] have focused just on a small sample of measured configurations. Valov et al. [2015] infer *performance* using four non-linear regression methods (*i.e.*, *classification and regression trees*, *bagging*, *random forest*, and *support vector machines*). Empirical results show that *bagging* achieves the best accuracy in approximately 2 hours for 1.152 valid configurations. In a similar scenario, Murwantara et al. [2014] use *linear regression*, *MLP*, *RT (REPTree)*, *Bagging+MLPs* and *Bagging+RTs* to infer *energy consumption*. However, although the authors conclude that *Bagging+RT* obtained the better results, they do not present further information about efficiency.

In the same context, Sánchez et al. [2014] propose a variety of metrics for the measurement of runtime NFPs, such as *memory consumption*, *response time*, and *security*. For a small-size SPL with 26 features, a few cross-tree constraints and 2 optimization criteria, both *Best-First Search Star* (BF*), and *Greedy Best-First Search* (GBFS) algorithms get the optimal solution with an execution time around 0,5 and 0,7 ms. Although this approach seems to be faster than the previous approaches, it does not consider feature interactions. In this scenario, the prediction of NFP measurements using static analysis have been further explored.

Khoshnevis and Shams [2017], dos Santos Neto et al. [2016] and Zanardini et al. [2016] propose a resource-usage-aware configuration approach based on a rigorous static analysis by a set of common design metrics related to source code inspection (such as cyclomatic complexity and coupling). These approaches mainly focus on the use of a non-dominated sorting genetic algorithm (NSGA-II metaheuristic) to search for valid configurations that satisfy previously defined product requirements. The evaluation shows that the proposed approaches are effective in generating a set of products to compose an optimal product portfolio (see Section 3.4.4.1). However, the authors do not present any information about the efficiency of the analyzer. In a similar scenario, Zanardini et al. [2016] consider a partial configuration from where only a set of minimal valid configurations is generated for each feature. Then, they use an off-the-shelf static analyzer to predict resource-usage for each minimal product, in order to search for the best configuration that fulfills the product requirements. The experiments have shown that the off-the-shelf static analyzer spent approximately 1,57 seconds to compute the NFP *footprint* for 768 valid products.

One step forward, Kolesnikov et al. [2013] use statically available information from the feature model and the internal source code structure to find relevant feature sets. Then, their approach predicts NFPs based on these features sets. Although this research presents a relevant progress in this field, an empirical evaluation to prove the approach validity is missing.

Camacho et al. [2016] propose a configuration language based on algebra to handle *cost* measurement (see Section 3.4.2.3). The language takes into account the order in which the features are configured (*i.e.*, product with the same features can have

different costs). The authors execute their proposed language in a distributed system and they show that the cost measurement for 97.648 valid configurations can be obtained in approximately 5 minutes by using 8 nodes and 32 workers.

Finally, in the interactive configuration scenario, Lettner et al. [2012] present an approach that uses the DOPLER tool suite to define and deploy business calculations instantly to end users after making configuration choices. This mechanism is also referred to as *non-functional properties prediction* and *non-functional properties value estimation*. Moreover, *feature interaction* can be referred to as *subsystem interaction* and *component interaction*.

3.4.1.3 Reuse of Non-Functional Property Measurements

This mechanism takes as input an EFM with measured NFPs (Section 3.4.1.2) and applies a reuse approach during a new configuration process. The reuse approach aims at reusing NFP values previously measured for features or products from the same SPL, or even at reusing previously specification of metrics. It minimizes the effort of performing a new analysis for each new product, making the configuration process faster. As an example, consider the smart-home product line in Figure 2.1. Suppose the *reliability* NFP for the feature `siren` already has been measured from an earlier configuration process and this measurement satisfies the new requirements related to the siren feature (*e.g.*, both target customers will use the same supplier to the feature `siren`). In this context, the measurement of the *reliability* NFP can be reused for the configuration of the new product. This mechanism is helpful for the automatic optimization of product configurations when dealing with NFP measurements (see Section 3.4.4). Additionally, even if the context has changed reuse is possible in case that NFPs are defined with the same metric specification. We observe in Table 3.3 three studies providing support for this mechanism [Horcas et al., 2017, Siegmund et al., 2015, Soares et al., 2015].

Soares et al. [2015] propose an approach (named NFP-RA) to define and reuse NFPs. This approach aims to define a systematic way for reusing previous NFPs measurements for different products, in order to minimize the effort of performing a new analysis for each new configuration. NFP-RA maintains a repository with NFP values that have already been analyzed (*i.e.*, estimated, measured, simulated, etc.). Thus, if the user desires to generate a new configuration, similar to some of the configurations already derived before, adequate NFPs are reused for the new configuration. Munoz [2017] implement this idea through an efficient collaborative repository, called HADAS, that stores the energy consumption measurements of complete valid configurations from several sources. To help developers perform a richer analysis, HADAS stores energy-related additional information, such as the energy consumption and the hardware used to obtain this data, along with its computational metric. Therefore, HADAS also allows the inclusion of different measurements for the same configuration. Overall, HADAS and NFP-RA aim at providing researchers a shared place to disseminate their empirical results.

Similar to HADAS, SPL Conqueror [Siegmund et al., 2015] allows users to export and import measurement specifications (*i.e.*, the metric definition of how an NFP can be measured). Thus, users may reuse the measurement setup in different contexts,

which reduces the experts' effort to define new metrics when new configurations have to be measured.

3.4.2 Mapping Product Requirements

This section presents three mechanisms supported by the *Mapping Product Requirements* activity: *(i)* Define Stakeholders' Preferences (A2a); *(ii)* Define Product Constraints (A2b); and *(iii)* Configuration Language Specification (A2c). Table 3.4 sketches which studies are supported by each mechanism. The first column identifies the study reference and the other columns are about the mechanisms. We use ● for studies providing support for the mechanism and ○ for studies without any support. Overall, 73 out of 112 approaches (65.2%) provide some contribution to this group of mechanisms, with a larger portion (56.3%) dedicated to handling product constraints.

3.4.2.1 Defining Stakeholder Preferences

To guide the product configuration process, this mechanism takes as input the stakeholders' priorities in terms of features, NFPs, or configurations. Note that stakeholders can have completely different requirements and priorities when they use a particular variant in different application scenarios, such as country, city, or region. Therefore, this mechanism is employed in scenarios where a decision considers multiple product requirements, and a trade-off among these requirements is required in order to better satisfy stakeholders' needs (*e.g.*, in multi-objectives optimization scenarios where there are a set of optimal configurations as solution), while still meeting configuration rules and resource restrictions.

In Table 3.4, we observe that 19 approaches out of 73 approaches, constituting 26%, support this mechanism. Among them, 10 approaches support NFP preferences [Asadi et al., 2014, Bagheri et al., 2012a, Noorian et al., 2017, Ognjanovic et al., 2012, Parra et al., 2012, Rezapour et al., 2015, Sánchez et al., 2014, Zanardini et al., 2016, Zdravkovic et al., 2015, Zheng et al., 2017a], 8 approaches support feature preferences [Bajaras and Agard, 2015, Bashari et al., 2014, dos Santos Neto et al., 2016, Noir et al., 2016, Pereira et al., 2017, Tan et al., 2014b, Wittern et al., 2012, Zhang et al., 2014], and one approach provides support for product preferences [Martinez et al., 2015a]. Different techniques are used to specify to what degree a specific NFP is preferred over others. There are three main techniques in SPL configuration for eliciting prioritization and finding priorities: *Analytical Hierarchical Process* (AHP), *Fuzzy Logic* (FL), and *Weighting Factors*.

Analytical Hierarchical Process (AHP). In AHP, pair-wise based comparison among different factors (*i.e.*, features, NFPs, and configurations) is performed to produce a ranked list of configurations or features. Pair-wise comparison is based on the *relative importance* of stakeholders' desires through a comparison square matrix $M[n,n] = \{M_{i,j} = \sigma | 1 \leq i,j \leq n\}$, where n represents the number of configurations, features, or NFPs (*i.e.*, factors); and σ represents the relative importance of the i^{th} factor with regards to the j^{th} factor. In this case, $n \times \frac{(n-1)}{2}$ comparisons among features are performed. Asadi et al. [2014] formalize the *relative importance* between two factors using the terms: *equality* (=), *slight value* (<), *strong value* (>), *very*

Study	A2a	A2b	A2c	Study	A2a	A2b	A2c
Zanardini et al.	●	●	●	Mauro et al.	○	●	○
Noorian et al.	●	●	○	Ruiz et al.	○	●	○
Pereira et al.	●	●	○	Sharifloo et al.	○	●	○
Zheng et al.	●	●	○	Sion et al.	○	●	○
dos Santos Neto et al.	●	●	○	Xue et al.	○	●	○
Noir et al.	●	●	○	Benali et al.	○	●	○
Zdravkovic et al.	●	●	○	Henard et al.	○	●	○
Asadi et al.	●	●	○	Leite et al.	○	●	○
Sánchez et al.	●	●	○	Lian and Zhang	○	●	○
Bagheri et al.	●	●	○	Mazo et al.	○	●	○
Ognjanovic et al.	●	●	○	Pascual et al.	○	●	○
Parra et al.	●	●	○	Siegmund et al.	○	●	○
Wittern et al.	●	●	○	Foster et al.	○	●	○
Bajaras and Agard	●	○	○	Guo et al.	○	●	○
Martinez et al.	●	○	○	Murguzur et al.	○	●	○
Rezapour et al.	●	○	○	Olaechea et al.	○	●	○
Bashari et al.	●	○	○	Wang and Pang	○	●	○
Tan et al.	●	○	○	White et al.	○	●	○
Zhang et al.	●	○	○	Chen et al.	○	●	○
Ter Beek et al.	○	●	●	Cubo et al.	○	●	○
Bąk et al.	○	●	●	Ge et al.	○	●	○
Camacho et al.	○	●	●	Karimpour and Ruhe	○	●	○
Santos et al.	○	●	●	Kramer et al.	○	●	○
Myllärniemi et al.	○	●	●	Saller et al.	○	●	○
Ochoa et al.	○	●	●	Sayyad et al.	○	●	○
Roos-Frantz et al.	○	●	●	Lettner et al.	○	●	○
Adjoyan and Seriai	○	●	○	Mazo et al.	○	●	○
Guedes et al.	○	●	○	Mussbacher et al.	○	●	○
Horcas et al.	○	●	○	Ostrosi et al.	○	●	○
Kifetew et al.	○	●	○	Qin and Wei	○	●	○
Nieke et al.	○	●	○	Schroeter et al.	○	●	○
Oh et al.	○	●	○	Wang and Ng	○	●	○
Umpfenbach et al.	○	●	○	Eichelberger et al.	○	○	●
Wang et al.	○	●	○	Triando et al.	○	○	●
Zheng et al.	○	●	○	Winkelmann et al.	○	○	●
Ayala et al.	○	●	○	Acher et al.	○	○	●
Hierons et al.	○	●	○				

Table 3.4: Literature supporting the activity: *Mapping Product Requirements* (A2a: *Define Stakeholders' Preferences*, A2b: *Define Product Constraints*, A2c: *Configuration Language Specification*).

strong ($\gg$), and *extreme value* ($\propto$). For example, if the stakeholders mention that *security* is *extremely* more preferable and relevant than *cost* (*i.e.*, *security* $\propto$ *cost*) and *cost* has a strong value against *response time* (*i.e.*, *cost* $>$ *response time*), the AHP technique converts the stakeholders' judgments to numerical values and numerical priorities are computed for each of the competing features. The traditional values used for representing preferences among two factors are 1, 3, 5, 7, and 9; where the highest values denote a higher importance. Values of the matrix are then normalized based on *eigenvalues* estimation [Saaty, 1987]. The v eigenvector is a non-null vector such that given the matrix M and an eigenvalue λ, we have the following relation $Mv = \lambda v$. Finally, it produces a feature (or configuration) ranking.

6 out of 19 approaches adopt AHP [Asadi et al., 2014, Bashari et al., 2014, Noorian et al., 2017, Ognjanovic et al., 2012, Zanardini et al., 2016, Zhang et al., 2014]. For instance, Zhang et al. [2014] introduced the usage of positive and negative impacts over NFPs. An NFP positive impact measures the importance of selecting a feature for a final product configuration, while the negative impact measures the importance of deselecting that feature. Then, just the subset of features with positive or negative impact over a particular NFP are considered in the matrix, reducing the number of manual comparisons to be performed by stakeholders. In a similar scenario, Tan et al. [2014b] adopt the ELO rating system which is also a pair-wise comparison approach, originally used to compute the performance between players (*i.e.*, competitors). In this approach, features assume the role of players and a feature ranking is computed based on stakeholders judgments on a pair of features in terms of satisfying a given NFP.

Fuzzy Logic (FL). FL is mainly used to deal with vague or imprecise judgments. *Fuzzy Requirements* refer to stakeholders' requirements surrounded by vagueness and uncertainty [Bajaras and Agard, 2015, dos Santos Neto et al., 2016]. As an example, consider the statement *"Stakeholders only agree in paying a high cost for a product if its response time is minimum, otherwise, they only pay a low price"*. In this scenario, the stakeholder judgments: *high cost* and *minimum response time* can be represented as fuzzy functions by mapping a *cost* and *response time* scale, such as the one in Figure 3.10. This figure shows that a specific smart-home configuration has rather slow response time but results in high cost. Thus, we can deal with a large range of values, instead of considering only two or few classifications: cheap and expensive, or slow and fast. Consequently, we can offer user much more suitable products. This method can be successfully combined with the AHP method, known as *Fuzzy Cognitive Mapping* (FCM). There are 4 approaches in Table 3.4 that implement FL [Asadi et al., 2014, Bagheri et al., 2012a, Bajaras and Agard, 2015, dos Santos Neto et al., 2016], where one of them adopts FCM [Asadi et al., 2014].

Weighting Factors. One of the methods used for expressing *stakeholders' preferences* is the simple employment of absolute *weighting factors*, such as the manual ranking specification of features' relevance. In Zheng et al. [2017a], users define weights to NFPs based on historical information from previous customers' comments. In a similar scenario, Zdravkovic et al. [2015] produce a consumer preference model based on NFP ranking for specific consumer segments that is used as the

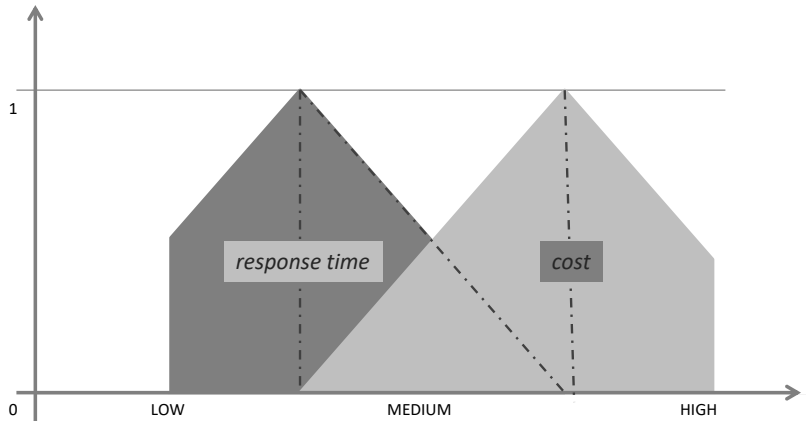


Figure 3.10: The semantic annotation of the properties *response time* and *cost* for a specific configuration of a *smart-home* product for the feature model in Figure 2.1.

input to a goal-oriented model (see Section 3.4.4.1). The remaining approaches [Martinez et al., 2015a, Noir et al., 2016, Parra et al., 2012, Pereira et al., 2017, Rezapour et al., 2015, Sánchez et al., 2014, Wittern et al., 2012, Zdravkovic et al., 2015, Zheng et al., 2017a] consider simple weighting schemes to reflect stakeholders' preferences. They define *weighting factors* related to a complete configuration as $F(x) = w_1 f_1 + w_2 f_2 + \cdots + w_n f_n$ where w_i is the weight assigned to the i^{th} feature, f_i is the status (*i.e.*, 1 for selected and 0 for deselected) of the set of n features, and $i, n \in \mathbb{N}$. Then, to better satisfy the user needs, the function $F(x)$ is maximized [Pereira et al., 2017]. The same idea is adopted in the context of NFP weights, where the value of an NFP corresponds to the aggregation sum of the NFP from each selected feature. However, since each NFP has different measuring units (*e.g.*, milliseconds for response time, megabytes for memory consumption, etc) and differing orders of magnitude, its value should be normalized before computing the weighted measure of a complete configuration. Sánchez et al. [2014] propose the employment of Equation 3.1 to normalize NFP measurements.

$$F(x) = \sum_{i \in P} w_i \times \frac{I_i(C) - \mu_i}{\sigma_i} \tag{3.1}$$

where P is the set of quality properties, w_i is the weight of the i^{th} quality property, such that $\sum_{i=1}^{|P|} w_i = 1$ (they are defined by stakeholders or measured by a domain-specific formula), $I_i(C)$ corresponds to the configuration aggregated measurement of the i^{th} property, and μ_i and σ_i are the average value and the standard deviation of $I_i(C)$ for all valid configurations. This formulation is responsible of converting evaluation measures in a normalized and comparable value, *i.e.* $I_i \in [0, 1]$.

Stakeholders' preferences are also referred to as *non-functional property preferences*, *feature preferences*, *product preferences*, and *fuzzy preferences*.

3.4.2.2 Defining Product Constraints

This mechanism takes as input specifications of decision rules over NFPs to guide the product configuration process. The definition of decision rules allow the product

alignment with business interests and stakeholder needs. These rules, known as *product constraints*, define a set of limited resources that cannot be exceeded. They are constraints that are defined as *equalities* or *inequalities* over the aggregation of a specific NFP. In this way, the set of NFPs (*e.g.*, *cost*) are aggregated (*e.g.*, *sum*) and an inequality (*i.e.*, $\leq, <, >, \geq$) or equality (*i.e.*, $=$) relation is defined. These constraints are useful for defining resource limits like budget boundaries [Ochoa et al., 2015, Pereira et al., 2017]. For instance, Bagheri and Ensan [2014b] employ hard limits in order to define *reliability* lower and upper boundaries when configuring an SPL. Also *filters* may be defined as an automated product constraint, where stakeholders select a set of key hard limits to reduce the SPL variability space. Then, a subset of products S' are obtained $S' \subseteq S$ from a complete set of products or solutions S. Bąk et al. [2016] provide visual support to filter product constraints.

Product constraints complement the interdependencies expressed through the feature model, restricting the set of valid configurations. In Table 3.4, we observe that 63 studies support this mechanism. This mechanism is usually used with the configuration optimization mechanism defined in Section 3.4.4.1. 65% of the studies supporting this mechanism also support optimization. This mechanism is also referred to as *decision rules*, *stakeholders constraints*, *resource constraints*, *hard limits*, and *non-functional property constraints*.

3.4.2.3 Configuration Language Specification

This mechanism describes EFMs, product requirements, and configurations using a representative textual configuration language. This language is helpful to support the activities of *automatic configuration* (Section 3.4.4) and *configuration adaptation* (Section 3.4.5) of large-scale SPLs, since the graphical representation of a large set of information tend to add an overwhelming layout, impacting understandability.

Inspired by Olaechea et al. [2012] and Mendonça et al. [2009], the Listing 3.1 shows the textual extended variability modeling language for the smart-home product line in Figure 2.1. The lines 8–27 represent the feature-tree with symbol $: r$ denoting the feature root. The symbols $: m$ and $: o$ denoting mandatory and optional features, respectively. In addition, $: g[1, *]$ and $: g[1, 1]$ denote a group cardinality alternative non-exclusive and exclusive, respectively. Furthermore, $: p$ denotes NFPs.

Lines 29–33 define the cross-tree-constraints and lines 35–38 define the hard cross-tree-constraints. Lines 41-68 represent the product requirements to search for an optimal instance of smart-home represented by *optimalHouse* in line 43. The line 43 declares *optimalHouse* as a singleton concrete sub-type of *SmartHome*, effectively defining an instance of the smart-home product line. Note that, in line 42, *SmartHome* is declared as abstract, meaning that there are no instances of this type. *SmartHome* in lines 44 and 45 also defines two additional attributes, each starting with `total_`. These attributes sums up the *response time* and *reliability* of the selected features. `Feature.reliability` returns the set of reliability values for the specified features. Features interactions are represented by adding conditional terms to the sum (see line 49 for an example). Lines 52–57, 59–63, and 65–67 represent product constraints (Section 3.4.2.2), optimization objectives (Section 3.4.4.1), and stakeholders' preferences (Section 3.4.2.1) respectively.

Listing 3.1: Extended variability language adapted from Olaechea et al. [2012] and Mendonça et al. [2009] for the *smart-home* product line in Figure 2.1.

```
<attribute-types>
  abstract Feature;
  response-time; reliability; cost; memory: double;
  customer-satisfaction; security: String;
</attribute-types>

<feature-model>
  <feature-tree>
  :r SmartHome;
    (...)
      :m security: Feature
        :o alarm: Feature
          :siren: g[1,*]
            :p response-time = 500
            :p reliability = 98
            :p cost = 600
            :p satisfaction = "high"
            :p security = "high"
          :visual: Feature
          :blinking_lights: Feature
            :p response-time = 5
            :p reliability = 99
            :p cost = 150
            :p customer-satisfaction = "medium"
            :p security = "low"
    (...)
  <\feature-tree>

  <constraints>
    c1: ~ (sensor or detection) or alarm;
    c2: ~ bliking_lights or automatic;
    (...)
  </constraints>

  <hard-constraints>
    h1: ~ monitoring or (camera.memory > 8);
    h2: ~ blinking_lights or (SmartHome.total_response-time < 1000);
  </hard-constraints>
<\feature-model>

<requirement>
  abstract SmartHome;
  optimalHouse: SmartHome;
  total_response-time = sum(Feature.response-time)
  total_reliability = sum(Feature.reliability)
  (...)

  <interactions>
    sum(siren.reliability) + sum(blinking_lights.reliability) = 100;
  </interactions>

  <product-constraints>
    total_customer-satisfaction and total_security = "high";
    optimalHouse.total_response-time < 1;
    optimalHouse.total_reliability = 100;
    optimalHouse.total_cost < 5000;
  </product-constraints>

  <optimization>
    max(optimalHouse.total_security);
    max(optimalHouse.total_ reliability);
    min(optimalHouse.total_response-time);
  </optimization>

  <preferences>
    security {$\propto$} cost;
  </preferences>
</requirement>
```

In Table 3.4, we observe 13 approaches supporting this mechanism. These approaches propose a textual modeling language for large scale management of SPL configurations. We split the contributions in the literature into four main groups: *(a)* managing specification of configurations, *(b)* managing NFPs, *(c)* managing product requirements, and *(d)* managing configurations at runtime.

Managing Specification of Configurations. Triando et al. [2016] propose an *Extended Product Selection Language* (EPSL) to specify configurations as sets of selected and deselected features by referring to other created configurations. EPSL uses a set of logical operations (*i.e.*, union, intersection, complement) over features and sets of previous configurations. This results in a more concise and less repetitive language in comparison with FeatureIDE [Pereira et al., 2016b], that describes the complete list of features for each new created configuration.

Managing NFPs. To model variable systems, Camacho et al. [2016] use a formal configuration language based on algebra operational semantic rules [Hillston, 2005] to support product management decisions and cost estimations. In a similar scenario, Myllärniemi et al. [2015] use a textual modeling language based on weight constraint semantic rules [Simons et al., 2002] to represent security variability.

Zanardini et al. [2016] use the *Micro Textual Variability Language* (μTVL) [Clarke et al., 2010] to deal with NFPs. μTVL is a text-based feature modeling language to describe EFMs. However, μTVL is not a programming language that generates executable Java code from derived product configurations. Therefore, Zanardini et al. [2016] propose an approach to link feature models specified in μTVL with deltas to support the direct propagation of NFPs to the associated code artifacts during product derivation. One step further, Winkelmann et al. [2016] introduce an extension of the programming language DeltaJ, called parametric DeltaJ. Parametric DeltaJ allows the propagation of NFP values from an EFM to Java code artifacts via product configuration. Parametric DeltaJ also allows the description of hard cross-tree-constraints and management of feature interactions.

Managing Product Requirements. On a broad view, ClaferMoo [Bąk et al., 2016], CoCo [Ochoa et al., 2015], and FaMa-OVM [Roos-Frantz et al., 2012] propose a variability modeling language to support the representation of EFMs and product requirements. These approaches use state-of-the-art solvers to find a set of features that satisfy the feature model constraints and fulfill the product requirements. In addition, ClaferMoo supports the representation of feature interactions and CoCo aims at managing conflicts among stakeholders' configurations performed over a feature model (see Section 3.4.3.3). One step further, FAMILIAR [Acher et al., 2013] and EASy-Producer [Eichelberger et al., 2016] provide domain-specific modeling languages to manage a set of inter-related feature models. ClaferMoo, CoCo, FaMa-OVM, FAMILIAR, and EASy-Producer integrate an interactive configuration view with a variability modeling language.

Managing Configurations at Runtime. In the context of dynamic configuration of SPLs at runtime, Ter Beek et al. [2016] present a set of probabilistic feature-oriented languages: FLan, PFLan, and QFLan. QFLan extends the two languages

FLan and PFLan. QFLan considers arithmetic relations among features, NFPs, and actions at run-time to analyze probabilistic aspects of SPLs configurations and behavior, such as the impact to install and replace features at a specific moment or in a specific order. In a similar scenario, Santos et al. [2016] use Promela [Holzmann, 2003] to specify the dynamic SPL behavior and validate product requirements. Promela consists of variables defining features, the environment and configurations behavior. It defines a process to trigger the adaptation rules (activation and deactivation) to manage context changes at runtime.

Product configuration language is also referred to as *product line language*, *extended variability language*, *variability instantiation language*, *product selection language*, *product instantiation language*, *configuration instantiation language*, and *textual variability modeling language*.

3.4.3 Manual Configuration Process

This section presents five mechanisms supported by the *Manual Configuration* activity: *(i)* Visualization Techniques (A3a); *(ii)* Constraint Checking and Propagation (A3b); *(iii)* Solving Conflicts among Configurations (A3c); *(iv)* Mapping Stakeholders Tasks (A3d); and *(v)* Recommender System (A3e). Table 3.5 sketches which studies are supported by each mechanism. The first column identifies the study reference and the other columns are about the mechanisms. We use ● for studies providing some support to the mechanism and ○ for studies without any support.

3.4.3.1 Visualization Techniques

This mechanism considers the use of visualization techniques combined with interactive techniques to support decision makers during the SPL configuration process. It is supported by automated tools, called *configurators*. Configurators aim to assist decision makers particularly configuring large feature models with complex dependencies between the variants. Configurators are needed especially *(i)* when the variability has grown large; *(ii)* there are complex dependencies between the variants; or *(iii)* the configuration task is not done by the SPL engineers. Therefore, configurators need to easily guide decision makers in the product configuration process step-by-step by communicating and explaining the variability of an SPL. It should allow even a non-technical user to perform the configuration task.

44 out of 112 studies (39.3%) provide visual support to handle the manual interactive configuration process, as summarized in Table 3.5. In the next sections, we discuss how each study supports this activity. Note that studies supporting mechanisms in this activity also may support the *automatic* and *adaptation* activities (see Figure 3.5). For example, after or before automatically optimizing a configuration, decision makers may interactively select or deselect features by means of the use of configurators. Otherwise, some configurators do not provide support to any other mechanism in this group, this is why they offer some support to the activities of *mapping NFPs* (A1) and (or) *mapping product requirements* (A2). For instance, in previous work [Pereira et al., 2017], the authors proposed an approach that supports stakeholders specifying NFPs, their preferences and product constraints to then automatically configure a product. Therefore, by considering the overall contribution

Study	A3a	A3b	A3c	A3d	A3e
Eichelberger et al.	●	●	●	●	○
Ostrosi et al.	●	●	●	●	○
Schroeter et al.	●	●	●	●	○
Lu et al.	●	●	●	○	●
Schwäger and Westfechtel	●	●	●	○	○
Ochoa et al.	●	●	●	○	○
Nöhrer et al.	●	●	●	○	○
Thurimella and Bruegge	●	●	●	○	○
Rabiser et al.	●	●	○	●	○
Pereira et al.	●	●	○	○	●
Bashari et al.	●	●	○	○	●
Mazo et al.	●	●	○	○	●
Zhang et al.	●	●	○	○	●
Tan et al.	●	●	○	○	●
Hajri et al.	●	●	○	○	○
Pereira et al.	●	●	○	○	○
Sion et al.	●	●	○	○	○
Myllärniemi et al.	●	●	○	○	○
Behjati et al.	●	●	○	○	○
Urli et al.	●	●	○	○	○
Klambauer et al.	●	●	○	○	○
Pleuss and Botterweck	●	●	○	○	○
Zheng et al.	●	○	○	●	●
Zhang and Becker	●	○	○	●	○
Fang et al.	●	○	○	●	○
Murguzur et al.	●	○	○	●	○
Martinez et al.	●	○	○	●	○
Noir et al.	●	○	○	○	●
Galindo et al.	●	○	○	○	●
Tan et al.	●	○	○	○	●
Khoshnevis and Shams	●	○	○	○	○
Kifetew et al.	●	○	○	○	○
Nieke et al.	●	○	○	○	○
Noorian et al.	●	○	○	○	○
Pereira et al.	●	○	○	○	○
Bąk et al.	●	○	○	○	○
Zanardini et al.	●	○	○	○	○
Mazo et al.	●	○	○	○	○
Siegmund et al.	●	○	○	○	○
Asadi et al.	●	○	○	○	○
Acher et al.	●	○	○	○	○
Heider et al.	●	○	○	○	○
Lettner et al.	●	○	○	○	○
Mazo et al.	●	○	○	○	○
Brink et al.	○	●	●	○	○
Zhao et al.	○	●	●	○	○
Safdar et al.	○	●	○	○	○
Chavarriaga et al.	○	●	○	○	○
Martinez et al.	○	○	○	○	●
Bagheri et al.	○	○	○	○	●

Table 3.5: Literature supporting the activity: *Manual Configuration Process* (A3a: *Visualization Techniques*, A3b: *Constraint Checking and Propagation*, A3c: *Solving Configuration Conflicts*, A3d: *Mapping Stakeholder Tasks*, A3e: *Recommender System*).

of such techniques, they propose a semi-automatic approach to support the configuration process. For a complete overview of state-of-the-art configurators we refer to our previous paper [Pereira et al., 2015].

3.4.3.2 Constraint Checking and Propagation

This mechanism takes as input a set of selected and deselected features. Then, it translates the feature model into a specific representation (*e.g.*, propositional logic) and using exact algorithms checks the structural constraints of the feature model and automatically informs the decision maker as soon as a selection state becomes invalid. Therefore, this mechanism concentrates on validating feature models and managing the configuration process by avoiding the introduction of product inconsistencies.

In Table 3.5, we observe 26 studies offering support for this mechanism: 5 studies use the decision propagation mechanism implemented in FeatureIDE [Bashari et al., 2014, Pereira et al., 2016b,c, Rabiser et al., 2016, Sion et al., 2016]. FeatureIDE [Meinicke et al., 2017] provides an interactive configuration support including automated conformance checking of configuration and decisions propagation. It uses *constraint logic programming* to provide immediate feedback on the effect of choices made (interactively) by decision makers, not allowing them to set an invalid selection state. As an example, consider the smart-home product line in Figure 2.1. After the user specifies that the `sensor` feature is selected the tool responds by automatically selecting `alarm` as well, since there is a *requires* constraint between `sensor` and `alarm`. This ensures that the configuration is valid at all points in time. Behjati et al. [2014] apply this idea to the *Integrated Control Systems* (ICS) domain.

Similar to FeatureIDE, PUMConf [Hajri et al., 2016] uses *Natural Language Processing* (NLP) to interactively check the consistency of SPLs. For each decision, PUMConf automatically generates use cases and domain models for the configured product. Since PUMConf focuses on managing requirements, it is integrated with an industrial requirements management tool: IBM DOORS. Kumbang [Myllärniemi et al., 2015] complements this approach by proposing a mechanism to link requirements to feature models. Then, decision makers follow making decisions over requirements, instead of features. After each decision, Kumbang checks the configuration for consistency and completeness. In case of inconsistency, it is reported to the user. Otherwise, decision propagation is applied to both feature model and requirements.

In a similar scenario, Zen-CC [Lu et al., 2016a] and QAPCTool [Zhang et al., 2014] propose a configuration inference system, which can automatically infer decisions based on configuration data from requirements and selected features. Therefore, Zen-CC advances one step further by incrementally producing conformance checking results and employing decision propagation strategies by only updating a small set of nodes of a tree, which are related to the currently specified requirements and features, instead of checking all the conformance rules at each configuration step. Zen-CC and QAPCTool are implemented as a component of Zen-Configurator and QAMTool, respectively.

Thurimella and Bruegge [2012] propose a consistency checker for a rationale approach (*i.e.*, questions, options, and criteria) to guide distributed stakeholders to

make new decisions. Their approach is based on past data from previous decisions to support new decisions.

Mazo et al. [2014a] and Tan et al. [2013] propose a collection of recommendation heuristics to reduce the number of configuration steps during the interactive configuration process by pointing out the order to select features. The decision maker should always start from this set of features since, based on the feature dependencies, the decisions made on these features will imply decisions on the rest of the features. Beyond to reducing the number of features visited in the interactive configuration process, these approaches also minimizes the computation time required by the solver to propagate the configuration choices.

Besides calculating the consequences of previously made decisions, S2T2 Configurator [Pleuss and Botterweck, 2012] supports the decision maker with visual explanations why a decision cannot be made as intended (due to a constraint specified in the feature model). Thus, whenever a decision is applied to the model, the consequences of that decision are highlighted to decision makers to provide them with an overview of what happened. Therefore, in contrast to the previous approaches, S2T2 Configurator allows the decision maker to ask for an explanation and track what is happening when a certain feature is selected or deselected. In addition, FeatureIDE [Pereira et al., 2016b] proposes a highlight technique to guide decision makers to a valid configuration, based on unsatisfied constraints of the feature model (see Chapter 7 for more detail).

Several authors [Chavarriaga et al., 2014, Klambauer et al., 2013, Safdar et al., 2017, Zhang et al., 2014] propose an automatic decision propagation mechanism to ensure consistence validity of a configuration belonging to multi-SPLs, which Safdar et al. [2017] named *Cross Product Lines* (CPL) *rules*. Since individual configurations from multiple dependent SPLs are not necessarily compatible, using CPL rules constraints between SPLs are expressed by inter-model dependencies and immediate consistence among them is enforced. This way, each stakeholder decides over its own decisions, then inter-model dependencies restricts the selection of particular features in other feature models. Conflicts are managed by avoiding the introduction of inconsistencies while using a decision propagation strategy. Klambauer et al. [2013] present a tool-supported approach for dynamically monitoring distributed configurations of multi-SPLs by multiple users, called DOPLER. To ensure valid SPL configurations, the tool also allows detecting violations of product requirements and providing immediate feedback to stakeholders. Complementary, Chavarriaga et al. [2014] propose an algorithm that supports conflict explanation through the decision propagation process. In contrast to previous work, other approaches temporarily allow the user to specify conflicting decisions, to decide about the inconsistencies later [Brink et al., 2015, Eichelberger et al., 2016, Lu et al., 2016a, Nöhrer et al., 2012a, Ochoa et al., 2015, Ostrosi et al., 2012, Schroeter et al., 2012, Schwäger and Westfechtel, 2016, Thurimella and Bruegge, 2012, Zhao et al., 2012]. It allows users to continue working despite the presence of errors. For further information about these approaches, we refer the reader to Section 3.4.3.3.

Also in the automatic configuration scenario, capturing rules and propagating these rules are the key to enable configuration automation. For further information about these approaches, we refer the reader to Section 3.4.4.

This mechanism is also referred to as *consistency checking*, *correctness checking*, *model consistency*, *conformance rules*, *decision propagation*, *propagation strategies*, *product line reasoning*, and *product line verification*.

3.4.3.3 Solving Configuration Conflicts

Configuration conflicts may occur due the user making conflicting decisions during the interactive manual configuration process. Conflicts may also be due to configuration of multi-SPLs and configuration by multiple stakeholders.

Different stakeholders can configure a set of different products from a unique or multiple feature models. To create a unique final product, this mechanism takes as input the set of created configurations and follows merging these distinct configurations satisfying as much as possible the final product requirements. A conflict among configurations arises when the (de)selection of a feature from a specific stakeholder configuration invalidates another configuration made by a different stakeholder due to feature model constraints or requirements specification. This happens because the stakeholders have different expertise and points of view, as well functional and non-functional requirements. This becomes an issue especially when the SPL evolves over time (*i.e.*, to address changing customer, market, or technology requirements).

In Table 3.5, we observe 10 approaches supporting this mechanism. EASy-Producer [Eichelberger et al., 2016] develops a variability-rich configurator to manage the consistency of individual configurations from different stakeholders in distributed multi-SPLs. In a similar scenario, Brink et al. [2015] propose an approach to model an SPL as multi-SPLs and then use a tool-based algorithm to support the integration of individual configurations. The implemented algorithm searches for a common configuration from multi-SPLs that is in accordance with the specified requirements and feature model constraints. In addition at solving conflicts among several static configurations, PuMA [Schroeter et al., 2012] handles the variability among multi-SPLs at runtime.

Some authors [Ochoa et al., 2015, Ostrosi et al., 2012, Zhao et al., 2012] propose an approach to solve conflicts among multiple distributed stakeholders configurations performed over the same feature model. They use *Satisfiability solvers*, *Fuzzy Configuration Grammar based agents* and *similarity retrieval* to manage conflicts. Beyond satisfying the set of decision rules, these approaches consider product requirements to search for a set of non-conflicting features that better fulfill business needs. In addition, Ostrosi et al. [2012] implement different stakeholder perspectives to assist the collaborative and distributed configuration processes (see Section 3.4.3.4).

Sysiphus [Thurimella and Bruegge, 2012] captures and shares rationale information from multiple stakeholder decisions to solve conflicts. Rationale includes the reasoning of stakeholders and the justification for a decision (*i.e.*, selection criteria and arguments). It allows issue-based communication between distributed stakeholders. On the basis of this information, resolutions are negotiated for resolving configuration conflicts. In Sysiphus, the rationale is integrated into variability models and it is named as *Issue-Based Variability Management* (IVM). In a similar scenario, SuperMod [Schwäger and Westfechtel, 2016] offers capabilities for collaborative SPL configurations through distributed version control by having the users working in a

single-variant workspace. The tool adopts the iterative check-out, modify, and commit functionalities known from version control systems for variability management.

Finally, instead of considering multi-stakeholder configurations or even multi-SPLs, HUMUS [Nöhrer et al., 2012a] focuses on solving configuration conflicts performed by a unique user in a specific model later without misguiding the user along the way. Also in this scenario, Zen-Configurator [Lu et al., 2016a] presents an automatic approach (named as Zen-FIX) to optimally recommend solutions to solve conflicts generated when constraints are violated. Furthermore, in the automatic SPL configuration scenario, dynamic SPL configurations address the problem of finding a suitable new configuration when the current configuration is invalidated by a context change (see Section 3.4.5.2 for further information on this mechanism).

This mechanism is also referred to as *constraint violations*, *configuration inconsistencies*, *non-conformity resolving*, and *conflicting configurations*.

3.4.3.4 Mapping Stakeholder Tasks

Usually, there are multiple stakeholders that participate in the same product configuration process. In accordance with Zhao et al. [2012], cooperative configuration can reduce the search time for the desired product. However, in the process of cooperative configuration, conflicts may occur. Consequently, to avoid such conflicts, configurators need to be able to guide a variety of users (including business leads, project managers, engineers, customers, end users, among others), based on their knowledge and expertise. Therefore, this mechanism takes as input the feature model and the stakeholders' profile, and provides customized views for stakeholders that is in accordance with their expertise (*i.e.*, depending of their tasks, rights, roles, and responsibilities). For example, while product engineers need low-level details to make choices during the product configuration process, such as code measurements; customers just need high-level information. In this case, code and other complex information should be hidden from customers. This mechanism supports decision makers by restricting and displaying only a set of information that are relevant to them based on their non-functional requirements. We observed 9 studies supporting this mechanism (see Table 3.5).

This mechanism is also referred to as *task-specific visualizations*, *user-specific visualizations*, *customer-specific visualizations*, *instantiation knowledge*, *licensing reasons*, *stakeholders points of view*, *stakeholder perspectives*, and *stakeholders guidance*.

3.4.3.5 Recommender System

Decision makers often have problems understanding the implications and effects of the choices they make during the product configuration process. The main reason is due the complexity of feature models and the heterogeneity of stakeholders with different knowledge about the SPL. To overcome this challenge, a recommender system takes as input a feature model and the product requirements, and returns a set of recommendations to guide decision makers.

We observed 12 approaches supporting this mechanism (Table 3.5). We classify these approaches into two groups: *feature-based* and *product-based* recommendations. While a *feature-based* recommendation aims to predict the utility of each

feature for the stakeholders, a *product-based* recommendation aims to predict the utility of a complete set of features, which forms a valid configuration. In this section, we highlight the particularities of these techniques.

Feature-Based Recommendation. Zen-Configurator [Lu et al., 2016a] recommends in which order decisions should be made to minimize the number of manual configuration steps. For each decision maker interactive choice, the tool automatically checks the conformance of the configuration (Section 3.4.3.2). In case any conformance constraint is violated, recommendations about how to solve the non-conformity are also provided. Then, decision makers can choose one of the recommendations to fix the non-conformity. Otherwise, a new optimized configuration order is dynamically recommended to decision makers. In a similar scenario, VariaMos [Mazo et al., 2014a] and Tan et al. [2013] propose a collection of recommendation heuristics to improve the interactivity of SPL configuration. In addition to minimizing the number of configuration steps, VariaMos also minimizes the time required by the solver to propagate the configuration choices.

As we have shown in Section 3.4.2.1, Fuzzy and AHP techniques have also been employed to rank features (and configurations) of an SPL. Bagheri et al. [2012a] use fuzzy variables to model and represent feature preferences over NFPs. Each feature is annotated with a fuzzy function for a given NFP to show how well that feature is able to contribute to the given property. The approach develops a requirements knowledge base which is used to make feature recommendations. However, the fuzzy annotation task of *each feature* involves significant manual efforts from domain experts, which is a tedious and time-consuming task.

In accordance with Zhang et al. [2014], it is much easier for stakeholders to make pair-wise comparisons than to judge the overall impact of each feature or a combination of features. Zhang et al. [2014] propose a pair-wise approach to estimate features' contributions over decision makers non-functional requirements. From a randomly selected pair of features, domain experts identify which one is more relevant in terms of satisfying a given NFP. Then, a ranking of recommended features is produced in terms of their relative importance. Instead of randomly selecting a pair of features, AUFM tool [Bashari et al., 2014] dynamically builds a decision model by considering the structural characteristics of SPLs. However, both techniques may generate inconsistencies due to conflicting judgments from multiple stakeholders. To overcome this challenge and provide more accurate feature rankings, Tan et al. [2014b] adopt an ELO rating approach. In this approach, each feature is assigned with a numerical rating based on their relevance over NFPs. In the end, it produces a comprehensive chart of feature contributions ranking on different NFPs.

A recommender system has been proposed in [Pereira et al., 2016c] to predict features relevance for a current user based on configuration historical data from previous users. The predictions are displayed as a 5-star feature score to guide the user through a step-wise selection of features. In a similar scenario, Zheng et al. [2017a] propose a recommender system of stakeholders' rationale preferences related to non-functional requirements based on the purchase history of most past customers. This is the unique approach that recommends non-functional requirements, instead of features or configurations, to guide stakeholders' decisions.

Product-Based Recommendation. Galindo et al. [2015b] propose an approach, named Invar, which aims at supporting the product configuration of multi-SPLs. Invar presents a set of questions and answers to decision makers through a unified configuration perspective over heterogeneous variability models. Based on decision makers' answers, a valid and complete configuration is recommended to them.

Martinez et al. [2015a] use an interactive genetic algorithm to create a historical dataset of valid SPL configurations. Based on stakeholders judgments over configurations likability on this dataset, it uses similarity measurements to create a prediction ranking for all possible configurations (including those that are not in this dataset).

Noir et al. [2016] propose a recommender system for assisting decision makers choose a single configuration resulting from a multi-objective optimization (Section 3.4.4.1). Based on the specification of product requirements, this approach uses aggregation metrics to compute prediction relevance for the set of resulting configuration. Noir et al. [2016] provide a graphical and textual explanation of the multi-criteria configuration results. The graphical explanation maps NFP values onto a common scale representing the degree of satisfaction from each resulting configuration. The textual explanation explains how the prediction scores are computed. After the decision maker selects the most desirable configuration, he can justify its design choices in the tool to improve future recommendations.

This mechanism is also referred to as *product prediction*, *configuration prediction*, *feature prediction*, *product recommender*, and *feature recommender*.

3.4.4 Automatic Configuration Process

The manual interactive identification of the best configuration might lead to a product that does not meet specific criteria related to the non-functional requirements. Therefore, decision makers need automatic support and more requirements-level information about decisions. This section presents three mechanisms supported by the *Automatic Configuration* activity: *(i)* Product Configuration Optimization (A4a); *(ii)* Minimal or Maximal Configuration (A4b); and *(iii)* Multi-Step Configuration (A4c). Table 3.6 sketches which studies are supported by each mechanism. The first column identifies the study reference and the other columns are about the mechanisms. We use ● for studies providing support to the mechanism and ○ for studies without any support.

The next three sections present the mechanisms employed to support the automatic SPL configuration process and the implementation particularities of the analyzed approaches. Then, Section 3.4.4.4 highlights the configuration characteristics employed by the studied approaches, and their performance and scalability results.

3.4.4.1 Product Configuration Optimization

This mechanism takes an EFM and the product requirements as input, and automatically returns a set of features that fulfill the feature model interdependencies and best satisfy the stakeholders' requirements. First, to automatically configure a final product, the product requirements provided by the stakeholders must be transformed

Study	A4a	A4b	A4c	Study	A4a	A4b	A4c
Lin and Kremer	●	○	●	Ochoa et al.	●	○	○
White et al.	●	○	●	Pascual et al.	●	○	○
Guedes et al.	●	○	○	Rezapour et al.	●	○	○
Horcas et al.	●	○	○	Siegmund et al.	●	○	○
Khoshnevis and Shams	●	○	○	Asadi et al.	●	○	○
Kifetew et al.	●	○	○	Foster et al.	●	○	○
Noorian et al.	●	○	○	Guo et al.	●	○	○
Oh et al.	●	○	○	Olaechea et al.	●	○	○
Pereira et al.	●	○	○	Sánchez et al.	●	○	○
Umpfenbach et al.	●	○	○	Wang and Pang	●	○	○
Bąk et al.	●	○	○	Karimpour and Ruhe	●	○	○
Hierons et al.	●	○	○	Sayyad et al.	●	○	○
dos Santos Neto et al.	●	○	○	Bagheri et al.	●	○	○
Noir et al.	●	○	○	Roos-Frantz et al.	●	○	○
Ruiz et al.	●	○	○	Mazo et al.	●	○	○
Zanardini et al.	●	○	○	Mussbacher et al.	●	○	○
Benali et al.	●	○	○	Ognjanovic et al.	●	○	○
Henard et al.	●	○	○	Ostrosi et al.	●	○	○
Leite et al.	●	○	○	Parra et al.	●	○	○
Lian and Zhang	●	○	○	Wittern et al.	●	○	○

Table 3.6: Literature supporting the activity: *Automatic Configuration Process* (A4a: *Product Configuration Optimization*, A4b: *Minimal or Maximal Configuration*, A4c: *Multi-Step Configuration*).

into an appropriated mathematical objective function. The product configuration optimization can be modeled in two different ways, known as *single-objective* and *multi-objective* optimization. The *Single-Objective Optimization* (SOO) problem is defined as the *minimization* or *maximization* of a function $f(x) = (x_1, x_2, \ldots, x_n)$ for $n \in \mathbb{N}$, where x is a vector of decision variables. Moreover, a set of m *inequality* constraints $g_i(x) \leq 0$ or a set of p *equality* constraints $h_j(x) = 0$ are defined, such that $i, j, m, p, n \in \mathbb{N}$, $i < m$ and $j < p$. This results in a single global solution. In contrast to the SOO problem, in a *Multi-Objective Optimization* (MOO) problem, $F(x)$ is a vector of k objective functions where $F(x) = \left[f(x)_1, f(x)_2, \ldots, f(x)_k\right]$ and $k \in \mathbb{N}$. Therefore, in a MOO, the user intends to *maximize* or *minimize* functions over multiple NFPs. Since stakeholders may have conflicting or contradicting non-functional requirements (*e.g.*, increasing the security has negative impact on the cost), priorities can then be used to specify which optimization requirement should be satisfied first (see Section 3.4.2.1). Thus, a solution for an MOO problem is a trade-off among the considered objectives in function $F(x)$. SOO supports the *complete* configuration process, while MOO supports the *incomplete* configuration process (see Figure 3.5).

Once the objective function is defined, the EFM is transformed into a mathematical representation, where the configuration process can be performed. Thus, suitable algorithms are used to automatically find configurations, allowing the alignment of the solution with the stakeholders' interests. There are two classes of algorithms: *exact* and *approximation* algorithms. Exact algorithms guarantee that the generated configuration is optimal. However, due to the large variant space and the computa-

tional complexity NP-hard of finding an optimal SPL configuration, exact algorithms have inefficient exponential-time. Thus, approximation algorithms are employed to approximate a good solution, generating partially-optimal product configurations in an efficient polynomial-time.

In Table 3.6, we observe 40 studies implementing this mechanism. After performing the complete reading of these studies, we identified four groups of techniques used to optimize the configuration process: *(1) Constraint Programming* (CP), *(2) Evolutionary Algorithms* (EA), *(3) Integer Linear Programming* (ILP), and *(4) Mapping and Models* (MM). Figure 3.11 shows the number of studies per employed technique and year. Note that we found other approaches that could not be classified in the previous groups. We represented these approaches in the figure as O* (other techniques). The O* group includes techniques such as greedy heuristics used exclusively for solving *knapsack problems* [Pereira et al., 2017, Sánchez et al., 2014], and *ad-hoc* algorithms [Siegmund et al., 2012a], as well as hybrid solutions. Next, we present a brief description of each encoding technique and the implementation particularities of the *main primary studies*. As shown in Figure 3.11, EA and CP are the most used techniques appearing in 15 and 11 studies respectively.

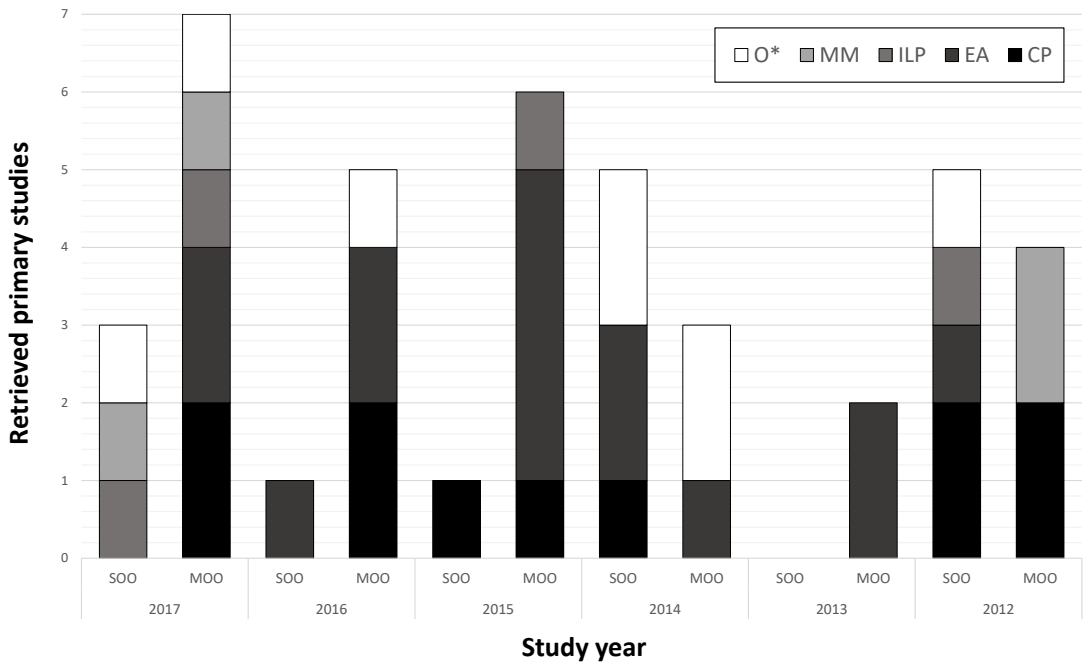


Figure 3.11: Optimization approaches per technique and year.

Encoding in Constraint Programming (CP). CP is a paradigm that proposes an *exact* approach, where *Constraint Satisfaction Problems* (CSP) are specified as a set of variables with their corresponding domain, and a set of constraints that affect these domains [Olaechea et al., 2014]. In order to configure a product, features are considered as binary variables in CSP. Then, for n features in the model, n variables are created, such that each variable $f_i \in \{0, 1\}$. Feature model constraints encodings are shown in Table 3.7, where f_p stands for *parent feature*, f_c for *child feature*, $f_c i$ for the i^{th} child feature of a variability constraint.

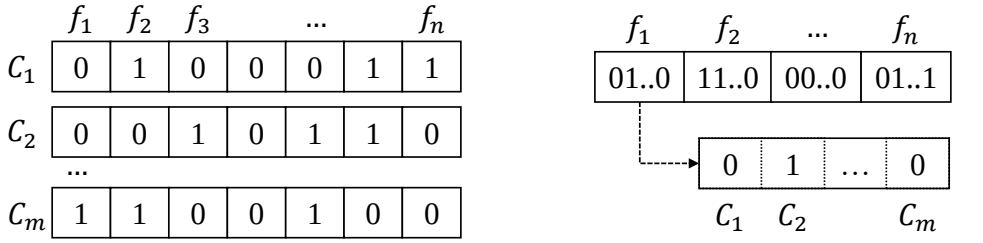
Table 3.7: CP-based encoding.

Variability constraint	Encoding
Mandatory	$f_p = f_c$ [Salinesi et al., 2010]
Optional	$f_p \geq f_c$ [Salinesi et al., 2010]
Or	$\sum_{i=1} f_{ci} > 0$ [Mannion, 2002]
Alternative	$\sum_{i=1} f_{ci} = 1$ [Salinesi et al., 2010]
Requires $f_1 \rightarrow f_2$	$f_1 \leq f_2$ or $f_1 = 1 \rightarrow f_2 = 1$ [Ochoa et al., 2015, Salinesi et al., 2010]
Excludes $f_1 \rightarrow \neg f_2$	$f_1 + f_2 \leq 1$, $f_1 \times f_2 = 0$, or $f_1 = 1 \rightarrow f_2 = 0$ [Ochoa et al., 2015, Salinesi et al., 2010]
Cardinality $[min, max]$	$(\sum_{i=1} f_{ci} \geq min) \wedge (\sum_{i=1} f_{ci} \leq max)$ [Salinesi et al., 2010]
Children selection	$f_2 = 1 \rightarrow f_1 = 1$ [Ochoa et al., 2015]
Parent deselection	$f_1 = 0 \rightarrow f_2 = 0$ [Ochoa et al., 2015]

The solution of the modeled CSP is obtained by finding a suitable configuration that meets variability constraints and product requirements. Some solutions first prune the search space based on product non-functional requirements [Zanardini et al., 2016]. Other approaches propose other encodings given the existence of additional needs in the configuration problem. For instance, White et al. [2014] manage different CSPs at different time slots, then there are $n \times k$ variables related to the n features of the SPL in k time slots. Moreover, Zanardini et al. [2016] enrich the CSP representation with decision trees that are known as *configuration trees* where each node represents an SPL partial configuration, and each edge an increase of the set of features. Then, nodes are not singleton sets with a single feature, but instead they represent a complete configuration that may be invalid.

Encoding in Evolutionary Algorithms (EA). EA are *stochastic* or *approximate* algorithms useful during optimization or learning tasks [Olaechea et al., 2014]. They emulate the natural species evolution, where given a first population of m individuals living in an environment with limited resources, there is a competition were only the fittest individuals survive [Hierons et al., 2016a, Lian and Zhang, 2015a, Sayyad et al., 2013c]. An *individual* is related to one or more *chromosomes* that is composed by n *genes*. First, the initial population is randomly generated or *seeded* with *good* individuals, based in a set of optimization objectives specified by stakeholders. Then, the algorithm starts from a mature population that already responds to a group of requirements, and follows improving the quality of the derived off-springs. The implementation of *seeding* constraints is done by the use of heuristic or deterministic algorithms (*e.g.*, NSGA-II [dos Santos Neto et al., 2016, Karimpour and Ruhe, 2013, Khoshnevis and Shams, 2017, Pascual et al., 2015a] and IBEA [Olaechea et al., 2014, Sayyad et al., 2013c]). *Genetic Algorithms* (GA) and *Multi-Objective Evolutionary Algorithms* (MOEA) are the most used for SPL configuration (see Section 3.4.4.4). These algorithms evaluate each individual according to the *fitness function* (*i.e.*, optimization objectives and product requirements) to produce *multiple generations* derived from the initial population. To obtain solutions that conform to the variability constraints, *correctness* is defined as one of the optimization objectives. Finally, each individual is considered a representation of a *solution* in a particular domain and a trade-off is performed among the multiple optimization objectives.

The general encoding of EA-based approaches in the SPL configuration context considers a *binary string* representation as proposed with GAs. Then, for n features in the SPL, a $1 \times n$ vector is generated in order to represent a configuration (*i.e.*,



a) String for each configuration. b) String for each feature.

Figure 3.12: EA-based encoding binary string.

chromosome), *i.e.* $C_j = [v(f_1), v(f_2), \ldots, v(f_n)]$ such that $f_i \in F$, $v(f_i) \in \{0, 1\}$, $i, j \in \mathbb{N}$, and $i < n$ (see Figure 3.12a) [Lian and Zhang, 2015a, Olaechea et al., 2014, Pascual et al., 2015a].

This encoding can be further reduced by removing some *prunable features* as stated by Lian and Zhang [2015a]. In this context, *core* and *dead features* are removed from the binary string representation. An analysis that considers both sets is unnecessary, since core features are included in all configurations, and dead features are never included. Hierons et al. [2016a] also removes selected parent features if one or more child features are selected. These features are added back when the remaining decisions are defined. Karimpour and Ruhe [2013] also employed a single binary encoding, however they have a particular need on representing multiple configurations in the same vector. Thus, in each of the n positions of the vector, they defined m bit values. Each of these bit values represents the selection state of a given feature related to one of the m considered configurations (see Figure 3.12b).

Encoding in Integer Linear Programming (ILP). ILP is a technique that considers a set of n *decision variables* $V = \{f_1, f_2, \ldots, f_n\}$ that represent features in the feature model, such that $f_i \in \{0, 1\}$; and a set of *constraints* $C = \{C_1, C_2, \ldots, C_m\}$ that represent both variability and requirements related to a linear combination of a subset of variables in V. For both cases $n, m \in \mathbb{N}$. The selection is based on an objective optimization function that should be *minimized* or *maximized* [Bagheri et al., 2012a, Noorian et al., 2017, Rezapour et al., 2015, Umpfenbach et al., 2017]. Solutions to an ILP problem are defined by the set $S = \{s_1, s_2, \ldots, s_t\}$. If all constraints in C are satisfied we refer to S as a *feasible* solution.

Similar to CP-based approaches, to translate variability constraints multiple encodings have been proposed. Some examples are shown in Table 3.8. Multiple encodings are similar or equal to the ones presented in the CP encodings, given their nature of expressing constraints over integer variables.

When all variables have a binary domain, the approach is known as *0-1 programming* [Noorian et al., 2017]. In order to specify additional configuration constraints, some of the ILP-based approaches have introduced extensions to the feature model representation. For instance, Bagheri et al. [2012a] introduced the fuzzy language, which allows the specification of product requirements, and a set of selected or required features (which they call *hard* and *soft* constraints). Moreover, Rezapour et al. [2015] transformed the feature model into a *Graph Product Line* (GPL), and

Table 3.8: ILP-based encoding.

Variability constraint	**Encoding**
Mandatory	$f_p = f_c$ or $f_p \leq f_c$ [Li et al., 2012, Noorian et al., 2017]
Optional	$f_p \geq f_c$ [Noorian et al., 2017]
Or	$\sum_{i=1} f_{ci} \geq f_p$ [Noorian et al., 2017]
Alternative	$\sum_{i=1} f_{ci} = f_p$ [Li et al., 2012, Noorian et al., 2017]
Requires $f_1 \rightarrow f_2$	$f_1 \leq f2$ [Li et al., 2012, Noorian et al., 2017]
Excludes $f_1 \rightarrow \neg f_2$	$f_1 + f_2 \leq 1$ [Li et al., 2012, Noorian et al., 2017]
Paternity	$f_c \leq f_p$ [Li et al., 2012]

proposed a technique that allows the evaluation of SPL methodologies by using the GPL encoding as main evaluation problem.

Encoding in Model and Mapping Approaches (MM). MM approaches employ a modeling technique and a set of mapping rules to derive SPL configurations based on decision propagation from variability model(s). The three main techniques used among the studied approaches are *goal models*, *ontologies*, and *grammars*.

Goal models are defined as AND/OR graph-like structure that supports the specification of *goals* (*i.e.*, objectives related to functional requirements), *soft-goals* (*i.e.*, objectives related to NFPs), and *operational plans* (*i.e.*, task to operationalize goals). *Contribution links* are specified among any entity and soft-goals, using partial positive ('+'), partial negative ('-'), full positive ('++'), and full negative ('−−') implications. Guedes et al. [2017] use a particular type of goal models known as *i* orthogonal models*. This model is related to feature model by means of *presence conditions*, which are references to goals. They are evaluated based on the goal satisfaction degree of the feature: *Full Satisfaction* (FS), *Full Denial* (FD), *Partial Satisfaction* (PS), and *Partial Denial* (PD). Goal satisfaction degrees are defined as boolean formulas, and their value is derived based on feature annotations provided by stakeholders. Similarly, Noorian et al. [2017] use a matrix that interrelates goals and features. They described annotations in each cell to define the correlation among the selection of goals and features (*i.e.*, if a goal is interrelated with a feature, its selection supposes the selection of the related feature). Moreover, they provide different feature models to separate the problem space from the solution space. Lastly, Mussbacher et al. [2012] presented *Aspect-oriented User Requirements Notation* (AoURN), an SPL framework for specifying stakeholders' goals. Crosscutting concerns are treated as aspects in the modeling approach. AoURN considers goal models and feature models to represent stakeholders' needs and SPL, and then both are related in an impact model, where the impact of a feature in each goal is identified. In a similar context, Ostrosi et al. [2012] propose the use of a *Grammars-Based Agents for Product Integrated Configuration* (G-APIC).

The product configuration optimization mechanism is also referred to as *decision scenario*, *single-objective optimization*, *multi-objective optimization*, *one-objective optimization*, *multi-objective search*, *single-objective search*, *one-objective search*, *single-criteria optimization*, *multi-criteria optimization*, *one-criteria optimization*, *optimal feature selection* and *optimization objective*.

3.4.4.2 Minimal or Maximal Configuration

The mechanism taken in this section makes a step beyond the mechanism presented in the last section. In addition to considering multi-optimization objectives, which may result in conflicting non-functional requirements, and consequently generating a set of valid configurations, it also considers the *optimization of reuse* to assist the configuration of a single product. We adopt the term *optimization of reuse* to express the maximization or minimization of reuse during the automatic product configuration. We classify *optimization of reuse* in three main groups: *minimization of reuse*, *maximization of reuse*, and *delimitation of reuse*.

- Minimization of reuse. The minimal number of features needed to get a valid configuration, which represents a partial valid configuration respecting both the SPL variability constraints and the product requirements specifications (*i.e.*, by removing any feature from this configuration leads to an invalid configuration).

- Maximization of reuse. The maximal number of features needed to represent a complete valid configuration which respects both the SPL variability constraints and the product requirement specifications (*i.e.*, by removing any feature from this configuration leads to a partial valid or invalid configuration).

- Delimitation of reuse. It sets a minimum or (and) maximum number of features (*i.e.*, at most (at least) n features) that any generated valid configuration must include.

However, we could not find any work from our SLR implementing such mechanism.

3.4.4.3 Multi-Step Configuration

This mechanism takes as input the product requirements and a fully developed EFM that needs to be multi-stage configured. Given a starting configuration and a desired ending configuration that needs to be active in n steps, an optimization method is used to select a set of features to meet each step continually. The output is a sequence of configurations from a starting configuration through a series of intermediate configurations to a final configuration. This path is optimized in a way that best meets the desired set of end product requirements. Each set of features for a step must constitute a complete and valid configuration of the SPL to avoid selling a defective and non-viable product. There are two approaches in Table 3.6 that implement this mechanism.

White et al. [2014] propose an automatic approach for modeling and solving multi-stage SPL configurations, called MUSCLES (MUlti-step Software Configuration probLEm Solver). MUSCLES transforms SPLs into Constraint Satisfaction Problems (CSPs), which enables CSP solvers for deriving a single optimized path of configurations. In accordance with White et al. [2014], the main challenge deriving configuration paths for an SPL is to analyze a myriad of trade-offs related to the order that the features are selected.

Lin and Kremer [2014] propose an approach to predict introduction timing of a product configuration in the market by using SPL practices from a multiple-generation

product strategy (MGPS). In an MGPS, a company first launches an initial configuration to the market. After this initial launch, the proposed approach analyzing the market to sequentially introduce successive configurations over time, updating variable features. The company predicts the sales behavior of each lunched configuration to achieve the highest profits and better utilize their resources.

This mechanism is also referred to as *multi-stage configuration*, *configuration path*, *multiple-generation product strategy*, and *temporal configuration*.

3.4.4.4 Performance and Scalability Results

In this section, we consider the automatic approaches that employ performance and scalability tests as evaluation techniques. Table 3.9 presents the characteristics employed for each approach by considering the different techniques. We aim at presenting the execution time taken by each solution to configure an SPL by considering the number of optimization constraints and the size of the feature model. In cases where more than one SPL was tested, we considered only the feature model with the highest amount of features. This analysis is shown for each of the considered techniques, except for MM-based approaches given the lack of performance and scalability tests during their evaluations. For each grouped approach, we used ● for specifying that the corresponding product requirement is completely supported, and ○ for specifying that it is not supported. The acronyms of each column title are defined as follows: *product constraints* (C), *stakeholders' preferences* (P), *feature model* (FM), *number of optimization constraints defined in a configuration scenario* (#OC), and *execution time for configuration* (ET). Notice that the #OC column shows the number of optimization objectives, and not the number of different types of configuration constraints. In addition, we did not analyze studies which performed *observations and experiences* (OBE) evaluation [Bagheri et al., 2012a, Khoshnevis and Shams, 2017, Noir et al., 2016] neither studies missing data about the performance evaluation [Benali et al., 2015, Foster et al., 2014, Guedes et al., 2017, Karimpour and Ruhe, 2013, Lin and Kremer, 2014, Mussbacher et al., 2012, Ognjanovic et al., 2012, Oh et al., 2017, Ostrosi et al., 2012, Ruiz et al., 2016, Umpfenbach et al., 2017, Wittern et al., 2012, Zanardini et al., 2016].

For all cases, solutions are obtained in at least 0.5 milliseconds [Sánchez et al., 2014]. However, it is important to note that the selection among encodings can affect the execution time of the approaches. In addition, the execution time of the solution varies depending on the considered constraints and the size of the feature model, *i.e.* if the complexity and size of the feature model increases or the number of considered optimization constraints is higher, these values can be drastically affected.

The considered feature models for *CP-based approaches* have a size between 35 and 500 features. Most of the CP-based approaches use Choco[3], a Java library for constraint programming. *CP-based approaches* can scale up to feature model instances with 500 features in less than 160 seconds [White et al., 2014]. However, the commonly used feature model is a toy model automatically generated by an external tool. Although toy models are important to demonstrate how an approach

[3]http://choco-solver.org/

	Study	Method	C	P	FM	FM Size	#OC	ET
CP-based	Parra et al.	JaCoP solver	●	●	Generated	50 variation points	3	27.7 ms
	Horcas et al.	Choco solver	●	○	WeaFQAs	1,000 valid configurations	1	3 min
	Bąk et al.	Choco solver	●	○	UML Violet	100 features	4	time out
	Leite et al.	Choco solver	●	○	Google Compute Engine	35 features	3	< 10 min
	Ochoa et al.	Choco solver	●	○	Generated	366 features	2	36 hours
	White et al.	Choco solver	●	○	Generated	500 features	1	< 160 sec
	Roos-Frantz et al.	Choco solver	●	○	Automotive domain	Undefined	1	time out
	Mazo et al.	GNU Prolog	●	○	Generated	89 features	1	15 sec
EA-based	dos Santos Neto et al.	NSGAII	●	●	ArgoUML	1,000 features	2	0.98 ms
	Kifetew et al.	GA	●	○	Drupal	48 features	8	240 sec
	Hierons et al.	SIP	●	○	Generated	10,000 features	8	100 sec
	Henard et al.	SATIBEA	●	○	Linux	6,888 features	5	15 min
	Lian and Zhang	IVEA	●	○	E-Shop	290 features	5	2.95 min
	Pascual et al.	NSGA-II	●	○	Generated	5,000 features	3	17.9 sec
	Olaechea et al.	IBEA	●	○	E-Shop	290 features	4	> 15 days
	Wang and Pang	MMAS and ACS	●	○	Generated	200 features	1	1,166.19 ms
	Sayyad et al.	IBEA	●	○	E-Shop	290 features	5	12.3 sec
ILP	Noorian et al.	CPLEXa solver	●	●	Generated	10,000 features	1	54.326 sec
	Rezapour et al.	-	○	●	Laptop	42 features	2	< 1 min
O*	Pereira et al.	Greedy heuristic	●	●	Generated	10,000 features	1	12.238 sec
	Asadi et al.	SHOP2	●	●	Generated	200 features	1	86 sec
	Sánchez et al.	BF* and GBFS	●	●	Video surveillance	26 features	2	0.5- 0.7 ms
	Siegmund et al.	Ad-hoc algorithm	●	○	Berkeley DB	36 features	1	< 50 min
	Guo et al.	GIA	●	○	E-Shop	290 features	4	6 days

Table 3.9: Performance and scalability evaluation of each approach (C: product constraints, P: stakeholders' preferences, FM: feature model, OC: optimization constraints, ET: execution time).

works, the results cannot be generalized to real-world scenarios. For the bigger real-world feature model (*i.e.*, UML Violet with a total of 100 features), the employed approach was not able to find an optimal solution [Bąk et al., 2016]. This highlights the need to use large real-world feature models in future works to prove the feasibility of the existing approaches in practice.

Since in most of cases it is unfeasible to test all possible combinations of selected and deselected features, some approaches explore the use of approximation algorithms, such as EA-based and heuristic approaches instead of exact CP-based approaches. One identified advantage related to approximation approaches correspond to the employment of the same or similar feature models, such as the E-Shop feature model [Guo et al., 2014, Lian and Zhang, 2015a, Olaechea et al., 2014, Sayyad et al., 2013c]. For *EA-based approaches*, feature models size varies between 48 and 10,000 features, and the considered optimization constraints ranges between 1 and 8 objectives. SIP [Hierons et al., 2016a] and NSGAII [dos Santos Neto et al., 2016, Pascual et al., 2015a] EA-based approaches are suitable for solving MOO problems, being scalable to feature models with 1,000 to 10,000 features in 0.98 milliseconds to 100 seconds. However, as in CP-based approaches, this feature model is a toy model automatically generated by an external tool.

For *ILP-based approaches*, only Noorian et al. [2017] and Rezapour et al. [2015] reported performance and scalability tests and can be considered in our results. The execution time for both solutions is of approximately 1 minute. However, while Noorian et al. [2017] consider a randomly generated feature model with 10,000 features and 1 optimization constraint, Rezapour et al. [2015] consider a real-world feature model with 42 features and 2 optimization constraints. Moreover, for both approaches, the employed time during the AHP ranking (see Section 3.4.2.1) was not considered. Therefore, for real-word large feature models, the ranking phase which is manually performed could be a tedious and time consuming task, becoming a bottleneck in the SPL configuration process.

In *O* approaches*, the GIA algorithm proposed by Guo et al. [2014] presents the higher execution time when configuring the E-Shop SPL with 290 features and 4 optimization constraints. This time can be decreased if more product constraints are employed. Otherwise, the greedy heuristic approach proposed by Pereira et al. [2017] seems to be the most suitable technique to solve SOO problems being scalable to randomly generated feature models with up to 10,000 features in approximately 12.2 seconds.

As conclusion, we identified that all approaches tend to support product constraints (C) and only a small number of these studies explicitly offer support to handle stakeholders' preferences (P) and hard cross-tree constraint [Lian and Zhang, 2015a, Sayyad et al., 2013c]. 16 of the analyzed studies explicitly provide tool support. The remaining studies only stated they developed solely algorithms to demonstrate the claims, and they encourage and point out the need of tool support that implements the proposed algorithm(s). Although in this section we described the performance evaluation characteristics from these studies which is essential for comparison reasons, it is also important to evaluate other kinds of feature models not reported by existing works, since there are many variables (*e.g.*, market domain and complexity)

Study	A5a	A5b	A5c	Study	A5a	A5b	A5c
Safdar et al.	●	●	○	Ruiz et al.	○	●	○
Schroeter et al.	●	●	○	Santos et al.	○	●	○
Zheng et al.	●	○	○	Leite et al.	○	●	○
Eichelberger et al.	●	○	○	Mazo et al.	○	●	○
Rabiser et al.	●	○	○	Pascual et al.	○	●	○
Brink et al.	●	○	○	Bürdek et al.	○	●	○
Galindo et al.	●	○	○	Bures et al.	○	●	○
Chavarriaga et al.	●	○	○	Murguzur et al.	○	●	○
Urli et al.	●	○	○	Sánchez et al.	○	●	○
Acher et al.	●	○	○	Cubo et al.	○	●	○
Jannach and Zanker	●	○	○	Kramer et al.	○	●	○
Klambauer et al.	●	○	○	Lee	○	●	○
Mazo et al.	●	○	○	Saller et al.	○	●	○
Pleuss and Botterweck	●	○	○	Parra et al.	○	●	○
Nieke et al.	○	●	●	Wang and Ng	○	●	○
Sharifloo et al.	○	●	●	Wittern et al.	○	●	○
Gamez and Fuentes	○	●	●	Gençay et al.	○	○	●
Adjoyan and Seriai	○	●	○	Nieke et al.	○	○	●
Guedes et al.	○	●	○	Tanhaei et al.	○	○	●
Zheng et al.	○	●	○	Heider et al.	○	○	●
Ayala et al.	○	●	○	Mitchell	○	○	●
Ter Beek et al.	○	●	○	Neves et al.	○	○	●
Mauro et al.	○	●	○				

Table 3.10: Literature supporting the activity: *Configuration Adaptation Process* (A5a: *Configuration of Multi-Software Product Lines*, A5b: *Dynamic Product Configuration*, A5c: *Product Configuration Evolution*).

that can influence the results. Therefore, an interesting area for work would be the evaluation of existing approaches by using other (real-world) benchmarks.

3.4.5 Configuration Adaptation Process

This section presents three mechanisms supported by the *Configuration Adaptation* activity: *(i)* Configuration of Multi-SPLs (A5a); *(ii)* Dynamic Product Configuration (A5b); and *(iii)* Product Configuration Evolution (A5c). Table 3.10 sketches which studies support these mechanisms. The first column identifies the study reference and the other columns are about the mechanisms. We use ● for studies providing support to the mechanism and ○ for studies without any support.

3.4.5.1 Configuration of Multi-Software Product Lines

This mechanism takes as input a set of configurations from multiple related SPLs. Considering that a feature can interact with other features in another SPL and resources may be shared among SPLs, the configuration must be coordinated and adjusted in accordance with the constraints between SPLs to instantly determine violations of distinct product requirements and stakeholders' preferences. In this context, this mechanism follows managing the configuration from multiple stakeholders over interrelated SPLs. Table 3.10 shows 14 studies supporting this mechanism.

A reason that makes the design of a centralized SPL infeasible is to keep information privacy for business reasons [Nieke et al., 2016]. For example, different competitors

on suppliers may be interested in keeping their detailed configuration and pricing rules private for their related SPLs. Some authors [Acher et al., 2013, Brink et al., 2015, Eichelberger et al., 2016, Rabiser et al., 2016] report that managing a single large model for an entire system may lead to unmanageable complexity. Since SPLs vary with respect to their granularity (*i.e.*, feature models describe variability at various levels of abstraction), some authors split the complexity and information density of a single model towards several models which make it easier to support the collaborative configuration process of multiple distributed stakeholders (see Section 3.4.3.4).

Cross-Model Constraint (CMC) rules are constraints defined among features in different models [Safdar et al., 2017]. They describe how configurations of communicating products belonging to different SPLs influence each other. Similar to cross-tree-constraints, the most common representation are *requires* and *excludes* implications. As an example, suppose we have two feature models: FM_1 and FM_2, each one with a set of features F_1 and F_2 respectively. Then, a *requires* CMC can be established among $f_{1i} \in F_1$ and $f_{2j} \in F_2$, such that $f_{1i} \rightarrow f_{2j}$, $i, j \in \mathbb{N}$, $i < |F_1|$, and $j < |F_j|$. Moreover, an *excludes* CMC can be represented as $f_{1i} \rightarrow \neg f_{2j}$ for the same context [Chavarriaga et al., 2014]. Notice that as in cross-tree-constraints, CMC can also be defined by propositional logic predicates, where features are represented as boolean variables and operators as $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and $\neg$ [Batory, 2005]. Moreover, hard cross-tree-constraints can also be specified between models.

Manually specifying CMC rules based on domain knowledge of experts is a time-consuming and tedious task. Therefore, Safdar et al. [2017] propose an approach, called *Search-Based Rule Mining* (SBRM), which combines multi-objective search with machine learning techniques to mine CMC rules at runtime. In addition, Acher et al. [2013] propose a textual and executable language, called FAMILIAR, for specifying CMC rules and managing interactions among multi-SPLs at runtime. To separate concerns in feature modeling, FAMILIAR uses composition and decomposition operators (*e.g.*, slice, merge, aggregate).

Some authors [Chavarriaga et al., 2014, Jannach and Zanker, 2013, Mazo et al., 2012c, Rabiser et al., 2016] support the definition of feature dependencies among models by means of prohibits and forces relationships by decision propagation strategies (see Section 3.4.3.2). Chavarriaga et al. [2014] provide explanation through the decision propagation process. In addition, S2T2 Configurator [Pleuss and Botterweck, 2012] provides a graphic layout to visualize multiple interrelated feature models. EASy-Producer [Eichelberger et al., 2016], PuMA [Schroeter et al., 2012], and Zheng et al. [2017a] provide a graphic user-friendly interface for multiple stakeholders. Each stakeholder decides over its own model (see Section 3.4.3.4). Zheng et al. [2017a] classify stakeholders regarding their capabilities of design specification knowledge. They are classified into *normal users* (*i.e.*, customers with little design knowledge) and *expert users* (*i.e.*, knowledgeable customers).

Schroeter et al. [2012] handle the configuration of multi-SPLs in dynamic environments. The approach supports the creation of multiple separate configurations from a global configuration, as well as the integration of separate configurations from multi-SPLs.

Invar [Galindo et al., 2015b] and SpineFM [Urli et al., 2014] provide a unified perspective over heterogeneous variability models with different semantics to handle the configuration of multi-SPLs. In addition, DOPLER [Klambauer et al., 2013] uses multi-system requirements within multi-SPLs to detect violations during the configuration of individual systems and provide immediate feedback to the stakeholders.

This mechanism is also referred to as *collaborative configuration*, *system-of-systems configuration*, *multi-system configuration*, *concurrent configuration*, *multiple configuration*, *distributed configuration*, *distributed product line composition*, *variability-rich software ecosystems*, and *multi-stakeholders configuration*.

3.4.5.2 Dynamic Product Configuration

Dynamic SPLs provide a promising strategy for planning and employing runtime reconfiguration to adaptive software systems. This mechanism requires as input the feature model, the current configuration and the captured contextual information. When environments represented by SPLs change, both variability models and configurations should embrace these changes to satisfy product requirements. Thus, self-reconfiguration of system at run-time in response to a context change is needed in order to maintain validity and completeness of the derived products.

Dynamic SPL approaches monitor variables (*e.g.*, contexts, non-functional requirements, etc) in order to detect changes that require the system to adapt. This mechanism supports three main functionalities *(i)* the definition of relevant contextual information; *(ii)* the definition of context influence on configuration options; and *(iii)* the automatic reconfiguration of products based on the context [Nieke et al., 2017]. Thus, for a feature to be selectable under a given context, all defined conditions on that context have to be satisfied. Table 3.10 shows 27 studies supporting this mechanism.

Saller et al. [2013] and Mauro et al. [2016] support the representation of relevant contextual information and the automatic reconfiguration of products based on the context. The authors propose a framework where contextual information is captured directly within the feature model. Saller et al. [2013] define a transition system to specify appropriate reconfigurations supporting the requirements of potentially interfering runtime context patterns. Mauro et al. [2016] annotate every feature in the feature model with a propositional formula which captures influences of contexts on features. Then, the authors implement a *hybrid variability reconfiguration engine* (HyVarRec) for constraint checking that uses the annotated feature model to self-adapt at runtime when a current configuration reaches an invalid state. To self-adapt, HyVarRec tries to find a new valid configuration most similar with the initial one. In a similar scenario, Wang and Ng [2012] propose a hybrid constraint solving algorithm to effectively reconfigure products based on the context. Also, some authors [Guedes et al., 2017, Leite et al., 2015, Parra et al., 2012, Pascual et al., 2015a, Ruiz et al., 2016, Sánchez et al., 2014, Wittern et al., 2012] propose a self-optimization of configurations at runtime and others [Safdar et al., 2017, Schroeter et al., 2012] propose the self-reconfiguration of multi-SPLs.

DarwinSPL [Nieke et al., 2017] and Sharifloo et al. [2016] correlated contextual variability with evolutionary variability (see Section 3.4.5.3). DarwinSPL [Nieke

et al., 2017] provides an integration with HyVarRec to reconfigure SPLs. Sharifloo et al. [2016] developed a set of adaptation rules based on the historic of evolutions and their effectiveness in achieving the current product requirements.

Several approaches [Ayala et al., 2016, Bures et al., 2014, Cubo et al., 2013, Lee, 2013, Mazo et al., 2015, Murguzur et al., 2014, Zheng et al., 2017b] create a set of models to represent contextual information and to model the impact of contextual information on possible feature selection. Complementary, Bürdek et al. [2014] propose the assignment of binding times to features. Binding times are introduced to distinguish prior configuration decisions from left-open variation points to enable re-configurations at runtime. This approach represents the contextual information with models and relate context to features using hard-cross-tree constraints (including dependencies between binding times, feature selections, and NFPs).

While the previous works have enabled program logic adaptation by the use of models, Adjoyan and Seriai [2017] specify contextual influences and dynamic reconfigurations at architecture level by using an architecture description language. In addition, Kramer et al. propose a solution for dealing with *Graphical User Interface* (GUI) document variability. In a similar scenario, other approaches [Adjoyan and Seriai, 2017, Gamez and Fuentes, 2013, Santos et al., 2016, Ter Beek et al., 2016] present a language for defining influence of context on configuration options. For instance, Ter Beek et al. [2016] present a set of probabilistic feature-oriented languages (*i.e.*, *FLan*, *PFLan*, and *QFLan*) to analyze context and self-reconfigure at runtime. The authors consider *(i)* arithmetic relations between NFPs; *(ii)* the relations between features and NFPs; and *(iii)* dependencies between actions.

This mechanism is also referred to as *dynamically adaptive systems*, *dynamic feature deployment*, *dynamic reconfiguration*, *runtime reconfiguration*, *compile time configuration*, and *self-adaptation*.

3.4.5.3 Product Configuration Evolution

Over the SPL evolution scenario, new configuration options become available and valid, while existing configuration options may become obsolete and invalid. It is important to make sure that the behavior of existing configurations is not affected. This mechanism takes as input a previous SPL and its correspondent configurations, and an evolved SPL. Then, it follows changing the previous configurations to meet the evolved SPL constraints to form new valid and concrete configurations. Modifying the SPL without keeping track of the changes results in loss of information. The new configuration needs to be readjusted in accordance with *(i)* changes in the source code; *(ii)* addition or remotion of features, relationships, cross-tree-constraints, and NFPs; and (iii) modification of a feature type (*e.g.*, a mandatory feature is transformed in optional). Moreover, the configurator needs to be adapted to hide invalid configuration option from decision makers.

To make sure that the behavior of existing configurations is not affected during the SPL evolution, experts usually have to manually analyze different artifacts which is infeasible considering that the number of changes in a single evolution scenario can increase exponentially [Neves et al., 2012]. Therefore, several approaches use previous configurations to automatically infer new ones. Table 3.10 shows 9 studies supporting this mechanism.

DarwinSPL [Nieke et al., 2017] and Sharifloo et al. [2016] propose an approach to keep track of the SPL evolution. DarwinSPL allows stakeholders to visualize the whole evolution history of an SPL (*i.e.*, the state of an SPL for arbitrary dates). DarwinSPL configurator only shows options which are temporally valid. Sharifloo et al. [2016] track evolutionary historic to learn how to self-reconfigure a product at runtime in response to a context change (see Section 3.4.5.2).

Hydra [Gamez and Fuentes, 2013] uses model mappings and transformations to automatically propagate the evolution changes into the existing configurations. In addition, it measures the effort in performing the changes in every configuration. In a similar scenario, VaMoRT [Heider et al., 2012] informs stakeholders about the impacts of variability model changes on existing configurations. VaMoRT uses regression tests to determine whether existing configurations can be reconfigured without unexpected effects.

Tanhaei et al. [2016b] propose an approach for refactoring of feature model to synchronize changes in the source code. In a similar way, Neves et al. [2012] and Nieke et al. [2016] describe several safe evolution templates to preserve the behavior of existing configurations. In this context, Gençay et al. [2017] consider only the removal of obsolete options from existing configurations. Similar to the previous approaches, Mitchell [2012] develops a modeling technique, called SysML, to manage configurations consistencies as the model evolves.

Nieke et al. [2016] specify transformations that go beyond program refactoring notions. The authors present a method to lock specific configurations to ensure their validity during evolution of the SPL. This method guarantees that locked configurations remain valid during SPL evolution and make statements on which part of the evolution would break the configurations. Then, the proposed approach prohibits evolutions that would break locked configurations so that their validity can be guaranteed.

The implementation of this mechanism reduces the manual changes effort and the inconsistencies among configurations as SPL evolves. This mechanism is also referred to as *evolution of existing configurations*, *management of configuration evolution*, and *management of temporal variability*.

3.5 Main Findings

In this section, we answer RQ4 introduced in the beginning of this chapter. This research question aims at identifying new areas of research that can lead to further enrichment of the SPL configuration field. They will be listed in two separate groups, one addressing the *employed activities* and the other addressing the *evaluation process*.

First, we identified the following open challenges and limitations related to the employed activities:

A1: Mapping Non-Functional Properties

- *Management of qualitative NFPs.* Quantitative NFPs are managed by most approaches. However, few solutions support qualitative NFPs. According to two

SLRs [Galster et al., 2014, MahdaviHezavehi et al., 2013] current methods focus on performance and availability NFPs. Moreover, in most cases NFPs are managed as nominal values. Although fuzzy logic has been adopted for few approaches to manage qualitative NFPs, further research is still required in this field in order to employ different techniques and better satisfy stakeholders' needs. Current studies do not provide enough evidence for practitioners to apply the proposed approaches to handle variability. Therefore, we suggest further studies in the industry to show the feasibility of the proposed approaches.

- *Measurement of NFPs at runtime.* Although the studies mentioned in Section 3.4.1 present a high advance in the static measurement of NFPs, there is still a lack of techniques in the literature to efficiently support the automatic measurement of NFPs at runtime. As a main limitation, we observe that most studies support only one NFP measurement. Moreover, empirical experiments performed in large industrial SPLs is still time consuming. Therefore, a tool support that allows the definition of a set of metrics and the efficient prediction of different NFPs at runtime is still missing.

A2: Mapping Product Requirements

- *Specification of configuration requirements.* Although there are several approaches to partially assist decision makers during the semi-automatic configuration process of extended SPLs, those approaches do not offer support to the whole set of identified configuration constraints from this SLR (*e.g.*, product constraints, multi-optimization objectives, and stakeholders' preferences) by considering feature interactions and hard cross-tree-constraints. Moreover, a configuration language such as the one presented in Listing 3.1, as well as an automatic encoding to solve the specification from Listing 3.1, is still missing in the literature. Therefore, the implementation of a new approach which combines the several reported configuration constraints with a product configuration language is a relevant area to be explored by researchers.

- *SPL Configuration Language.* The specification of configurations as sets of selected and deselected features is time consuming. Moreover, the specification of configurations by referring to other created configurations may cause side-effects in configuration evolution (Section 3.4.5.3). We did not find any approach in the literature that provides a formal syntax and semantics to support this scenario.

A3: Manual Configuration Process

- *Extended feature model tools.* Feature model diagrams have a very strict representation in state-of-the-art tools in which all functional and NFPs of the SPL must be represented. Moreover, there is a lack of SPL tools in the literature that allow users to represent NFPs and specify product requirements (*e.g.*, product constraint filters). Furthermore, we are not aware of any tool that allows decision makers to specify hard cross-tree-constraints or even visualize the impact of these constraints during the interactive configuration process.

- *Interactive configurator view.* An ample interaction with decision makers during the variability resolution process is essential, in order to abstract the technical details from feature models (*i.e.*, complex and hard to reasoning constraints) and guide stakeholders through the SPL configuration process. Since decision makers are often not aware of all dependencies between the features, they need tools that help them to make an efficient analysis, *e.g.*, comparing different consequences of decisions. Therefore, as future work, researchers should consider to extend consolidated state-of-the-art SPL configurators with a set of further background information about the feature model, such as its rationale, important constraints and dependencies, and implications from NFPs on features (*e.g.*, *sensor* is the most expensive feature and *alarm* is the safest one). Moreover, decision makers should be supported in understanding the consequences calculated by the reasoning engine and automatically applied to the model (*i.e.*, decision propagations).

- *Support multi-stakeholder tasks.* Although there are configurators in the literature that deal with multi-stakeholder environments (Section 3.4.3.4), these tools do not support the configuration process of extended SPLs. Thus, future research is needed in this field to automatically generate configurations with a higher satisfaction degree for a set of different decision makers. As an example, Martinez et al. [2014] present a visualization paradigm, called FRoGs (*Feature Relations Graphs*). FRoGs supports different decision makers to obtain a better understanding of feature constraints during the product configuration process. However, FRoGs does not support the configuration process of extended SPLs.

- *Explore recommendation techniques.* Although AHP techniques are used by most of the SPL configuration approaches to define stakeholders' preferences (Section 3.4.2.1), these approaches require pair-wise comparison, which is a time consuming and error prone task. Therefore, new techniques should be explored, such as the adaptation of state-of-the-art recommendation algorithms for SPL configuration. New configurators that make use of recommender systems have emerged as Sysiphus [Thurimella and Bruegge, 2012] and PROFilE [Pereira et al., 2016c]. In both works recommendation techniques are employed to guide the configuration of SPLs by considering historical data related to previous configurations. Sysiphus keeps a database of rationales from users about non-functional requirements and PROFilE uses a historical dataset of configurations to recommend features. As future work, the rationale captured by Sysiphus [Thurimella and Bruegge, 2012] could be also useful by PROFilE [Pereira et al., 2016c] to guide new decisions through the use of content-based recommendation techniques. Also, researchers could look at the use of recommendation techniques in requirements specification to improve the decision-making process even more during the instantiation of their products. Finally, it is important to analyze the impact of the proposed techniques by mean of a user-controlled study to investigate the users' satisfaction with the recommendations.

A4: Automatic Configuration Process

- *Development of hybrid solutions.* Each automatic configuration technique offers a set of advantages and disadvantages which affect the performance of the configuration process (see Table 3.9). Performance is favored in some cases (*e.g.*,

EA-based approaches), while in other cases performance and scalability present critical issues (*e.g.*, CP-based approaches). Furthermore, some approaches cannot guarantee the satisfaction of constraints (*e.g.*, EA-based approaches), while others demand their compliance (*e.g.*, CP-based approaches). These reasons demand the implementation of hybrid solutions that consider the strengths presented by each different technique. For example, a CP-based approach could be used to derive a small set of valid configurations that can be used as the initial population of an EA-based approach. This action could enhance the quality of the obtained configurations without deeply affecting the performance of the solution.

- *Tuning configuration techniques and algorithms.* In the case of large and complex models, the tuning of the configuration techniques and algorithms is required to derive solutions with a desired performance. As acknowledged by some of the selected studies from our SLR, tuning algorithm parameters can improve the obtained results [Henard et al., 2015a, Sayyad et al., 2013c]. Moreover, heuristics can be employed to outperform previous results, as it is the case of Guo et al. [2014] where the performance of an SPL configuration approach based on CP was improved by defining search heuristics. However, as an effective automatic configuration process is a challenge for large SPLs, there is still opportunity for the use of such heuristics for improvements of the performance results found in the literature. As future work, it is also important to compare the different configuration techniques and algorithms proposed in the literature by using the same evaluation process (*i.e.*, the same feature model and configuration constraints).

- *Automatic incomplete configurations.* Multi-optimization objective (MOO) problems may return a set of product configurations, therefore support is needed to guide users in the selection of one configuration from the set of possible results. The current practice in the literature of SPL configuration is to perform a subjective manual analysis. Then, a central challenge is thus to provide stakeholders support (*e.g.*, justification reports) to explore and choose the best configuration. Possibly, this could be achieved through the use of visualization mechanisms plus the use of recommendation techniques. In addition, the employment of recommendation techniques to suggest the modification of existing constraints in order to obtain most personalized configurations is also a field that could be researched.

A5: Configuration Adaptation Process

- *Reconfiguration in dynamic environments.* Currently, most dynamically SPL configuration approaches are developed for static *models* and *configuration constraints*, where configurations are strictly tied to human actions and not to changing contexts. We are not aware of any work that make use of recommendation techniques on self-adaptive systems. In this scenario, we suggest the use of different context-aware recommendation algorithms. Moreover, we recommend analyzing the impact of the proposed algorithms on configuration efficiency and performance. Also, we suggest evaluating the proposed approach under the use of optimization techniques.

- *Chose from a set of configurations.* When dealing with dynamic SPL configuration, we may also face the problem where there are too many configurations

for a single context we can choose from. HyVarRec [Mauro et al., 2016] targets this problem by choosing the configuration that is most similar to the initial one. However, it may be even more interesting to allow users to choose their preferred configuration. In this context, we could also use recommendation techniques to rank configurations based on historical data from previous users [Pereira et al., 2016c]. Thus, this poses additional challenges in this field, as well as the visualization and elicitation of the optimal criteria for selecting a configuration.

Next, we identified the open challenges related to the evaluation process used by the selected studies:

Evaluation

- *Evaluation type.* Most of the analyzed approaches present just a running example (*i.e.*, case studies) that explain how the proposed configuration technique works, without presenting a further evaluation (*i.e.*, performance and scalability tests) or the adoption of empirical user studies.

- *Real-world SPLs.* Several evaluations do not rely on real-world feature models like it is the case of Asadi et al. [2014], Hierons et al. [2016a], Mazo et al. [2012c], Noorian et al. [2017], Ochoa et al. [2015], Parra et al. [2012], Pascual et al. [2015a], Pereira et al. [2017], Wang and Pang [2014], White et al. [2014]. Moreover, the studies that rely on real-world feature models can not get a solution in few seconds for large feature models. This supposes a threat to validity in current researches. Thus, to support study validity, features and NFPs should be modeled based on real data from business experiences. Also, in addition to large real-world SPLs, we also recommend the use of state-of-the-art product lines, which allows the comparison between results presented by different approaches.

- *Tool support.* Tool support should assist the complete SPL configuration process, and not just some mechanisms, because it would lead to the need to use several tools and information traceability among them which would, probably, have to be done manually by the application engineer. Furthermore, there is a lack of empirical user study to compare SPL configurators and show their practical support on real-world scenarios. For example, studies are missing the evaluation of qualitative aspects, such as learnability, expressiveness, and usability. Although Asadi et al. [2016] have recently empirically analyzed visualization and interaction characteristics of configurators, the authors performed a controlled user empirical study of a single tool. Hence, experiments should be performed to compare the effects of different configurators.

We found that SPL configuration techniques have gained attention in the recent years and several studies address similar mechanisms in different ways. Therefore, the SPL community needs to tackle new challenges to improve the expressiveness of SPLs and to manage the overwhelming complexity of SPL configurations. In this thesis, we aim to address some of the previous identified challenges. First, in Chapter 4, we extend the recommendation techniques presented in Pereira et al. [2016c]. Second, in Chapter 5, we adopt recommendation techniques to support the

automatic configuration of extended SPLs. Third, in Chapter 6, we introduce a context-aware recommender system to support the dynamic configuration of SPLs at runtime. Finally, in Chapter 7, we follow developing a tool support that offers all those contributions, as well as a set of visualization mechanisms to easy the configuration process.

3.6 Threats to Validity

This section discusses potential *threats to validity* that might have affected the results of the SLR. We faced similar threats to validity as any other SLR. The findings of this SLR may have been affected by bias in the selection of the primary studies, inaccuracy in the data extraction and in the classification of the primary studies, and incompleteness in defining the set of activities, mechanisms, and open challenges. Next, we summarize the main threats to the validity of our work and the strategies we have followed to minimize their impact. We discussed the SLR validity with respect to the two groups of common threats to validity: *internal* and *external* validity [Wohlin et al., 2000].

Internal validity. An internal validity threat concerns the reliability of the selection and data extraction process. To further increase the internal validity of the review results, we conducted the inclusion and exclusion processes in parallel by involving three researchers and we cross-checked the outcome after each phase. In the case of disagreements, we discussed until a final decision was achieved. Furthermore, we documented potentially relevant studies that were excluded. Therefore, we believe that we have not omitted any relevant study.

For the selected papers, a potential threat to validity is the reliability and accuracy of the data extraction process, since not all information was obvious to extract (*i.e.*, many papers lacked details about the design and evaluation of the reported study). Consequently, some data had to be interpreted which involved subjective decisions by the researchers. Therefore, to ensure the validity, multiple sources of data were analyzed, *i.e.*, papers, websites, technical reports, manuals, and executable. Moreover, whenever there was doubt about some extracted data in a particular paper, we discussed the reported data from different perspectives in order to resolve all discrepancies. However, we are aware that the data extraction process is a subjective activity and likely to yield different results when executed by different researchers.

External validity. A major external validity to this study was during the identified primary studies. We limited the search by starting in the year of 2012. This may affect the completeness of our search results. However, we aimed at highlighting the current research on SPL configuration in the recent years, and according to Ochoa et al. [2017] most relevant contributions in this field are from 2012 onwards. To further decrease the probability of missing relevant papers, the search for relevant studies was conducted in several relevant scientific databases, and it was focused not only on journals but also on conferences, symposiums, and workshops. However, the quality of the search engines could have influenced the completeness of the identified primary studies (*i.e.*, our search may have missed those studies whose authors did not use the terms we used in our search string to specify keywords). To minimize

these limitations and avoid all sorts of bias, we also analyzed the references of the primary studies to identify other relevant primary studies. In addition, this SLR was based on a strict protocol described in Section 3.2 which was discussed before the start of the review to increase the reliability of the selection and data extraction processes of the primary studies and allow other researchers to replicate this review.

Another external validity concerns the definition of SPL configuration activities and mechanisms, as well as open challenges. It can be possible that these definitions might have been affected by personal interest and opinions. Moreover, we are aware that the completeness of activities, mechanisms, and open challenges is another limitation that should be considered while interpreting the results of this review. Therefore, additional findings not highlighted in Section 3.3 and Section 3.5 should be included in a future review. For instance, Hubaux et al. [2012] and Benavides et al. [2013] systematically compare product configuration and feature model configuration and they conclude that the synergies of both areas is rather glaring. Based on this assumption, it is also important to explore general contributions in the product configuration domain (*i.e.*, outside the software domain) to fully gather the spread knowledge from a different area to benefit the SPL community, which may extend the list of findings in this field.

Finally, although we focused only on feature models, other approaches (*e.g.*, OVM, decision models, and CVL) are also important in SPL [Vale et al., 2016]. We did not include such approaches due a lack of studies in this field. Still, analyzing such studies remains as an important next step.

3.7 Related Work

A broad SLR has been conducted by Heradio et al. [2016] to identify the most influential researched topics in SPL, and how the interest in those topics has evolved over the years. Strict to the SPL domain engineering phase, Vale et al. [2016] have provided a survey on variability modeling techniques. They reported several dozens of variability modeling approaches (inclusive feature models). In addition, Chen and Babar [2011] conducted an SLR and reported how different types of variability modeling are evaluated. Also Lisboa et al. [2010] and Pereira et al. [2015] presented an SLR of variability management tools to support the SPL domain engineering phase. Although these reviews are not directly related to ours, the high level of detail of their research methodology supported to structure and define our own methodology.

Benavides et al. [2010] presented the results of a literature review conducted up to December 2009. The aim of this review was to identify a set of thirty operations and techniques that provide automated analysis of feature models, covering the phases of domain and application engineering. These operations are mainly focused on the extraction of information from feature models. Since Benavides et al. [2010] have shown that even with improved feature model analysis the configuration task is still complex, we aim at presenting a complementary review on product configuration. In our review, we have excluded studies that return all the products represented by the model, or basically answer the single question of how many valid configurations a model has (EC4). Besides the thirty operations defined in Benavides et al. [2010], we

have additionally considered *configuration optimization* and *multi-step configuration* as relevant mechanisms to support the SPL configuration process. Thus, the review from Benavides et al. [2010] has been substantially extended in terms of the research methodology and findings from recent contributions.

Benavides et al. [2013] have searched how similar are the operations defined in Benavides et al. [2010] with the operations in product configuration. However, the authors performed a rapid survey and highlighted the need to conduct a more exhaustive SLR and define a historical catalogue of configuration operations similar to what has been reported in the feature model analysis literature by Benavides et al. [2010]. Based on this study, we have created a preliminary catalogue of mechanisms supported by the SPL configuration process.

Rabiser et al. [2010] present the results of an SLR conducted up to February 2008. The goal of this review was to identify key requirements to support the entire application engineering process. Complementary, Lopez-Herrejon et al. [2015] and Harman et al. [2014] conducted a systematic mapping study to identify for which requirements have search-based software engineering techniques been used. In contrast to both reviews, our SLR focuses just on the SPL configuration process and contributes with a catalogue of mechanisms that serves as a summarization of the results in this specific field.

In a more strict scope, Afzal et al. [2016] conducted an SLR to investigate how to solve conflicts on multiple stakeholder configurations, and Guedes et al. [2015] conducted a review on dynamic SPL configuration at runtime. We also presented contributions in these areas. However, we presented a more broad view of these mechanisms and how they are related with other mechanisms. Finally, complementary to our review, Asadi et al. [2016] and Pereira et al. [2016a] investigated the effects of visualization techniques in SPL configurators by mean of a controlled empirical user study. They focus on the empirical analysis of the manual configuration activity reported in Section 3.4.3.

In conclusion, none of the aforementioned surveys directly address the SPL configuration process. The main concerns addressed by previous SLRs include: SPL testing, variability management, requirements engineering, evolution, and variability model analysis. The configuration problem is not studied in isolation in any of these SLR. In contrast, the current review provides further details on the SPL configuration process, such as information related to configuration constraints, techniques employed for solving the SPL configuration problem, and performance and scalability results of existing approaches.

3.8 Summary

Currently, there are multiple heterogeneous contributions in the SPL configuration domain. Thus, this chapter presents a *Systematic Literature Review* (SLR) that shows an overview of the progress, trends, and gaps faced by researchers in this field. We compare and classify a total of 157 primary studies from January 1^{st} 2012 to December 31^{sh} 2017 by following Kitchenham and Charters [2007] SLR guidelines. Mainly, we give an in-depth view of mechanisms used by each work, how these

mechanisms are empirically evaluated and their main shortcomings. The results of the review reveal that the research in SPL configuration is still fragmented and diverse. Available techniques have been developed fairly independently with different purposes to address different mechanisms. Our review mainly identified *(i)* the need to improve the quality of the empirical evaluation of existing approaches; *(ii)* the lack of holistic solutions to support multiple configuration constraints; and *(iii)* the need to improve scalability and performance conditions. This seems to indicate that given the increasing interest and importance of this field, there are many exciting opportunities of research. Therefore, in the next chapters, we make use of recommendation techniques to overcome some of the previous identified challenges.

4. Personalized Software Product Line Configurations

This chapter shares material with the Computer Languages, Systems & Structures paper "Personalized Recommender Systems for Product-line Configuration Processes" [Pereira et al., 2018b]. We presented initial ideas at GPCE'16 [Pereira et al., 2016c]. Furthermore, we have faced the need to implement such approach by conducting a systematic literature review published at ICSR'15 [Pereira et al., 2015] and an empirical user study published at ENASE'16 [Pereira et al., 2016a].

In this chapter, we propose the use of recommender techniques to support the *Software Product Line* (SPL) configuration process. Although there are several approaches in the literature that make use of recommender techniques [Bagheri et al., 2010a,b, Bagheri and Ensan, 2014a, Bagheri et al., 2012b, Galindo et al., 2015a, Martinez et al., 2015b, 2014, Mazo et al., 2014b, Tan et al., 2014a], this chapter presents the first approach that uses an automated and personalized recommender system that learns about the relevant features from past configurations. Therefore, no intervention of human experts is necessary in the creation of the recommendations. Figure 4.1 shows a complete overview of our approach.

Based on the literature [Bosch et al., 2015, Constantino et al., 2016, Payne et al., 1993, Pereira et al., 2015, 2013], we identify the following configuration challenges that arise from existing works and industry needs when using configurators:

- Real-world SPLs yield feature models that can be large and complex with too many options and complex relationships (*e.g.*, the Linux kernel [Sincero et al., 2010]). The amount and complexity of options shown by the existing configurators may lead decision makers to *make poor decisions* due to the difficulty in reasoning about the dependencies between features.
- Psychological studies [Payne et al., 1993] have shown that decision makers (*e.g.*, requirement engineers and business analysts) are usually unsure about users' needs

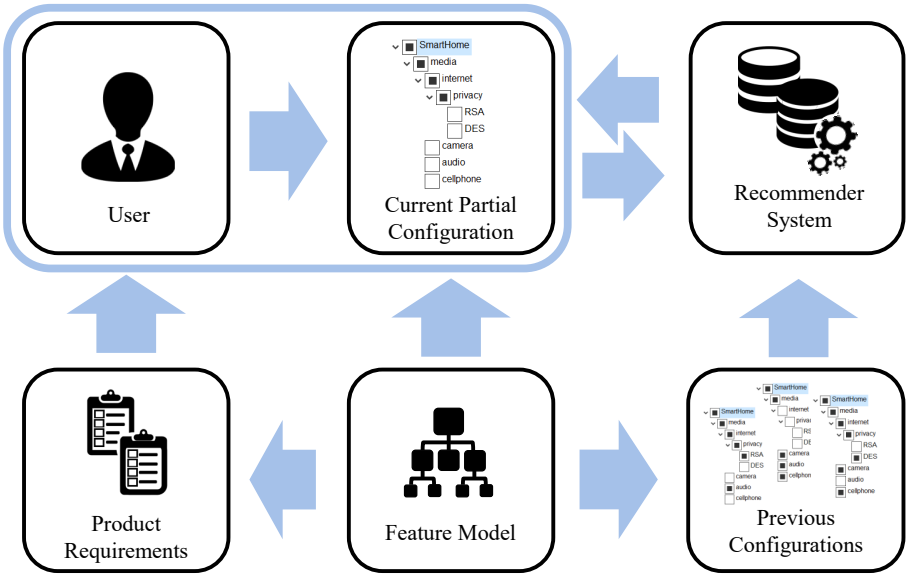


Figure 4.1: Proposed configuration components and their interplay.

when confronted with a large set of choices. According to Mazo et al. [2014b], a high percentage (more than 60%) of decision makers are unable to find satisfactory configurations for highly customized products.

- It is difficult to define a valid configuration since often users specify requirements that are inconsistent with the feature model's constraints, and also features of no importance to the user need to be taken into account in order to fulfill the constraints [Pereira et al., 2016b]. Moreover, logic inference rules may explicitly infer additional constraints that are not shown by the configurator.

Thus, handling the product configuration process in large SPLs is still a critical issue for many companies and the current literature still lacks a configurator to support this process. To overcome these challenges, this chapter presents a new approach to guide decision makers through a step-wise selection of features avoiding useless and invalid decisions. The proposed approach encompasses a *feature-based personalized recommender system* that aids a user in the configuration process. The main contribution of our approach is to ease the configuration process by effectively guiding the decision makers at understanding users' needs and preferences. The approach is mainly directed to decision makers who lack sufficient personal experience to evaluate the complex technical feature properties. In addition, domain experts and product developers can also take advantage of this approach as it offers a faster and less error-prone configuration process. In summary, we provide three main contributions:

1. We propose a personalized recommender system that guides users through the decision-making process and allows them to focus on valid and relevant parts of the configuration space. The system is based on configurations from previous users to generate personalized recommendations for a current user.

2. We design an open-source configurator[1] to support our approach by extending a state-of-the-art configurator FeatureIDE [Meinicke et al., 2017].

3. We empirically evaluate the recommendations' quality of six different recommender algorithms on two real-world datasets of configurations derived from two industrial case studies.

To evaluate our approach, we answer the following three research questions:

- *RQ1.* Can recommender approaches support the SPL configuration process in realistic configuration scenarios?

- *RQ2.* In which phase of product configuration can a recommender system make good recommendations?

- *RQ3.* What is the impact of the implemented algorithms on the quality of recommendations?

To address these research questions (*RQ1–3*), we conducted numerous experiments with six different recommendation algorithms on two datasets from our industry partners (see Section 4.3.1). To address *RQ1*, we evaluate the performance of the implemented algorithms by comparing them with the performance of a random algorithm that simulates the performance of an uninformed user without any support from a recommender system. This algorithm recommends randomly chosen features and, therefore, it indicates the minimal performance level every algorithm should reach. Regarding *RQ2*, we investigate if useful feature recommendations can be performed by the algorithms with a reasonable percentage of selected features as input. Finally, *RQ3* compares the accuracy of results from seven different recommender methods. Accuracy evaluates how well our proposed algorithms are capable of understanding the preferences of the users and give recommendations that are in accordance with the decisions that the users actually have in mind. The main purpose of *RQ3* is to evaluate which algorithm is most effective in producing a feature model configuration.

The remainder of this chapter is structured as follows. Section 4.1 motivates the business need for a feature-based recommender system by presenting the main challenges faced by a company when configuring a product. Section 4.2 provides an overview of the proposed approach. Section 4.3 presents the results of our experiments using a large dataset of real-world configurations derived from two business domains. Section 4.4 discusses the threats to the validity of our evaluation results. Section 4.5 clarifies the position of this work in the literature, highlighting the gaps filled by our approach. Finally, Section 4.6 summarizes the contributions of this chapter.

[1]The tool and the full code can be found in the Web supplementary material http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/.

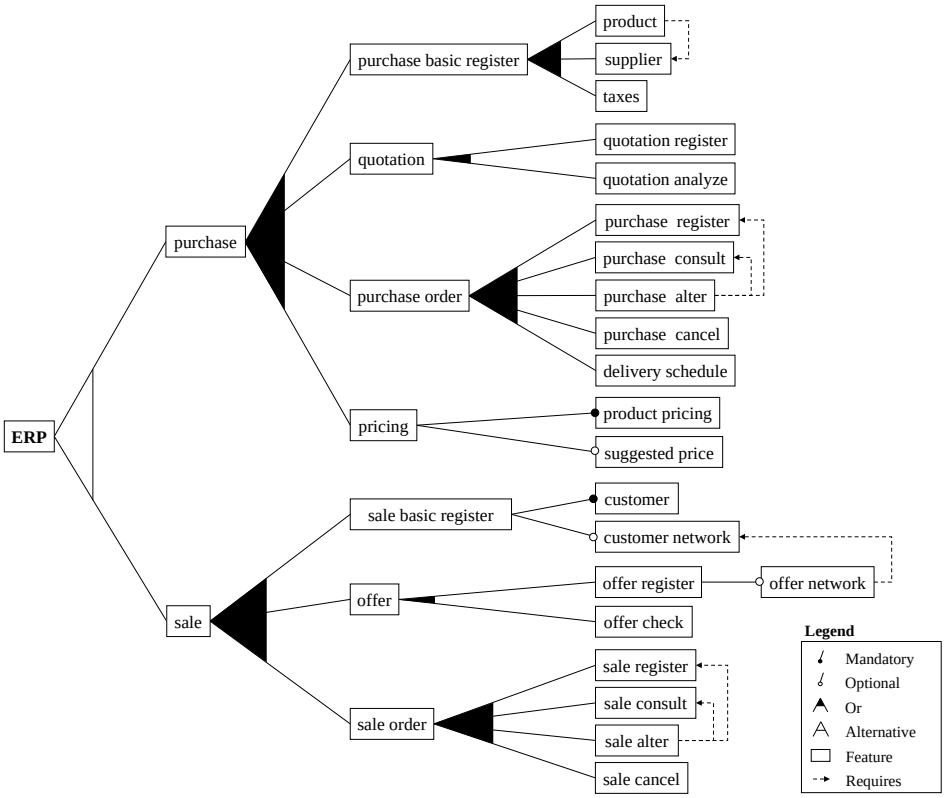


Figure 4.2: A simplified feature model for an ERP SPL.

4.1 Open Issues in SPL Configuration

Many companies provide solutions for customization of their products to meet the needs of each user. We illustrate the challenges of the product configuration process based on a real-world SPL of an *Enterprise Resource Planning* (ERP) system from a medium-sized business partner (Figure 4.2 shows an extract of the ERP feature model[2]). Our business partner operates with several departments and more than a hundred active users. Thus, this system is composed of features (*i.e.*, modules) that are configured in order to fulfill different user needs. The model illustrates an integrated view of core business processes (*i.e.*, purchase and sale system modules). Beyond the purchase and sale modules, the complete version of the ERP system consists of multiple enterprise software modules that are: supply chain management, inventory control, finances, manufacturing, accounting, fiscal affairs, customer relationship and marketing. The ERP system integrates these various modules into one complete system to manage processes and information across the entire business. The complete system is considered a large application as it is composed of 1,653 features and 171 active users (*i.e.*, previous configurations).

Product configuration is performed as a process of customization and generation of a specific product for each user. The company configuration strategy takes into account two activities: *requirements specification* and *decision making*. The requirements specification involves multiple stakeholders, while decision making is

[2]The complete representation of the feature model can be found at http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/datasets-download/ERP-System.xml

conducted only by an *Information Technology* (IT) expert. The *requirements specification* activity is concerned with collecting the system's requirements to satisfy the users' needs. The *decision-making* activity is performed interactively, mapping the system's requirements to features in the feature model. Finally, as a result of the process, a complete and valid product configuration (see definitions 3 and 4 in Section 4.2.1) is derived and delivered to the user.

In the SPL configuration context, there are many specialized configurators (see Section 3.4.3) to guide IT experts. However, although most of these tools ensure that any partially configured product is in accordance with the feature model constraints (which prevents users from introducing conflicts into their configuration), configurators do not yet adequately support business needs [Bosch et al., 2015]. After an interview with eight IT experts, we identify the following three main configuration challenges faced by the company when using configurators:

Challenge 1: Large SPLs. Industrial SPLs often define several thousand of features. As an example, the complete ERP SPL of our business case in Section 4.3.1 contains 1,653 features and can generate billions of different valid product variants. Thus, the amount and complexity of the options presented by the configurator lead decision makers to *get lost* with so much information to be taken into account, and *spend too much time* reasoning about too many options and complex relationships. Thus, showing all features and their dependencies is impractical as a decision maker can only focus on one part of the configuration a time. Thus, it is crucial for companies to provide an easy to use support that will alleviate the configuration-related challenges faced by the decision maker.

Challenge 2: Unclear Requirements and Features. The requirements are very often not clear and exact; some vague descriptions may have been introduced to the requirements specification. In our business scenario, this happens because the IT expert (*i.e.*, decision maker) is not the one who specifies the system's requirements. Moreover, the feature model may present many subjective features that cannot be matched with the requirements. Finally, if a final product does not meet the requirements, most likely, a re-configuration process has to be carried out. Completely re-configuring products is not desirable as the user has to wait for a new configuration. Thus, decision makers need further support to make appropriate decisions.

Challenge 3: Get a Final Configuration. It may be very difficult to define a valid configuration since quite often stakeholders specify requirements that are inconsistent with the feature model's constraints, and also features of no iterest are normally needed to fulfill the feature model's interdependencies. As an example, imagine that when mapping the system's requirements with their respective features implied by the feature model, the features that are part of an alternative exclusive relationship are not relevant to the user. In this case, since one feature must be selected, adequate support is still missing in the literature to guide decision makers to compare the features in order to select the most appropriate one. Moreover, in another scenario, consider that the decision maker has already selected all the users' features of interest according to the specified requirements. However, they still may

not have a valid configuration. Once the remaining features are not relevant to them, support is also needed to guide the decision makers into a valid final configuration.

In summary, our business experience shows that the SPL configuration process can be a challenging since decision makers regularly do not know every feature and their interdependencies, particularly for large SPLs. Thus, easy and comprehensive tool support is crucial to guide users in making decisions. In the next section, we propose a feature-based recommendation approach to overcome the above challenges.

4.2 The Proposed Approach

As seen in our motivating example (Section 4.1), the current configuration process is still challenging for decision makers. To ease this process, we propose a feature-based recommender system that guides decision makers during each step of the SPL configuration process by delivering capabilities to effectively understand users' needs and preferences. In this section, we formally define the terms feature model and configuration as well as certain relevant properties of configurations. We then use these definitions to give an overview of our approach and describe our chosen recommender system.

4.2.1 Formal Definitions

In this section, we formally define feature models and configurations for the formal description of the proposed configuration process. As we adapt recommendation algorithms to the configuration process, we need a single formalism that is suited to describe and combine both concepts. An existing formalism to describe a feature model are propositional formulas, as they can represent arbitrary boolean dependencies [Batory, 2005]. As the dependencies of a feature model are usually defined as a conjunction of constraints, a corresponding propositional formula can be efficiently transformed into a *Conjunctive Normal Form* (CNF) (*i.e.*, a conjunction of clauses, which are a disjunction of literals, which represent the single features), which is easy to formalize. Consequently, we use the CNF representation in our implementation for decision propagation and validity checking.

Definition 4.1. *A* feature model $\mathcal{FM} = (\mathbb{F}, \mathcal{R})$ *is a tuple that consists of a feature space* $\mathbb{F} = \{-1, 0, 1\}^h$*, where* h *is the number of features in the feature model, and a set of constraints* $\mathcal{R} = \{\vec{r}_1, \vec{r}_2, ..., \vec{r}_m\}$*, where* m *is the number of constraints of the feature model. A constraint* $\vec{r}_i$ *represents a clause from the feature model's propositional formula in CNF, such that* $\vec{r}_i \in \mathbb{F}$ *and the component* j *of* r_i *specifies whether the feature* j *should be selected* ($r_{i_j} = 1$)*, deselected* ($r_{i_j} = 0$)*, or is not relevant* ($r_{i_j} = -1$) *within this constraint.*

As an example, consider a feature model with four features `A`, `B`, `C`, and `D` and the constraints $\texttt{A} \Rightarrow (\texttt{B} \vee \texttt{C})$ and $\texttt{D} \Rightarrow \texttt{C}$. The formal representation of this feature model has a feature space $\mathbb{F}$ with $h = 4$ and the two vectors in $\mathcal{R}$, $\vec{r}_1 = (0, 1, 1, -1)$ and $\vec{r}_2 = (-1, -1, 1, 0)$. The vectors correspond to the constraints in CNF $(\neg \texttt{A} \vee \texttt{B} \vee \texttt{C}) \wedge (\neg \texttt{D} \vee \texttt{C})$, where $\vec{r}_1$ represents the first clause and $\vec{r}_2$ the second one.

Definition 4.2. *Given a feature model $\mathcal{FM}$, a* configuration $\vec{c} \in \mathbb{F} = (c_1, c_2, ..., c_h)$ *represents a selection of features such that $c_i = 1$, if feature i is selected, $c_i = 0$ if it is deselected, and $c_i = -1$ if its state is undefined.*

For example, considering our previously defined feature model, the configuration $\vec{c}_1 = (1, 0, 1, 1)$, means that the features `A`, `C`, and `D` are selected and the feature `B` is deselected. The configuration $\vec{c}_2 = (-1, -1, 0, 1)$ states that feature `C` is deselected, feature `D` is selected and the features `A` and `B` are currently not defined.

Definition 4.3. *Given a feature model $\mathcal{FM}$, a configuration $\vec{c}$ is* complete *iff $\vec{c}$ defines each feature (i.e., $\forall i \in \{1, .., h\} : c_i \neq -1$), otherwise the configuration is* partial. *For a given feature model, we denote the set of all its complete configurations with $\mathcal{CC}$ and the set of all its partial configurations with $\mathcal{PC}$.*

Definition 4.4. *Given a feature model $\mathcal{FM}$, a configuration $\vec{c}$ is* valid *iff it satisfies all constraints in $\mathcal{R}$ when considering all undefined features in $\vec{c}$ as deselected, otherwise it is* invalid. *More formally, $\vec{c}$ is valid iff $\forall \vec{r} \in \mathcal{R}, \exists i \in \{1, .., h\} : r_i \neq -1 \wedge complete(c_i) = r_i$, where the function complete is defined as:*

$$complete(c_i) = \begin{cases} 0, & \text{if } c_i = -1 \\ c, & otherwise \end{cases}$$

For a given feature model, we denote the set of all its valid configurations with $\mathcal{VC}$ and the set of all its invalid configurations with $\mathcal{IC}$.

Our previously considered configuration $\vec{c}_1 = (1, 0, 1, 1)$ is complete and valid, as it defines every feature and satisfies both constraints in $\mathcal{R}$. In contrast, the configuration $\vec{c}_2 = (-1, -1, 0, 1)$ is partial, as the features `A` and `B` are undefined, and it is invalid, as it contradicts with the constraint $\vec{r}_2 = (-1, -1, 1, 0)$.

To recommend features during the configuration process, our recommender system uses a *configuration matrix* X as input. A configuration matrix provides the status of a set of configurations. In this context, given a feature model $\mathcal{FM}$ and a set of complete and valid configurations $\{\vec{c}_1, ..., \vec{c}_n\} \subseteq \mathcal{CC} \cap \mathcal{VC}$, a *configuration matrix* X is defined as:

$$X = \begin{bmatrix} c_{1_1} & c_{1_2} & \cdots & c_{1_h} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n_1} & c_{n_2} & \cdots & c_{n_h} \end{bmatrix}$$

Considere the feature model of our previous example. Assuming the configuration $\vec{c}_1$ selects the features `A`, `C`, and `D`, and deselects the feature `B` (*i.e.*, $\vec{c}_1 = (1, 0, 1, 1)$); and the configuration $\vec{c}_3$ selects the features `A` and `B`, and deselects the features `C` and `D` (*i.e.* $\vec{c}_3 = (1, 1, 0, 0)$), the corresponding matrix X is defined as:

$$X = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Given the variables and constants described above, our approach aims to restrict the configuration space and predict the relevance of undefined features in a way that the users can make decisions more easily.

4.2.2 An Overview of the Proposed Configuration Process

Our approach creates an interactive perspective for users and offers recommendations to maximize the chances to have an adequate configuration in the end. The workflow in Figure 4.3 presents an overview of the proposed configuration process. The users are engaged in all steps, knowing which features are considered and their importance for the final product. The interaction ensures that the users understand the configuration space and its limitations, and are also comfortable with the decisions that are made during the whole process.

The configuration process is carried out by considering five main activities: *configure*, *propagate decisions*, *check validity*, *calculate recommendations*, and *visualize*. As input, it takes the feature model; and the requirements for a particular product from users, customers, and other stakeholders. The same feature model is used to customize different products for new customers' application scenarios. As an example, the two real-world feature models from our case-study (see Section 4.3.1) have been in use by the respective companies for more than 10 years.

The process starts with the user defining features of interest from a focused view of the feature model. A focused view restricts the decision maker's view to direct subfeatures from selected features (see Section 7.2.1 in Chapter 7). Each time a user decides to define a feature, *decision propagation* is applied. This means that features that are implied by this decision according to the feature model's constraints are automatically (de)selected [Janota, 2008]. As a consequence, the user cannot define two conflicting features [Pereira et al., 2016b]. However, a configuration can still be invalid, if due to the feature model's constraints more features need to be defined. Next, the partial configuration is checked for validity. For every valid partial configuration, the user can decide to finish the configuration process by automatically deselecting all remaining undefined features. An invalid partial configuration requires the user to select or deselect more features. A highlighted view shows the user which decisions are necessary to have a final valid configuration by highlighting the corresponding features (see Section 7.2.1 in Chapter 7). Due to the application of decision propagation the user will eventually come to a valid configuration. Furthermore, a complete configuration created using this process will alway be valid.

In each step, the recommender algorithm predicts the relevance of each undefined feature (see Section 4.2.3). The proposed recommender system requires as input a current partial configuration with at least one selected feature and at least one previous configuration to use as historical data. In Section 4.3 we investigate when the proposed recommender algorithms hit the threshold where they have enough historical data to start making better predictions (see *RQ2* at the beginning of this chapter). The predictions are displayed as a 5-star feature score in the focused view to guide the current user through a step-wise selection of features. The predictions are constantly updated as new information is received based on selected and deselected features. Therefore, if the user is not familiar with the features and cannot decide what is the best choice, suggestions are presented. Finally, if the user wishes to finish the configuration process (*i.e.*, he does not have the interest to select any additional feature), the *focused* and *highlighted* view combined with the *recommender system* can support them to have a desired and valid configuration.

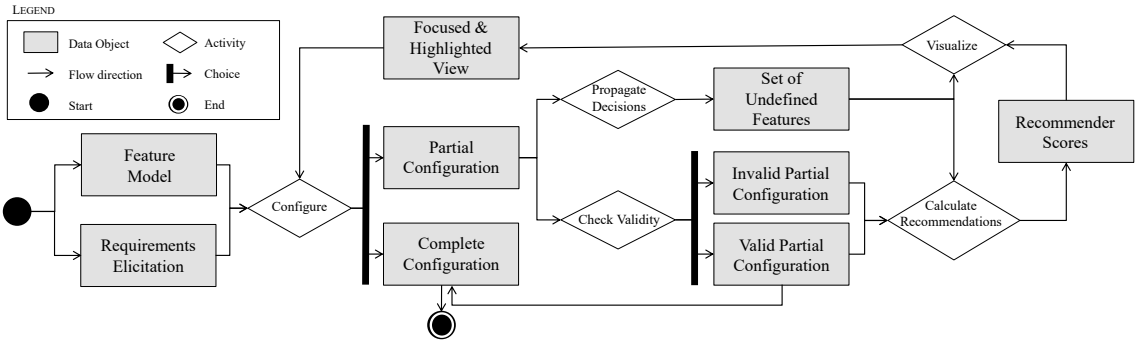


Figure 4.3: An overview of the configuration process.

In our case-study scenario (see Section 4.1), the predictions serve as input for IT experts to understand the users' needs, and consequently provide them with a set of features that are more likely to be useful. Moreover, given a previous configuration, our approach can also support configuration update and upgrade. Through the proposed configuration process IT experts can: *(a)* rely on a small relevant set of features, instead of going through the entire feature model; *(b)* visualize, in an interactive way implicit and explicit interdependences among features; and *(c)* go through a list of the most recommended features. In the next sections, we detail our proposed approach that centers around the above activities.

4.2.3 Recommender System Algorithms

Even with visual mechanisms for reasoning on feature models, manually configuring a product can be a massive and difficult process. To ease this process, we adapt six personalized recommendation algorithms to the scenario of SPL configuration (Section 2.2): *(i)* neighbourhood-based CF, *(ii)* CF-significance weighting, *(iii)* CF-shrinkage, *(iv)* CF-Hoeffding, *(v)* average similarity, and *(vi)* matrix factorization. These algorithms use previous configurations for estimating and predicting the relevance of features in order to guide a decision maker through the process of feature selection. The aim of implementing different algorithms is a comparative analysis and investigation of its performance in real configuration scenarios (see the research questions in the introduction of Chapter 4).

4.2.3.1 Neighbourhood-Based CF Recommender

In this section, we present how neighbourhood-based *Collaborative Filtering* (CF) algorithms can be adapted to recommend features in the SPL configuration domain.

Given the configuration matrix X (see Section 4.2.1) and a new partial configuration $\overrightarrow{pc} \in \mathcal{PC}$ that is currently being processed by a user, a recommendation algorithm has the task of finding relevant features for this partial configuration. To achieve that, the CF algorithm calculates the relevance scores for the non-selected features. Consider a simplified example in Table 4.1. The non-selected features in the partial configuration $\overrightarrow{pc}$ are f_1 and f_4. For those features a recommender system calculates the relevance scores. Once this calculation is completed, the most relevant features (the ones with the highest score) are recommended. The relevance scores are computed as follows.

For a partial configuration $\overrightarrow{pc}$ a set of neighbours is determined. A neighbour is a configuration $\overrightarrow{c}_x \in X$ (a row vector from X with $x \in \{1, ..., n\}$) that is similar to $\overrightarrow{pc}$ according to a similarity measure. A configuration qualifies as similar, if $sim(\overrightarrow{pc}, \overrightarrow{c}_x) > \tau$, where τ is a similarity threshold that is given as an input parameter (see Section 4.3.2.1). All neighbouring configurations build a neighbourhood $\mathcal{N}(\overrightarrow{pc}, \tau)$. In our example, the configuration that is the most similar to $\overrightarrow{pc}$ is $\overrightarrow{c}_1$, as they both have the features f_2 and f_3 co-selected. Consequently, the similarity $sim(\overrightarrow{pc}, \overrightarrow{c}_1)$ is high. This is not the case of $\overrightarrow{c}_2$ and $\overrightarrow{c}_3$. Therefore, if we consider $\overrightarrow{pc}$ and $\overrightarrow{c}_2$, it is likely that they have only a few features that have been selected commonly (co-selected). In this case, the measure of their similarity $sim(\overrightarrow{c}_1, \overrightarrow{c}_2)$ is based mostly on non-selected features (zeroes in the configuration vectors), which are not informative for a recommender system. This problem is known as unreliable estimation of similarity (unreliable similarity for short). The similarity measure is, in this case, not reliable, because its calculation is based only on few observations. We address this problem in the following subsections (Section 4.2.3.2 – Section 4.2.3.4).

	f_1	f_2	f_3	f_4
$\overrightarrow{c}_1$	0	1	1	1
$\overrightarrow{c}_2$	1	1	0	0
$\overrightarrow{c}_3$	0	0	1	1
$\overrightarrow{pc}$	?	1	1	?

Table 4.1: A simplified example of a configuration matrix with three previous configurations ($\overrightarrow{c}_1$, $\overrightarrow{c}_2$, $\overrightarrow{c}_3$) and a new partial configuration ($\overrightarrow{pc}$). f_1 - f_4 denote features of products.

Since our configuration matrix X is binary, in our experiments we use similarity measures such as Jaccard Coefficient, Mean Hamming Similarity and Dice Coefficient, which are appropriate for binary vectors. For explanation of those measures we refer to Choi et al. [2010].

Once the neighbourhood $\mathcal{N}(\overrightarrow{pc}, \tau) \subseteq \mathcal{CC}$ has been determined, the relevance score Rel for a feature f from the set of all non-selected features (*e.g.*, f_1 in our example) is calculated as follows:

$$Rel(\overrightarrow{pc}, f) = \frac{1}{|\mathcal{N}(\overrightarrow{pc}, \tau)|} \sum_{\overrightarrow{c}_x \in \mathcal{N}(\overrightarrow{pc}, \tau)} sim(\overrightarrow{pc}, \overrightarrow{c}_x) \cdot c_{x_f} \tag{4.1}$$

Note that the sum in the formula iterates over the neighbours of $\overrightarrow{pc}$. In other words, if many similar configurations from the neighbourhood have the feature f selected, then the relevance score for this feature is high. This is due to the term $sim(\overrightarrow{pc}, \overrightarrow{c}_x) \cdot c_{x_f}$, which weights the selected features with the similarity of the configurations. The fraction $\frac{1}{|\mathcal{N}(\overrightarrow{pc}, \tau)|}$ normalizes the sum, so that the whole expression results in being a weighted average. Note that $c_{x_f} \in \{0, 1\}$, therefore our formula differs from typical CF-based algorithms working with non-binary ratings. If $c_{x_f} = 0$, *i.e.* the feature f was not selected in the configuration c_x, then its similarity does not influence the relevance score. Finally, in the last step, the relevance scores $Rel(\overrightarrow{pc}, f)$ are returned and used for recommendations.

4.2.3.2 CF with Significance Weighting

CF with significance weighting, abbreviated hereafter as CF-significance, addresses a shortcoming of the conventional CF algorithm related to the unreliable similarity problem. CF does not consider the number of co-selected features in the calculation of similarity between configurations. Not considering the number of commonly selected features poses a problem, since a similarity between two configurations can be very high based merely on absent features, which are not informative. Unlike in our simplified example in Table 4.1, in real-world applications there are many more features. Often, in a configuration only a few of the features are selected. Consequently, the corresponding configuration vector contains mostly zeroes. A similarity between two such vectors can be high based merely on a few selected features. Despite high value, such a similarity is not reliable. To solve this problem, Herlocker et al. [1999] proposed significance weighting. According to this approach similarity based on more co-selected features is given more weight than a similarity based on only few observations.

Similarly to the conventional CF, significance weighting builds a neighbourhood $\mathcal{N}(\overrightarrow{pc}, \tau) \subseteq \mathcal{CC}$ of a configuration $\overrightarrow{pc}$ by considering similarities to all other historical configurations $\overrightarrow{c_x}$. However, the configurations that belong to the neighbourhood must additionally fulfill the following criterion $w(\overrightarrow{pc}, \overrightarrow{c}_x) > \tau$ with (notation from Desrosiers and Karypis [2011] adjusted):

$$w(\overrightarrow{pc}, \overrightarrow{c}_x) = \frac{min(n, \gamma)}{\gamma} \cdot sim(\overrightarrow{pc}, \overrightarrow{c}_x) \tag{4.2}$$

where n is the number of co-selected features between $\overrightarrow{pc}$ and $\overrightarrow{c}_x$ and γ is the weighting parameter. The term $\frac{min(n,\gamma)}{\gamma}$ penalizes (lowers) the similarity measures that are based on fewer co-selected features than γ. If γ is for example set to 20, then the configurations $\overrightarrow{pc}$ and $\overrightarrow{c}_x$ need to have at least 20 co-selected features in order not to be penalized by this weighting. This parameter is typically set by a domain expert or optimized. Also, the Equation 4.1 is updated as follows to consider the new weighting:

$$Rel(\overrightarrow{pc}, f) = \frac{1}{|\mathcal{N}(\overrightarrow{pc}, \tau)|} \sum_{\overrightarrow{c}_x \in \mathcal{N}(\overrightarrow{pc}, \tau)} w(\overrightarrow{pc}, \overrightarrow{c}_x) \cdot c_{x_f} \tag{4.3}$$

4.2.3.3 CF with Shrinkage

CF with shrinkage, denoted hereafter as CF-shrinkage, was proposed by Ma et al. [2007] to also address the problem of unreliable similarity. However, Ma et al. used a different weighting formula (notation from Desrosiers and Karypis [2011] adjusted):

$$w(\overrightarrow{pc}, \overrightarrow{c}_x) = \frac{n}{n + \beta} \cdot sim(\overrightarrow{pc}, \overrightarrow{c}_x) \tag{4.4}$$

with β being a parameter that controls the extent of weighting. n is again the number of co-selected features between $\overrightarrow{pc}$ and $\overrightarrow{c}_x$. If n is much greater than β then the weighting has only little effect. Otherwise, configurations having only a few co-selected features are penalized more strongly [Desrosiers and Karypis, 2011].

4.2.3.4 CF with Hoeffding Bound

To address the problem of unreliable similarity, Matuszyk and Spiliopoulou [2014] proposed a different approach, denoted hereafter as CF-Hoeffding. Instead of weighting similarities based on the number of co-selected features, they introduced the concept of a *reliable neighbourhood*.

A *reliable neighbourhood* contains only configurations that are reliably similar to the current partial configuration $\overrightarrow{pc} \in \mathcal{PC}$. The reliability is determined using a *baseline configuration* $\overrightarrow{c}_B$. Matuszyk et al. proposed several methods to compute a baseline configuration. We adjust those methods to our scenario with binary data and we use a *random baseline* and a *majority baseline*.

The *random baseline* is defined as a random binary vector determining if a feature is selected or not. The *majority baseline* is also a binary vector that contains a feature (*i.e.*, an entry of 1), if the majority of configurations contains that feature. Therefore, this baseline represents a default configuration, as it would be defined by an average user.

The baselines are used in determining the reliability of a neighbour. A given historical configuration $\overrightarrow{c}_x \in X$ is *reliably similar* to the current partial configuration $\overrightarrow{pc} \in \mathcal{PC}$, if $sim(\overrightarrow{pc}, \overrightarrow{c}_x) \gg sim(\overrightarrow{pc}, \overrightarrow{c}_B)$. The symbol $\gg$ denotes *significantly greater than*. This inequality is evaluated using a significance-based solution derived from Hoeffding's Inequality that considers the number of co-selected features n and a confidence level δ. For details on calculation we refer to Matuszyk and Spiliopoulou [2014]. All historical configurations that fulfil this inequality are considered neighbours. Once the neighbourhood is defined, Equation 4.1 can be applied as in the conventional CF to predict the relevance scores of all unselected features.

4.2.3.5 Average Similarity Recommender

This algorithm uses the same principle as CF, CF-significance weighting, CF-shrinkage, and CF-Hoeffding, but it does not use the notion of a neighbourhood. Consequently, the configurations of all users are considered for computing the relevance score of a feature. Accordingly, Equation 4.1 is changed to:

$$Rel(\overrightarrow{pc}, f) = \frac{1}{n} \sum_{\overrightarrow{c}_x \in X} sim(\overrightarrow{pc}, \overrightarrow{c}_x) \cdot c_{x_f} \tag{4.5}$$

Note that the sum iterates over all configurations from X, which contains n configurations. This means that the relevance score is an average similarity of $\overrightarrow{pc}$ over the configurations that have the feature f selected.

We conduct experiments with this algorithm to investigate if restricting the neighbourhood size has an influence on the quality of recommendations in our application.

4.2.3.6 Matrix Factorization Recommender

In contrast to neighbourhood-based methods, matrix factorization algorithms do not rely on the similarity of configurations. Instead, they transform the configuration

matrix X (as presented *e.g.* in Table 4.1) into a latent space. In the following, we give an explanation of this transformation. However, matrix factorization in recommender systems constitutes an entire field of research that cannot be presented here in an exhaustive way. Therefore, for more information on this concept we refer to Bishop [2006] and Takács et al. [2009].

The transformation of the matrix X is obtained by incremental minimization of an error function. In this work, we use the BRISMF algorithm by Takács et al. [2009] and adapt it to our SPL configuration problem. Formally, this transformation is represented as follows:

$$X_{n\times h} = P_{n\times k} \cdot Q_{k\times h} \tag{4.6}$$

where P is a latent matrix of configurations and Q a latent matrix of features, n is the number of configurations in X (see Section 4.2.1), h the number of features, and k is the number of latent dimensions. k is an exogenous input parameter that needs to be optimized (see Section 4.3.2.1). The latent matrices are results of the factorization (the matrix X is factorized into two other matrices). They are called latent, because their values are not directly interpretable and their meaning is not known. Nevertheless, the values in the matrices are fitted in a way that minimizes the error in the prediction of the relevance score.

Using the transformation in Equation 4.6, the relevance score for a feature f in a partial configuration $\overrightarrow{pc}$ (*e.g.*, for the feature f_1 in our example in Table 4.1) is calculated using the following equation:

$$Rel(\overrightarrow{pc}_f) = \overrightarrow{p}_{\overrightarrow{pc}} \cdot \overrightarrow{q}_f \tag{4.7}$$

where $\overrightarrow{p}_{\overrightarrow{pc}} \in P$ is a row vector from the latent matrix P describing the configuration $\overrightarrow{pc}$ in the latent space. $\overrightarrow{q}_f \in Q$ is the corresponding latent vector of feature f, *i.e.* a column vector from the matrix Q. Both $\overrightarrow{p}_{\overrightarrow{pc}}$ and $\overrightarrow{q}_f$ have length k. Equation 4.7 describes how the prediction of a relevance score is obtained given the matrices P and Q. In the following, we describe how the values in those matrices are obtained.

First, the matrices P and Q are initialized randomly. To improve them, *Stochastic Gradient Descent* (SGD) is used to iteratively update the matrices by minimizing an error function. The error function used in the transformation is a prediction error between the true value $c_{x_f} \in X$ (*i.e.*, a value known from the data, *e.g.* $c_{x_f} = 1$, if a feature was selected in a configuration) and the predicted relevance score $Rel(c_{x_f})$ on a training set Tr (see Section 4.3.2.2 for definition of a training set):

$$Error = \sum_{c_{x_f} \in Tr} (e_{c_{x_f}})^2 \tag{4.8}$$

$$e_{c_{x_f}} = c_{x_f} - Rel(c_{x_f}) \tag{4.9}$$

For the training, only the selected features are used (*i.e.*, the entries, where $c_{x_f} = 1$), because the meaning of a zero is ambivalent. A zero in the configuration vector can mean deselecting a feature on purpose, or not selecting it, because the user did not

know the feature, even though it was relevant. Using the zero entries as indication of irrelevance would be misleading.

To minimize the error function and to update the matrices P and Q, SGD uses the following formulas:

$$\overrightarrow{p}_{c_x} := \overrightarrow{p}_{c_x} + \eta(e_{c_{x_f}} \cdot \overrightarrow{q}_f - \lambda \cdot \overrightarrow{p}_{c_x}) \tag{4.10}$$

$$\overrightarrow{q}_f := \overrightarrow{q}_f + \eta(e_{c_{x_f}} \cdot \overrightarrow{p}_{c_x} - \lambda \cdot \overrightarrow{q}_f) \tag{4.11}$$

where η is a learning rate and λ is a regularization parameter that prevents overfitting. Both of them are input parameters to be set in advance (see Section 4.3.2.1). f stands again for a feature as *e.g.* in Table 4.1. Note that those formulas update the vectors from the matrices P and Q in a way that minimizes the prediction error. Applying those formulas iteratively in several epochs of the *Stochastic Gradient Descent* (SGD) eventually leads to an approximately minimal prediction error. For the derivation of those formulas we refer to Takács et al. [2009].

Once the training using the SGD is completed (*e.g.*, due to convergence or maximal number of iterations) the matrices P and Q can be used to make relevance predictions, as presented in Equation 4.7.

4.3 Evaluation

This section describes the evaluation protocol used to evaluate the six different recommender algorithms introduced in Section 4.2.3. Since a configuration cannot be invalid due the decision propagation strategies automatically applied to the configuration process (see Section 4.2.2), we empirically evaluate the *quality of recommendations* of the proposed recommender algorithms. For the purpose of comparison, we also experiment with a random recommender system and report our results.

4.3.1 Target Software Product Lines and Datasets

In order to address the research questions (*RQ1—3*) introduced in the beginning of this chapter, we use two real-world datasets of configurations[3] from our industry partners as a configuration matrix (see Section 4.2.1). Table 4.2 summarizes the properties from both datasets (*i.e.*, ERP System and E-Agribusiness) used in the evaluation. For each dataset, we present four properties including the number of features ($\#f$), percentage of cross-tree constraints ($\mathcal{R}$), number of all valid configurations ($\#\mathcal{C}$), and number of previous configurations ($\#\overrightarrow{c}_x$). We give an upper bound on the number of valid configurations ($\#\mathcal{C}$) for the models since it is not feasible to determine the number of products for such a large model[4].

As shown in Table 4.2, although the number of features of both models is quite similar (*i.e.*, 1,653 and 2,008), these models cover a range of sizes in terms of historic configurations (*i.e.*, 171 and 5,749). While the ERP System dataset provides

[3]The configuration datasets can be found at wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/.

[4]We compute $\#\mathcal{C}$ by using the automated analysis mechanism of SPLOT at http://splot-research.org/.

a high-level representation of an SPL in the business management domain, the E-Agribusiness dataset represents variability in the e-commerce agribusiness domain. To the best of our knowledge, both feature models are the largest real-world datasets of configurations ever cited in the literature. They have a very high degree of variability which would be hard for the decision maker to process without any additional support. These characteristics from both target feature models make them ideal to be employed in our experiments to explore our research questions.

Dataset	Domain	$\#f$	$\mathcal{R}$	$\#\mathcal{C}$	$\#\overrightarrow{c}_x$
ERP System	Business Management	1,653	$\approx 7\%$	$> 10^9$	171
E-Agribusiness	E-Commerce	2,008	none	$> 10^9$	5,749

Table 4.2: Main properties of the datasets.

4.3.2 Experiment Design

To evaluate our algorithms, we performed an offline evaluation that encompasses three components: *(i)* parameter optimization, *(ii)* splitting into training and test datasets, and *(iii)* evaluation metrics.

4.3.2.1 Parameter Optimization

Parameter optimization is essential for comparing algorithms that require parameter tuning. A typical example of an algorithm that requires parameter tuning is the k-means algorithm for clustering. In the k-means algorithm, a domain expert, who uses this algorithm, needs to specify the parameter k, which stands for the number of centroids. This parameter is not known a priori and it influences the quality of results greatly. If the domain expert sets the parameter incorrectly, the quality of the results will be poor. Therefore, it is common that domain experts optimize the parameters by trying several settings.

However, optimizing parameters manually is highly subjective and might lead to unreliable conclusions when comparing algorithms that were tuned manually. Therefore, it is beneficial to use an optimization algorithm for the parameter optimization to obtain objective and replicable results.

Recommender systems also require setting of parameters. Neighbourhood-based CF, for instance, requires a similarity measure and a threshold value τ, above which users are considered neighbours. Matrix factorization algorithms also have parameters to set, *e.g.*, the number of latent dimensions k, the learning rate η and the regularization λ. Regularization is a mechanism often used in machine learning to prevent overfitting of models to data (see Bishop [2007] for details on this technique).

To ensure replicability of our results, in the following we describe, how we optimized the parameters of our recommendation algorithms. In our optimization, we used a genetic algorithm. We used a random hold-out sample of 30% of all configurations from our dataset, *i.e.* this set was held out from further training and evaluation for the sake of reliable conclusions. On this hold-out dataset the aforementioned genetic algorithm was used to optimize the F-Measure of different recommendation

algorithms (see Section 4.3.2.3). Every experiment in this phase was validated using a 10-fold cross validation. The optimal parameter setting was then used in our main validation phase (see Section 4.3.2.2) and applied to the remaining 70% of configurations. The optimal parameter values are shown in Table 4.3. In the column "method" of the table, we show our recommendation algorithms. The "parameter" column shows which parameters the corresponding algorithm requires. The last two columns show the optimal values of the corresponding parameters on the two datasets used in our experiments. Note that optimal parameter settings depend on the application domain and they are, therefore, different on different datasets. The results presented in the following sections were achieved using those optimal parameter settings.

		Dataset	
Method	**Parameter**	**ERP System**	**E-Agribusiness**
Avg. Sim.	*Sim.Measure*	Jaccard Coeff.	Jaccard Coeff.
kNN-CF	τ	0.000001	0.757576
	Sim.Measure	Jaccard Coeff.	Jaccard Coeff.
CF significance	τ	0.994	0.469
	Sim.Measure	Dice Coeff.	Jaccard Coeff.
	γ	10	207.98
CF shrinkage	τ	0.999	0.33
	Sim.Measure	Jaccard Coeff.	Mean Hamming Sim.
	β	14.95	158.48
Hoeffding CF	*baseline*	majority	random
	Sim.Measure	Jaccard Coeff.	Mean Hamming Sim.
	δ	0.999	0.988
BRISMF	k	40	80
	η	0.0166	0.0948
	λ	0.0062	0.1

Table 4.3: Optimal parameter values.

4.3.2.2 Splitting into Training and Test Datasets

Once the optimal parameter settings were found, we performed our main evaluation on the remaining 70% of configurations. For the sake of a fair evaluation it was necessary to split the dataset into disjoint training and test datasets. In this main evaluation phase, we use the leave-one-out evaluation protocol to create realistic evaluation conditions. According to this protocol, one configuration is left out from the training set and used for testing. The remaining configurations are given to the algorithm as training data (*i.e.*, the configuration matrix X). This simulates the behaviour of a real system, where a user logs in and carries out a new configuration. The data of the past configurations are available to the system and only the current partial configuration should be predicted. To perform well, a recommender system has to recommend the features that were used in the left-out test configuration based on the training data (*i.e.*, all other configurations).

Formally, a test configuration is a partial configuration $\overrightarrow{pc} \in \mathcal{PC}$ that was left out from training set, *i.e.* it is not contained in the configuration matrix $\overrightarrow{pc} \notin X$. The recommender system returns an estimated configuration $\widehat{pc} = \{0,1\}^h$. Then, the degree of overlap between the real configuration $\overrightarrow{pc}$ (known from the data, but held

out from the recommendation algorithm) and the predicted one $\widehat{pc}$ is calculated using an evaluation measure (see Section 4.3.2.3; note that it is not a similarity measure from Section 4.2.3.1).

To further simulate the progress of a user in the configuration process, we gradually give parts of the configuration to the recommender system as training data, *e.g.*, 10% of a complete configuration. Then, a new prediction $\widehat{pc}_{@10\%}$ is made and its quality is estimated using the aforementioned quality measure. The quality of a prediction is good if the predicted configuration overlaps with the remaining 90% of the configuration that is not known to the system. A good recommender system should learn from the given parts of the configuration and improve the quality of recommendations of the remaining features as it obtains more information about the current configuration.

The entire process is repeated for all configurations and all recommendation algorithms separately. The final quality measure of a recommendation algorithm is the average quality over all configurations.

4.3.2.3 Evaluation Metrics

In our evaluation, we use precision, recall and F-measure at w, where w is the number of recommendations displayed to a user. We use a value of $w = 10$, which is typical to recommender systems, since usually no more than 10 recommendations can be displayed (see Shani and Gunawardana [2011] for more details). Even if more recommendations are displayed, users mostly consider only the top positions. In the following, we explain how those measures are calculated.

Given a set Rec of features recommended by an algorithm and a set of truly relevant features Rel known from a test configuration, precision is calculated as follows: $Precision = \frac{|Rec \cap Rel|}{w}$. Analogously, recall is calculated using the formula: $Recall = \frac{|Rec \cap Rel|}{|Rel|}$. Precision states how many of the recommended features were relevant. Recall measures what percentage of relevant features has been recommended. Thus, in an ideal recommender system these values should be maximal (*i.e.*, equal to 1). However, maximizing one of them, while ignoring the other, is not a challenging task. Therefore, in our evaluation we use a measure that combines both, *i.e.* the F-Measure:

$$F\text{-}Measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{4.12}$$

On the one hand, F-Measure is high when both precision and recall are high. On the other hand, when only one of the components is high (*i.e.*, either precision or recall), then F-Measure is low. This is an ideal characteristic, since our goal is to maximize both precision and recall.

Evaluation metrics that are typical to recommender systems, such as RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) [Shani and Gunawardana, 2011], are in this case not applicable, since in our scenario we have binary data in the matrix X which is not compatible with these metrics (see Section 4.2.1).

4.3.2.4 Baseline Comparison

As a comparison baseline we use a random recommender system. It returns a randomly ordered list of non-selected features as recommendations. It is important to compare with this algorithm, because it indicates a basic performance level every algorithm should reach. If a method does not outperform the random recommender, then it should not be used in the given application scenario. Furthermore, the performance of this algorithm is equivalent to the performance of a hypothetical, fully uninformed user without any support from a recommender system. If a method outperforms the random baseline, it means that can be used in this scenario and that it performs a configuration better than an uninformed configurator. However, in most real-world applications no human configurator is fully uninformed. Therefore, to additionally indicate how useful our recommendation algorithms are, we report the precision and recall values. Those values indicate the quality of recommendations.

Precision and recall are objective measures of quality. The subjective value of a recommender systems to users can be determined only in a large user study. This, however, is part of Chapter 7.

4.3.3 Analysis of Results and Discussion

In our evaluation we performed more than 149,000 experiments on a cluster running the (Neuro)Debian operating system [Halchenko and Hanke, 2012]. As an experiment, in this case, we understand making and evaluating recommendations for one user in the process of configuration. Therefore, the total number of experiments depends on the total number of previous configurations in both datasets. Furthermore, each experiment was conducted at different stages of the configuration process. For each experiment, there are nine such stages for different level of completeness of a configuration (*e.g.*, 10%, 20%, etc.). All experiments were conducted for each of our seven methods separately. Consequently, this gives us 4,720 experiments on the ERP dataset and 144,864 experiments on the E-Agribusiness dataset. Note that a part of the configurations from the datasets have been held out for the purpose of parameter optimization (see Section 4.3.2.1). Those additional experiments for parameter optimization are not counted here.

In Figure 4.4 and Table 4.4 we present the results on the dataset from the ERP domain. Since we performed a leave one out validation for each configuration separately, the results presented here are averages over all configurations. The figure represents the F-Measure achieved by seven different recommender methods. On the horizontal axis of the figure, we present the completeness of a configuration, *i.e.* the percentage of features that are given to the algorithm as training data, where only the remaining part of the configuration needs to be predicted.

On the ERP dataset, we observe that the CF-shrinkage and CF-significance weighting algorithms outperform all other recommendation algorithms at all stages of the configuration process, with a better result for the CF-shrinkage algorithm. Moreover, although the BRISMF algorithm does not outperform those two algorithms, it presents also good results outperforming the other three algorithms, as well the random recommender. We observe an increase of performance for those algorithms

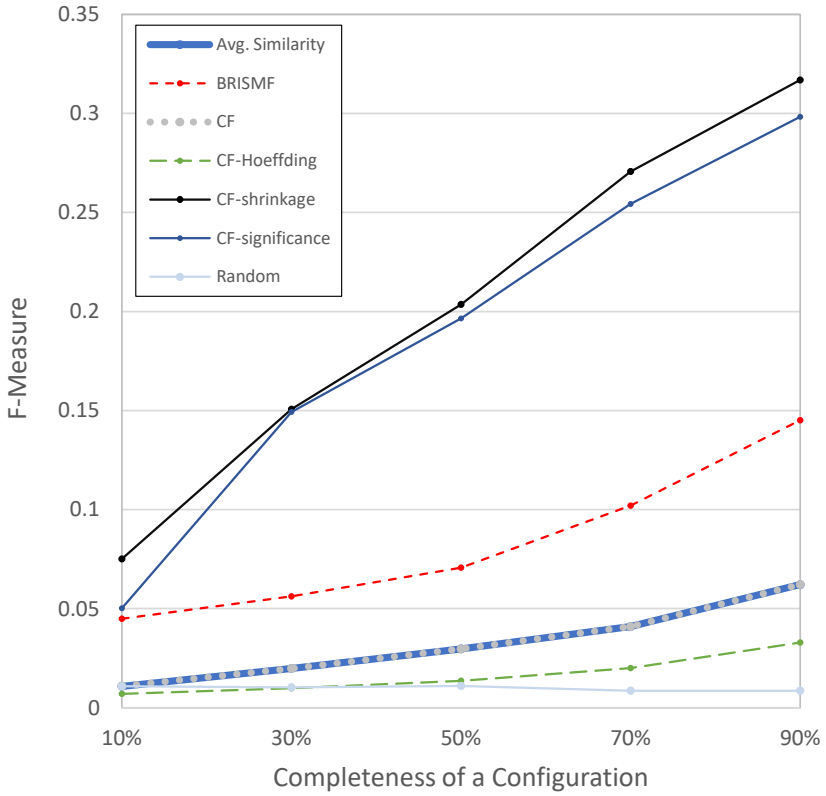


Figure 4.4: F-Measure achieved by seven different recommender methods on the ERP dataset (higher values are better). The horizontal axis shows how much of the current configuration has been completed. The performance is calculated on the remaining part of a configuration.

as the configuration becomes more complete, *i.e.* as the recommendation algorithm receives more data for training.

The CF algorithm and the average similarity algorithm perform nearly the same (their curves overlap). They dominate the random recommender at nearly all stages of configuration process, except for the initial part, when little information about the current configuration is available. In addition, note that the CF-Hoeffding algorithm only outperforms the random recommender when we have a partial configuration with more than 50% of selected features. The random recommender only shows the minimal level of F-Measure by guessing the relevance scores (*i.e.*, it serves only as a comparison baseline). Therefore, the CF-Hoeffding algorithm is not recommended to be used in this domain.

The corresponding numeric values of F-Measure and also precision and recall are shown in Table 4.4. Also in the table, we see that the CF-shrinkage and CF-significance weighting algorithms dominate the other algorithms not only with respect to F-Measure, but also with respect to precision and recall at nearly all stages of the configuration process.

In Figure 4.5 and Table 4.5, we present the analogous results on the *E-Agribusiness dataset*. On this dataset, the BRISMF algorithm performs the best, except for the

Measure	Method	Completeness of Configuration 0.1	0.3	0.5	0.7	0.9
F-Measure	Avg. Similarity	0.0108	0.0198	0.0297	0.0409	0.0621
	BRISMF	0.0449	0.0562	0.0707	0.1020	0.1451
	CF	0.0109	0.0198	0.0297	0.0409	0.0622
	CF-Hoeffding	0.0070	0.0098	0.0136	0.0200	0.0329
	CF-shrinkage	**0.0751**	**0.1507**	**0.2036**	**0.2707**	**0.3169**
	CF-significance	**0.0502**	**0.1493**	**0.1965**	**0.2543**	**0.2983**
	Random	0.0108	0.0103	0.0110	0.0086	0.0086
Precision	Avg. Similarity	0.0517	0.0644	0.0737	0.0814	0.0847
	BRISMF	**0.4203**	0.3975	0.3754	0.3203	0.2475
	CF	0.0525	0.0644	0.0737	0.0814	0.0856
	CF-Hoeffding	0.0678	0.0627	0.0788	0.0822	0.0754
	CF-shrinkage	**0.4093**	**0.5347**	**0.5390**	**0.5059**	**0.4068**
	CF-significance	**0.3644**	**0.5237**	**0.5102**	**0.4686**	**0.3788**
	Random	0.1153	0.0805	0.0788	0.0492	0.0263
Recall	Avg. Similarity	0.0069	0.0133	0.0217	0.0324	0.0587
	BRISMF	0.0240	0.0306	0.0397	0.0625	0.1073
	CF	0.0069	0.0133	0.0217	0.0324	0.0588
	CF-Hoeffding	0.0038	0.0062	0.0093	0.0136	0.0242
	CF-shrinkage	**0.0450**	**0.0977**	**0.1479**	**0.2219**	**0.3072**
	CF-significance	**0.0277**	**0.0970**	**0.1435**	**0.2055**	**0.2923**
	Random	0.0060	0.0064	0.0064	0.0052	0.0078

Table 4.4: Performance of seven recommender methods *w.r.t.* F-Measure, precision and recall on the ERP dataset. The CF-shrinkage algorithm performs the best *w.r.t.* all three measures at nearly all stages of a configuration process.

initial part of a configuration (*i.e.*, less than 12% of selected features). In this part, the CF, CF-Hoeffding, CF-shrinkage, and CF-significance weighting algorithms yield a better result recall and F-Measure. At all other stages of the configuration process BRISMF clearly outperformed the remaining algorithms.

Different than on the ERP dataset, we observe an decrease of performance for those algorithms as the configuration becomes more complete. Moreover, here the CF and average similarity algorithms perform very differently. While CF is the second best algorithm, the average similarity algorithm performs worse than the random recommender. Consequently, this algorithm should not be used in the E-Agribusiness scenario.

In comparison to the ERP dataset the absolute values of the quality measure are lower on the E-Agribusiness dataset. Since this dataset is more difficult in terms of predicting the relevant features, good predictions with a random recommender are unlikely, and the relative performance difference between the random recommender and the other algorithms, especially BRISMF, is larger.

When analyzing the datasets in details, we could notice that for each configuration in the E-Agribusiness dataset there are in average a lower number of selected features

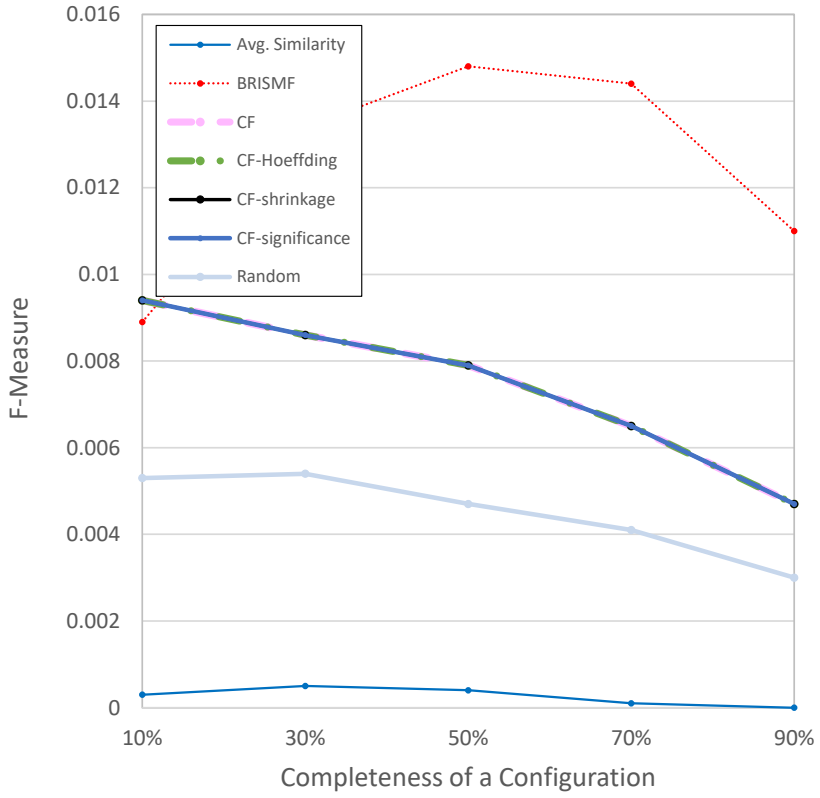


Figure 4.5: F-Measure achieved by seven different recommender methods on the E-Agribusiness dataset.

(*i.e.*, 73 features) than for each configuration in the ERP dataset (*i.e.*, 207 features). This can also explains the different tendency of the curves. While the F-Measure curves on the ERP dataset show an increasing tendency (except for the random recommender), in the E-Agribusiness dataset such a tendency cannot be observed. In the E-Agribusiness dataset when a configuration becomes more complete, then there are fewer and fewer features that are relevant, *i.e.* their relative proportion in the set of all features drops. This results from the fact that features used by a user cannot be recommended any more. Thus, the number of remaining relevant features decreases as the configuration becomes more complete. Therefore, precision that influences the F-measure also decreases. On this dataset this effect outweighs the learning effect and, therefore, the curves are not monotonically increasing. Nevertheless, the curve of the BRISMF algorithm increases initially and reaches optimum around 50% of a configuration.

Considering the results of our experiments, we answer the research questions (*RQ1–3*) as follows:

RQ1. Can recommender approaches support the SPL configuration process in realistic configuration scenarios? Yes, we have shown that the implemented recommendation algorithms find relevant features better than our comparison baseline and by recommending them it can support SPL configuration. Consequently, the automation of our approach assumes that the user is capable of providing the system

		Completeness of Configuration				
Measure	**Method**	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**
F-Measure	Avg. Similarity	0.0003	0.0005	0.0004	0.0001	0.0000
	BRISMF	0.0089	**0.0134**	**0.0148**	**0.0144**	**0.0110**
	CF	**0.0094**	0.0086	0.0079	0.0065	0.0047
	CF-Hoeffding	**0.0094**	0.0086	0.0079	0.0065	0.0047
	CF-shrinkage	**0.0094**	0.0086	0.0079	0.0065	0.0047
	CF-significance	**0.0094**	0.0086	0.0079	0.0065	0.0047
	Random	0.0053	0.0054	0.0047	0.0041	0.0030
Precision	Avg. Similarity	0.0018	0.0014	0.0009	0.0002	0.0000
	BRISMF	**0.0584**	**0.0529**	**0.0413**	**0.0259**	**0.0127**
	CF	0.0401	0.0286	0.0207	0.0115	0.0052
	CF-Hoeffding	0.0401	0.0286	0.0207	0.0115	0.0052
	CF-shrinkage	0.0401	0.0286	0.0207	0.0115	0.0052
	CF-significance	0.0401	0.0286	0.0207	0.0115	0.0052
	Random	0.0234	0.0177	0.0125	0.0075	0.0035
Recall	Avg. Similarity	0.0002	0.0004	0.0003	0.0001	0.0001
	BRISMF	0.0049	**0.0082**	**0.0108**	**0.0125**	**0.0119**
	CF	**0.0062**	0.0060	0.0061	0.0057	0.0055
	CF-Hoeffding	**0.0062**	0.0060	0.0062	0.0057	0.0055
	CF-shrinkage	**0.0062**	0.0060	0.0061	0.0057	0.0055
	CF-significance	**0.0062**	0.0060	0.0061	0.0057	0.0055
	Random	0.0035	0.0039	0.0037	0.0039	0.0038

Table 4.5: Performance of seven recommender methods *w.r.t.* F-Measure, precision and recall on the E-Agribusiness dataset. The BRISMF algorithm performs the best except for the initial part of a configuration.

with a partial configuration (*i.e.*, it requires only one manual selection of a single feature that are in accordance with stakeholders' needs) and at least one previous configuration to use as training data. However, we would like to highlight that our approach is designed to work most effectively with a large historical dataset of previous configurations.

RQ2. In which phase of product configuration can a recommender system make good recommendations? Our results show that the CF-shrinkage, CF-significance weighting, and BRISMF algorithms provide better recommendations than a random recommender already at the initial stage of the configuration process (*i.e.*, with 10% of selected features). The optimal percentage differs and depends both on the application domain and on the recommendation algorithm.

RQ3. What is the impact of the implemented algorithms on the quality of recommendations? The choice of the algorithm has a large impact on the quality of recommendations. Due to the best performance in our experiments we recommend the usage of one of the three algorithms: CF-shrinkage, CF-significance weighting, and BRISMF. However, since there is no single best algorithm for all application scenarios, we recommend the practitioners who apply our methods to test the aforementioned three algorithms and to select the best one.

4.4 Threats to Validity

Even though the experiments presented in this chapter provide evidence that the solution proposed is feasible, a key issue when performing these experiments is some assumptions that may affect the validity of the results. Questions we need to answer include: Was the study designed and performed in a sound and controlled manner? To which domain can the results be generalized? To address these questions, we discuss the experiment threads with respect to the four groups of common validity threats: *external validity*, *internal validity*, *construct validity*, and *conclusion validity* [Wohlin et al., 2000].

External validity. External validity concerns the ability to generalize the results to other environments, such as other domain contexts [Wohlin et al., 2000]. To minimize threats to external validity, our experiments rely on the two largest available datasets of SPL configurations already cited in the literature. Although the number of benchmark instances can also be considered a threat to external validity, once there is no other large real-world dataset of SPL configurations available, we could not proceed doing more experiments. Still, conducting experiments with other datasets of configurations remains as an important next step, which is part of our future work.

Internal validity. Threats to internal validity are influences that have not been considered and can affect the performance of our approach [Wohlin et al., 2000]. Thus, to increase internal validity, we use either standard or straightforward techniques to implement the recommender algorithms. We choose six well established state-of-the-art recommender algorithms to apply in the SPL configuration context and we show that three of them can efficiently support decision makers configuring a product. However, we cannot guarantee that those three algorithms are the best recommender algorithms to be applied in the SPL configuration context. Additional experiments using other personalized recommender algorithms are conducted in Chapter 5 and Chapter 6.

Moreover, to increase the internal validity of our experiments' results, we carried out not one execution of a test configuration, but we executed all available configurations from the dataset as test. Then, we reported the average of the F-Measure value of the execution of each configuration as the result in order to reduce any variation of performance and limit this validity threat.

Another validity threat may be related to the employed CNF representation in our implementation for decision propagation and validity checking (see Section 4.2.1). As a CNF is not unique, an equivalent CNF might lead to other results. This includes transitive dependencies that are not explicitly stated in the CNF. Thus, the (de)selection of features other than the recommended ones may also lead to a valid configuration. However, in order to minimize this effect, the feature model is always constructed with the same transformation algorithm.

Construct validity. The decision of which state-of-the-art configurator to extend for use in our approach can be a threat to construct validity. To minimize this validity threat, we chose to extend FeatureIDE based on results from two user empirical

studies [Constantino et al., 2016, Pereira et al., 2013] (see Chapter 7). However, any other configurator could be used in combination with our approach.

Conclusion validity. Conclusion validity concerns the relation between the treatments and the outcome of the experiment [Wohlin et al., 2000]. A potential conclusion validity threat is the type of the used SPLs. To enhance conclusion validity, we performed our experiments using two relatively large real-world SPLs of different sizes, having different types of features, structures, and from different application domains. However, we are aware that the results of our experiments cannot be generalized for all kinds of feature models. Thus, we try to minimize this validity threat by documenting the characteristics of the SPLs used[5] (see Table 4.2). We expect that especially large systems, such as the *Linux Kernel* with thousands of features [Sincero et al., 2010], would benefit from our approach.

Moreover, the list of identified challenges from our business case studies (see Section 4.1) are not the only missing aspects that are needed to support the configuration process of large-scale industrial SPLs. Still, analyzing other real-would scenarios and pointing out other challenges remains as an important next step, which is part of our user empirical study described in Chapter 7.

We conclude that while several validity threats exist, our results are promising as we have achieved high performance for three state-of-the-art recommender algorithms applied to the SPL configuration scenario. We assume that most real-world problems will be of similar scale. However, more analysis of which recommender algorithms provide the best results is needed. Moreover, conducting user empirical experiments is also an important part of our future work.

4.5 Related Work

A large body of literature has been dedicated to the SPL configuration process [Benavides et al., 2013]. These studies deal with the information overload resulting from interactive mechanisms to configure large and complex SPLs with multiple competing and conflicting features. While some of these studies [Antkiewicz and Czarnecki, 2004, Bagheri et al., 2010a,b, Bagheri and Ensan, 2014a, Bagheri et al., 2012b, Benavides et al., 2007, Martinez et al., 2014, Mazo et al., 2014b, 2012a, Mendonça et al., 2009, Spinczyk and Beuche, 2004, Tan et al., 2014a] aim to predict the utility of each feature for the user, others [Bagheri et al., 2010b, 2012b, Galindo et al., 2015a, Henard et al., 2015b, Hierons et al., 2016b, Lian and Zhang, 2015b, Martinez et al., 2015b, Pascual et al., 2015b, Tan et al., 2015a] aim to predict the utility of an entire set of features, which forms a valid product configuration. Next, we classify the related works into two main groups: *feature-based recommender systems* and *product-based recommender systems*.

Feature-Based Recommender System

On the feature recommender system scenario, Mazo et al. [2014b] present a set of recommendation heuristics to prioritize choices and recommend a collection of

[5]The complete representation of both SPLs can be found at http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/

candidate features to be configured. The purpose of their approach is to reduce the number of configuration steps and optimize the computation time required by the solver to propagate the configuration choices. However, the authors do not propose any mechanism to guide the decision makers in choosing among the set of candidate features. In this chapter, we propose a recommender approach that provides the decision maker with additional information about features' relevance.

Bagheri and Ensan [2014a] present dynamic decision models for guiding decision makers through the product configuration process. In their approach, decision makers are iteratively asked to choose between two competing features. Then, based on the decision maker's choices, algorithms are used to automatically predict the utility of the features to the decision maker and provide a ranking of recommended features that are close to their preferences. However, this approach may introduce inconsistencies in the ranking (*e.g.*, if a user ranks feature a more important than feature b, feature b more important than feature c, and feature c more important than feature a). Moreover, if two or more features that must be (de)selected (due feature model's constraints) are of equal (or no) interest to the decision makers, no support is provided to guide the configuration process. Therefore, when decision makers are not sure about which decision must be taken, the approach proposed in this chapter provides them with further information about the features' relevance. Thus, Bagheri and Ensan [2014a] approach benefits from our feature-based recommendation approach.

In a similar scenario, Tan et al. [2014a] and Bagheri et al. [2010a] propose a feature pair-wise comparison system to support decision makers in configuring a product. In their approach, the decision maker compares a randomly selected pair of features and identifies their relevance in terms of satisfying a set of non-functional requirements. Based on the decision maker's choices, feature rankings are produced for each decision maker's non-functional requirement. These feature rankings are then used as a recommender system to guide the product configuration process. However, as one feature may contribute to many non-functional requirements, the amount and complexity of options presented by the system may overwhelm the decision maker. Moreover, for performing this approach efficiently, multiple decision makers should be consulted. Consequently, a large amount of user input is needed. To address these challenges, we propose in this chapter a recommender system mainly based on data from previous users. Therefore, the proposed approach does not involve a large amount of manual interactive input from users. Moreover, we performed several experiments to prove that our approach can be efficiently automated.

Bagheri et al. [2010b, 2012b] propose a feature ranking approach based on *soft* and *hard* constraints. *Soft constraints* represent the stakeholders' preferences regarding complex technical product properties. *Hard constraints* represent features that the stakeholders are sure that fulfill their requirements. Thus, feature recommendations are employed on the set of open features based on their degree of contribution to the satisfaction of the stakeholders' soft constraints. Then, the decision maker can interactively select the features they desire until the feature model is fully configured. However, when hard constraints are inconsistent (*e.g.*, the stakeholders might request for different features that cannot be satisfied simultaneously), decision makers need to prioritize them. In this context, Bagheri et al. provide a perspective

view of non-functional properties related to each feature, and we propose a complementary approach that prioritizes features to decision makers based on the use of state-of-the-art collaborative-based recommender algorithms.

There are also approaches that use visualization mechanisms to aid the users in the configuration process. Among them, Martinez et al. [2014] present a visualization paradigm, called FRoGs (*Feature Relations Graphs*). FRoGs shows the impact, in terms of constraints, of the considered feature on all other features. The purpose of their approach is to support decision makers in obtaining a better understanding of feature constraints, and to serve as a recommendation system during the product configuration process. However, FRoGs as well as several others interactive configurators (*e.g.*, FeatureIDE [Thüm et al., 2014], SPLOT [Mendonça et al., 2009], FaMa [Benavides et al., 2007], VariaMos [Mazo et al., 2012a], pure::variants [Spinczyk and Beuche, 2004], Feature Plug-in [Antkiewicz and Czarnecki, 2004]) have not implemented further configuration support to guide the user prioritizing the features. Consequently, the amount and complexity of information presented may exceed the capability of a user to identify an appropriate configuration. In this context, our system additionally guides the product configuration process by delivering capabilities to effectively communicate with the decision maker and understand their needs and preferences.

Product-Based Recommender System

On the product recommender system scenario, Martinez et al. [2015b] present the use of product line techniques in the computer-generated artwork domain. The authors approach use tailored data mining interpolation techniques to predict configuration likability based on people's vote feedback for a dataset of configurations. Their approach is developed in two phases. First, a dataset of configurations is created using a genetic algorithm. Second, based on people's vote feedback for the dataset of configurations, tailored data mining interpolation techniques are used to predict configuration likability. Thus, a ranking is created among all valid configurations. This ranking serves as input to the artists to understand people's perception. However, people's votes are not directly related to a feature due to a large number of features and relationships presented in a configuration. Consequently, it cannot be generalized to the SPL configuration domain.

Galindo et al. [2015a] propose an approach, named Invar, which provides the users with a decision model with a set of questions and a defined set of possible answers. Based on the users' answers, a product is configured using decision propagation strategies through web service mechanisms. However, some vague descriptions may be introduced in the questionnaires and additional users' preferences are not allowed. Consequently, it may be hard for users to know which answer fulfills stakeholders' requirements better. Contrary to this approach, our approach focuses on the staged and gradual selection of features based on users' preferences through recommendations.

Considering a broader view on SPL configuration, in the optimization scenario, there are many recent works in the literature that support the automatic selection of features addressing each stakeholder's non-functional requirements (see Chapter 3).

However, automatic approaches have focused on techniques to derive product configurations in a single step, not allowing users to interactively change the obtained product(s). Moreover, since those techniques may provide a set of feasible solutions, users may not know which one would be the better choice. To guide users in choosing among the set of solutions, these works can benefit from the proposed recommender approach by computing the scores of the configurations' relevance.

Although there are many interesting studies that assist users through the *feature-based* and *product-based* configuration process, the process is still far from trivial and is often limited to tool support. The available configurators show to the users the set of all features and do not guide them in interactively configuring a product. To address this issue, we propose a *feature-based recommender system* to support users through the product configuration process by directing the order of selecting features and predicting which of them are more useful. In contrast to the current literature, our approach benefits from a simplified view of the configuration space by dynamically predicting the importance of the features. It only requires the manual selection of a single feature to offer recommendations that can be successfully used as an additional support by other approaches.

4.6 Summary

This chapter provides a further contribution towards the adoption of personalized recommender algorithms in the SPL configuration domain. Using this approach, decision makers can go through a small set of relevant features to configure a product. This interactive configuration process is supported by a state-of-the-art configurator that is intended to assist the decision makers in dynamically selecting features by considering the structural characteristics of feature models (Chapter 7). It ensures a valid and complete product configuration while simultaneously interacting with the decision maker, both to learn their preferences and provide new recommendations.

We evaluate the performance of six state-of-the-art recommender algorithms by using two real-world SPLs. Our experiment results show that our proposed approach for helping users understand their feature preferences is able to positively impact the quality of the product configuration process. Three of the six proposed recommendation algorithms clearly and consistently outperform the random recommender in finding relevant features. In summary, the proposed approach is very useful as: *(i)* it provides feature predictions that are in accordance with the preferences of users and constraints over the feature model, and *(ii)* it has good performance on partial configurations with just 10% of selected features. Furthermore, our approach can be automated. By selecting the features with higher predictions, our approach automatically creates a minimum or a maximum product configuration that meets all the feature model constraints (see Section 3.4.4). It requires only one manual selection of a single feature to create an initial partial configuration. Thus, the proposed approach can further facilitate the adoption of SPL practices and increase their benefits, such as mass personalization. In the next chapter, we extend our work to consider features' non-functional properties as input contextual data and consequently be able to capture currently relevant features for a user even though no configuration with these features have been observed in the past, and no features have been initially selected by a current user.

5. Personalized Extended Software Product Line Configurations

This chapter shares material with the VaMoS'18 paper "A Context-Aware Recommender System for Extended Software Product Line Configurations" [Pereira et al., 2018d]. We presented initial ideas at GPCE'16 [Pereira et al., 2016c] and COMLAN [Pereira et al., 2018b]. Furthermore, we have given an overview on semi-automatic approaches for extended software product line configurations at VaMoS'17 [Ochoa et al., 2017] and JSS [Ochoa et al., 2018].

A key part of a *Software Product Line* (SPL) is a model that represents features and their dependencies (*i.e.*, SPL configuration rules). As shown in Chapter 2, this model can be extended by adding *Non-Functional Properties* (NFPs) as feature attributes resulting in *Extended Feature Models* (EFMs). Deriving products from an EFM requires considering the configuration rules of the model and satisfying the product functional and non-functional requirements. Although the configuration of a valid product arising from EFMs may reduce the configuration space, selecting the most appropriate set of features is still an overwhelming task due to many factors including technical limitations and diversity of contexts. Consequently, configuring large and complex SPLs by using configurators is often beyond the users' capabilities of identifying valid combinations of features that match their (non-functional) requirements. Therefore, the configuration process may result in inappropriate or inefficient configurations. To overcome these limitations, we introduced the adoption of recommender systems in Chapter 4.

Our previous approach relies on the single use of binary data from previous configurations to generate personalized recommendations. Hence, this approach prevents the recommendation of new facts or new perspectives that would be valuable to the user. In this chapter, we extend this idea by considering product requirements as additional contextual data and thereby improving the overall recommendation quality. Similar to our previous work [Pereira et al., 2017], our current approach

is mainly built on the idea that feature priorities may change, based on the target stakeholders and the context of the configuration.

Our solution works interactively on a stream of selections and deselections of features and uses *contextual modeling* to incorporates NFPs. These data have an essential advantage, that is, being adaptive to changes of user preferences and release of new features. Consequently, our system is currently able to capture relevant features for a user even though no configuration with these features have been observed in the past. The aim of our system is to reduce the users' configuration effort and enhance their configuration experience.

To summarize, in this chapter, we provide the following three contributions:

1. We adopt a context-aware recommender system tailored for the EFM configuration scenario.
2. We target a challenge in the recommender system field, which is the recommendation of unexpected events (*e.g.*, new features).
3. We conduct extensive experiments on a large real-world industrial SPL to evaluate the proposed approach.

For evaluating our approach, we formulate three research questions to be answered by means of an experimental study:

- *RQ1.* Can a context-aware recommender system support the configuration process of EFMs in realistic scenarios?
- *RQ2.* Which are the recommendation quality benefits of a context-aware approach against a non-contextual approach?
- *RQ3.* What is the effect of using different combinations of contextual data?

To answer our research questions, we conducted numerous experiments with five context-aware recommendation algorithms on a real-world dataset of SPL configurations[1]. To address *RQ1*, we compare the results from all algorithms with the interactive configuration process and a randomized approach. To address *RQ2*, we compare the context-aware recommendation approach proposed in this chapter against the non-context-aware recommendation approach introduced in Chapter 4. Finally, since our context-aware approach is intended to work in three main stages, *RQ3* follows analyzing the impact of each stage in the quality of recommendations.

The remainder of this chapter is structured as follows: Section 5.1 motivates our work by presenting the open issues faced by our previous approach described in Chapter 4. We introduce our approach in Section 5.2, and describe evaluation design and results in Section 5.3 and Section 5.4, respectively. Section 5.5 discusses the threats to the validity of our evaluation results. We give an overview on related work in Section 5.6. Finally, we summarize this chapter in Section 5.7.

[1]The configuration dataset can be found in the Web supplementary material at http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/.

5.1 Open Issues from Previous Contribution

To support the SPL configuration task, recommender systems can provide suggestions that effectively prune the large configuration space so that users are directed towards features that best meet their needs and preferences. In the previous chapter, we proposed the use of a personalized feature-based recommender that relies on configurations from previous users to generate personalized recommendations for a current user. In particular, we adapted *Collaborative Filtering* (CF) algorithms to predict how much a user will like a particular feature. The CF-based system uses a *configuration matrix* $X_{\vec{c}\times f}$ as input, where $\vec{c} = \{\vec{c}_1, ..., \vec{c}_n\} \subseteq \mathcal{CC} \cap \mathcal{VC}$ represents the set of n complete and valid previous configurations whose preferences are known for the set of h features in $f \in \mathbb{F}$ (for further details we refer to Section 4.2.1). Our previous approach aims at predicting the relevance of a set of undefined features $f \subseteq \mathbb{F}$ for a current user with a partial configuration $\vec{pc} \subseteq \mathcal{PC}$, based on likely relevant features from X. Notice that $\vec{pc}$ is continuously augmented as the user interacts with the system over time guiding him on how to get a complete configuration.

Though our previous approach has shown to be useful for supporting the SPL configuration process, it exhibits at least one limitation: It only examines binary variables, indicating the selection of a feature in previous (user) configuration(s). Hence, it is sensitive to the number of features configured for a user in the past (known as *sparsity problem* in recommender systems [Adomavicius and Tuzhilin, 2011]). This problem is addressed in this chapter by combining multiple techniques within the recommender system [Burke, 2002].

5.2 Hybrid Context-Aware Recommender

To overcome the sparsity problem faced by the previous approach and the information overload generated by state-of-the-art SPL configurators, we propose a hybrid recommender system. Our system combines four recommendation techniques: *(i) context-aware*, *(ii) knowledge-based*, *(iii) CF-based*, and *(iv) rule-based*. The *context-aware recommender* attempts to suggest features based on inferences about the user's needs and preferences. It has contextual knowledge about product requirements, *e.g.* the financial context of a user. In addition, the *knowledge-based recommender* builds a complete utility function from historical data. The utility function infers many different factors that contribute to the value of a configuration by weighting the significance of each feature for each user, such as the popularity of new features by computing the similarity with other features and analyzing historical data, rather than just selected features. Then, the *CF-based recommender* recognizes similarities between users on the basis of previous configurations and generates recommendations. Finally, the *rule-based recommender* recommends only features that satisfy the configurations rules from an EFM.

To summarize, we use knowledge data from features, users, and configurations to assign weights to the matrix X based on contextual data from product requirements. Next, we use this as input to a CF-based recommender that will recommend features that satisfy the configurations rules from an EFM and better meet the product requirements. Hence, contextual-aware and knowledge-based systems don't suffer

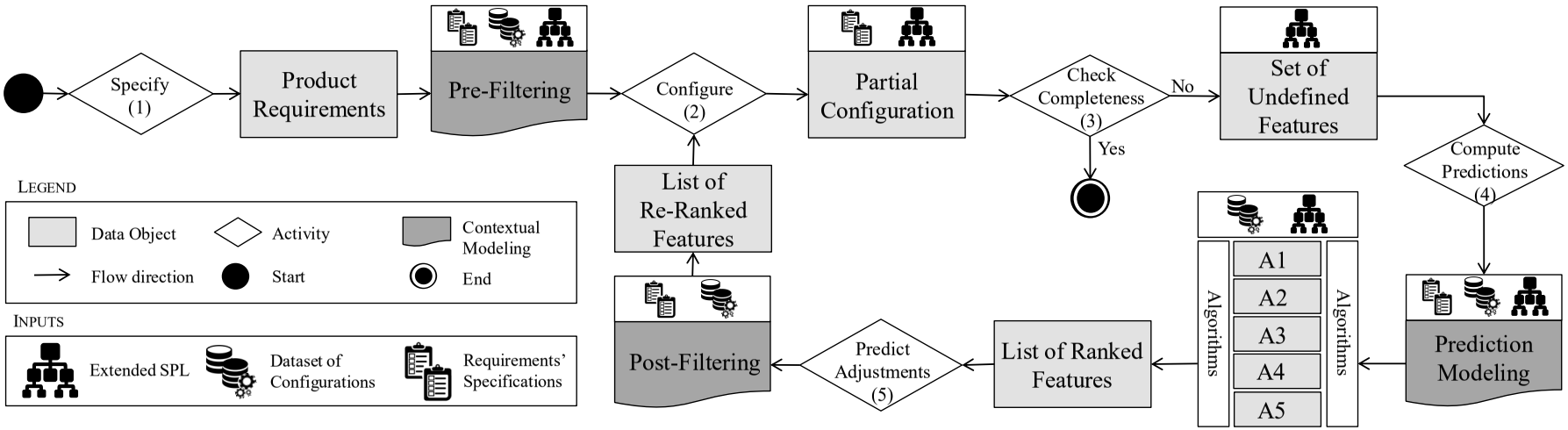


Figure 5.1: An overview of our feature-based recommender approach (A1: User-Based Collaborative Filtering, A2: Feature-Based Collaborative Filtering, A3: User-Based Average Similarity, A4: Feature-Based Average Similarity, and A5: Matrix Factorization).

from the sparsity problem because they do not (only) rely on binary data about previous configurations.

In Figure 5.1, we present a general overview of the configuration process, which consists of five main activities: (1) *specify contextual data*, (2) *configure product*, (3) *check product completness*, (4) *compute predictions*, and (5) *predict adjustments*. It uses the users' explicit requirements, previous configurations, and implicit data of features and users to predict the relevance of unselected features in a given configuration. Next, we briefly describe each of these activities:

(1) First, the user *explicitly* specifies contextual information in the system by collecting the product requirements from stakeholders. As we focus on the later stages, such as the (pre-)filtering and then the actual recommendation process, we assume that the users are capable of understanding the goals of the target system and translate those goals into product requirements. For example, laptops for gamers differ from laptops targeting other types of customer profiles (*i.e.*, low performance laptops would not meet the needs of a gamer due to the high processing demand of current games). We are aware that one of the key problems of product configuration is to find out what actually is the stakeholders' requirements [Rabiser et al., 2012a]. This, however, goes beyond the scope of this chapter. Nevertheless, after this interactive activity, the process is fully automated by our approach.

(2) Next, a *pre-filtering* stage builds a filter to include only historical data pertaining to the user-specified criteria in which the recommendation is relevant. Then, the user selects features of interest from a configuration view on the list of ranked features from our recommender system. Each time the user configures a particular feature, decision propagation strategies are automatically applied to validate the configuration and the ranked list of relevant features is updated (see Chapter 7 for more details).

(3) Subsequently, we check the completeness of the configuration by verifying whether there are some undefined features. In case we have a partial configuration, we compute feature predictions. Thus, the system attempts to model and learn users' preferences automatically by interactively obtaining preference feedback on their partial configuration.

(4) To compute predictions, the *prediction modeling* stage uses knowledge-based information directly in the recommendation algorithm (Section 5.2.3) as an explicit predictor of a user's preference.

(5) Then, the *post-filtering* stage reorders the recommended features by weighting the predictions with the probability of NFPs relevance in the user's specific context. The basic idea at this stage is to analyze the user's preferences in the given context to find specific feature configuration patterns (*e.g.*, user prefers a cheaper product) and then use these patterns to adjust the features' predictions, resulting in more useful recommendations.

Next, Section 5.2.1 formally defines the terms used to describe our recommender system. Then, Section 5.2.2 explains how contextual data are specified and modeled

by our approach and Section 5.2.3 describes the implemented CF-based algorithms that use contextual data to compute recommendations.

5.2.1 Formal Definitions

An SPL can be extended by adding NFPs as feature attributes, known as *extended SPL* [Benavides et al., 2010]. An Extended SPL describes the dependencies and constraints among functional features and NFPs through an EFM (see Section 2.1). In this section, we formally define EFM and configurations for the formal description of the proposed configuration process in Section 5.2.2 and Section 5.2.3.

Definition 5.1. *An* EFM $\mathcal{EFM}(F, P, R)$ *is a tuple that consists of a set of* n *features* $F = \{f_1, f_2, ..., f_n\}$*, a set of* m *NFPs* $P = \{p_1, p_2, ..., p_m\}$*, and a set of* k *configuration rules* $R = \{\vec{r}_1, \vec{r}_2, ..., \vec{r}_k\}$*.*

A configuration rule $\vec{r}_i$ *represents a clause from the EFM's propositional formula in CNF, such that* $\vec{r}_i \in \{-1, 0, 1\}^n$ *and the component* j *of* $\vec{r}_i$ *specifies whether the feature* j *should be selected* ($r_{i_j} = 1$)*, deselected* ($r_{i_j} = 0$)*, or is not relevant* ($r_{i_j} = -1$) *within this constraint.*

Definition 5.2. *Given an* $\mathcal{EFM}$*, a* configuration $\vec{c} = (c_{f_1}, c_{f_2}, ..., c_{f_n})$ *represents a selection of features such that* $c_{f_i} = 1$*, if feature* f_i *is selected,* $c_{f_i} = 0$ *if it is deselected, and* $c_{f_i} = -1$ *if its state is undefined.*

Definition 5.3. *A configuration* $\vec{c}$ *is* complete *iff* $\vec{c}$ *defines each feature (i.e.,* $\forall i \in \{1, .., n\} : c_{f_i} \neq -1$*), otherwise it is* partial*. For a given* $\mathcal{EFM}$*, we denote the set of all its complete configurations with* $\mathcal{CC}$ *and the set of all its partial configurations with* $\mathcal{PC}$*.*

Definition 5.4. *A configuration* $\vec{c}$ *is* valid *iff it satisfies all constraints in* R *when considering all undefined features in* $\vec{c}$ *as deselected (i.e.,* $\forall \vec{r} \in R, \exists i \in \{1, .., n\} : r_i \neq -1$*), otherwise it is* invalid*. For a given* $\mathcal{EFM}$*, we denote the set of all its valid configurations with* $\mathcal{VC}$ *and the set of all its invalid configurations with* $\mathcal{IC}$*.*

Based on the above formalization, our approach aims to use contextual data from features and users to restrict the configuration space and predict the relevance of undefined features in a way that the users can make decisions more easily.

5.2.2 Contextual Modeling

In our approach, contextual data can be obtained: *explicitly*, *implicitly*, and by *inferring*. Firstly, the user must specify which contextual information is relevant given the product requirements. Then, we deal with modeling the contextual data by incorporating them into the recommendation process. Next, we describe how to obtain contextual data.

- *Explicitly*: a set of previous configurations, a current partial configuration, product contraints, and stekeholders' preferences.
- *Implicitly*: domain expert judgement to specify NFPs [Bagheri and Ensan, 2014a, Machado et al., 2014a, Ochoa et al., 2015].

- *Inferring*: use of functional metrics to a static or runtime quantifiable measurement of NFPs (see Section 3.4.1), which allows the detection of interactions resulting from a valid set of features.

We assume that *implicitly* and *inferring* NFPs were previously specified by using state-of-the-art techniques. For instance, Cruz et al. [2013] and Zanardini et al. [2016] infer NFPs based on a rigorous static source code inspection, *e.g.*, analysis of coupling, number of code lines, and cyclomatic complexity. In this case, the success of implicitly specifying or inferring NFPs depends very significantly on the quality of such techniques, and it also varies considerably across different systems.

The context-aware recommendation process is performed into three main stages: *pre-filtering*, *prediction modeling*, and *post-filtering*, as visualized in Figure 5.1. Next, we describe how our approach incorporates each one of these stages.

Pre-Filtering. This stage takes as input a user specification of *product constraints* as pre-filtering data. To this end, the user specifies product constraints based on implicitly and inferring NFPs from features and configurations. The aim of this stage is to filter out noisy or irrelevant data before they are used for computing recommendations [Adomavicius and Tuzhilin, 2011]. An example of contextual pre-filter data for the *smart-home* product line in Figure 2.1 is given by ensuring that the `cost` and `response time` of the product do not exceed $2,550.00 and 300 ms, respectively. This pre-filter data is then used to reduce the initial matrix X (Section 4.2.1), containing data about previous configurations, to a matrix W using the following equation:

$$X \rightarrow W^{D[cost \leq 2{,}550, time \leq 300](User, Feature, Config)} \tag{5.1}$$

where $[cost \leq 2,550, time \leq 300]$ denotes two contextual pre-filters, and $D[cost \leq 2,550, time \leq 300](User, Feature, Config)$ denotes the historical configuration dataset obtained from D by keeping only the set of data where *cost* and *time* constraints are satisfied.

By using this reduction-based approach, we may not have enough data for accurate features' prediction. This is the case because this stage builds a local prediction model for a particular context, and this may limit the power of the predictions due to few remaining data [Adomavicius and Tuzhilin, 2011]. To overcome such limitation, we use the n-dimensional Euclidean metric [Amatriain and Pujol, 2015] to search for similar configurations to the ones in the matrix W based on data from configurations' NFPs. Then, to have a significant amount of historical data, *i.e.* filters with more than N configurations, we add a set of the highest similar configurations to the matrix W, where N is some predetermined threshold (*e.g.*, 50% of the dataset was used in this study).

Prediction Modeling. After filtering the set of relevant configurations and features, contextual data are used directly inside the recommendation-generating algorithms as part of the features' prediction estimation. Unlike the traditional recommender systems that deal with ratings, our knowledge-based recommender relies on the notion of weights as a way to measure the *utility* of a feature for a user. Thus,

we assign a utility weight u for all selected features in W. The general formula to measure the *utility* u of feature j for configuration $\vec{c}_w \in W$ (a row vector from W with $w \in \{1, ..., n\}$) is given as:

$$u(w, j) = \frac{Freq(c_{w_j})}{Period(c_{w_j})} \tag{5.2}$$

where $Freq$ and $Period$ are information inferred by the system. $Freq$ represents how often the customer of the configuration $\vec{c}_w$ used feature j and $Period$ the number of days since the last usage of feature j by $\vec{c}_w$, which measures the popularity of feature j. Following the idea that features have a short life cycle, this measurement is important to prevent the recommendation of outdated features. Moreover, features' popularity drift over time. Thus, the system is capable to capture such signals and timely adapt its recommendations accordingly. However, in this scenario, the recommendation of unexpected events, *e.g.* new features, are no longer possible. To overcome this limitation, we calculate an additional utility weight s for all *deselected* features in W, based on their *similarity* with other features' NFPs.

Since the range of values among NFPs may vary widely (*e.g.*, 1-60,000 ms for `response time` and $50-1,550 for `cost`), we firstly use a feature scaling method (also called *unity-based normalization*) to *normalize* the range of NFP values to a common scale in the range of $[0, 1]$. Then, we calculate the *weighted arithmetic mean* s for deselected features in W:

$$s(w, j) = \sum_{k=1}^{h} c_{i,k} \cdot \frac{Sim(f_j, f_k)}{\sum_{l=1}^{h} Sim(f_j, f_l)} \cdot \frac{1}{h} \tag{5.3}$$

where h represents the number of features, and $Sim(f_j, f_k)$ measures the *similarity* between a target feature j and another feature k based on their normalized NFP values. In our experiments, we use the n-dimensional Euclidean metric [Amatriain and Pujol, 2015] to calculate similarity. Once the weights w and s are estimated for the matrix W, traditional recommendation algorithms are used to compute the list of the N highest features' predictions for an active user (see Section 5.2.3).

To summarize, the prediction modeling stage follows four main steps: *(i)* assign a utility weight w for all selected features; *(ii) normalize* features' NFPs; *(iii)* calculate the *similarity* between features based on their normalized NFPs; and *(iv)* calculate a *weighted arithmetic mean* s for deselected features based on their similarity weight with the other features. This stage makes the recommender system to continuously adapt to the set of new features and to discard outdated features. Therefore, even without any historical knowledge about the features' utility, recommendations can be done by modeling and inferring a weight s. Thus, the use of contextual data in the prediction modeling stage avoids the sparsity problem (known as the *cold-start problem* in recommender systems) where users have a few set of selected features [Adomavicius and Tuzhilin, 2011]).

Post-Filtering. In this stage, contextual information is used after computing recommendations to adjust the resulting set of predictions, *i.e.*, when generating the final ranked list of relevant features. We use a model-based post-filtering approach

[Adomavicius and Tuzhilin, 2011], in which we build a predictive model by weighting the predicted features with an additional probability of relevance based on the contextual data from *users' preferences*. According to Asadi et al. [2014], users' preferences can be categorized into six levels: high, medium, and low *positive*; and low, medium, and high *negative*. In case the user classifies an NFP p_i as a *positive influence* over the computed predictions, we weight the computed predictions by considering the values 1 (high), 0.66 (medium), and 0.33 (low) as an additional probability of relevance a that is *directly proportional* to the predicted value. In case the user classifies an NFP p_j as a *negative influence*, we consider the values 1 (high), 0.66 (medium), and 0.33 (low) as an additional probability of relevance b that is *inversely proportional* to the predicted value. To summarize, the relevance score $Rel(\vec{pc}, f)$ computed in Section 5.2.3 is updated using the following equation:

$$Rel(\vec{pc}, f) = \frac{Rel(\vec{pc}, f) \cdot \sum_{i=1}^{n} a_i \cdot p_{i_f}}{\sum_{j=1}^{m} b_j \cdot p_{j_f}} \tag{5.4}$$

where n and m correspond to the number of NFPs with positive and negative influences, respectively. As an example, for the *smart-home* product line in Figure 2.1, the user may mention that a high `security` house has a *high positive* effect over the product requirements, while a high `response time` has a *high negative* effect. On the *optimization objective* scenario, the user wish to minimize the system `response time`, while maximizing the system `security`. Consequently, a feature with high security and fast response time is more preferable and relevant than another feature with slightly higher prediction but lower security and higher response time. The specification of users' preferences is especially important when there are competing features' predictions and a trade-off is needed (see Section 3.4.4).

5.2.3 Collaborative-Based Recommender

Since there is no single recommender algorithm that performs the best in all applications, in this section, we adapt five traditional CF-based recommender algorithms to the extended SPL configuration scenario: (A1) *User-Based CF*, (A2) *Feature-Based CF*, (A3) *User-Based Average Similarity*, (A4) *Feature-Based Average Similarity*, and (A5) *Matrix Factorization*. Given the matrix W and a new partial configuration $\vec{pc} \in \mathcal{PC}$ that is currently being configured by a target user, these algorithms are used to estimate unknown features' preferences for $\vec{pc}$. The first two algorithms are neighborhood-based CF approaches, which require the definition of a distance function between users or features, respectively [Desrosiers and Karypis, 2011]. We make a comparative analysis of the two main types of neighborhood-based algorithms: *user-based* and *feature-based CF*. In addition, we conduct experiments with the *user-based* and *feature-based average similarity* (AS) algorithms to investigate if restricting the neighbourhood size has an influence on the quality of recommendations in our application. Finally, we compare these algorithms with the *Matrix Factorization* (MF) algorithm introduced in Section 4.2.3. In the following, we will discuss how these algorithms are used by our approach to predict the relevance of specific user-feature combinations.

A1. User-Based CF. The main idea of a user-based CF algorithm is that similar users have similar patterns of configurations, therefore similar features receive

similar weights. First, we search for similar users (*i.e.*, nearest neighbours) to a target user in the matrix W. A similar user is a previous user, who has weights attributed to features (a row vector from W) that are similar to the ones from a target user according to a similarity measure. To find the most similar users, we use either the *Pearson Correlation Coefficient* (PCC) similarity measure or the *cosine similarity*. For a detailed explanation of both measures we refer to Amatriain and Pujol [2015]. A set of feature weights from a previous user qualifies as similar, if $Sim(\overrightarrow{pc}, \overrightarrow{c_w}) > \tau$, where τ is a similarity threshold that is given as an input parameter (Section 5.3.2). If a similarity measure exceeds a given threshold τ, then the corresponding configuration is considered a neighbour. All neighbours build a *neighbourhood* $\mathcal{N}(\overrightarrow{pc}, \tau) = \overrightarrow{c_w} \in \mathcal{CC} \wedge sim(\overrightarrow{pc}, \overrightarrow{c_w}) > \tau$.

Second, once the neighborhood $\mathcal{N}(\overrightarrow{pc}, \tau)$ has been determined, the algorithm calculates the relevance score $Rel(\overrightarrow{pc}, f)$ of an undefined feature f for a target configuration $\overrightarrow{pc}$ as the weighted average of her neighborhood weights for the feature f. The overall prediction function is as follows:

$$Rel(\overrightarrow{pc}, f) = \bar{pc} + \frac{\sum_{\overrightarrow{c_w} \in \mathcal{N}(\overrightarrow{pc}, \tau)} Sim(\overrightarrow{pc}, \overrightarrow{c_w}) \cdot (c_{w_f} - \bar{c_w})}{\sum_{\overrightarrow{c_w} \in \mathcal{N}(\overrightarrow{pc}, \tau)} |Sim(\overrightarrow{pc}, \overrightarrow{c_w})|} \tag{5.5}$$

where $\overrightarrow{pc}$ represents the set of features which an active user has selected and $\bar{c}_w = (\sum_{j \in \overrightarrow{pc}} c_{w_j})/(|\overrightarrow{pc}|)$ is the mean weight (analogously for $\bar{pc}$) for each configuration w for the specified set of j selected features in $\overrightarrow{pc}$. Note that the sum in the formula iterates over the neighbours of $\overrightarrow{pc}$, where each prediction is weighted with the similarity weight of its owner to the target configuration. Since our approach works with context-weights instead of ratings, we do not face the problem of one user rating all items highly, while another user might rate all items negatively. Therefore, the weights in the matrix W do not need to be mean-centered before determining the predictions. Moreover, note that this formula differs from the one in Section 4.2.3.1 because here we work with non-binary ratings.

A2. Feature-Based CF. In feature-based CF, the neighbourhood is constructed in terms of features (or columns in the weighting matrix W) rather than users. The weighted average value of these (raw) utilities is reported as the predicted value.

In particular, we calculate the NFP weight similarity between a target feature f for which the predictions are being computed and all the other features. Similarly, we use PCC and cosine similarity measurements and denote the *neighbourhood* of a target feature f as $\mathcal{N}(\overrightarrow{f}, \tau)$. Then, the weighted average of the neighbours weightings is used to compute the prediction of feature f for the target user $\overrightarrow{pc}$. The relevance score $Rel(\overrightarrow{pc}, f)$ is calculated as follows:

$$Rel(\overrightarrow{pc}, f) = \frac{\sum_{\overrightarrow{f_j} \in \mathcal{N}(\overrightarrow{f}, \tau)} Sim(\overrightarrow{f_j}, \overrightarrow{f}) \cdot pc_{f_j}}{\sum_{\overrightarrow{f_j} \in \mathcal{N}(\overrightarrow{f}, \tau)} |Sim(\overrightarrow{f_j}, \overrightarrow{f})|} \tag{5.6}$$

The basic idea of this algorithm is to leverage the user's own weights on similar features when making the prediction (*i.e.*, similar features are of similar relevance for the same user). For example, in the configuration scenario, the feature peer group will typically be features of a similar popularity. Therefore, the weight history of the same user on such features is a reliable predictor of the interests of that user.

A3. User-Based AS. This algorithm uses the same principle as CF, but it does not use the notion of a neighbourhood. Consequently, all weights attributed to features in the matrix W are considered for computing the relevance score of a feature for a target user. Accordingly, Equation 5.5 is changed to:

$$Rel(\overrightarrow{pc}, f) = \bar{pc} + \frac{\sum_{\overrightarrow{c_w} \in W} Sim(\overrightarrow{pc}, \overrightarrow{c_w}) \cdot (c_{i,f} - \bar{c_w})}{\sum_{\overrightarrow{c_w} \in W} |Sim(\overrightarrow{pc}, \overrightarrow{c_w})|} \tag{5.7}$$

Note that the sum iterates over all the users in W. This means that the relevance score of feature f is an average similarity of all other users' weights attributed to feature f.

A4. Feature-Based AS. Similar to the user-based AS recommender, Equation 5.6 is changed to:

$$Rel(\overrightarrow{pc}, f) = \frac{\sum_{\overrightarrow{f_j} \in W} Sim(\overrightarrow{f_j}, \overrightarrow{f}) \cdot pc_{f_j}}{\sum_{\overrightarrow{f_j} \in W} |Sim(\overrightarrow{f_j}, \overrightarrow{f})|} \tag{5.8}$$

Here the sum iterates over all features in W. This means that the relevance score of feature f is an average similarity over all other features' weights attributed to the same target user $\overrightarrow{pc}$.

A5. MF Recommender. This algorithm uses the same principle and formulas described in Section 4.2.3.6 to compute the relevance score $Rel(\overrightarrow{pc}_f)$. However, here instead of using the binary matrix X as input, we use W (see definition in Section 5.2.2 - *Pre-Filtering* stage).

In the last step, the relevance scores $Rel(\overrightarrow{pc}, f)$ are returned to the post-filtering stage. In this stage, to provide an optimized guidance for the user, features are ranked with respect to user's preferences (see Section 5.2.2). Note that we did not use three of the recommender algorithms used in Chapter 4 (*i.e.*, CF with Significance Weighting, CF with Shrinkage, and CF with Hoeffding Bound). These algorithms address the problem of not reliable similarity by considering the number of co-selected features between configurations (see Section 4.2.3). However, as we always work with a weighted dense matrix W, we do not face this problem here.

5.3 Experiment Design

This section describes the experiment design to evaluate our hybrid context-aware approach introduced in Section 5.2.

5.3.1 Target Software Product Line and Dataset

We evaluate the effectiveness of our configuration approach by applying it to a real-world dataset of 2,000 configurations and 203 features from our business partner in the ERP domain. It delivers an application scenario where customers, features, and configurations are described as relations having the following attributes:

- Customer: receives the feature recommendations; defined as Customer(CustomerID, Name, Market Domain, Location, Type).
- Feature: set of all features that can be recommended; defined as Feature(FeatureID, Cost, Profit, Provider, Category).
- Configuration: set of selected features for a previous customer; defined as Configuration(CustomerID, FeatureID, Frequency of Usage, Date Last Usage).

Moreover, the contextual information consists of the following specifications:

- Product constraints: market domain, location, and type.
- Stakeholders' preferences: minimize the system's cost for the customer, while maximizing the system profit for the company.

To evaluate our algorithms, on this dataset, we performed an offline evaluation that encompasses three main steps: *(i) parameter optimization*, *(ii) splitting into training and test datasets*, and *(iii) evaluation metrics*. In the next sections, we discuss each one of these steps. These steps are similar to the ones in Section 4.3.2.

5.3.2 Parameter Optimization

The implemented algorithms require the specification of a similarity measure and a threshold value τ to define which users are considered neighbours. The matrix factorization (BRISMF) algorithm also have three parameters to set: the number of latent dimensions k, the learning rate η, and the regularization λ. To find the optimal parameters to make the algorithms fit an unknown dataset, we perform an initial optimization step. This step is essential as it may influence the quality of the recommendations considerably. For example, since we are considering using either PCC or cosine similarity as the similarity measure of four algorithms to predict features' relevance, we must compare the performances of these two methods to determine which of them produce the best predictive model. Therefore, we hold out a random sample of 50% of all configurations from our original dataset and run a genetic algorithm. The genetic algorithm was used to optimize the F-Measure of the five used recommendation algorithms (Section 5.3.4). Every parameter in this phase was validated using a *10-fold cross validation*. The optimal similarity measure for CF and AS algorithms are *cosine similarity* and the approximately optimal threshold values τ are 0.7576 and 0.5661 for both user-based CF and feature-based CF algorithms, respectively. Moreover, for the BRISMF algorithm, the values for the parameters k, η, and λ are 65, 0.086 and 0.81, respectively.

The optimal parameter settings from this algorithm are shown in Table 5.1. In the column *algorithm*, we show our recommendation algorithms. The *parameter* column shows which parameters the corresponding algorithm requires. The last column shows the optimal values of the corresponding parameters. Next, these parameter settings are used in our main evaluation and applied to the remaining 50% of configurations.

Algorithm	Parameter	ERP System
User-based CF	τ	0.7576
	Sim. Measure	Cosine Sim.
Feature-based CF	τ	0.5661
	Sim. Measure	Cosine Sim.
User-based Avg. Sim.	*Sim. Measure*	Cosine Sim.
Feature-based Avg. Sim.	*Sim. Measure*	Cosine Sim.
Matrix Factorization	k	65
	η	0.086
	λ	0.81

Table 5.1: Optimal parameter values.

5.3.3 Splitting into Training and Test Datasets

In this main evaluation phase, we use the *leave-one-out evaluation* protocol. According to this protocol, one configuration is used for testing and the remaining ones are given to the algorithm as training data (*i.e.*, all other configurations). Formally, the specified entries of the configuration matrix $\overrightarrow{pc} \notin X$ are referred to as the training data, whereas the partial configuration $\overrightarrow{pc} \in \mathcal{PC}$ is referred to as the test data. It simulates the behavior of an active user configuring a single target product in a configuration system, where the remaining configurations are available to the system to assist him in finding the features matching their individual preferences and expectations. However, to further simulate the interactive configuration process by an active user, the set of features is randomly partitioned into 10 equal sized sub-sets. Then, we increasingly give a sub-set of (de)selected features from a test configuration to the recommender system as training data and the remaining ones are hidden from the algorithm and used as testing, *e.g.*, 10% of a complete configuration is used as training and 90% as test data. To perform well, the system has to recommend the features that were hidden from the algorithm.

To ensure reliability, the cross-validation process is repeated 1,000 times (*i.e.*, where 1,000 represents the remaining 50% of the configurations from our original dataset) for each stage of the configuration process (*e.g.*, 10%, 20%,...,90%) and for each of the six methods with each configuration used exactly once as the testing data. This give us a total of 54,000 experiments. Consequently, to produce a single estimation, the final quality measure of a recommendation method is then the average quality over all configurations.

5.3.4 Evaluation Metrics

Once the recommender system returned a ranked list of relevant features, we perform a quality measurement of the recommendations. Firstly, we compare the real set of truly relevant features *Rel* known from the test configuration with the set of recommended ones *Rec* using *precision* and *recall* as quality measure. Since we approach recommendation as a ranking task, we are mainly interested in relatively few most relevant features. Thus, *precision* and *recall* are computed based on the top-10 ranked features (*i.e.*, $w = 10$) which is common in recommender systems [Shani

and Gunawardana, 2011]. While *precision* measures the proportion of recommended features that were truly selected, *recall* measures the proportion of all truly selected features that appear in the top-w ranked features. They are calculated as follows: $Precision = \frac{|Rec \cap Rel|}{w}$ and $Recall = \frac{|Rec \cap Rel|}{|Rel|}$. Consequently, the optimal value for both measures is 1.0, indicating that all truly selected features have been correctly recommended, without any deselected features among the recommendation. Since our goal is to maximize both, precision and recall, in our evaluation we use a measure that combines both, *i.e.*, the *F-Measure*:

$$F\text{-}Measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{5.9}$$

The F-Measure is high, when both precision and recall are high. Thus, the quality of a recommendation is good, if the top-w predicted features overlaps with the set of truly selected features from the test configuration.

5.4 Analysis of Results and Discussion

In this section, we investigate how efficient and effective the proposed recommender system is in supporting the SPL configuration process by answering the research questions *RQ1–3* introduced in the beginning of this chapter.

5.4.1 Approach Effectiveness

We evaluate the effectiveness of the algorithms by comparing them with a random baseline method that simulates the performance of an uninformed user without any support from a recommender system. However, as in most real-world applications no human is fully uninformed, to additionally indicate how useful our recommendation algorithms are, we reported also the results from an interactive configuration process from two domain experts (*Exp1* and *Exp2*). In this context, for each testing configuration, we informed the expert the target user context (*i.e.*, the product requirements) as well as the training set of pre-selected features. Then, for each percentage of features that were given to them as training data, they choose the features that most suited the specified requirements.

Figure 5.2 presents the *F-Measure* achieved by the eight methods: a *random* baseline method, five *recommendation algorithms*, and two *domain experts*. On the horizontal axis of the figure, we present the completeness of a configuration, *i.e.* the percentage of features that were given to the method as training data, where only the remaining part of the configuration needed to be predicted.

We observe that all methods presented good results outperforming the baseline random recommender. For all algorithms, we observe an increase of performance as the configuration becomes more complete. This is because the algorithms receive more data for training or rasoning. The user-based CF (A1) and BRISMF (A5) algorithms achieved the best performance over all the other algorithms at nearly all stages of the configuration process, except for the initial part of a configuration, when few information about the current configuration is available. Overall, CF algorithms yielded a better performance than AS algorithms (*i.e.*, A1 outperformed A3, and

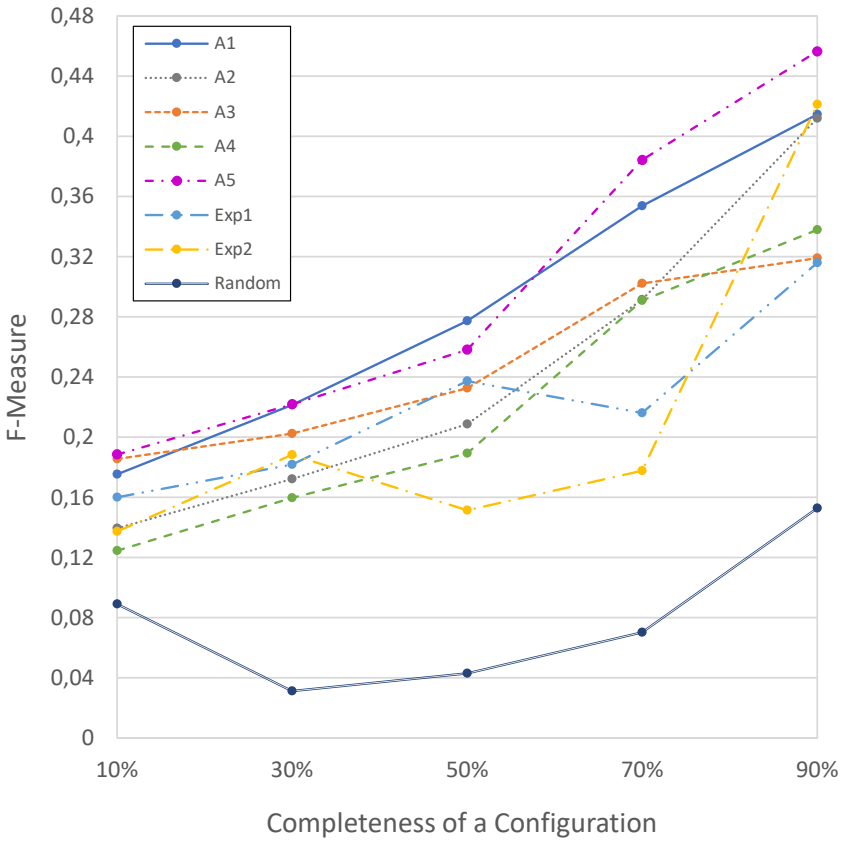


Figure 5.2: F-Measure achieved by eight different recommender methods (A1: User-Based CF, A2: Feature-Based CF, A3: User-Based AS, A4: Feature-Based AS, A5: MF, Exp1 and Exp2: domain experts). The horizontal axis shows how much of the current configuration has been completed. The performance is calculated on the remaining part of a configuration.

A2 outperformed A4). One potencial reason is that AS algorithms do not rely on any neighborhood information. In average the CF algorithms provided even better results than the interactive configuration performed by domain experts. Moreover, domain experts are engaged in a time-consuming and tedious task.

5.4.2 Context-Aware Approach Benefits

In this section, we estimate how effective our context-aware approach is in comparison with the previous non-contextual version of our approach (see Chapter 4). It is worth mentioning that we do not use the datasets from Section 4.3.1, since these datasets do not work with NFPs. Therefore, we run the non-contextual version of the five implemented algorithms in our target dataset. In Figure 5.3, we report the *F-Measure* results achieved by both approaches for each of the five algorithms.

The contextual recommendation algorithms (txt) significantly outperform the non-contextual algorithms (n-txt) in terms of predictive accuracy at all the stages of the configuration process. The main reason is the benefit of having additional relevant data for calculating unknown features' relevance, instead of only having binary information. The non-contextual version of the algorithms are limited by the users that

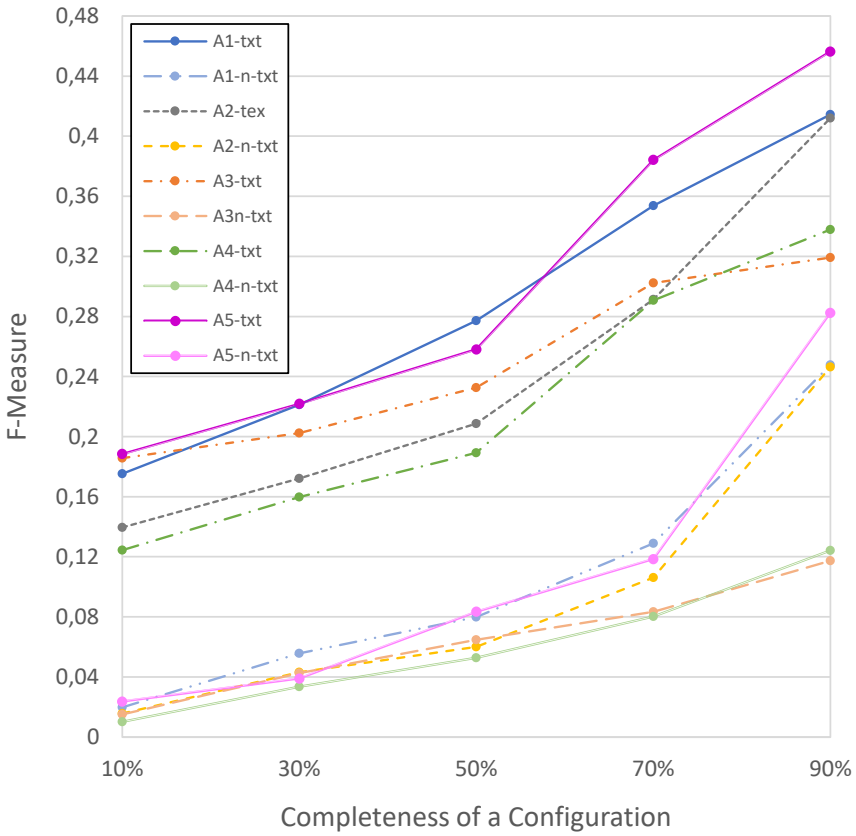
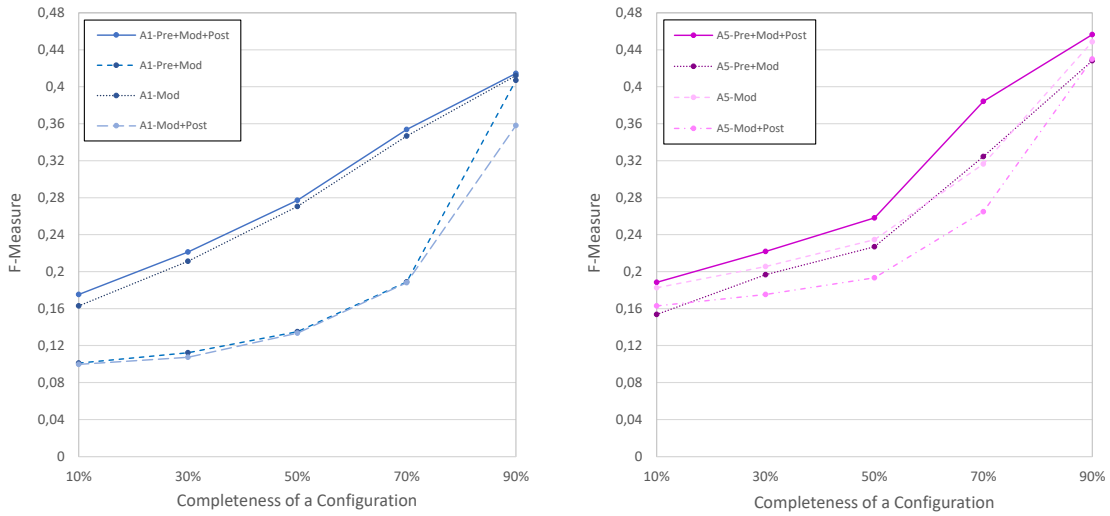


Figure 5.3: F-Measure achieved by five contextual (txt) and non-contextual (n-txt) CF-based recommenders (A1: User-Based CF, A2: Feature-Based CF, A3: User-Based AS, A4: Feature-Based AS, and A5: MF).

are explicitly associated with the features that they recommend and therefore has difficulty when the space of selected features is sparse (*i.e.*, in average few users have selected the same features). Sparsity is a significant problem in the SPL configuration domain, since there are many features available and, unless the configuration dataset is very large, the chances that another user will share a large number of selected features is small. Consequently, pure CF-based recommenders work best on datasets where the density of user configuration is relatively high across a small and static number of features. However, which of these two trends dominates depends on the application domain and available data. Therefore, we extended the state-of-the-art FeatureIDE configurator with our recommender approach by using the contextual data only for those contextual situations where this method outperforms the standard non-contextual version of the same algorithm (see Chapter 7). In our implementation, if the context-based system cannot make a recommendation with sufficient confidence, then just a collaborative recommendation is attempted. Thus, the approach proposed in this chapter is expected to perform equally well or better than the approach presented in Chapter 4 in practice. Next, to better understand the contextual behavior of our approach, we perform a preliminary experimental study to evaluate the contextual stages specified in Section 5.2.2.



a) A1: User-Based CF Recommender. b) A5: BRISMF Recommender.

Figure 5.4: F-Measure achieved by the user-based CF (A1) and BRISMF (A5) contextual recommender algorithms for different combinations of contextual data.

5.4.3 Different Combinations of Contextual Data

In Section 5.2.2, we described the main contextual stages under which our approach is developed (see Figure 5.1). Since not all stages might be useful for recommendation purposes, in this section we empirically evaluate the effect that each stage has in the quality of the recommendations.

Figure 5.4 shows the results for the most effective algorithms (*i.e.*, BRISMF and user-based CF in Section 5.4.1). We observed that all proposed algorithms achieved the best performance when making use of all available contextual data. For example, for the user-based CF algorithm the performance differences between the A1-Pre+Mod+Post and A1-Pre+Mod, A1-Mod and A1-Mod+Post methods range 38%, 3%, and 41% in average respectively for the *F-measure* across the same dataset. It is evident that explicitly modeling a large amount of data significantly boosts the recommendation performance under the same algorithm. Furthermore, note that although the A1-Mod recommender is outperformed by the A1-Pre+Mod+Post recommender, the results from both methods are quite similar. Therefore, this implies that inappropriate contextual modeling in the pre-filtering and post-filtering stages can even hurt the performance.

We would also like to point out that accurate configuration predictions unquestionably depends on the degree of which the recommender system incorporates the relevant contextual information. There are several approaches, *e.g.* from machine learning, data mining, and statistics, to determine the relevance of a given type of contextual information. These approaches aim at screening all the NFPs and filtering out those that do not affect a particular recommendation application. This, however, goes beyond the scope of this chapter and remains as an important next step, which is part of our future work.

5.5 Threats to Validity

Next, we discuss the four groups of common validity threats: *internal validity*, *external validity*, *construct validity*, and *conclusion validity* [Wohlin et al., 2000].

Contextual settings may affect the performance of an algorithm, which might form a threat to *internal validity*. In our context, such a threat might be due to the fact that we used contextual requirements settings from the company for all the algorithms. They are conform to the company needs and have been proven to give good results. However, the accuracy of the recommender algorithm may be different depending on whether contextual information (*i.e.*, product requirements) is used in the pre-filtering, post-filtering, or modeling stage. For example, in other scenarios, cost information (cheap vs. expensive) may be most useful to pre-filter relevant data, but location information (*i.e.*, state) may be the most appropriate to use as a post-filter. However, determining the benefit of different contextual information in the SPL configuration scenario with respect to different stages of context-aware recommender systems constitutes an interesting and promising direction.

External validity threats are related to the generalization of the results of the experiments [Wohlin et al., 2000]. In our evaluation, we ran more than 54,000 experiments using a large-scale case study which has 203 features and 2,000 configurations. To the best of our knowledge, this is the largest real-world dataset of configurations with NFPs already cited in the SPL literature. However, the results from our experiments may not be generalized to other SPLs. Therefore, future experiments with additional case studies are required to further generalize the results.

To deal with *construct validity* threats, we used the same measure, *i.e.* F-Measure to compare the accuracy of the algorithms. There are other evaluation metrics that are typical of recommender systems, such as RMSE of MAE. However, since our non-contextual approach uses binary data (Chapter 4), these metrics are not applicable. Therefore, for a fair comparison, in this chapter we also use F-Measure as an evaluation metric. Moreover, we ran all the experiments on the same machines with the same configuration (*i.e.*, 2 sockets Intel Xeon E5620 @2.40GHz with 4 cores per socket and 20GB of RAM) for all algorithms.

Conclusion validity threats are concerned with factors that can influence the results of the experiments [Wohlin et al., 2000]. Since SPLs present a large variability space, the current partial configuration may vary from user to user. To reduce the bias that the results were obtained by a particular current set of optimal (de)selected features, we repeated the experiments 1,000 times for each testing configuration in our dataset.

5.6 Related Work

Recommender systems reduce the complexity of comprehension tasks and help to get insights for making decisions [Ricci et al., 2011]. Recommendation techniques have been studied by the SPL research community to support several tasks beyond configuration (*e.g.*, feature location Marcén et al. [2017], creating variability models [Dumitru et al., 2011, Hamza and Walker, 2015]). In this chapter, we focus

on recommendations to guide the configuration process of extended SPLs and we acknowledge several works in this field.

Several approaches address the configuration in extended SPLs by using *dynamic decision models* [Bagheri and Ensan, 2014a, Galindo et al., 2015a, La Rosa et al., 2009, Tan et al., 2014a, Zhang et al., 2014]. Through decision models, the user can interactively construct a complete preference function by weighing the significance of each relevant feature in terms of satisfying their non-functional requirements. However, as real-world EFMs tend to be inherently large and complex, this often creates a significant burden of interaction, *i.e.* the user has to assess several types of variability relations among features and NFPs. Furthermore, as product requirements may be conflicting and one feature may contribute to many requirements, users may still be unsure about their preferences.

To overcome the limitations of the above approaches, several authors have proposed optimization techniques to automatically support the configuration process (see Section 3.4.4). However, these techniques usually return a set of optimal configurations and none of them guides the user in selecting the most appropriate one. Hence, these techniques can be complementary to our approach that aims at guiding the user in the selection of a single configuration.

In addition to recommendation techniques to support the configuration task, there are numerous academic and industrial tools for reasoning about SPLs (*e.g.*, FeatureIDE [Thüm et al., 2014], SPLOT [Mendonça et al., 2009], VariaMos [Mazo et al., 2012a], pure::variants [Spinczyk and Beuche, 2004]). Whereas these tools focus on satisfying the SPL constraints, our approach builds a prediction model to identify features' relevance based on product requirements through analysis of features' NFPs from extended SPLs.

5.7 Summary

In this chapter, we propose a hybrid recommender approach for predicting feature selections in an extended SPL configuration scenario, *i.e.* taking *non-functional properties* (NFPs) of features into consideration. Our approach adopts traditional CF recommendation algorithms to estimate features' preferences based on users' contextual information. The proposed approach is not only able to support new configurations, but also provides configuration upgrades for previous derived configurations. To assess its effectiveness, we present an empirical evaluation based on a large real-world dataset of configurations derived from industrial experience in the *Enterprise Resource Planning* (ERP) domain. Our results indicate significant improvements in the predictive accuracy of the proposed context-aware recommendation approach over a non-contextual recommendation approach presented in Chapter 4. In the next chapter, we evaluate our approach under the use of a multi-dimensional tensor factorization recommender algorithm.

6. Personalized Self-Configuration of Software Product Lines

This chapter shares material with the SPLC'18 paper "N-dimensional Tensor Factorization for Self-Configuration of Software Product Lines at Runtime" [Pereira et al., 2018c]. We presented initial ideas at ICSE'17 Student Research Competition [Pereira, 2017].

Dynamic *Software Product Lines* (SPLs) provide configuration options to adjust a software system at runtime to deal with changes in the users' context [Hinchey et al., 2012]. To enable such a dynamic configuration, several *semi-automatic* and *automatic* approaches have been proposed in previous work [Guedes et al., 2015]. Nevertheless, the applicability of these approaches is still limited. In particular, for SPLs with a huge exponential configuration space, they have shown to be infeasible.

On the one hand, *semi-automatic approaches* require many tasks to be carried out manually by decision makers. Consequently, decision makers must know a lot of detailed, technical information about the features and the context. However, decision makers usually lack such knowledge, which often leads them to invalid configurations and an increase in configuration time. On the other hand, although *automatic approaches* do not require any user intervention, these approaches may generate a set of resulting suboptimal configurations. Thus, as a complementary solution, we propose a context-aware recommendation technique to automatically prioritize features and self-configure SPLs at runtime.

Recommendation techniques have become essential to efficiently filter the huge amount of SPL variants and support the configuration of personalized products [Pereira et al., 2016c, 2018b,d]. In recommender systems, user's preferences may be inferred from consumption patterns from other users (a technique known as *Collaborative Filtering* (CF)). In previous chapter, we studied five different CF recommendation algorithms to support the context-aware SPL configuration process. However, these algorithms do not provide a straightforward way of integrating context data (*i.e.*, features' *non-functional properties* (NFPs) [Benavides et al., 2010])

into the model [Koren et al., 2009]. Instead, they use a reduction-based approach to reduce the problem of N-dimensional $User \times Feature \times Contexts$ recommendations to the traditional two-dimensional $User \times Feature$ recommendation. For example, to recommend a laptop to a gamer, this approach uses only the data from previous users which has high processing laptops. Although reduction approaches lead to more relevant data for calculating unknown features interest, they also lead to fewer data used in this calculation based only on the configurations with the same or similar context. Thus, to improve the quality of the recommendations, we incorporate a multidimensional recommendation technique (*i.e.*, *Tensor Factorization* (TF)) so that it allows the specification of contextual data in the form of a tensor [Karatzoglou et al., 2010]. Our aim is to take advantage of the same principles behind MF in Chapter 4 to deal with N-dimensional contextual information. Therefore, the TF approach proposed here has three main advantages compared to our previous reduction-based approach. First, there is no need for pre-filtering or post-filtering of the data based on context since TF uses all the available data to model the users and features. Second, it provides a computational simplicity. Instead of relying on a sequence of techniques, TF relies on a single and less computationally expensive model. Third, it provides capabilities to handle N-dimensional data. The TF approach generalizes well to an arbitrary amount of contextual information. In Section 6.4.1, we empirically study the tradeoff between a reduction-based approach and a TF approach. In particular, we are interested in whether and how good a TF approach can be to support the self-configuration process of SPLs at runtime.

To this end, we formulate the following three research questions that guide us in evaluating our approach.

- *RQ1.* How effective does an N-dimensional TF recommender system support the SPL self-configuration compared to others state-of-the-art contextual recommender techniques?
- *RQ2.* How accurate is a context-aware TF recommender system compared to a state-of-the-art non-contextual recommender?
- *RQ3.* How long does it take, on average, for a TF recommender system to self-configure a complete valid product?

To answer these questions and prove the applicability of our approach, we present an empirical study on two subject systems: a laptop and a library product lines[1]. Both product lines take various contextual information into account, such as by *whom*, *when*, and *where* the laptop and library are used. To address *RQ1*, we demonstrate the effectiveness of our approach based on interactive recommendation updates against four context-aware reduction-based approaches introduced in Chapter 5 and a random configuration of features as baseline. To address *RQ2*, we compare the results from our proposed context-aware TF approach with the non-contextual MF approach introduced in Chapter 4. Finally, since recommender

[1]The product lines and the configuration datasets can be found at http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/.

systems are frequently intended to work on very large datasets, the performance of the recommender is essential. Thus, for *RQ3*, we investigate how fast is our proposed technique. To ensure a good user experience, the approach must be able to handle changes in the environment ensuring a very short response time for delivering self-configurations.

Overall, we make the following three contributions:

1. We adopt a tensor-based recommender system tailored for the extended SPL configuration scenario in which we explicitly take the user's context into account.
2. We target a challenge in the field of dynamic SPLs, which is the efficient self-configuration by interactive recommendation updating.
3. We conduct extensive experiments on two medium-sized product lines to evaluate our proposed approach.

The remaining chapter is structured as follows. Section 6.1 motivates our work by presenting the open issues in the SPL self-configuration literature. Section 6.2 presents our approach. Subsequently, Section 6.3 describes the design of the performed experiments and Section 6.4 discusses the experimental results. In addition, Section 6.5 further discusses threats to the validity of our results and Section 6.6 gives an overview on related work. Finally, Section 6.7 summarizes the chapter.

6.1 Open Issues in Self-Configuration of Dynamic Software Product Lines

The automatic process of building personalized products from an SPL is known as self-configuration. *Dynamic SPLs* are software systems in which self-configurations occur at runtime (see Section 3.4.5). Previous works in the SPL literature have proposed several *static* and *dynamic* self-configuration approaches. However, the current self-configuration process of dynamic SPLs is still limited when dealing with the variability of highly configurable systems. We identify the following open issues in this field:

- In static self-configuration approaches, features are selected based only on product requirements and human desires. Nowadays, this may not be enough due to context changes that without reconfiguration would lead to context violations.
- Because a valid context type is not necessarily Boolean, the problem evolves from SAT to a general constraint satisfaction problem (CSP). From the problem solving perspective, checking the consistency of a configuration can also be non-trivial. Therefore, due to the computational complexity of the task, improvements related to scalability and performance are still needed to manage the complexity of larger variability models.

- Given a set of product requirements and adaptation rules that explicitly define under which circumstances a reconfiguration should take place, there may be a set of resulting valid configurations instead of a single one. Therefore, those configurations should be prioritized before the configuration process continues. An example of this scenario is when a smart home application reaches the energy limit, but the user is executing a critical task that requires energy consumption. The low energy level context requires the heating to be shutdown, but for the temperature, use of heating is a critical non-functional requirement and it requires energy. Therefore, these two possible configurations are conflicting (the heating must be either turned on or turned off, it cannot be in both states at the same time) and one of them must be chosen before the adaptation process continue. The manual prioritization of features may result in an overwhelming task. Thus, developing an automatic and effective set of prioritization is a challenging task for developers due to the complexity and dynamicity of the context. According to our SLR in Chapter 3, studies do not use any strategy for dealing with configuration selection conflicts in runtime environments, except by HyVarRec [Mauro et al., 2016] that selects the configuration most similar to the last derived configuration.
- Some studies rely on many tasks to be carried out manually. For example, some approches rely on the experience of an expert to describe adaptation rules. Consequently, unexpected environmental changes (*e.g.*, new features might be integrated at runtime) may be unknown for the expert and not take into consideration during the reconfiguration process.

Based on the identified limitations, the envisioned contribution of this chapter is therefore to fill the gap in the literature by proposing a polynomial-time recommendation approach for self-configuration of dynamic SPLs. Although in previous chapters (Chapter 4 and Chapter 5), we show that *Matrix Factorization* (MF) can be used successfully in many static environments, in dynamic environments additional contextual variables come into play. As an example, consider the online social media Facebook, where the contextual variables: *privacy*, *security*, *performance*, *network*, and *access load* play an important role in defining a configuration. For instance, Web applications can suffer from load fluctuations depending on the number of access. An important challenge in this scenario is to avoid the impact on performance by dynamically self-reconfiguring the system at runtime to meet the current user's context. Similar to state-of-the-art approaches, we aim at automatically adjusting the software features based on *external* and *internal* system settings (*i.e.*, implicit information). *External* system settings are product resource constraints that limit the environment context, such as *network* access. *Internal* system settings are preferences options that allow users to customize their Facebook environment, such as *privacy* options. The set of *internal* and *external* settings is known as product requirements which are determined based on the users' current context. Although the product requirements and SPL constraints limit the configuration space, Facebook has hundreds of features and relationships, leading to several different possible context options. To efficiently overcome this challenge, our approach relies on explicit information from other users to self-configure a product over contextual changes in the environment. For instance, the system can rely on *friends* and *apps* to suggest new Facebook features to current users. To this end, the two-dimensional matrix

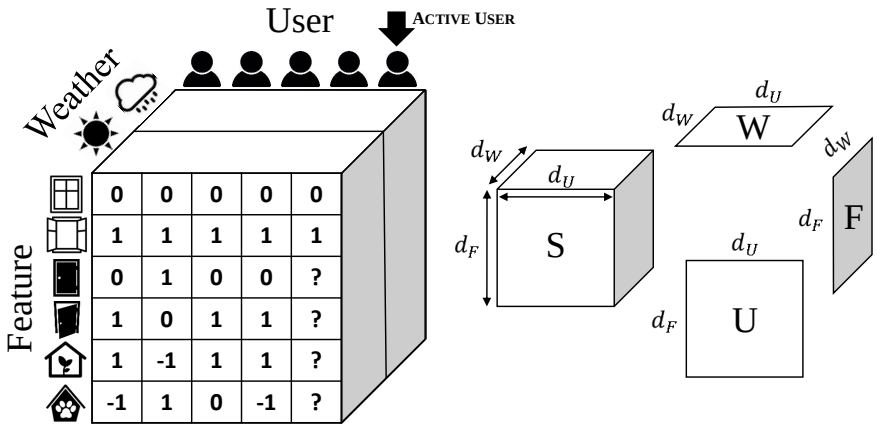


Figure 6.1: A 3-dimensional example of a tensor factorization model derived from the SPL in Figure 6.2. Selected features are encoded as 1 and deselected as 0. All other entries (-1 and ?) are unknown features' interests.

is turned into a multidimensional tensor and feature predictions are done by using *Tensor Factorization* (TF), an N-dimensional extension of MF.

The use of TF can provide recommendations based on multiple dimensions that go beyond the typical two dimensions (*i.e.*, users and features) used in MF [Karatzoglou et al., 2010]. The order of a tensor is the number of dimensions, *i.e.* the number of relevant contextual information. In the example given in Figure 6.1, the usual two-dimensional $User \times Feature$ matrix is converted into a three-order tensor $User \times Feature \times Weather$. The intuition behind using TF to support the self-configuration of dynamic SPLs is that there should be some latent features that determine how to configure a product at runtime. Hence, if we can discover the set of latent features from a current configuration through the user-specified context, we should be able to predict the configuration for a specific user (*i.e.*, the characteristics associated with the user, the feature, and the context should match among them). TF takes advantage of most of the benefits of MF, such as fast prediction computations as well as simple and efficient optimization techniques. For more information of this algorithm, we refer to Section 6.2.2.

6.2 Tensor-Based Recommender

In this section, we explain the details of how we have adapted the 2-dimensional MF for an N-dimensional TF. First, we introduce how context is specified and modeled and how we relate context to features. Afterwards, we describe details of the proposed TF approach and illustrate it by means of an example.

6.2.1 Modeling Features and Context

Our approach is based on the idea of making use of EFMs to integrate context information as a specific kind of NFPs (similar to Saller et al. [2013] and Mauro et al. [2016]). This way, we make the context information explicit and relate it to features of the SPL. It makes possible to easily assess the context information and reason about corresponding feature selections in the configuration.

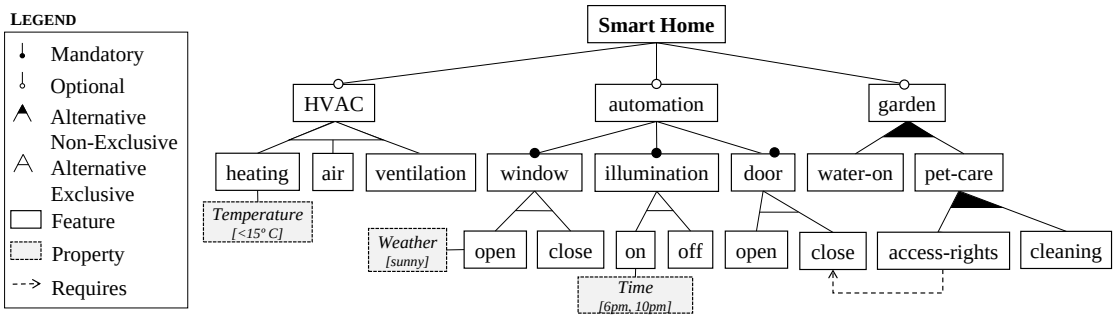


Figure 6.2: A sample of a *smart-home* SPL adapted from Figure 2.1.

As an example, we introduce three different contexts in the feature model of Figure 6.2: *Temperature*, *Weather*, and *Time*. Each context is associated with a particular feature, for instance, weather is associated with the `window` feature, thus, may influence whether the window is open or closed. Furthermore, each context may encompass an abitrary number of numeric or categorical values, *e.g.*, the weather can be *sunny* or *rainy*.

For our approach, we consider each context as a separate dimension in our recommendation model (note that also features and the configuration itself constitute separate dimensions), with each dimension encompassing a number of predefined numeric or categorical values. Since this may lead to a high number of dimension, as a preliminary step, we aim at minimizing our recommendation model by omitting contexts that are not relevant (*i.e.*, which have no influence on the feature (de)selection).

To determine which properties are relevant, we apply the *binary operations* of union and intersection over the set of NFP values. For example, consider a simple case of a single-attribute dimension *Weather* which has only two possible qualitative values *Sunny* and *Rainy*. To determine the relevance of context *Weather*, we first split the dataset of configurations into two sets, one containing the set of *Sunny* configurations and another containing the set of *Rainy* configurations. Then, for each set we perform an union operation over all selected features across all configurations (*i.e.*, the transitive closure of all features selected in at least one of the configurations) followed by an intersection operation between the sets. Consequently, if the distributions of selected features for sunny and rainy days are the same (*i.e.*, $Config(Sunny) \cap Config(Rainy) = 1$), then the dimension *Weather* would not matter for recommendation purposes. Hence, the weather context does not affect the configured product, and thus, the *Weather* dimension can be omitted from the TF model.

Next, we explain details of our TF model and how we make use of it to automate the configuration process.

6.2.2 Using TF for Self-Configuration of SPLs

Our basic idea for applying an N-dimensional TF approach for context-aware self-configuration is to model the relevant product context by taking the interactions

between users, features, and context into account. The proposed tensor-based technique is actually a context optimization method based on the set of previous configurations. For the sake of simplicity, we describe our approach for a single contextual variable C, and therefore the tensor X, containing the previous configurations, is a 3-dimensional tensor[2] $X \in x^{n \times m \times c}$, where n is the number of configurations, m the number of features, and c the number of numeric or categorical values from a relevant contextual variable.

A configuration is given as $X \in \{0, 1, -1\}^{n \times m \times c}$, where the values 0 and 1 indicate that a user deselected and selected a feature, respectively. In addition, -1 indicates a lack of knowledge from the user regarding the feature relevance (*i.e.*, the feature is *undefined)*. The list of recommended features can then be learned from the set of previous configurations and contexts observed in X. To this end, we apply *High Order Singular Value Decomposition* (HOSVD) as shown in Figure 6.1, to factorize the 3-dimensional tensor into three matrices $User$: $U \in \mathbb{R}^{n \times d_U}$, $Feature$: $F \in \mathbb{R}^{m \times d_F}$, and $Context$: $C \in \mathbb{R}^{c \times d_C}$, and one central core tensor $S \in \mathbb{R}^{d_U \times d_F \times d_C}$. This factorization allows to predict which features are most likely to be selected for a given context. In particular, the prediction function for a single user i, feature j, and context k is:

$$Y_{i,j,k} = S \times_U U_i \times_F F_j \times_C C_k \tag{6.1}$$

such that Y approximates X, *i.e.* minimizes a loss function $L(Y, X)$ between the real and the predicted values. Additionally, we use a tensor-matrix multiplication operator denoted by $\times_U$ where the index U shows the direction to multiply the matrix, *e.g.* $T = X \times_U U$ is $T_{i,j,k} = \sum_{i=1}^{n} X_{i,j,k} U_{i,j}$.

However, minimizing the loss function for models with a large number of parameters will lead to overfitting [Bishop, 2007]. A common way to prevent overfitting is to regularize the optimization criterion. Therefore, we add to the loss function a regularization term $\Omega(Y)$. The objective function for the minimization problem is:

$$R[U, F, C, S] = min(L(Y, X) + \Omega(Y)) \tag{6.2}$$

Loss Function. We define the loss function as:

$$L(Y, X) = \frac{1}{||S||_1} \sum_{i,j,k} D_{i,j,k} * l(Y_{i,j,k}, X_{i,j,k}) \tag{6.3}$$

where $D \in \{0; 1\}_{n \times m \times c}$ is a binary tensor $D_{i,j,k}$ whenever $X_{i,j,k}$ is observed; and $l(Y_{i,j,k}, X_{i,j,k})$ is computed by the least squares loss function $l(Y_{i,j,k}, X_{i,j,k}) = \frac{1}{2}(Y_{i,j,k} - X_{i,j,k})^2$.

Regularization. We define the regularization function as:

$$\Omega(Y) = \Omega[U, F, C] + \Omega[S] \tag{6.4}$$

[2] We only consider third order tensors for explaining the concepts, though the generalization to an N-dimensional tensor is trivial.

where $\Omega[U, F, C]$ and $\Omega[S]$ are computed by the Frobenius norm [Golub and Van Loan, 2012]:

$$\Omega[U, F, C] = \frac{1}{2}[\lambda||U||^2_{Frob} + \lambda||F||^2_{Frob} + \lambda||C||^2_{Frob}] \tag{6.5}$$

$$\Omega[S] = \frac{1}{2}[\lambda_S||S||^2_{Frob}] \tag{6.6}$$

with λ and λ_S being the regularization parameters for the matrices and core tensor respectively, and $||.||^2_{Frob}$ represents the *Frobenius norm* [Golub and Van Loan, 2012]. The Frobenius norm $||U||$ of a matrix U is given by: $||U||_{Frob} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} u^2_{i,j}}$.

We use the simplest algorithm to solve the optimization problem of Equation 6.2 which performs *Stochastic Gradient Descent* (SGD) in the factors U_i, F_j, C_k and S for a given tensor $X_{i,j,k}$ (see Algorithm 1). SGD is a standard algorithm for training a wide range of models in machine learning. This algorithm uses a stochastic update approach, that means we need to compute the gradients of the loss function and the objective function with respect to the individual components of the model:

$$\partial_{U_i} l(Y_{i,j,k}, X_{i,j,k}) = \partial_{Y_{i,j,k}} l(Y_{i,j,k}, X_{i,j,k}) S \times_F F_j \times_C C_k$$

$$\partial_{F_j} l(Y_{i,j,k}, X_{i,j,k}) = \partial_{Y_{i,j,k}} l(Y_{i,j,k}, X_{i,j,k}) S \times_U U_i \times_C C_k$$

$$\partial_{C_k} l(Y_{i,j,k}, X_{i,j,k}) = \partial_{Y_{i,j,k}} l(Y_{i,j,k}, X_{i,j,k}) S \times_U U_i \times_F F_j$$

$$\partial_S l(Y_{i,j,k}, X_{i,j,k}) = \partial_{Y_{i,j,k}} l(Y_{i,j,k}, X_{i,j,k}) U_i \times F_j \times C_k$$

As an example, consider the three-dimensional cube $User \times Feature \times Weather$ shown in Figure 6.1 for the smart home simplified EFM in Figure 6.2. Assume that we want to automatically and intelligently self-configure features to an active user. As for the standard two-dimensional case, we start with an initial set of previous configurations. It has the following dimensions:

- *User*: people for whom features are self-configured in an application.
- *Feature*: features that can be self-configured in a given application.

Initialize U, F, C, and S with small random values.
set $t = t_0$
while (i, j, k) in observations Y **do**
  $\eta \leftarrow \frac{1}{\sqrt{t}}$ and $t \leftarrow t + 1$
  $Y_{i,j,k} = S \times_U U_i \times_F F_j \times_C C_k$
  $U_i = U_i - \eta\lambda U_i - \eta\partial_{U_i} l(Y_{i,j,k}, X_{i,j,k})$
  $F_j = F_j - \eta\lambda F_j - \eta\partial_{F_j} l(Y_{i,j,k}, X_{i,j,k})$
  $C_k = C_k - \eta\lambda C_k - \eta\partial_{C_k} l(Y_{i,j,k}, X_{i,j,k})$
  $S = S - \eta\lambda_S S - \eta\partial_S l(Y_{i,j,k}, X_{i,j,k})$
end while
return U, F, C, S

Algorithm 1: Tensor Factorization (X)

- *Weather*: weather forecast when the application is dynamically self-configured, *e.g.* $Weather\{Sunny, Rainy\}$.

Then, we define a function $Y_{i,j,k}$ on the recommendation space $User \times Feature \times Weather$ specifying how much feature $j \in Feature$ is important for a user $i \in User$ in weather $k \in Weather$. For example, the first user configured the features `open window`, `open door`, and `water-on` for a *sunny day*. A second user configured the features `open window`, `close door`, and `pet-care` also for a *sunny day*. Overall, we assume that we have the historical configuration data for the first four users in the multidimensional cube as described in Figure 6.1. Then, we monitor the context to detect changes that require the system to adapt. Consequently, when adaptation is required, we select and deselect all hard features that are in accordance with the adaptation rules described in the EFM. Next, the remaining features are prioritized by our algorithm and predicted until we self-configure a valid complete configuration.

Suppose that the current context for the active user Elizabeth is sunny, and thus, the relevance of the interest on features `close` and `open door`, `water-on`, and `pet-care` need to be computed. To self-configure these features for Elizabeth, the decisions should be made at runtime based in how often the specific unknown features **"?"** are configured for the specific context *Sunny*. So, we use the Algorithm 1 to compute $Y_{i,j,k} = S \times_U U_i \times_F F_j \times_W W_k$, where i = Elizabeth; j = {`close` and `open door`, `water-on`, and `pet-care`}; and k = *Sunny*. $U \in \mathbb{R}^{5\times6}$, $F \in \mathbb{R}^{6\times2}$, $W \in \mathbb{R}^{2\times5}$, and $S \in \mathbb{R}^{6\times2\times5}$. Note that, in our example, some features constitute states (*e.g.*, *open*) to easily exemplify our approach. However, in our case study, we use real system features in their actual sense.

Finally, the system should self-configure the best N features that make the configuration valid for the specified context. Given a predictor Y, the list of the top N highest scoring features for a given user u and context c can be calculated by:

$$Top(u, c, N) = max^{N}_{f \in F}(Y_{u,f,c}) \tag{6.7}$$

where N denotes the number of features to get a valid and complete configuration. A *valid and complete configuration* is a configuration where each feature is defined (*i.e.*, (de)selected) and it satisfies all functional and non-functional variability constraints. Therefore, as each top feature is selected, decision propagation strategies are applied to automatically validate the configuration where dependent features are automatically (de)selected and the remaining ones stay undefined, until we have a complete configuration (*i.e.*, all features are defined).

6.3 Experiment Design

To evaluate the benefits and drawbacks of our approach, and thus, to answer our research questions, we conduct an empirical study. In this section, we provide details about the datasets used, the experimental protocol, and the state-of-the-art approaches we compare our proposal with. All material (*e.g.*, EFM, datasets, results) of our evaluation is accessible at our complementary webpage[3].

[3] http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/

6.3.1 Target Software Product Lines and Contexts

For our study, we make use of two state-of-the-art product lines: Dell laptop [Mendonça et al., 2009] and library [Tan et al., 2013]. In Table 6.1, we present five characteristics for each product line, including the number of features ($\#f$), number of cross-tree-constraints ($\mathcal{R}$), an upper bound estimation of the number of valid configurations by FeatureIDE statistics [Meinicke et al., 2017] ($\#\mathcal{C}$), number of previous configurations ($\#\overrightarrow{c}_x$), and number of context dimensions (d_N). Using these subject product lines, we evaluate the *effectiveness* of our approach. The effectiveness evaluates how well the proposed approach is capable of understanding the context of the users and self-configure a product at runtime.

Dataset	$\#f$	$\mathcal{R}$	$\#\mathcal{C}$	$\#\overrightarrow{c}_x$	d_N
Dell laptop [Mendonça et al., 2009]	68	7	>154,832	42	4
Library [Tan et al., 2013]	135	none	>273,534	74	5

Table 6.1: Main properties of the datasets.

Dell Laptop Product Line

The first subject system constitutes a publicly available dataset of laptop configurations [Mendonça et al., 2009]. The dataset is a tensor containing laptop configurations, encompassing 42 products (*i.e.*, previous configurations) and 68 features. It is constructed as a 6-dimensional tensor representing *User* × *Feature* × *Usage* × *Line* × *Price* × *Performance*. It delivers an application scenario where the four contexts (*i.e.*, *Usage*, *Line*, *Price*, and *Performance*) are described as relations having the following attributes:

- Usage [Game, Play, Program, Study, Work]
- Line [Personal, Professional, Gamer]
- Price [Cheap, Medium, Expensive]
- Performance [Low, Medium, High]

By counting the occurrence of each entry, a tensor of size $42 \times 68 \times 5 \times 3 \times 3 \times 3$ was created.

A Dell laptop is available in all different shapes, prices, and configurations. In order to evaluate our approach, we have to analyze the influence of a varying context on the user configuration. To this end, we defined three target contexts: *gamer*, *programmer*, and *kid* laptops. The contextual information consists of the following specifications:

Gamer laptop: The aim of the final product is to serve as a *portable gaming laptop*. We assume the following (informal) requirements for this context:

- *lightning-fast gaming* installs and loads;

- able to play *intensive games* (*e.g.*, Battlefield 4, Watch Dogs, Assassin's Creed IV, etc.) as well as *online games* (via wifi);
- as such games require a lot of power, this laptop needs more *longevity out* (*i.e.*, should not run hot all the time);
- *portable* yet powerful;
- high *Memory* (RAM) is also crucial;
- games are storage-intensive, *i.e.* this laptop should be able to store 20+ games (5-10 GB each); and
- should support *multiple applications* running at once and *streaming* games.

Programmer laptop: For this context, the customer needs a laptop with a great combination of performance and power, thus, assuming the following requirements:

- adequate support for all programming language compilers, interpreters, local servers, and code editors;
- *Speed* is important (*e.g.*, to program several intensive game applications), thus, a very good processor is required;
- good amount of memory with additional storage to efficiently run local servers, compilers, code editor, and a web browser simultaneously;
- battery life is not a priority (as the laptop is supposed to be used in the office);
- as users are supposed to spent lots of time in programming, *comfortable features* are crucial, *i.e.* this laptop should provide comfortable keyboard and a large and high-resolution display (*e.g.*, to reduce/prevent eye strain).

Kids laptop: This product should serve as an entertainment laptop for kids, assuming the following requirements:

- good wireless connection (for playing games and watching videos online);
- lightweight and highly portable (as it is for kids); and
- longer battery life (assuming that kids use it away from standard power source).

Library SPL

For our second subject system, we derived the dataset from the state-of-the-art library SPL [Tan et al., 2013]. The scope and purpose of this SPL is to have a library system equipped with all operations and facilities needed to provide services to its membership holders. In general, a library has its own management system, operating environment, payment methods, network and security system. In addition, the library offers several services to its users through an offline and online environment.

The library SPL consists of 74 previous configurations and 135 features. From the configurations, we constructed a 7-dimensional, dense tensor constituting *User* × *Feature* × *Resource Access* × *Device* × *Internet Connection* × *Environment* × *Age Range*. By counting the occurrence of each entry, a tensor of size 74 × 135 × 2 × 4 × 2 × 6 × 4 was created. It delivers an application scenario where the five contexts (*i.e.*, *Resource Access*, *Device*, *Internet Connection*, *Environment*, and *Age Range*) are described as relations having the following attributes:

- Resource Access [Digital, Phisical]
- Device [Mobile, Computer, Tablet, None]
- Internet Connection [Yes, No]
- Environment [City, Company, University, Farm, Prison, Any]
- Age Range [<10, 10-20, 20-60, >60]

To obtain a dataset of configurations, we conducted *a priori experiment* with 37 Software Engineering and Database Master and PhD students from two universities (University of Magdeburg in Germany and Federal University of Minas Gerais in Brazil). The students were asked to solve a given configuration task[4], consisting of three subtasks: *(i)* analyze the library feature model; *(ii)* configure two products based on the library feature model; and *(iii)* briefly describe the requirements specification for the created configuration.

For our evaluation, we use the whole set of generated configurations based on four contextual target environments derived from our apriori experiment: *digital scientific library*, *digital company e-book library*, *software engineering research group library*, and *farm library*. The contextual information consists of the following specifications:

Digital Scientific Library. It constitutes a digital scientific library of research articles and books.

- access is purely online (via Web);
- allows to search by classification and keywords due to multiple indices;
- registration via email;
- enables notification about newly published titles;
- no loan and renewal, as items are downloaded in PDF format;
- usage is free of charge for university's employees and students (from university network), and for other users an annual fee applies;
- password authentication and digital certificates for security; and

[4]Further details about the configuration task carried out by the participants are available at http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/.

- reasonable security for payment data and transactions.

Digital Company E-book Library. It constitutes a digital library for a small company that allows to manage e-books.

- entirely free of charge with e-books accessible via mobile interface;
- all interactions (*e.g.*, registration, searching the library, etc.) must be possible via mobile devices;
- no overdue policies or reservations (due to e-books only);
- email notification about new resources; and
- no specific network security measures.

Software Engineering Research Group Library. It serves as a small information system to track lending of books, periodicals, and technical reports between members of a research group at a university.

- access for members by a specific website;
- notifications on new arrivals;
- web interface for log in, search and reserve resources, view account, etc.;
- policy for renewing reservations (*i.e.*, not possible if already reserved by someone else);
- catalog should state fees for loss and damage of items; and
- fees can only be paid cash.

Farm Library. It constitutes a traditional (public) library system.

- should be able to operate offline;
- physical items, fees for damage and loss as well as for returning items late;
- reminders (for returning items) as notification via email and mobile phone message; and
- comprehensive user profile (including borrowing history) to be used for reading campaigns.

6.3.2 Evaluation Protocol

We assess the effectiveness of our approach by conducting a leave-one-out cross-validation study. Leave-one-out cross-validation involves using one configuration as the validation set and the remaining configurations ($n-1$ where n is the total number of configurations) as the training set. For each validation set (*i.e.*, target configurations defined in Section 6.3.1), we select $\alpha * 100\%$ features from the dataset that are subject to recommendation. For example, if $\alpha = 0.8$, we randomly give 80% of the configured features to the recommender system, while we hide the remaining features (20%). We considered different percentages, that is, $\alpha = \{0; 0.2; 0.4; 0.6; 0.8\}$. Subsequently, the system automatically conduct a self-configuration of a product based on the target context defined in Section 6.3.1 by using the tensor-based recommendation algorithm, as well the recommendation methods introduced in Section 6.3.3. This is repeated 1,000 times for each dataset and target context to ensure different combinations for each validation set. Therefore, we carried out $1,000 \times 5 \times 3$ runs on the Dell laptop dataset and $1,000 \times 5 \times 4$ runs on the library dataset. All experiments were conducted for each of our seven methods separately. Consequently, they give us 105,000 experiments on the Dell laptop dataset and 140,000 experiments on the library dataset. Note that a part of the configurations from the datasets have been held out for the purpose of parameter optimization. Every parameter in this phase was validated using a 10-fold cross validation over the use of other four target contexts (*i.e.*, office and study laptops, and city and prison libraries). Those additional experiments for parameter optimization are not counted here.

We used the *Precision* and *Recall* metrics to make a comparison between the set *Rec* of self-configured features by the algorithm and a set of relevant features *Rel* known from the oracle containing the desired configuration. Precision is calculated as follows: $Precision = \frac{|Rec \cap Rel|}{w}$. where w represents the number of self-configured features by the algorithm to have a valid complete configuration. Analogously, recall is calculated using the formula: $Recall = \frac{|Rec \cap Rel|}{|Rel|}$. *Precision* states how many of the self-configured features were relevant. *Recall* measures what percentage of relevant features has been self-configured. Since our goal is to maximize both precision and recall, in our evaluation we use a measure that combines both (*i.e.*, the F-Measure), defined as follows:

$$F\text{-}Measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{6.8}$$

All the experiments were performed on a 2 sockets Intel Xeon E5620 @2.40GHz with 4 cores per socket and 20GB of RAM. All reported results constitute the F-Measure average of all runs.

6.3.3 Comparison Approaches

We evaluate the effectiveness of our approach by comparing it with a random baseline recommender that simulates the performance of an uninformed user without any support from a recommender system (see Section 4.3). Additionally, we also compare our approach to the context-aware reduction-based approach introduced in Chapter 5, which is based on traditional CF methods. This approach compute recommendations using only the configurations made in the same context as the target

one according to the contextual information introduced in Section 6.3.1. For the Dell laptop product line, we use the context *Usage* as pre-filtering data, *Performance* as prediction modeling data, and *Price* as post-filtering data. For the library SPL, we use the contexts *Resource Access*, *Environment* and *Age Range* as pre-filtering data; and *Device* and *Internet Conection* as post-filtering data. For more information on those stages, we refer to Chapter 5.

Finally, to demonstrate the improved effectiveness of a context-aware approach, we compare the proposed TF approach with the non-contextual MF approach introduced in Chapter 4. This approach uses the BRISMF algorithm by Takács et al. [2009] to transform a two-dimensional configuration matrix into a latent space by incremental minimization of an error function. In addition, to make a fair comparison, we encoded unknown feature interests from past configurations as -1, instead of using zero entries as in Chapter 4.

6.4 Analysis of Results and Discussion

In this section, we investigate how efficient and effective is our proposed N-dimensional context-aware TF algorithm. To this end, we answer the research questions stated in the beginning of this chapter based on the results of our evaluation. First, in Section 6.4.1, we evaluate the efficiency of our approach by comparing it with a random selection of features and the reduction-based approach introduced in Chapter 5. Second, in Section 6.4.2, we assess the impact of using contextual information by comparing the proposed context-aware TF approach to the non-context-aware MF approach introduced in Chapter 4. Finally, in Section 6.4.3, we demonstrate how fast our approach can self-configure an SPL.

6.4.1 Approach Effectiveness

Figure 6.3 and Figure 6.4 present the *F-Measure* achieved by the six methods: *TF*, four *reduction-based algorithms* (user-based CF and AS, and feature-based CF and AS), and a *random* baseline method. F-Measure combines recall and precision, *i.e.* higher values are better. On the horizontal axis of the figure, we present the completeness of a configuration, *i.e.* the percentage of (de)selected features that were given to the method as training data, where only the remaining part of the configuration needed to be self-configured. In practice, it simulates the hard requirements imposed by the context. The reported results for each dataset are averaged over all F-Measures computed for the set of target contexts introduced in Section 6.3.1 which we use as validation set. For example, for the Dell laptop dataset, we average all F-Measure values from an exhaustive cross-validation for each of the three target contexts (*i.e.*, *gamer*, *programmer*, and *kids*). Moreover, note that the F-Measure value cannot be directly compared to the previous computed F-Measure values in Chapter 4 and Chapter 5, since here we do not compute precision based on the list of the top-10 recommended features. Instead, we run the algorithm each time to self-configure each top-1 feature at once until we have a *valid* and *complete* configuration.

In summary, our data reveal that, on average, the TF method outperforms the reduction-based methods on the Dell laptop dataset, while it slightly underperforms

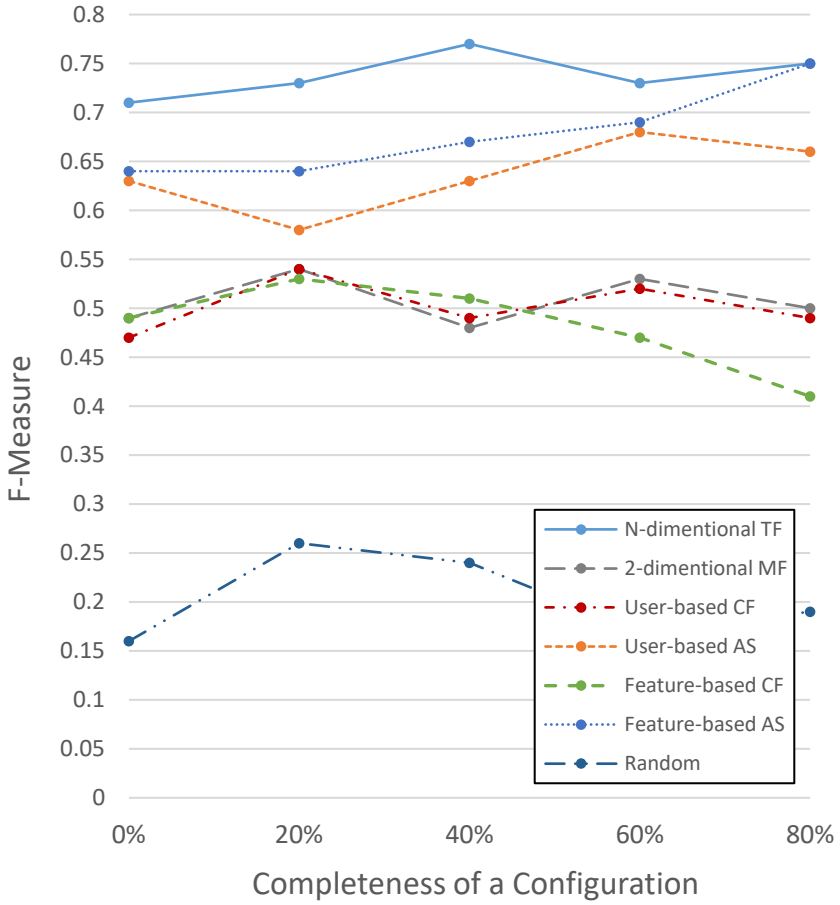


Figure 6.3: F-Measure achieved by seven different recommender methods on the Dell laptop dataset (higher values are better). The horizontal axis shows how much of the current configuration has been completed. The performance is calculated on the remaining part of a configuration.

on the library dataset for feature-based and user-based AS algorithms. Experimentally, when comparing the prediction quality of the reduction-based and the tensor-based approaches, it is clear that the reduction-based approaches work better on larger datasets. On the library dataset with 74 configurations, the results are even better than on the Dell laptop dataset with 42 configurations. This is because reduction-based approaches (see Chapter 5) provide recommendations on a particular segment and build a local prediction model for this segment. Consequently, these recommendations are based on a small number of configurations limited to the same or similar context. This tradeoff between having fewer yet more relevant data for calculating predictions (*i.e.*, the sparsity effect) explains why the reduction-based method underperforms the tensor-based method on some segments and outperforms on others.

Moreover, we observe that all algorithms outperform the baseline random recommender. However, for some algorithms, we observe a decrease of the F-Measure as the configuration becomes more complete, *i.e.* when we use more than 80% of selected features as training data. This is because the precision considers w as

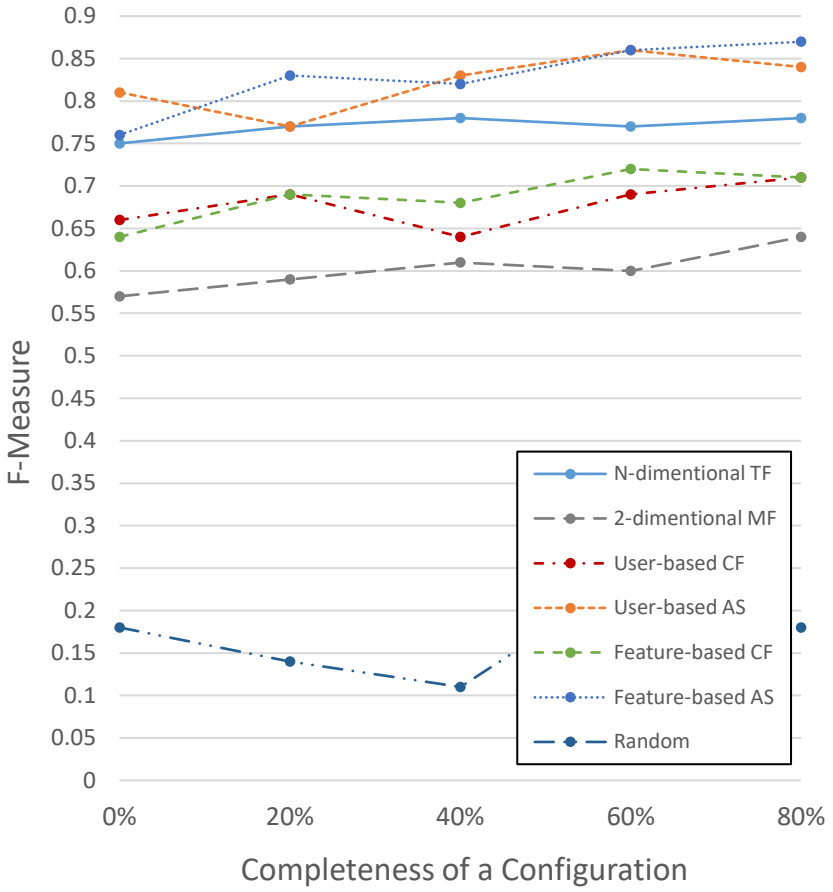


Figure 6.4: F-Measure achieved by seven different recommender methods on the library dataset.

the number of self-configured features to automatically generate a valid and complete configuration. However, for most of the target products, the data validation does not represent a maximum valid configuration (*e.g.*, the relevance of the feature `Additional_Warranty` was not specified by gamer and programmer dell laptops). Nevertheless, it does not mean that the self-selected features are not relevant since they are represented as unknown relevance (-1) in the validation set. Overall, the N-dimensional TF and the AS reduction-based algorithms achieved the best performance over all other algorithms at all stages of the configuration process. The main reason is the benefit of having more historical data for calculating unknown features' relevance. However, reduction-based approaches are a rather computationally expensive operation as for each combination of contextual requirements, a CF model needs to be trained and tested (Chapter 5). Therefore, in the given dynamic scenario where the system needs to be constantly reconfigured to deal with changes in the environment, the use of a TF algorithm seems more appropriate.

6.4.2 Contextual vs. Non-Contextual Approaches

Since it is still not clear if context matters in our scenario, in this section we conducted some experiments to assess the relevance of contextual information. We

compare our context-aware multidimensional TF approach with the non-context-aware two-dimensional MF approach introduced in Chapter 4 and we analyze the trade-offs between them. We choose this algorithm instead of others presented previously because it is usually more effective since it allows us to discover the latent configurations underlying the interactions between users and features. We conducted a comparison analysis between the algorithms from both approaches on our target datasets. It is worth mentioning that we do not use the datasets of configurations from Chapter 4, since these datasets do not work with NFPs. In Figure 6.3 and Figure 6.4, we present the *F-Measure* results achieved by both approaches for each considered dataset.

Our data reveal that the performance of the tensor-based approach outperforms the matrix-based approach on both datasets disregarding the completeness of the configuration, which indicates that context matters. The main reason is the benefit of having contextual data for calculating feature predictions, instead of only having binary information. The use of binary data turns the SPL configuration problem into a sparse two-dimensional matrix in which we have no contextual information and very few (de)selected features from previous users that must be used to compute feature predictions. However, the extent to which the contextual approach can outperform the non-contextual approach may depend on many different factors, such as the application domain and the specifics of the contextual available data. Consequently, pure matrix-based approaches work best on static environments where context does not matter.

Overall, the proposed tensor-based approach is more efficient than a pure matrix-based approach via singular value decomposition, whereas on the library SPL dataset the F-Measure values clearly outperform the values achieved in the Dell laptop dataset. This may be because the library SPL consists of a larger dataset with a higher number of context dimensions. Therefore, we believe that for most of the applications with context information available, context-aware recommendation techniques are supposed to outperform non-contextual recommendation techniques. Still, conducting experiments with other SPLs to prove this claim remains part of our future work.

6.4.3 Approach Performance

In a last experiment, we evaluate how fast our approach is by measuring its response time for self-configuring a valid and complete configuration. We record the average response time during the effectiveness experiment presented in Section 6.4.1 for each target context being analyzed.

Both datasets lead to a reasonable runtime (up to 112.5ms). The prediction runtime of our tensor-based approach is independent of the size of the dataset (*i.e.*, number of previous configurations) and is dominated by the factorization dimensions. For the larger factorization of the library SPL (7 dimensions), the runtime of our approach is worse than that for the smaller factorization of the Dell laptop (6 dimensions), resulting into 112.5ms and 51.6ms, respectively. The runtime in the Dell laptop dataset for 1-4 context dimensions are 15.6ms, 27.7ms, 39.5ms, and 51.6ms respectively; and in the library dataset for 1-5 context dimensions are

30.2ms, 47.5ms, 71.3ms, 88.8ms, and 112.5ms respectively. Hence, we conclude that our approach scales linearly to the dimensionalities d_U, d_F, and d_C of the factors $User \times Feature \times Context$.

While we found it to be very sensitive to the number of contextual dimensions (*i.e.*, the more context dimensions exist, the more time is required for training), we observed that by increasing the completeness of configurations, we obtain better results in a faster time. To improve the results, on larger datasets with large context dimensions the training phase may be done offline. Moreover, we aim at optimizing Algorithm 1, as it is trivial to parallelize the algorithm by performing several updates independently, because it accesses only one row of U, F, and C at a time. Therefore, we may use a low level language such as C++; and use parallel concurrency to exploit all processor's cores and add the results of each different thread. Also, the tradeoff between quality and speed can be minimized by controlling the number of context dimensions. That means depending on the application we can statistically measure each contextual relevance and thus reduce the number of context dimensions. Thus, further efforts can be taken in future to improve the current implementation regarding performance.

We conclude that, although the training phase posed additional challenges for our algorithms, the tensor-based approach works efficiently (within milliseconds) for both datasets. In addition, the number of NFPs for these product lines reaches a threshold of five which is an usual number of context dimensions used in real-world applications (see Mairiza et al. [2010] and Sommerville and Sawyer [1997]).

6.5 Threats to Validity

In this section, we describe some concerns related to the validity of our experiments. We have followed the guidelines proposed by Wohlin et al. [2000] in order to identify and discuss how the main validity threats to our approach were addressed. We discuss the four groups of common validity threats: *internal validity*, *external validity*, *construct validity*, and *conclusion validity* [Wohlin et al., 2000].

An *internal validity* threat concerns the specification of NFPs. In this work, we have investigated and created suitable NFPs based on experts opinion and general characteristics of NFPs found in the literature [Commission et al., 2001, Mairiza et al., 2010] (*i.e.*, our focus is in the recommender system). In our approach, we assume that a particular NFP has been already measured or specified and we use it in our N-dimensional model. We are aware that measuring NFPs might influence the scalability and time performance of our approach. This, however, goes beyond the scope of this thesis. In addition, our approach does not consider aggregation measures of NFPs. This is a complex problem, and its general solution also lies outside of the scope of this thesis.

We have identified two *external validity* threats. The first validity threat is the characteristics of the product lines and datasets used for evaluating our approach. We have addressed this validity threat by reporting the characteristics of the product lines and datasets[5], such as the product line size, the variability degree, the num-

[5]The complete representation (*xml* files) of the product lines can be found at http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/.

ber of cross-tree constraints and previous configurations (see Table 6.1). Note that in our experiments we used different product lines from Chapter 4 and Chapter 5. These product lines were more suitable for our experiments due to their easily understandable domain and an available real medium-sized dataset of configurations. However, we are aware that the use of other datasets of configurations with different characteristics could have impacted our results.

The second validity threat is the selection of the comparison approaches. To address this threat, we have included all known feature-based recommendation algorithms. However, since multi-objective optimization algorithms may also support the process of self-configuration of SPLs, we plan to extend our experiments with optimization algorithms as part of our future work.

Construct validity threats have been addressed by using the same evaluation process for all algorithms and using a standard evaluation metric. To simulate the practical configuration task while measuring the effectiveness and performance of our approach, the set of self-configured features were chosen randomly from the whole set of relevant features. Moreover, to ensure significance of the results, the reported F-Measure values are averaged over 105,000 and 140,000 runs, respectively. However, it is important to mention that our approach is designed to work most effectively when a large dataset of previous configurations and context are available. This is a requirement for our recommender system that allows us to build a set of more reliable configurations.

Finally, to address *conclusion validity* threats every effort has been made to eliminate machine dependencies and minimize the influence of the environment (*i.e.*, platform, coding, compiling, caching, etc.). Moreover, all tests were performed using the same machine and the same compiler. Furthermore, all the data used to run these experiments is publicly available for replication.

We conclude that while several validity threats exist, our results are promising as we have shown the viability of applying our approach to dynamic SPL configuration scenarios. We assume that most real-world problems will be of similar scale.

6.6 Related Work

Several authors have proposed exact and approximate optimization approaches to automatically support the *statical* and *dynamic* self-configuration of SPLs [Afzal et al., 2016, Guedes et al., 2015, Lopez-Herrejon et al., 2015, Ochoa et al., 2018, 2017]. *Statical approaches* have focused on techniques to derive product configurations in a single step (*e.g.*, [Bagheri et al., 2010b, 2012b, Henard et al., 2015b, Hierons et al., 2016b, Lian and Zhang, 2015b, Machado et al., 2014b, Pascual et al., 2015b, Shi, 2017, Tan et al., 2015a, Xiang et al., 2018]). In static approaches, features are selected based only on product requirements and human desires. Nowadays, this may not be enough due to context changes that without reconfiguration would lead to context violations. Thus, *dynamic approaches* monitors the environment and when context changes, it dynamically adapts (self-reconfigure) its behaviour to the current situation to keep fulfilling its requirements (*e.g.*, [Alférez et al., 2014, Almeida et al., 2014, Bürdek et al., 2014, Cetina et al., 2008, Mizouni et al., 2014,

Pascual et al., 2015a, Ruiz et al., 2016, Sharifloo et al., 2016]). However, quite often it requires the optimization of multiple and sometimes contradicting objectives. Consequently, there may be a set of resulting valid configurations instead of a single one, *i.e.*, a wide variety of feature combinations may meet the requirements. Furthermore, as the size and complexity of a software system increases, not only exact techniques but also approximated ones present scalability issues. Consequently, improvements related to effectiveness, scalability and performance are still needed. To achieve this, we propose a TF recommender system to add contextual data and thus exploit additional information to reduce the search effort to self-configure SPLs at runtime. Moreover, our approach that aims at self-configuring a single product can be complementary to these existing techniques.

To the best of our knowledge, there is no previous work on the use of an N-dimensional context-aware TF algorithm in the SPL configuration domain, which is the main contribution of this chapter.

6.7 Summary

Dynamic *Software Product Lines* (SPLs) demand self-adaptation of their behavior to deal with runtime contextual changes in their environment and offer a personalized product to users. In this chapter, we proposed an efficient multidimensional context-aware recommender approach to handle the self-configuration of SPLs at runtime. Our approach adopts a *Tensor Factorization* (TF) recommendation algorithm to predict unknown features interest on undefined features from a set of previous configurations according to product requirements. It dynamically and proactively adapts the feature selection according to the context of the running applications (*e.g.*, availability of resources and current user needs) while avoiding unexpected behavior. To assess the effectiveness of our approach, we conducted a series of experiments on state-of-the-art product lines. In our experiments, we empirically demonstrated that the quality of the proposed context-aware TF approach improves against the non-contextual MF approach (Chapter 4) up to 37% in terms of the *F-Measure* metric. In addition, it has a good performance already at the initial configuration stage. Moreover, we have shown that although the proposed approach presents a similar behavior to reduction-based recommendation approaches (Chapter 5), tensor-based approaches seem to be more appropriate in dynamic contexts. In the next section, we present our tool support which encompass a *recommender system* and *visualization mechanisms* to guide the interactive configuration process.

7. Visual Guidance for Software Product Line Configurations

This chapter shares material with the SAC'18 paper "Visual Guidance for Product Line Configuration Using Recommendations and Non-Functional Properties" [Pereira et al., 2018a] and the ICSR'16 paper "FeatureIDE: Scalable Product Configuration of Variable Systems" [Pereira et al., 2016b]. We presented initial ideas at GPCE'16 [Pereira et al., 2016c], COMLAN [Pereira et al., 2018b], and VaMoS'18 [Pereira et al., 2018d]. Furthermore, we have presented a preliminary empirical user study of FeatureIDE without the implementation of our approach at ENASE'16 [Pereira et al., 2016a].

In *Software Product Line* (SPL) engineering, the product configuration process aims to define, for each customer, a valid and complete configuration satisfying its requirements (both functional and non-functional). This interactive process is not trivial [Hubaux, 2014] and it is problematic mainly for the following reasons:

- In real-world SPLs, the number of features can be numerous with several types of variability relations, constraints among features, and NFPs. Hence, it is impractical for the user to keep track of all features and their dependencies during the configuration process. According to an industrial survey [Berger et al., 2013], the comprehension and visualization of models capturing the features of the SPL are reported as an issue for around 60% of the participants. Another survey on variability management tool support [Bashroush et al., 2017] shows that 17% of the tools explicitly discuss having limitations related to visualization.

- The constraints that can exist between the features (*e.g.*, requires or excludes relations) can be numerous as well. Although tool support can trigger feature constraints without user involvement based on logical rules, the user may easily lose control of the consequences of the selections. On the one hand, it may be

difficult for a user to specify a valid configuration, especially since also features of no interest need to fulfill their dependencies. On the other hand, the user can unintentionally introduce conflicts by specifying mutually exclusive features. Consequently, the number of features and constraints impact the interactive configuration process both in the time that it takes and in the quality of the results.

- The selection of functional features can determine a valid configuration responding perfectly to the functional needs of the product. However, it might have non desired NFPs (*e.g.*, slow performance or too expensive). Although automatic approaches for obtaining configurations that optimize NFPs exist [Ochoa et al., 2017], they focus on techniques to derive configurations in a single step, not allowing users to interact with the configuration process.

There are many SPL tools that aim to provide configuration support and several recommendation heuristics have been proposed to guide the order of feature selection (see Section 3.4.3). However, no visual support is provided to decision makers to focus on likely relevant configuration options including recommendations. Thus, we address this challenge by providing a visualization mechanism that shows to the user the impact, in terms of SPL constraints and NFPs, of each relevant feature over a set of target product requirements. Moreover, relevant features are scored (five-stars classification) and ranked (top-10 features) according to their importance for the user. In summary, we provide the following three contributions:

1. A set of interrelated visualizations: 1) Information hiding. 2) 5-star view to suggest feature selections. 3) Feature's graph to visualize features constraints and the positive or negative impact of NFPs. 4) NFP's graph to visualize the impact of an NFP over features and other NFPs.

2. The implementation is publicly available as an extension of the SPL tool FeatureIDE [Meinicke et al., 2017]. This system provides visual support to users. It guides them through the decision-making process and allows them to focus on valid and relevant parts of the configuration space.

3. An empirical evaluation of the approach in terms of effectiveness, scalability and performance on eleven SPLs.

Furthermore, from our experimental results on eleven realistic product lines, we answer the following three research questions:

- *RQ1.* Does the set of proposed visualizations support the configuration process of realistic product lines?

- *RQ2.* Does the set of proposed visualizations avoid information overload?

- *RQ3.* How long does it take to construct the set of visualizations when increasing the complexity of the problem?

To address *RQ1*, we evaluate the effectiveness of the implemented visualization mechanisms by an user empirical study of our tool support. Regarding *RQ2*, we investigate if the proposed views are scalable for large feature models. Finally, *RQ3* analyzes if the implemented views achieved complete visual continuity.

This chapter is structured as follows: Section 7.1 presents relevant background information about the state-of-the-art tool we extended with our approach. Section 7.2 describes the proposed visualization mechanisms. Section 7.3 presents the evaluation. Section 7.4 discusses the threats to the validity of our evaluation results. Section 7.5 presents related work. Finally, Section 7.6 summarizes the chapter.

7.1 FeatureIDE Configurator

Users can be guided to configure valid products using specialized configurators. Based on a preliminary survey [Pereira et al., 2015] and two empirical user studies [Constantino et al., 2016, Pereira et al., 2013], we decided to follow extending a state-of-the-art tool FeatureIDE [Meinicke et al., 2017] with the proposed recommender system. FeatureIDE is an open-source Eclipse-based tool which widely covers all phases of SPL development. Besides being integrated with several programming and composition languages, FeatureIDE provides the key functionality of typical tools, such as SPL editor, analyzer, and configurator. Moreover, FeatureIDE is actively used by industry practitioners and academic researchers.

In this section, we present the configuration support of FeatureIDE. With a close connection to FeatureIDE's feature-model editor, the configuration editor can provide several mechanisms that guide the user. With automated decision propagation, FeatureIDE ensures that any partially configured product is in accordance to the feature model so that the result only describes valid combination of reusable artifacts. Therefore, by integrating the proposed recommender system with FeatureIDE, we can provide to users a most efficient and complete environment.

Preventing Conflicting Feature Combinations

Product configuration is a decision process to form a valid feature combination, where the interdependencies of all features are considered [Pohl et al., 2005]. Especially when dealing with large feature models with complex feature dependencies, a configuration process without tool support is an error-prone and tedious task. Completely configuring products and checking validity afterwards is henceforth not advisable as at least one feature dependency is probably violated.

To ease the SPL configuration process, FeatureIDE provides an iterative strategy, which only allows feature selections that comply with the feature model's dependencies. Thus, similar to the configurators SPLOT [Mendonça et al., 2009] and fmp [Antkiewicz and Czarnecki, 2004], FeatureIDE prevents the user to introduce conflicts in their configuration. This functional characteristic of FeatureIDE is based on two concepts: *(a)* a close coupling between configurations and their feature models and *(b)* decision propagation.

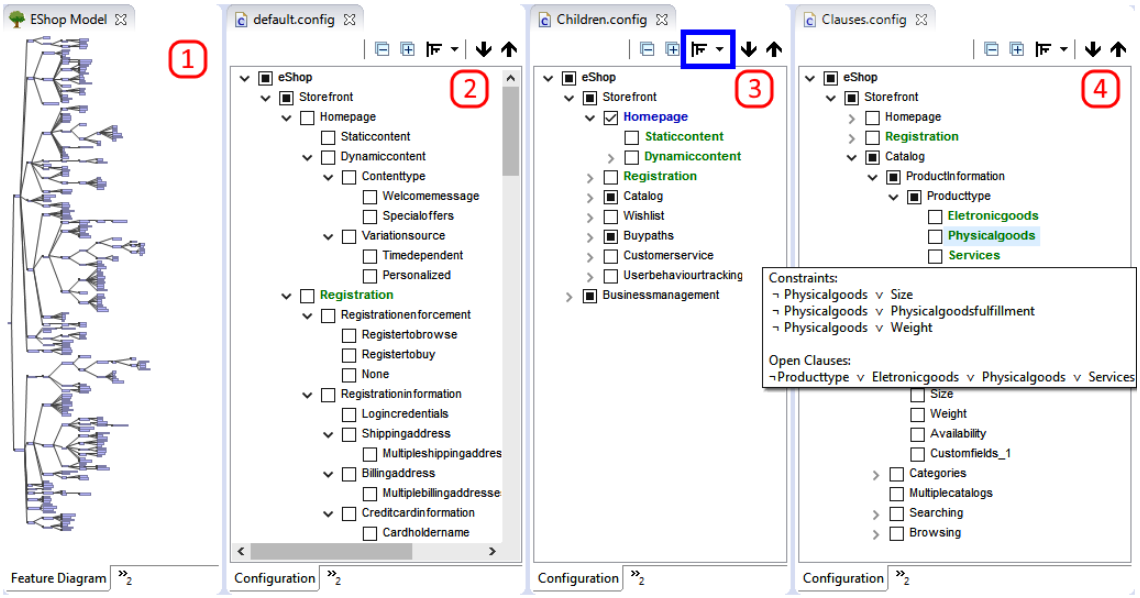


Figure 7.1: An overview of FeatureIDE's configuration support: ① feature model editor, ②-④ configuration editor (② showing all features, ③ showing direct children, ④ finalizing configuration).

Close Connection of Feature Models and Configurations. The feature model and the configuration editor of FeatureIDE are closely connected and influence each other. On the one hand, the configuration editor of FeatureIDE uses the same hierarchical structure as the corresponding feature model. Furthermore, the feature model influences configurations so that, for instance, a renaming of a feature also renames the feature in each configuration. On the other hand, each selection in a configuration forces a validity check considering the corresponding feature model. In addition, all implied and excluded features are automatically (de)selected and a change of their selection is forbidden. In Figure 7.1.①-②, we depict this functionality for the product line *EShop*. In Figure 7.1.①, the dependencies of the feature model are hard to resolve. However, the representation in the configuration editor (see Figure 7.1.②) allows an iterative selection of features.

Decision Propagation. Based on the close connection between feature models and configurations, FeatureIDE's configuration editor prevents conflicts in each iteration of the configuration process using *decision propagation*. In detail, if a (de)selection of a feature forces the (de)selection of another feature, FeatureIDE automatically adopts the implied configuration changes. For instance, if we select the feature `Welcomemessage` in the product line *EShop* (see Figure 7.1.②), all parent features will be also selected.

Although FeatureIDE presents several mechanisms to support an easier configuration process, we identified the following challenges:

- *Unclear requirement specification.* Some vague descriptions and even misleading information may be introduced to the requirements making hard the preliminary process to relate requirements to features in the feature model.

- *Too many options and complex relationships.* Configuring a product can be a difficult process as users usually do not know all features and their dependencies, especially for large feature models [Benavides et al., 2010]. Consequently, showing all features and relationships (see Figure 7.1.②) is impractical as a user can only focus on one part of the configuration at once.

- *Subjective features and unclear user preferences.* Decision makers may be unsure about users needs when confronted with a set of unknown features.

- *Requirement Inconsistencies.* Users often specify requirements that are inconsistent with the feature model's constraints and no support is provided to decision makers to prioritize the most relevant choices.

- *Get a valid configuration.* Although FeatureIDE ensures that any partially configured product is in accordance with the SPL constraints, it does not guide users into a valid configuration. To finish the configuration process, features of no importance to the user need to be taken into account (*i.e.*, to fulfill the constraints). Since decision makers again are not provided with additional support to guide them in this proces, this may lead to delays due to users' exploration of choices at each step of the configuration process.

Thus, handling the SPL configuration process in FeatureIDE is still a critical issue and the current literature still lacks a complete technique to support this process. In the next section, we illustrate FeatureIDE's facilities to support these challenges by providing an advanced configuration support. Our approach ensures a *valid* and *complete* configuration while simultaneously maintaining efficiency as the user can focus on their features of interest.

7.2 Visualization and Selection Mechanisms

To represent functional and non-functional features' interdependencies, as well as features' relevance on SPL configurators, we provide a set of four visualization components: 1) *information hiding*, 2) *5-star view*, 3) *feature's graph*, and 4) *NFP's graph*. The use of visualization and interactive techniques on SPL can widely reduce the complexity of understanding large and complex feature models [Card et al., 1999b, Nestor et al., 2008a]. The visualizations allow the user to freely explore the functional and NFPs related to the product-under-configuration in order to choose the features that best meet their requirements. Our aim is to support the visualization of relevant information configuring a target product.

Through the proposed visualization mechanisms, decision makers can: (a) rely on a small relevant set of features, instead of going through the entire feature model; *(b)* visualize, in an interactive way implicit and explicit interdependences among features and NFPs; and *(c)* go through a list of the most recommended features. In addition, the user is able to automatically configure a complete product in any point of the configuration process. Consequently, the recommendation algorithm selects the top features to meet a complete valid configuration that is in accordance with the SPL constraints and product requirements. This is especially applicable when configuring large-scale SPLs in which the interactive configuration is recognized to be

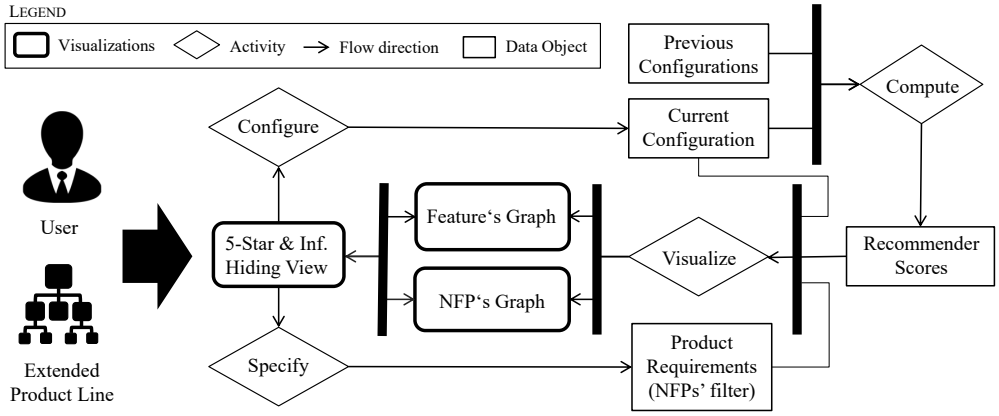


Figure 7.2: An overview of the proposed visualizations.

a time-consuming and tedious task. Next, we describe each visualization component shown in Figure 7.2.

7.2.1 Information Hiding View

To ease the configuration process, we provide information hiding mechanisms[1] that focus the user's view on the relevant configuration space.

Focused View. This mechanism focuses the decision makers's view on the part of the configuration that is currently (de)selected. The essential idea of this mechanism is that a feature tree hierarchy represents the features' degrees of abstraction. The higher a feature is located in the tree, the higher its level of abstraction is. In contrast, leaf features (*i.e.*, features without any children) are the most detailed and technical features. Consequently, a user should first decide between the most abstract features before going into detail. To support this intuitive process, we implement a level-wise view on the feature tree (see Algorithm 2 and Section 4.2.1 for the formulation). We assume that when the user selects a feature, they are probably interested in its sub-features (*e.g.*, fine-grained features of the same area). Thus, the view automatically expands and shows only the direct sub-features from selected features. This behavior is exemplary illustrated in Figure 7.1.③. Initially, only the feature `Storefront` is expanded. After the user selects the feature `Homepage`, the expand algorithm shows the sub-features `Staticcontent` and `Dynamiccontent`. With the focus on direct sub-features, we reduce significantly the user decision space in each configuration step. Thus, the user has a proper overview of the configuration process and is able to focus on one particular choice at-each-time. However, when the user has selected all features of their choice, their configuration might still be invalid due to unsatisfied feature model constraints (*i.e.*, decision propagation and expand algorithms can only benefit to configure partial configurations). In this case, it is necessary to (de)select further features to make the configuration valid. Next, we present a highlighted view that supports the user in this process. The sequence of the variation point at which the user makes their configuration decisions may have significant impact on the efficiency of the product configuration.

[1] A demo video of these mechanisms in FeatureIDE can be found at https://youtu.be/zM9K3wqUiVE

In addition, filters can be applied simultaneously to NFPs. Filters define NFPs thresholds from a minimum to a maximum value that fulfills the product non-functional requirements. We assume that the decision makers are capable of understanding the goals of the target system and they are able to translate those goals into priority information that would allow them to filter relevant features. Thus, the set of features in the visualizations is adapted to satisfy the set of predefined filters. By providing this mechanism, we can additionally reduce the decision makers configuration workload.

input : $\mathcal{FM}$, $\vec{pc}$
output: Focused View

1 $compress(\mathcal{FM})$;
2 **foreach** $c_i = 1 \in \vec{pc}$ **do**
3 $\quad \vec{ch} \leftarrow children(c_i)$;
4 $\quad$ **foreach** $ch_j \in \vec{ch}$ **do**
5 $\quad\quad expand(ch_j)$ from $\{c_i\}$;
6 $\quad$ **end**
7 **end**

Algorithm 2: Focused view algorithm.

Finalize Partial Configurations. Decision propagation and specialized expand algorithms can only help to configure partial configurations. Still, a configuration needs to fulfill all dependencies defined in the feature model. Automatic selection of features is an efficient way to create a valid configuration based on the given partial configuration (*e.g.*, the auto-completing mechanism presented by SPLOT [Mendonça et al., 2009]). However, such algorithms arbitrarily select features without considering the user intentions. Thus, undesired features might be selected as well. In order to address this challenge, the tools VISIT-FC [Nestor et al., 2008b] and FaMa [Trinidad et al., 2008] introduce dependency visualization mechanisms to support the user in configuring products, but both tools present all features to the user. In contrast, our tool provides a mechanism that guides the user to a valid configuration, reasoning from a smaller number of features.

Based on unsatisfied clauses of the feature model's CNF-representation [Benavides et al., 2010], this mechanism shows the user which decisions are necessary to finish the configuration process by highlighting the corresponding features. First, the algorithm transforms a propositional formula into *Conjunctive Normal Form* (CNF). For a valid configuration, all clauses of the CNF must be satisfied. Using the user current partial configuration $\vec{pc}$, the algorithm determines the set of all unsatisfied clauses $\mathcal{UR} \subseteq \mathcal{R}$ in the CNF. This set can be defined as $\mathcal{UR} = \{\vec{r} \in \mathcal{R} \mid \forall i \in \{1, .., |\mathbb{F}|\} : r_i = -1 \vee complete(c_i) \neq r_i\}$ (see Section 4.2.1 for the formulation). Then, the algorithm is able to highlight every feature that is contained in an unsatisfied clause (see Algorithm 3). More formally, for each $\vec{r} \in \mathcal{UR}$ the algorithm determines all features i with $r_i \neq -1$. To provide optimized guidance for the decision maker, the algorithm considers each unsatisfied clause separately and *highlights* its features. As each clause needs to be satisfied, the user can focus on one clause at a time. We associate the green and blue colors to features that may be selected and deselected

respectively. In order to satisfy the CNF clause, the decision maker would either need to deselect a blue parent feature or select one of its green children features. Naturally, (de)selecting one of those features automatically satisfies the corresponding clause $\vec{r}$. We exemplary show this behavior in Figure 7.1.④. As shown, only the current open clause (displayed in the tooltip of `Physicalgoods`) is expanded. The feature `Producttype` was automatically selected by decision propagation. Thus, at least one of its children (highlighted with green) has to be selected to satisfy the open clause. A deselection of a feature might also satisfy a clause as shown in Figure 7.1.③ with a blue highlighting of the feature `Homepage`. After a clause is satisfied by the decision maker (de)selection, the focus automatically changes to the next unsatisfied clause. Thus, again the number of configuration options presented to the user is reduced to a minimum. Using this mechanism, the user can efficiently finish the configuration process and simultaneously prevent undesired feature selections.

input : $\mathcal{FM} = (\mathbb{F}, \mathcal{R})$, $\vec{pc}$
output: Finalized Partial Configuration

```
1 UR ← getUnsatisfied(R, pc)
2 foreach r_i ∈ UR do
3 |   if r_i = 1 then
4 |   |   green(r_i)
5 |   else
6 |   |   blue(r_i)
7 |   end
8 end
```

Algorithm 3: Highlight view algorithm.

In the next section, we propose a set of three visualization mechanisms over a focused and highlighted view on the configuration.

7.2.2 5-Star View

In practice, the configuration process starts either from scratch or by loading a partial consistent configuration file. The 5-star view represents features' relevance over a focused view of the feature model plus a color highlighting notation (see Section 7.2.1). It uses the state-of-the-art recommender algorithm introduced in the previous chapters to display the score in the form of a 5-star scale as shown in Figure 7.3. From this view, the user will be able to (de)select features of interest to configure a product. Features' scores are computed after each user interaction. Each score is normalized in a scale from zero to five and associated with an unselected feature on the focused view. This view reflects a generalization of knowledge that is inferred from previous configurations, and ensures the consistency of complete and partial configurations.

This is particularly helpful to users when it is not clear which feature selection fulfills their requirements better. Thus, the user can efficiently explore the configuration space and finish the configuration process preventing undesired feature selections.

7.2.3 Feature's Graph View

To represent the impact of undefined features (*i.e.*, non-(de)selected) over a set of relevant features and NFPs, we extend the FRoGs view proposed by Martinez et al. [2014]. Similar to FRoGs, each graph is a *radial ego network* representation associated with a specific target feature f_c displayed in the center (*e.g.*, `sensor` in Figure 7.4a). We extend FRoGs notation by showing to decision makers how the selection of f_c affects the features' scores of a set of relevant features and the values of its NFPs. Also, while FRoGs proposes to use all the features in the feature model, we reduce the information density by using only the features of the focused view. Then, the set of n relevant features and m NFPs are displayed around f_c with a constant separation of $2\pi/(n+m)$. This separation allows to uniformly distribute all features and NFPs around the circle. Each graph displays different circular sectors (*i.e.*, zones) associated with two main perspectives: *features' relevance scores* and *values of NFPs*. Our approach assumes that the features are annotated with NFPs and NFPs interactions have been previously measured for state-of-the-art approaches.

Perspective: Features' Relevance Scores. A set of relevant features is displayed in this perspective depending on their computed relevance score (Section 7.2.2). Features' relevance scores are associated with a minimum value of 0 and a maximum value of 5. Features are positioned in the range of these values on the graph. To represent different ranges of values, the zones in this perspective are displayed with different shades of the yellow color. The light yellow of the positive zone contrasts with the dark yellow of the negative zone. Features in the positive zone are closest to f_c meaning that, if f_c is selected, these features are more likely to be selected (*i.e.*, the extreme of this zone is represented for the maximum value of relevance score, which is 5). Features in the negative zone are furthest from f_c meaning that if f_c is selected these features are more likely to be deselected (*i.e.*, the extreme of this zone is represented for the minimum value of relevance score, which is 0). As an example, Figure 7.4a shows the impact of the feature `sensor` on the relevance score of nine relevant features. Relevant features are the set of undefined features displayed in the focused view (Figure 7.3), as well as `sensor` dependent features. As we can see in this example, the selection of the feature `sensor` potentially encourages the selection of `fire`, `alarm`, `siren`, and `manual` located in the extreme of the positive

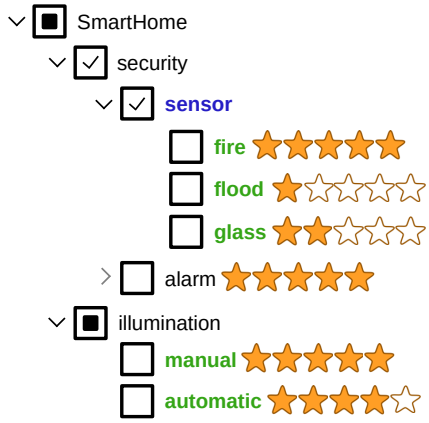
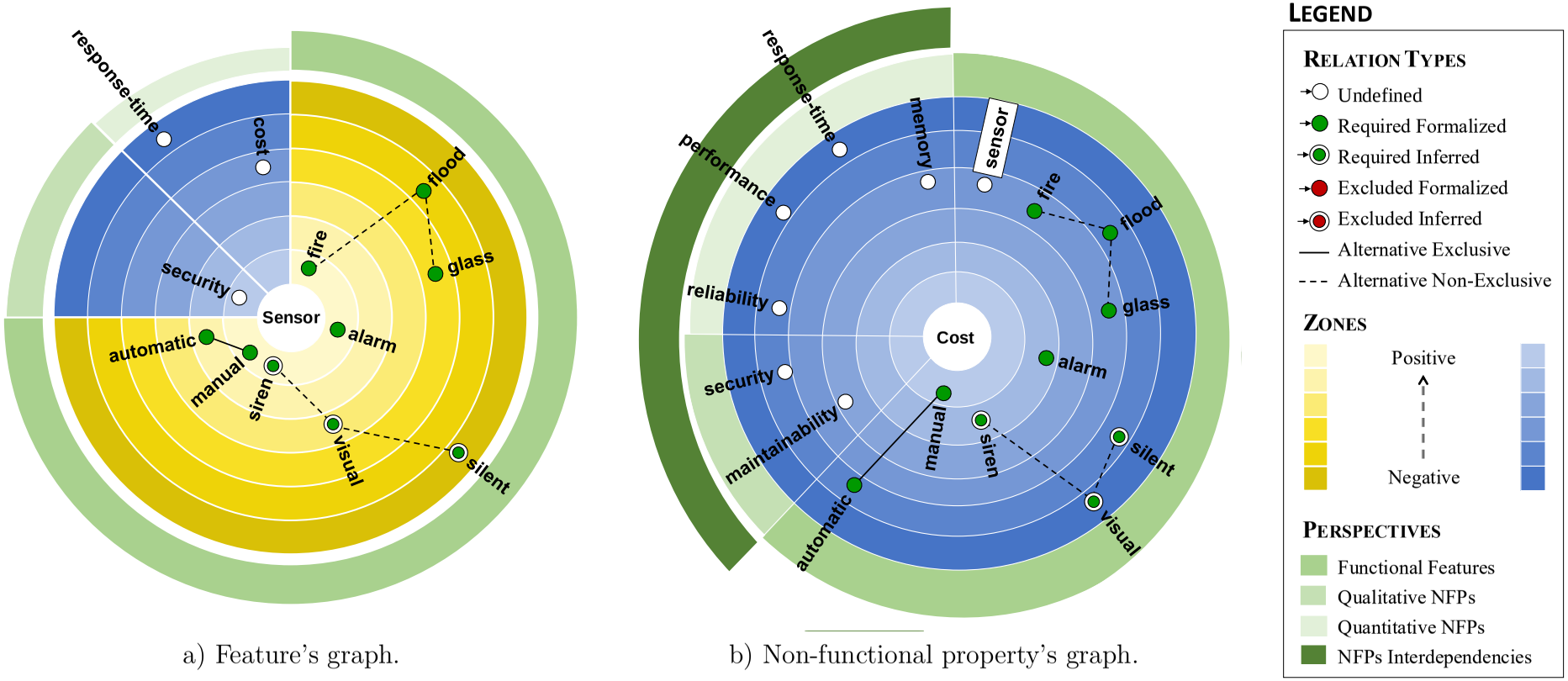


Figure 7.3: 5-star view.



a) Feature's graph.

b) Non-functional property's graph.

Figure 7.4: Visualization graphs.

zone. Although the sub-features of `alarm` (*i.e.*, `silent`, `visual`, and `siren`) are not displayed in the focused view, the target feature `sensor` requires `alarm`. Consequently, at least one of `alarm` sub-features need to be selected if `sensor` is selected. Therefore, we classify these features as relevant. A relevance score for a relevant feature is updated over time depending on $\overrightarrow{pc}$.

We use ● and ● for required and excluded features constraints, respectively. If it is neither a requires nor an excludes constraint, we use ○ . On the one hand, the notation ● f_i illustrates that there is no occurrence of f_c without f_i given $\overrightarrow{pc}$. However, when a set of required features are linked by a dashed line, *at least one* feature must be selected if f_c is selected, otherwise *exactly one* of those features must be selected. For example, the selection of `sensor` requires the selection of *exactly one* of the features `manual`, and `automatic`, and the selection of *at least one* of the features `fire`, `flood`, and `glass`. On the other hand, the notation ● f_i illustrates that there is no occurrence of f_c and f_i together given $\overrightarrow{pc}$.

Requires and excludes notations are consequences of formalized and inferred constraints. *Inferred constraints* are represented by adding an extra circle in the node with the color associated to the type of the constraint (*i.e.*, ◉ for requires and ◉ for excludes). An inferred constraint is a constraint that is not explicitly formalized in the feature model by the feature tree and the CTCs, but that exists because of logical rules [Martinez et al., 2014]. Figure 7.4a shows an example of inferred constraint for the features `silent`, `visual`, and `siren`. In the feature model shown in Figure 2.1, there is a CTC `sensor` requires `alarm`. Consequently, by looking at the feature tree, the feature `alarm` requires at least one of the features `silent`, `visual`, and `siren`. Therefore this implication is not explicitly formalized neither in the feature tree nor in the CTCs. We display all the dependent features of f_c and its relation types by analyzing the propositional formula of the feature model. Figure 7.4 shows the complete legend of the graph as it is shown in the tool.

Perspective: Values of NFPs. A set of NFPs is displayed in this perspective depending on their values over the feature f_c. NFP values are categorized into two main categories: *quantitative* and *qualitative*. For a detailed explanation on these categories we refer to Chapter 3, Section 3.4. In this work, as we focus on the visualizations, we assume that quantitative and qualitative values were already specified by using state-of-the-art techniques.

Since the types of quantitative NFPs are quite different, their range of values may suffer a wide variation (*e.g.*, 100-60,000ms for `response time` and $50-850 for `cost`). In this scenario, the range of values should be normalized so that each NFP contributes approximately proportionately to their minimum and maximum values. Thus, we use a simplest feature scaling method (*a.k.a. unity-based normalization*) to rescale the range of NFP values between any arbitrary minimum and maximum value to a common scale in the range of $[0, 1]$. The general formula is given as:

$$p'_i = \frac{p_i - min_{p_i}}{max_{p_i} - min_{p_i}} \tag{7.1}$$

where p_i is the NFP value and p'_i is the normalized value. We normalize the values by finding the minimum min_{p_i} and maximum max_{p_i} values p_i assumes. Our aim is that the corresponding normalized values allow the comparison of different NFPs.

NFPs are associated with a negative or positive impact of their values over f_c. Thus, an NFP is displayed in the zones by considering a scale ranging from a negative impact of 0 to a positive impact of 1. For quantitative NFPs where higher values have a negative influence over the configuration (*e.g.*, `cost`), we reverse the scale (*i.e.*, $p'_i = 1 - p'_i$). For qualitative NFPs, we automatically mapped them onto real values to be handled as quantitative properties in a scale of [0,1]. To represent the six different zones in the graph, we consider six qualitative levels [Asadi et al., 2014]: high [0.85, 1], medium [0.68, 0.85] and low [0.51, 0.68] positive; and low [0.34, 0.51], medium [0.17, 0.34] and high [0, 0.17] negative. Thus, 1 implies the highest positive quality level while the value 0 illustrates the highest negative quality level.

To represent different ranges of values, the zones in this perspective are displayed with different shades of the blue color (Figure 7.4a). As much light is the color of the zone where the NFP is positioned much positive influence it has over f_c (*e.g.*, `security`), while as much dark the color much negative the influence (*e.g.*, `response time`). Multiple NFPs can be associated with the same feature, as well as some features may not be annotated with any NFP. Therefore, the proposed graph adapts dynamically in accordance with the amount of information to be displayed (*i.e.*, if there are no NFP associated to f_c, then the graph fully represents the perspective of features' relevance scores).

7.2.4 Non-Functional Property's Graph View

Apart from interacting with the list of undefined features, the user can interact with NFPs displayed in the feature's graph view. This view is associated with an NFP p_c displayed in the center (*e.g.*, `cost` in Figure 7.4b). It shows the effect of p_c over relevant features and other NFPs. Each graph displays different circular zones associated with two perspectives: *features' NFP values* and *NFPs interdependencies*.

Perspective: Features' NFP Values. A set of relevant features is displayed in this perspective depending on their p_c value. The zones in this perspective are associated with a minimum to a maximum value of p_c. As much light the color where the feature is displayed, much positive the influence of p_c over the feature, otherwise much negative the influence. As an example, by visualizing the *functional features perspective* in Figure 7.4b, the decision maker has an overview of which features are cheaper (*e.g.*, `manual`) and more expensive (*e.g.*, `visual`).

Perspective: NFPs Interdependencies. The set of NFPs is displayed in this perspective depending on their effect on p_c. To measure the effect of an NFP on p_c, we use the *Pearson Correlation Coefficient* (PCC) also known as bivariate correlation. PCC measures the degree of covariation between two NFPs. For example, given two NFPs `cost` and `security`, if, on average, expensive products have high security (and cheap product low security), we say that `cost` and `security` are correlated. To measure the correlation between NFPs, our approach needs to find all valid combinations of features that contribute to both NFPs. However, the number of feature combinations in a feature model may be exponential to the number of features in the worst case. Therefore, we compute PCC using the subset of previous configurations from past users. Thus, PCC is the covariance of two NFPs p_c and p_i divided by the product of their standard deviations. The equation is given as:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} = \frac{\mathrm{cov}(X,Y)}{\sqrt{\mathrm{var}(X) \cdot \mathrm{var}(Y)}} \tag{7.2}$$

where x_i and y_i are the values that p_c and p_i assumes for each given previous configuration. $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the sample mean (analogously for $\bar{y}$). Thus, the value r is obtained based on the subset of n valid previous configurations (*i.e.*, samples), then it is an estimation of the true value of the correlation of all valid configurations. It assumes the values between [-1,1], where -1 implies a negative linear correlation (*i.e.*, p_i decreases as p_c increases), a value of 0 implies that there is no linear correlation between p_i and p_c, and 1 implies a positive linear correlation (*i.e.*, p_i increases as p_c increases). Thus, the degree of correlation is more significant as it is closer to 1 (direct correlation) or -1 (inverse correlation).

To represent the different ranges of correlations, NFPs are positioned in the range of the values [-1,1] on the graph. Thus, NFPs closest to p_c has a positive effect on p_c, meaning that for a positive value of p_c there is a positive effect of p_i (*i.e.*, the extreme of this zone is represented for the maximum value of PCC correlation, which is 1). NFPs furthest from p_c has a negative effect on p_c, meaning that for a positive value of p_c there is a negative effect of p_i (*i.e.*, the extreme of this zone is represented for the minimum value of PCC correlation, which is -1). In this scenario, if a lower value of p_c is more desirable (*i.e.*, it has a more positive effect over the configuration for stakeholders), then we plot the NFPs in the graph by taking into account the inverse correlation of p_c. As example, Figure 7.4b shows that the NFPs `response time` and `performance` have a higher negative effect in the configuration over a positive effect of the NFP `cost` (*i.e.*, cheaper products have a high response time and low performance).

7.3 Evaluation

To empirically evaluate our approach, we extend the state-of-the-art tool FeatureIDE with the proposed visualizations described in Section 7.2. All the experiments were performed on an Intel Core @3.30GHz with 8GB of RAM. In this section, we report the settings of the conducted experiments and analysis that were performed to answer the research questions *RQ1–3* introduced in the beginning of this chapter. Overall, the experiments reported in this section intend to present a preliminary observation on the feasibility of our approach.

7.3.1 Approach Effectiveness

We empirically analyze the effectiveness of our approach to reduce the complexity of the interactive configuration process by assisting the decision makers to reasoning on a small set of relevant information. We designed a preliminary experiment in which 10 participants used the FeatureIDE configurator to configure the Dell laptop product line[2] [Mendonça et al., 2009]. This product line is suitable for our experiment because the domain is easy to understand and a realistic dataset of configurations

[2]We adapt the Dell laptop product line found at http://www.splot-research.org/. The complete feature model with a dataset of 33 real configurations can be found at http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/.

was available. Table 7.1 (in the seventh row) summarizes the main characteristics of the Dell product line. The first column identifies the product line. We use |F| for the total number of features, |M| and |O| for the number of mandatory and optional features, |OR| and |XOR| for the number of alternative non-exclusive and exclusive groups, |CTC| for the number of cross-tree-constraints, and |NFP| for the number of NFPs. Then, we use *Height* for the height of the feature tree and |C| for an upper bound estimation of the number of valid configurations (computed by the *FeatureIDE Statistics* using a timeout-length of 15 minutes). Finally, *Graph Data* column shows the number of displayed features and NFPs, and the *Time* column shows the time in milliseconds to generate the graphs for the worst case scenario.

Firstly, we developed a requirement specification for a *gamer* Dell laptop[3]. Laptops for gaming differs from laptops targeting other type of customer profiles (*e.g.*, atom netbooks would not meet the needs of a gamer laptop due to the high processing demand of current games, which require a high performance). In this context, if the product obtained by the user is significantly different from the expected ones, we assume that our approach is not appropriately guiding the user. We use $Recall = |c_i \cap c_j|/|c_j|$ to evaluate the similarity between a user configuration c_i and an expected configuration c_j (*i.e.*, a valid set of truly relevant features known from the specified requirements). We consider as c_j the expected product configuration much similar to c_i. The recall vary between 0 and 1. As much close to 1 is the recall, much similar c_i is of c_j. Therefore, we assume that our approach is successfully guiding the user to meet the specified requirements.

The experiment consisted of two groups with 5 Master's students without SPL experience and with knowledge in computer games to make sure they are on the same level of background. One group used the FeatureIDE configurator without the proposed visualizations, while the other group used the FeatureIDE configurator with our visualization components. Moreover, in order to balance knowledge of participants, we conducted a training session of 1 hour to introduce the basic concepts of SPL configuration and the tool support. After the training session, we asked the participants to perform a configuration task of a gamer Dell laptop by following the requirement specification. All configurations were created from scratch and all tasks were based on the Dell product line to provide the same level of difficulty among the participants. For each participant ($P1$ to $P10$ to keep their anonymity), we performed an analysis of the total of configuration interactions (*i.e.*, number of decisions), the number of rollbacks (*i.e.*, modification of a previous decision), the time spent to complete the configuration, and the confidence level of obtaining the target product. Table 7.2 summarizes our findings. The first column identifies the participant. The recall obtained for each configuration is shown in the second column. The number of configuration interactions and rollbacks are displayed in the third and fourth columns respectively, while the time spent for the user (in minutes) to complete the configuration task is shown in the fifth column. Moreover, the confidence level of each final configuration for each participant is displayed in the last column. Participants had the following options to answer: ○ *I am not confident*, ◒ *I am a little confident*, ◒ *I am confident*, and ● *I am very confident*.

[3]Further details about the product specification and the configuration task carried out by the participants are available at http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/

Product Line	|F|	|M|	|O|	|OR|	|XOR|	|CTC|	|NFP|	Height	|C|	Graph Data	Time (ms)
Mobile Media [Figueiredo et al., 2008]	17	6	11	1	1	1	6	4	252	16	287
Email System [Thüm et al., 2014]	23	3	20	0	1	0	6	3	5,632	25	363
Smart Home [Alférez et al., 2009]	28	3	25	1	1	3	6	4	182,280	17	348
Devolution [Thüm et al., 2014]	32	11	21	3	1	0	6	6	19,656	17	198
Gasparc [Aranega et al., 2012]	38	23	15	1	4	0	6	6	352	17	247
FraSCAti [Seinturier et al., 2012]	63	19	44	0	2	43	6	5	>780,009	39	1,604
Dell laptop [Mendonça et al., 2009]	68	13	55	0	12	7	5	4	>391,529	52	4,356
Model Transformation [Czarnecki and Helsen, 2003]	88	18	70	14	11	0	6	7	>775,446	36	3,491
Battle of Tanks [Thüm et al., 2014]	144	8	136	1	9	0	6	4	>799,682	62	6,939
Web Portal [Mendonça et al., 2008]	237	51	186	34	0	0	6	11	>559,146	60	8,396
e-Shop [Lau, 2006]	286	76	210	39	0	27	6	11	>583,758	58	8,328

Table 7.1: Characteristics of the product lines (F: features, M and O: mandatory and optional features, OR and XOR: alternative non-exclusive and exclusive groups, CTC: cross-tree-constraints, NFP: non-functional properties, *Height*: height of the feature tree, C: valid configurations).

Partic.	Recall	Decision	Rollbacks	Time	Confidence
$P1$	1	23	8	27	○
$P2$	0.77	21	6	20	○
$P3$	0.69	17	5	22	◒
$P4$	0.93	19	6	18	○
$P5$	1	17	3	19	◒
Average	0.87	19.4	5.6	21.2	N/A
$P6$	1	17	4	38	◒
$P7$	1	15	2	32	●
$P8$	0.92	13	0	40	●
$P9$	1	14	1	36	●
$P10$	1	15	2	48	●
Average	0.98	14.8	1.8	38.8	N/A

Table 7.2: Experiment results from a usability evaluation of FeatureIDE configurator without (P1-P5) and with (P6-P10) the proposed visualization components.

From the average values of Table 7.2, we can see that the gaps between these two approaches are large for decision makers of this product line. The users of our approach have advantages in terms of the accuracy of the final complete configuration, the number of decisions to make, the number of rollbacks, and the confidence level. The recall for users of our approach is higher than for users of the traditional configuration approach (due the low recall of $P2$ and $P3$), which suggest that the users of our approach will be more confident about the fitness of the final product (see confidence column). Although by using our approach the users took much longer to finalize the configuration process, they are more confident and consequently they performed a relative minor number of rollbacks. Therefore, we believe that the time consumed was due the users' understanding of the features' and NFPs' dependencies.

In addition, we ask the participants the main difficulties they faced using the configurator. This analyzes the problems that participants may have to carry out the configuration task by using both tool supports. On the one hand, the five main concerns reported by the participants P1-5 were: *(i)* I was not sure where to start (P3); *(ii)* the relationships among features are not explicitly clear in the configuration view, as it is in the feature model editor view (P1 and P4); *(iii)* the automatic (de)selection applied to validate the configuration makes me lose control of the consequence of my selections (P2 and P3); *(vi)* I need support to reason over competitive features (P4); and *(v)* I cannot track the non-functional requirements (P1, P2, P3, and P5). However, the positive point was unanimous about the easy process to (de)select a feature through check boxes. Moreover, some of the participants (P1, P4, and P5) mentioned that the automatic (de)selection of dependent features is a favorable point. On the other hand, the five main concerns reported by the participants P6-10 were: *(i)* even with the training session in the beginning, some terms in the legend (*e.g.*, formalized and inferred) were somehow confusing during the task (P7 and P8); *(ii)* the graph view is showing in new windows, instead of in the configuration view next to the focused view (P6 and P8); *(iii)* I cannot click on the name of the

feature in the configuration view for the visualization graph. I have aways to go to the menu to select the visualization view (P6); *(vi)* I cannot (de)select the features by checking it on the graph (P9 and P10); and *(v)* I do not like the blue color in the graph (P10). However, all participants mentioned as positive points the focused 5-star view and the filter mechanism. In addition, they also cited as positive points the features' relation types view (P6, P9, and P10), the effect of NFPs associated to each feature (P7 and P8), comparison among features' NFP values (P6, P7, P8, and P10), and different tones of colors for the zones of the graph (P10).

7.3.2 Approach Scalability

We investigate whether the implemented visualization components are scalable for large and complex product lines. We present experimental results for 11 product lines acquired from existing publications in the product line community [Pereira et al., 2017]. The characteristics of the product lines are described in Table 7.1. Each feature of each feature model (except the Dell laptop) has been annotated with 6 NFPs: `cost`, `response time`, `performance`, `security`, `reliability`, and `maintainability`. The values for quantitative NFPs have been set randomly with a uniform distribution: `cost` takes real values between $0 and $500; `response time` takes integer values between 0ms and 60,000ms; `performance` takes percentage values between 0% and 100%. Finally, the NFPs `security`, `reliability`, and `maintainability` take the following qualitative values [*HighN*, *MedN*, *LowN*, *LowP*, *MedP*, *HighP*] introduced for Asadi et al. [2014]. For the Dell laptop product line, we consider 3 real-world numeric quantitative NFPs (`price`, `frequency`, and `rotation`) and 2 real-world categorical qualitative NFPs (`security` and `performance`). The values for these NFPs were specified by game domain experts. The number of NFPs for these product lines reaches a threshold of six which is the upper bound number suggested by some authors (Mairiza et al. [2010] and Sommerville and Sawyer [1997]) as the most often used for realistic systems.

To perform our experiment, we developed a computer program that randomly chose a feature to be analyzed and consequently selected. The program repeats the process until obtaining a complete configuration. Since running it for the set of all different valid sequences of feature selections is NP-hard [White et al., 2009], we performed this experiment by taking into account a set of 1,000 runs. Each configuration was created from scratch and no filter to stakeholders' requirements was applied. For each feature being analyzed, we collected the number of relevant features plus the number of NFPs being displayed by the graphs and we record the worst case scenario (*i.e.*, with higher information overload). The *Graph Data* column of Table 7.1 presents the experiment results for each product line.

In the worst case, the graph showed up 56 features and 6 NFPs (*i.e.*, 62 nodes) for the *Battle of Tanks* product line. Although this product line is not the largest, it has few mandatory features and a small height of the feature tree. Consequently, a parent feature has many dependent sub-features and, when they are selected, there are many required features (due the XOR relation). To have insights about the scalability of the resultant visualization, we showed the generated graph with the worst case scenario to the ten participants of our user study (Section 7.3.1) in order to find out if they have any trouble understanding the graph. All participants were able

to completely explain the information in the graph. In summary, they described that once the relevant information are in the extreme of the graph (*i.e.*, higher prediction and positive NFP values) and represented for different notations (*i.e.*, ● , ● , ◉ , and ◉), they can easily focus on a reduced amount of data and quickly decide which feature would be more relevant to them. However, they complained mainly about the large name of some features and the feature positions in each zone. Overall, although there are some points to be improved in the future regarding usability, our approach was applicable for this worst-case scenario regarding scalability.

7.3.3 Approach Performance

We evaluate the performance of our approach by measuring the response time of the tool for generating the visualization graphs. Displaying a feature's graph is an algorithm with order $\mathcal{O}(nm)$ where n is the number of features related to the feature in the center (*i.e.*, f_c) plus the unselected features showing up in the focused view, and m is the number of NFPs related to f_c. Moreover, displaying the NFP's graph is also an algorithm with order $\mathcal{O}(nm)$, here n is the number of unselected features showing up in the focused view, and m is the number of NFPs (except the one in the center). However, beforehand we need to run a CNF algorithm to build the functional features perspective which is an NP-hard problem. As a consequence, this process can negatively affect the user experience as visual continuity is not reached. We capture the response time during the scalability experiment presented in Section 7.3.2. For each feature being analyzed, we record the highest time spent for the configurator to generate the visualizations (see *Time* column of Table 7.1). In the worst case, we have around 8 seconds for the *Web Portal* and *e-Shop* product lines. These were worst cases however further efforts can be done to improve the current implementation regarding performance.

7.4 Threats to Validity

In this section, we discuss the two main groups of common validity threats: *internal validity* and *external validity*.

Regarding the *internal validity*, we identify the characteristics of the Dell laptop product line used for evaluating the efficiency and the number of participants as two relevant factors (Section 7.3.1). We used a publicly available product line and we have investigated and created suitable NFPs based on experts opinion and general characteristics of NFPs found in the literature [Commission et al., 2001, Mairiza et al., 2010]. In addition, we discussed the experiment design with experienced researchers in the product line field. However, additional experiments are required to determine the impact in other scenarios.

Another limitation of this study concerns to the focused view used by the proposed graphs. On the one hand, the focused view reduces the information density. On the other hand, it might cause that the user will lose the "global view". To minimize this validity threat, we allow the user to expand the focused view in the 5-star perspective but we agree that loosing the global view in the graph perspectives might be problematic. Also, the way that the features are filtered (child features) might cause that the effectiveness of our approach depend on the topology of the

target feature model. To simulate the practical configuration task while measuring the scalability and performance of our approach (Section 7.3.2 and Section 7.3.3 respectively), the set of selected features were chosen randomly from the whole set of relevant features. We carried out 1,000 runs to ensure significance of the results of this stochastic process and we reported the worst scenario to prove its feasibility.

Regarding *external validity*, our experiments to check the scalability rely on eleven large and medium size real-word product lines with different types of features and structures and from a set of diverse domains (Table 7.1). However, the use of other product lines with different characteristics could have impacted our results. We try to minimize this validity threat by documenting the characteristics of the product lines[4]. Still, conducting experiments with additional product lines with realistic NFPs remains part of our future work.

Our approach is designed to work effectively when a large dataset of realistic previous configurations is available. This is a requirement for our recommender system that allows us to compute a set of more reliable feature scores and build the visualizations. We also use this dataset to compute NFPs' interdependencies. However, in this work, we assume the use of state-of-the-art approaches to compute NFP values resultants of the interaction of a valid set of features (*i.e.*, our focus is in the visualizations). We are aware that measuring NFPs might influence the scalability and time performance of our approach. This, however, goes beyond the scope of this thesis.

7.5 Related Work

Visualization reduces the complexity of comprehension tasks and helps to get insights and make decisions on a tackled problem [Card et al., 1999a]. Visualization in SPL engineering is a relevant challenge [Bashroush et al., 2017, Berger et al., 2013] and therefore it has been studied by the research community to support several tasks beyond configuration (*e.g.*, SPL testing [Lopez-Herrejon and Egyed, 2013] or traceability between features and implementation assets [Kästner et al., 2008]). In this chapter, we focus on visualizations to guide the configuration process [Nestor et al., 2008a] and we acknowledge several related works in this field.

Czarnecki et al. [2008], as part of their work in probabilistic feature models, proposed a visualization for recommending features during the interactive configuration of a product based on existing configurations. The visualization represents a score associated to each feature. Regarding scores, Chapter 4, Chapter 5, and Chapter 6 studied different algorithms to be used as scores for recommender systems in SPL configuration. We took these algorithms and we extended the visualization aspects which was very limited in Czarnecki et al. work. In addition, both works did not consider NFPs as part of the configuration process.

Martinez et al. [2014] presented FRoGs (*Feature Relations Graphs*), a visualization paradigm based on radial ego networks that inspired two of our visualizations. Their work uses a simple algorithm for calculating the score based on the co-occurrence of the features, they did not consider NFPs and we extended it to show other relevant

[4]The complete representation (*xml* files) of all product lines can be found at http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/

information. In addition, several works on visualization during configuration suggest the importance of reducing the information density and using techniques to bring the attention to the user to the relevant features (*e.g.*, incremental browsing and increasing the size in Botterweck et al. [2008] or manually introducing visibility conditions in Dhungana et al. [2011]). We also follow these principles in our approach while this was not present in Martinez et al. work.

Schneeweiss and Botterweck [2010] presented *Feature Flow Maps* which focused on NFPs. This visualization helps to comprehend the contribution of each selected feature in the global value of an NFP. However, the positive or negative impact between features is not visualized. This visualization can be complementary to ours. Bagheri and Ensan [2014a] proposed to use colors inside the feature diagram to encourage or discourage features based on an interactive gambling process. The NFPs are considered by showing the interdependencies among them and the impact of each feature in the NFPs. We propose a set of visualization paradigms that are alternatives to this work but both can be complementary as well.

Several authors proposed automatic approaches to automatically configure a product optimizing NFPs [Ochoa et al., 2018, 2017]. These approaches often provide a set of possible solutions due to conflicting NFPs. Murashkin et al. [2013] proposed a visualization to help the users to select a solution among the ones automatically found. Contrary to this, our visualization focuses on staged and gradual configuration processes to support the interactive selection of features. In addition, our approach can be combined with Murashkin et al. approach in order to allow users to see further information about features and NFPs for the set of allowed solutions. Other approaches, such as Temple et al. [2017], narrow the configuration space by automatically discovering extra constraints related to specific requirements. This can be also complementary to our approach.

Finally, a series of tools present support to the SPL configuration process [Pereira et al., 2015]. Similarly to FeatureIDE tool, these tools follow managing variability in SPLs. Antkiewicz and Czarnecki [2004] and Mendonça et al. [2009] introduce a manual configurator to support users removing variability from feature models. In both tools, feature models are visualized as a feature-tree structure and cross-tree constraints textually as logical expressions between features. Although SPLOT [Mendonça et al., 2009] provides constraints information among features to the user after each decision made during the configuration process, one common capability missing in these tools is the use of visualization and interaction techniques to support the manual configuration process before each decision. In order to address this issue, VISIT-FC (*Visual and Interactive Tool for Feature Configuration*) [Nestor et al., 2008b] and FAMA [Trinidad et al., 2008] tools introduced visualization techniques of feature models considering dependences graphically as arrows between features. However, these tools display to the user all features, relationships and cross-tree constraints included in the model. As discussed in Section 7.1, real-world feature models can be large and complex with many features, relationships and constraints. For larger-scale problems, these tools make the configuration process a cumbersome, time-consuming, error-prone and tedious task. Current tools still lack adequate mechanisms to visualize variability model, thus making variability harder to manage. In contrast, our approach follows showing only the relevant features and

NFPs to the user, instead of the huge amount of information presented by the other tools. It supports users to visualize in an interactive way the conflicting features and guide them through of a hierarchical and graphical exploration of the model while predicting features and NFPs relevance. Therefore, our approach performs well on these large-scale problems and makes use of visualization techniques to help users interactively configuring a product.

7.6 Summary

In this chapter, we have proposed and evaluated a set of interrelated visualizations to improve the efficiency of decision makers to configure a product. We extended the state-of-the-art configurator FeatureIDE with our approach. The proposed approach ensures the consistency of the configured products. In addition, it reduces the configuration effort and complexity of decision making by providing a restricted view of the configuration space and by dynamically assisting the decision makers to reasoning on a focused set of relevant information about features and NFPs. We conducted a set of numerous experiments with eleven state-of-the-art product lines to evaluate three important practical characteristics of our approach: *effectiveness*, *scalability*, and *performance*. Our experimental results show that the proposed set of visualizations are useful as: *(i)* it is able to improve the effectiveness of the configuration process in three perspectives: the accuracy of the configuration, the number of decisions to take, and the confidence level of decision makers; and *(ii)* it is scalable for state-of-the-art product lines. We assume that most real-world problems will be of similar scale to our experiments.

8. Conclusion and Future Work

Mass customization of standardized products has become a trend to succeed in today's market environment. *Software Product Lines* (SPLs) address this trend by describing a family of software products that share a common set of features. Different combinations of predefined features enable tailoring the product to fit the needs of each customer. These needs are related to functional properties of the system (optional features) as well as non-functional properties (*e.g.*, performance or cost of the final product). However, choosing the appropriate set of features that matches the customer's needs is hampered due to the complexity of variability constraints and the limitation of tool support to check functional properties interdependencies. In addition, the importance of non-functional properties as relevant drivers during configuration has been overlooked. In this context, recommender systems can support decision makers by filtering the number of configurations and suggesting a suitable set of features in accordance with the stakeholders' needs. In the following, we conclude our thesis that centers on the use of recommendation techniques and briefly discuss potential future work on SPL configuration.

8.1 Conclusion

This work involved four phases. First, we conducted a *Systematic Literature Review* (SLR) on the SPL configuration domain to increase the understanding of the fundamental research issues in this field (Chapter 3). Second, to overcome a set of SPL configuration issues found in the literature, we introduced the adoption of a collaborative-based personalized recommender system for SPL configuration (Chapter 4, Chapter 5, and Chapter 6). To this end, we presented an initial formalization of seven recommender algorithms to the SPL configuration context: *Neighbourhood-Based Collaborative Filtering (CF)*, *CF-Hoeffding*, *CF-Shrinkage*, *CF-Significance Weighting*, *Average Similarity*, *Matrix Factorization*, and *Tensor Factorization*. Third, the algorithms were validated against several real-world and state-of-the-art SPLs. To draw conclusions from the algorithms' effectiveness, we compare them with a baseline random recommender algorithm. The preliminary experiments show that four of the seven proposed recommendation algorithms (*i.e.*, *CF-Shrinkage*,

CF-Significance Weighting, *Matrix Factorization*, and *Tensor Factorization*) clearly and consistently outperform the baseline recommender in finding relevant features. For non-extended SPLs, it has a good performance already at the initial stage of the configuration process (*i.e.*, with just 10% of selected features), while for extended SPLs it presents good performance without the preliminary selection of features. Fourth, based on insights from previous user empirical studies [Constantino et al., 2016, Pereira et al., 2013] and the SLR presented in Chapter 3, we extended an established state-of-the-art configurator with our recommender system (Chapter 7). In addition, we proposed a set of interactive and automatic visual mechanisms that are intended to support users selecting among a vast number of features, avoiding useless and invalid decisions. In contrast to the current literature, the proposed approach benefits from a simplified view of the configuration space by dynamically predicting the importance of the features. Moreover, this is the first approach that uses a collaborative-based recommender system that learns about the relevant features from previous users configurations.

8.2 Open Research Directions

One of the main outcome of our SLR in Chapter 3 is the identification of new areas of research that can lead to further enrichment of the SPL configuration field. In this section, we present the short-term goal which considers the open challenges related to the chapters of this thesis.

Direction 1: Use of other recommendation algorithms

As a next step, we will extend the proposed approaches to consider other well-established state-of-the-art recommendation algorithms. Moreover, we plan to adapt these algorithms to consider extra variables (*e.g.*, selection order of features, market domain similarity, date of feature creation, and others). For instance, we may use the assigment of *Feature Binding Time* from Bürdek et al. [2014] and time-based discounting of weights to account for drift in stakeholders' preferences over the order of feature (de)selections. Also, we aim at working with the set of configuration constraints defined in Section 3.4 and explore the concept of optimization of reuse created in Section 3.4.4.2. We plan therefore provide an integrated environment by combining state-of-the-art automatic configuration approaches (*e.g.*, EA-based techniques) with recommender algorithms to support the set of defined configuration constraints. Then, we intend to compare our approach with state-of-the-art optimization approaches. We believe that the combination of various algorithms may outperform single methods in terms of accuracy and performance. Finally, we plan to investigate how the different configuration constraints can influence the accuracy and performance results from proposed algorithms.

Direction 2: Use of other dataset of configurations

Since there are no other publicly available dataset with real configurations and obtaining them from companies is problematic, we could not test our approach on further datasets. However, as future work, we plan to search for other real-world

datasets of configurations (*e.g.*, by establish partnership with companies). By testing our approach on other systems, we can investigate how the diversity of application scenarios and the number of previous configurations affect the quality of the recommendations (*i.e.*, at which point they start to be useful). Moreover, since recommender algorithms are frequently intended to work on very large datasets of configurations, we also aim at analyzing the impact of the proposed algorithms on configuration performance. From the company perspective, the recommender algorithms must be able to handle a large number of features ensuring a short response time for delivering recommendation results to ensure a good user experience.

Direction 3: Tool support and empirical user study

We aim at extending PROFilE[1] with new recommender algorithms and proposing new visualization techniques to complement the ones proposed in Chapter 7. Moreover, we plan to perform an user controlled study of PROFilE to investigate the user's satisfaction with the recommendations. Also, we aim at comparing PROFilE with others configurators in the literature to investigate the gain in terms of acceptance and usability of the proposed interactive configuration processes. We hope to conduct the experiments in various settings, *i.e.* in industrial and academic contexts, in different application domains, and with various kinds of participants to have different degrees of expertise.

Direction 4: Context-aware SPL configuration process

We intend to explore various types of statistical tests to identify which of the contextual NFPs and configuration constraints are truly significant in the sense that they indeed affect the recommendations. Here, we also aim at considering the historical order of features (de)selection as a contextual variable by assuming that different types of actions may give rise to and call for different types of relevant features, thus assuming a bidirectional relationship between (de)selection of features and underlying contexts. Moreover, as most proposals for dynamic SPL configuration at runtime neglect model evolution, we also plan to investigate the use of recommendation techniques in this domain. Finally, we plan to evaluate our approach under the use of different recommendation algorithms on self-adaptive systems.

Direction 5: Systematic literature review

As future work, we aim at extending the SLR presented in Chapter 3 by considering the coming years and new proposed approaches in this field. Also, we plan to conduct an empirical study with experts in the SPL configuration field in order to rank the importance of the new findings.

[1] http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/

A. Appendix

A.1 Papers Venues

This appendix shows the complete list of considered publication venues referring to *journals* (see Table A.1), *conferences* (see Table A.2), *workshops* and *symposiums* (see Table A.1). For each publication venue, we presented the *venue acronym* and the set of published *primary studies*.

A.2 Studies Grouped by Contribution

Table A.4 shows 157 retrieved primary studies from our systematic literature review. Notice that we grouped similar contributions and we refer to each group in Section 3.4 by using a single citation from the most recent published paper.

A.3 Supported Non-Functional Properties

Table A.4 presents the set of NFPs supported by each study. Notice that most of the studies do not clearly specify which NFPs they supported.

Table A.4: Studies grouped per contribution.

ID	Study
S1	Adjoyan and Seriai [2015, 2017]
S2	Gençay et al. [2017]
S3	Guedes et al. [2017]
S4	Horcas et al. [2017], Munoz [2017], Munoz et al. [2017]
S5	Khoshnevis and Shams [2017]
S6	Kifetew et al. [2017]
S7	Nieke et al. [2017]
S8	Noorian et al. [2014, 2016, 2017]
S9	Oh et al. [2017]
S10	Machado et al. [2014a], Pereira et al. [2017]

Table A.4: Studies grouped per contribution.

ID	Study
S11	Safdar et al. [2017]
S12	Umpfenbach et al. [2017]
S13	Wang et al. [2017]
S14	Zheng et al. [2017b]
S15	Zheng et al. [2017a]
S16	Ayala et al. [2016]
S17	Ter Beek et al. [2013], ter Beek et al. [2015a,b], Ter Beek et al. [2016]
S18	Bąk et al. [2016], Murashkin et al. [2013], Olaechea et al. [2012]
S19	Camacho et al. [2016]
S20	Eichelberger et al. [2014, 2016], Eichelberger and Schmid [2015], El-Sharkawy et al. [2015], Schmid and Eichelberger [2015]
S21	Hajri et al. [2016]
S22	Hierons et al. [2016a]
S23	Lu et al. [2014, 2016a,b]
S24	Mauro et al. [2016]
S25	Cruz et al. [2013], dos Santos Neto et al. [2016]
S26	Nieke et al. [2016]
S27	Noir et al. [2016]
S28	Pereira et al. [2016b], Thüm et al. [2014]
S29	Pereira et al. [2016c]
S30	Rabiser et al. [2016]
S31	Ruiz et al. [2016]
S32	Santos et al. [2016]
S33	Schwäger and Westfechtel [2016]
S34	Sharifloo et al. [2016]
S35	Sion et al. [2016]
S36	Tanhaei et al. [2016a,b]
S37	Triando et al. [2016]
S38	Winkelmann et al. [2016]
S39	Tan et al. [2015b], Xue et al. [2016]
S40	Villela et al. [2012], Zanardini et al. [2016]
S41	Zhang and Becker [2016]
S42	Bajaras and Agard [2015]
S43	Benali et al. [2015]
S44	Brink et al. [2015, 2012]
S45	Fang et al. [2015]
S46	Dhungana et al. [2013], Galindo et al. [2015b], Rabiser et al. [2012b]
S47	Henard et al. [2015a]
S48	Leite et al. [2015]
S49	Lian and Zhang [2015a]
S50	Martinez et al. [2015a]
S51	Mazo et al. [2015], Sawyer et al. [2012]
S52	Myllärniemi et al. [2015]
S53	Ochoa et al. [2015]
S54	Pascual et al. [2015a, 2013]
S55	Rezapour et al. [2015]
S56	Siegmund et al. [2015, 2012a,b,c]
S57	Soares et al. [2015]
S58	Valov et al. [2015]
S59	Svee and Zdravkovic [2015], Zdravkovic et al. [2015]
S60	Asadi et al. [2014], Soltani et al. [2012]
S61	Bagheri and Ensan [2014b], Bashari et al. [2014]
S62	Behjati et al. [2014, 2012, 2013]

Table A.4: Studies grouped per contribution.

ID	Study
S63	Bürdek et al. [2014]
S64	Bures et al. [2014]
S65	Chavarriaga et al. [2014]
S66	Foster et al. [2014]
S67	Guo et al. [2014]
S68	Lin and Kremer [2014], Lin and Okudan [2012]
S69	Mazo et al. [2014a]
S70	Murguzur et al. [2014]
S71	Murwantara et al. [2014], Murwantara and Bordbar [2014]
S72	Martinez et al. [2014]
S73	Olaechea et al. [2014]
S74	Sánchez et al. [2014], Sanchez et al. [2013]
S75	Tan et al. [2014b]
S76	Urli et al. [2014]
S77	Wang and Pang [2014]
S78	White et al. [2014]
S79	Zhang et al. [2014]
S80	Acher et al. [2013]
S81	Chen et al. [2013]
S82	Cubo et al. [2013]
S83	Gamez and Fuentes [2013]
S84	Ge et al. [2013]
S85	Jannach and Zanker [2013]
S86	Karimpour and Ruhe [2013]
S87	Holl et al. [2012a, 2013, 2012b], Klambauer et al. [2013]
S88	Kolesnikov et al. [2013]
S89	Kramer et al. [2013]
S90	Lee [2013]
S91	Saller et al. [2013]
S92	Sayyad et al. [2013a,b,c]
S93	Tan et al. [2013]
S94	Bagheri et al. [2012a]
S95	Roos-Frantz et al. [2012]
S96	Heider et al. [2012]
S97	Lettner et al. [2012]
S98	Mazo et al. [2012b,c]
S99	Mitchell [2012]
S100	Mussbacher et al. [2012]
S101	Neves et al. [2012]
S102	Nöhrer et al. [2012a,b]
S103	Ognjanovic et al. [2012]
S104	Ostrosi et al. [2012]
S105	Parra et al. [2012]
S106	Pleuss and Botterweck [2012]
S107	Qin and Wei [2012]
S108	Schroeter et al. [2012]
S109	Thurimella and Bruegge [2012]
S110	Wang and Ng [2012]
S111	Wittern et al. [2012]
S112	Zhao et al. [2012]

Table A.1: List of journal publications.

Acronym	Primary study	Venue
-	Winkelmann et al. [2016]	Central Europe (CEUR) Workshop Proceedings
-	Khoshnevis and Shams [2017]	Frontiers of Computer Science
-	Gençay et al. [2017], Wang et al. [2017]	Journal of Intelligent Manufacturing
-	Noorian et al. [2017]	Journal of Software: Evolution and Process
-	Zheng et al. [2017b]	The Int. Journal of Advanced Manufacturing Technology
-	Rezapour et al. [2015]	Transportation Research Part E: Logistics and Transportation Review
ASC	dos Santos Neto et al. [2016], Xue et al. [2016]	Journal Applied Soft Computing
CDSPL	Sawyer et al. [2012]	IEEE Computer Dynamic Software Product Lines
CERA	Wang and Ng [2012]	Concurrent Engineering Research and Applications
CLOUD	Leite et al. [2015]	Int. Conference on Cloud Computing
IJIDeM	Bajaras and Agard [2015]	Int. Journal on Interactive Design and Manufacturing
IJKSS	Bashari et al. [2014]	Int. Journal of Knowledge and Systems Science
IST	Asadi et al. [2014], Behjati et al. [2013], Galindo et al. [2015b], Gamez and Fuentes [2013], Lu et al. [2016a], Tanhaei et al. [2016a], Thurimella and Bruegge [2012]	Information and Software Technology
ITOR	Pereira et al. [2017]	Int. Transactions in Operational Research
JCI	Lin and Kremer [2014]	Journal of Computers in Industry
JCST	Tanhaei et al. [2016b]	Journal of Computer Science and Technology
JEO	Foster et al. [2014]	Journal of Engineering Optimization
JIM	Ostrosi et al. [2012]	Journal of Intelligent Manufacturing
JISMD	Mazo et al. [2012c]	Int. Journal of Information System Modeling and Design
JLAMP	Camacho et al. [2016], Zanardini et al. [2016]	Journal of Logical and Algebraic Methods in Programming
JMSY	Zheng et al. [2017a]	Journal of Manufacturing Systems
JORS	Umpfenbach et al. [2017]	Journal of the Operational Research Society
JRE	Bagheri and Ensan [2014b], Zdravkovic et al. [2015]	Journal of Requirements Engineering
JS	Qin and Wei [2012]	Journal of Software
JSEKE	Noorian et al. [2014]	Int. Journal of Software Engineering and Knowledge Engineering
JSEP	Bagheri et al. [2012a]	Journal of Software: Evolution and Process
JSJU	Wang and Pang [2014]	Journal of Shanghai Jiaotong University (Science)
JSS	Pascual et al. [2015a], White et al. [2014]	Journal of Systems and Software
JSTTT	Pleuss and Botterweck [2012]	Int. Journal on Software Tools for Technology Transfer
RSSE	Mazo et al. [2014a]	Recommendation Systems in Software Engineering
SCP	Acher et al. [2013], Thüm et al. [2014]	Science of Computer Programming
SoSyM	Rabiser et al. [2016]	Software and Systems Modeling
SQJ	Mussbacher et al. [2012], Roos-Frantz et al. [2012], Siegmund et al. [2012c], Zhang et al. [2014]	Software Quality Journal
SSM	Bąk et al. [2016]	Software & Systems Modeling
TKDE	Jannach and Zanker [2013]	IEEE Transactions on Knowledge and Data Engineering
TOSEM	Behjati et al. [2014], Hierons et al. [2016a]	ACM Transactions on Software Engineering and Methodology

Acronym	Primary study	Venue
ACSC	Tan et al. [2013]	Australasian Computer Science Conference
APSEC	Holl et al. [2012a, 2013], Tan et al. [2014b]	Asia-Pacific Software Engineering Conference
ASE	Guo et al. [2014], Rabiser et al. [2012b], Sayyad et al. [2013b], Schwäger and Westfechtel [2016]	Int. Conference on Automated Software Engineering
ASME	Lin and Okudan [2012]	Int. Design Engineering Technical Conferences
BDCLOUD	Murwantara and Bordbar [2014]	Int. Conference on Big Data and Cloud Computing
BIR	Svee and Zdravkovic [2015]	Int. Conference on Perspectives in Business Informatics Research
CAiSE	Mazo et al. [2012b]	Int. Conference on Advanced Information Systems Engineering
CBSoft	Machado et al. [2014a]	Brazilian Congress on Software
CDVE	Zhao et al. [2012]	Int. Conference on Cooperative Design, Visualization, and Engineering
CEC	Cruz et al. [2013]	Congress on Evolutionary Computation
CloudTech	Benali et al. [2015]	Int. Conference on Cloud Technologies and Applications
ECMFA	Behjati et al. [2012]	European Conference on Modelling Foundations and Applications
ECSA	Munoz et al. [2017]	European Conference on Software Architecture: Companion Proceedings
EuSEC	Mitchell [2012]	European Systems Engineering Conference
GECCO	Martinez et al. [2015a], Safdar et al. [2017]	Genetic and Evolutionary Computation Conference
GPCE	Kramer et al. [2013], Neves et al. [2012], Pereira et al. [2016c]	Int. Conference on Generative Programming: Concepts and Experiences
ICACSIS	Triando et al. [2016]	Int. Conference on Advanced Computer Science and Information Systems
ICMIC	Ge et al. [2013]	Int. Conference on Measurement, Information and Control
ICSE	Henard et al. [2015a], Sayyad et al. [2013c], Siegmund et al. [2012a]	Int. Conference on Software Engineering
ICSESS	Chen et al. [2013]	Int. Conference on Software Engineering and Service Sciences
ICSOC	Wittern et al. [2012]	Int. Conference on Service-Oriented Computing
ICSR	Cubo et al. [2013], Pereira et al. [2016b]	Int. Conference on Software Reuse
ICST	Lu et al. [2016b]	Int. Conference on Software Testing, Verification and Validation
iiWAS	Murwantara et al. [2014]	Int. Conference on Inf. Integration and Web-Based Applications and Services
MODELS	Chavarriaga et al. [2014]	Int. Conference on Model Driven Engineering Languages and Systems
PAAMS	Ayala et al. [2016]	Int. Conference on Practical Applications of Agents and Multi-Agent Systems
PROFES	Brink et al. [2015]	Int. Conference no Product-Focused Software Process Improvement
QoSA	Myllärniemi et al. [2015]	Int. Conference on Quality of Software Architectures
REFSQ	Klambauer et al. [2013]	Int. Conference on Requirements Engineering: Foundation for Software Quality
SANER	Lian and Zhang [2015a]	Int. Conference on Software Analysis, Evolution, and Reengineering
SCC	Ognjanovic et al. [2012]	Int. Conference on Services Computing
SEKE	Adjoyan and Seriai [2015]	Int. Conference on Software Engineering and Knowledge Engineering
SLE	Ochoa et al. [2015]	Int. Conference on Software Language Engineering
SPLC	Eichelberger et al. [2014, 2016], Eichelberger and Schmid [2015], Fang et al. [2015], Heider et al. [2012], Horcas et al. [2017], Lee [2013], Lettner et al. [2012], Mazo et al. [2015], Munoz [2017], Murashkin et al. [2013], Murguzur et al. [2014], Nöhrer et al. [2012a], Noir et al. [2016], Olaechea et al. [2014], Saller et al. [2013], Schmid and Eichelberger [2015], Schroeter et al. [2012], Sion et al. [2016], Soltani et al. [2012], Ter Beek et al. [2013], ter Beek et al. [2015b], Urli et al. [2014], Valov et al. [2015], Villela et al. [2012], Zhang and Becker [2016]	Int. Systems and Software Product Line Conference
UCC	Ruiz et al. [2016]	Int. Conference on Utility and Cloud Computing
VISSOFT	Martinez et al. [2014]	Int. Working Conference on Software Visualization

Table A.2: List of conference publications.

	Acronym	Primary study	Venue
Symposium	CBSE	Bures et al. [2014]	Int. Symposium on Component-Based Software Engineering
	FSE	Hajri et al. [2016], Oh et al. [2017], Siegmund et al. [2015]	Int. Symposium on Foundations of Software Engineering
	ISoLA	Ter Beek et al. [2016]	Int. Symposium on Leveraging Applications of Formal Methods
	ISSRE	Lu et al. [2014]	Int. Symposium on Software Reliability Engineering
	ISSTA	Tan et al. [2015b]	Int. Symposium on Software Testing and Analysis
	SAC	Adjoyan and Seriai [2017], Noorian et al. [2016], Parra et al. [2012]	Symposium on Applied Computing
	SBCARS	Sánchez et al. [2014]	Brazilian Symposium on Software Components, Architectures and Reuse
	SBES	Guedes et al. [2017], Santos et al. [2016]	Brazilian Symposium on Software Engineering
	SEAMS	Pascual et al. [2013], Sharifloo et al. [2016]	Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems
	SSBSE	Kifetew et al. [2017]	Int. Symposium on Search-Based Software Engineering
Workshop	CMSBSE	Karimpour and Ruhe [2013], Sanchez et al. [2013], Sayyad et al. [2013a]	Int. Workshop on Combining Modelling and Search-Based Software Engineering
	ETX	El-Sharkawy et al. [2015]	Eclipse Technology eXchange
	FMSPLE	ter Beek et al. [2015a]	Int. Workshop on Formal Methods and Analysis in Software Product Line Engineering
	NFPinDSML	Olaechea et al. [2012]	Int. Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages
	SEES	Siegmund et al. [2012b]	Int. Workshop on Software Engineering for Embedded Systems
	VaMoS	Bürdek et al. [2014], Dhungana et al. [2013], Holl et al. [2012b], Kolesnikov et al. [2013], Mauro et al. [2016], Nieke et al. [2017, 2016], Nöhrer et al. [2012b], Soares et al. [2015]	Int. Workshop on Variability Modeling of Software-Intensive Systems
	VariComp	Brink et al. [2012]	Int. Workshop on Variability & Composition

Table A.3: List of symposium and workshop publications.

Study	Non-Functional Properties
Guedes et al.	battery, network, stock, weather
Horcas et al.	energy
Khoshnevis and Shams	cohesion, coupling, granularity, business entity convergence
Kifetew et al.	lines of code, cyclomatic complexity, test assertions, number of installations, number of developers, number of changes, number of faults
Nieke et al.	Location
Noorian et al.	satisfaction, cost, security, ease to use, preformance, safety
Pereira et al.	cost, degree of preference
Umpfenbach et al.	volume, costs, profit
Zheng et al.	delivery time, service quality, product quality
Ter Beek et al.	cost, weight, load
Camacho et al., Qin and Wei	cost
Mauro et al.	max speed, road, location
dos Santos Neto et al.	cost, suitability
Noir et al.	security, safety, CPU response, lifetime, overhead, maintenance, cost, power consumption
Ruiz et al.	cost, CPU, memory, disk
Santos et al.	speed, road, location
Henard et al., Lian and Zhang, Sayyad et al., Xue et al.	cost, used before, defects
Zanardini et al.	memory consumption, number of instructions
Benali et al.	installation time
Leite et al.	vCPU, memory, cost
Myllärniemi et al.	security
Ochoa et al.	costs, human resources, time
Pascual et al.	usability, battery consumption, memory footprint
Rezapour et al.	quality, price
Siegmund et al.	reliability, complexity, footprint, performance
Soares et al.	footprint, performance, memory consumption, usability
Valov et al.	performance
Asadi et al.	cost, security, performance, reliability, ease of use
Bashari et al.	time-to-market, development cost, customer satisfaction, security, usability, reliability, performance, supportability
Bürdek et al.	binding time
Guo et al.	cost, reliability, battery usage, response time, ramp-up time, development time, deployment time
Mazo et al.	performance, reuse, cost
Murwantara et al.	energy consumption
Olaechea et al.	reliability, security, footprint, mass, battery, response time, ramp-up time, cost, deployment time, prior usage, defects
Sánchez et al.	required memory, reconfiguration time, response time, accuracy, availability, security
Tan et al.	ease of use
Wang and Pang	performance, money, development time
Zhang et al.	cost, performance, security, usability
Bagheri et al.	speed, performance
Roos-Frantz et al.	accuracy, memory, ROM, range, latency, cost, cycle
Lettner et al.	profit, break even
Mussbacher et al.	importance
Ognjanovic et al.	cost, response time
Parra et al.	quality, response time
Wittern et al.	cost, storange capacity
Zhao et al.	exhaust, exhaust pressure, rated power, noise, weight, fuel consumption, stability

Table A.5: NFPs supported by each approach.

Bibliography

Acher, M., Collet, P., Lahire, P., and France, R. B. (2013). Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming (SCP)*, 78(6):657–681. (cited on Page 26, 38, 43, 45, 61, 62, 173, and 174)

Adjoyan, S. and Seriai, A.-D. (2015). An architecture description language for dynamic service-oriented product lines. In *International Conference on Software Engineering and Knowledge Engineering (SEKE)*. (cited on Page 171 and 175)

Adjoyan, S. and Seriai, A.-D. (2017). Reconfigurable service-based architecture based on variability description. In *ACM Symposium on Applied Computing (SAC)*, pages 1154–1161. ACM. (cited on Page 26, 38, 61, 64, 171, and 176)

Adomavicius, G. and Tuzhilin, A. (2011). Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer. (cited on Page 105, 109, 110, and 111)

Afzal, U., Mahmood, T., and Shaikh, Z. (2016). Intelligent software product line configurations: A literature review. *Computer Standards & Interfaces*, 48:30–48. (cited on Page 72 and 142)

Alférez, G. H., Pelechano, V., Mazo, R., Salinesi, C., and Diaz, D. (2014). Dynamic adaptation of service compositions with variability models. *Journal of Systems and Software (JSS)*, 91:24–47. (cited on Page 142)

Alférez, M., Santos, J. P., Moreira, A., Garcia, A., Kulesza, U., Araújo, J., and Amaral, V. (2009). Multi-view composition language for software product line requirements. In *International Conference on Software Language Engineering (SLE)*, pages 136–154. ACM. (cited on Page 159)

Almeida, A., Cavalcante, E., Batista, T., Cacho, N., and Lopes, F. (2014). A component-based adaptation approach for multi-cloud applications. In *International Conference on Computer Communications (INFOCOM)*, pages 49–54. IEEE. (cited on Page 142)

Amatriain, X. and Pujol, J. M. (2015). Data mining methods for recommender systems. In *Recommender Systems Handbook*, pages 227–262. Springer. (cited on Page 109, 110, and 112)

Antkiewicz, M. and Czarnecki, K. (2004). FeaturePlugin: feature modeling plug-in for eclipse. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, pages 67–72. ACM. (cited on Page 98, 100, 147, and 164)

Aranega, V., Etien, A., and Mosser, S. (2012). Using feature model to build model transformation chains. In *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 562–578. Springer. (cited on Page 159)

Asadi, M., Soltani, S., Gašević, D., and Hatala, M. (2016). The effects of visualization and interaction techniques on feature model configuration. *Empirical Software Engineering*, 21(4):1706–1743. (cited on Page 69 and 72)

Asadi, M., Soltani, S., Gasevic, D., Hatala, M., and Bagheri, E. (2014). Toward automated feature model configuration with optimizing non-functional requirements. *Information and Software Technology (IST)*, 56(9):1144–1165. (cited on Page 2, 25, 32, 37, 38, 39, 45, 52, 59, 69, 111, 156, 161, 172, 174, and 177)

Ayala, I., Amor, M., and Fuentes, L. (2016). Using spl to develop aal systems based on self-adaptive agents. In *International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*, pages 263–275. Springer. (cited on Page 26, 38, 61, 64, 172, and 175)

Bagheri, E., Asadi, M., Gasevic, D., and Soltani, S. (2010a). Stratified analytic hierarchy process: prioritization and selection of software features. In *International Systems and Software Product Line Conference (SPLC)*, pages 300–315. Springer. (cited on Page 75, 98, and 99)

Bagheri, E., Di Noia, T., Ragone, A., and Gasevic, D. (2010b). Configuring software product line feature models based on stakeholders' soft and hard requirements. In *International Systems and Software Product Line Conference (SPLC)*, pages 16–31. Springer. (cited on Page 75, 98, 99, and 142)

Bagheri, E. and Ensan, F. (2014a). Dynamic decision models for staged software product line configuration. *Requirements Engineering Journal (REJ)*, 19(2):187–212. (cited on Page 2, 75, 98, 99, 108, 121, and 164)

Bagheri, E. and Ensan, F. (2014b). Dynamic decision models for staged software product line configuration. *Requirements Engineering Journal (REJ)*, 19(2):187–212. (cited on Page 41, 172, and 174)

Bagheri, E., Noia, T. D., Gasevic, D., and Ragone, A. (2012a). Formalizing interactive staged feature model configuration. *Journal of Software: Evolution and Process (JSEP)*, 24(4):375–400. (cited on Page 25, 32, 37, 38, 39, 45, 50, 52, 55, 58, 173, 174, and 177)

Bagheri, E., Noia, T. D., Gasevic, D., and Ragone, A. (2012b). Formalizing interactive staged feature model configuration. *Journal of Software: Evolution and Process (JSEP)*, 24(4):375–400. (cited on Page 75, 98, 99, and 142)

Bajaras, M. and Agard, B. (2015). A methodology to form families of products by applying fuzzy logic. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 9(4):253–267. (cited on Page 26, 37, 38, 39, 172, and 174)

Bąk, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., and Wąsowski, A. (2016). Clafer: unifying class and feature modeling. *Software & Systems Modeling (SSM)*, 15(3):811–845. (cited on Page 25, 32, 38, 41, 43, 45, 52, 59, 60, 172, and 174)

Barbeau, M. and Bordeleau, F. (2002). A protocol stack development tool using generative programming. In *ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE)*, pages 93–109. Springer. (cited on Page 6)

Bashari, M., Noorian, M., and Bagheri, E. (2014). Product line stakeholder preference elicitation via decision processes. *International Journal of Knowledge and Systems Science (IJKSS)*, 5(4):35–51. (cited on Page 25, 32, 37, 38, 39, 45, 46, 50, 172, 174, and 177)

Bashroush, R., Garba, M., Rabiser, R., Groher, I., and Botterweck, G. (2017). Case tool support for variability management in software product lines. *ACM Computing Surveys (CSUR)*, 50(1):14:1–14:45. (cited on Page 9, 145, and 163)

Batory, D. (2005). Feature models, grammars, and propositional formulas. In *International Systems and Software Product Line Conference (SPLC)*, volume 3714, pages 7–20. Springer. (cited on Page 62 and 80)

Behjati, R., Nejati, S., and Briand, L. C. (2014). Architecture-level configuration of large-scale embedded software systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(3):25. (cited on Page 27, 45, 46, 172, and 174)

Behjati, R., Nejati, S., Yue, T., Gotlieb, A., and Briand, L. (2012). Model-based automated and guided configuration of embedded software systems. In *European Conference on Modelling Foundations and Applications (ECMFA)*, pages 226–243. Springer. (cited on Page 172 and 175)

Behjati, R., Yue, T., Briand, L., and Selic, B. (2013). Simpl: A product-line modeling methodology for families of integrated control systems. *Information and Software Technology (IST)*, 55(3):607–629. (cited on Page 172 and 174)

Bell, R., Koren, Y., and Volinsky, C. (2007). Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 95–104. ACM. (cited on Page 3 and 11)

Benali, A., El Asri, B., and Kriouile, H. (2015). A pareto-based artificial bee colony and product line for optimizing scheduling of vm on cloud computing. In *International Conference on Cloud Technologies and Applications (CloudTech)*, pages 1–7. IEEE. (cited on Page 25, 32, 38, 52, 58, 172, 175, and 177)

Benavides, D., Felfernig, A., Galindo, J. A., and Reinfrank, F. (2013). Automated analysis in feature modelling and product configuration. In *Safe and Secure Software Reuse*, pages 160–175. Springer. (cited on Page 13, 17, 18, 71, 72, and 98)

Benavides, D., Martín-Arroyo, P. T., and Cortés, A. R. (2005). Automated reasoning on feature models. In *International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 5, pages 491–503. Springer. (cited on Page 6 and 30)

Benavides, D., Segura, S., and Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later: a literature review. *Information Systems*, 35(6):615–708. (cited on Page 6, 17, 18, 71, 72, 108, 123, 149, and 151)

Benavides, D., Segura, S., Trinidad, P., and Cortés, A. R. (2007). FAMA: tooling a framework for the automated analysis of feature models. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 129–134. ACM. (cited on Page 98 and 100)

Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., and Wasowski, A. (2013). A survey of variability modeling in industrial practice. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 7:1–7:8. ACM. (cited on Page 145 and 163)

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer, New York. accessed November 15, 2017. (cited on Page 87)

Bishop, C. M. (2007). *Pattern recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition. (cited on Page 89 and 129)

Boehm, B., Brown, R., Kaspar, H., Lipow, M., McLeod, G., and Merritt, M. (1978). Characteristics of software quality. trw series of software technology, trw systems and energy. *Inc., also published by North Holland*, 1973. (cited on Page 30)

Boehm, B. W., Brown, J. R., and Lipow, M. (1976). Quantitative evaluation of software quality. In *International Conference on Software Engineering (ICSE)*, pages 592–605. IEEE Computer Society Press. (cited on Page 34)

Bosch, J., Capilla, R., and Hilliard, R. (2015). Trends in systems and software variability. *IEEE Software*, 32(3):44–51. (cited on Page 75 and 79)

Botterweck, G., Thiel, S., Nestor, D., bin Abid, S., and Cawley, C. (2008). Visual tool support for configuring and understanding software product lines. In *International Systems and Software Product Line Conference (SPLC)*, pages 77–86. IEEE. (cited on Page 164)

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., and Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software (JSS)*, 80(4):571–583. (cited on Page 16)

Brink, C., Heisig, P., and Sachweh, S. (2015). Using cross-dependencies during configuration of system families. In *International Conference no Product-Focused Software Process Improvement (PROFES)*, pages 439–452. Springer. (cited on Page 26, 45, 47, 48, 61, 62, 172, and 175)

Brink, C., Peters, M., and Sachweh, S. (2012). Configuration of mechatronic multi product lines. In *Conference of the International workshop on Variability & composition (VariComp)*, pages 7–12. ACM. (cited on Page 172 and 176)

Bürdek, J., Lity, S., Lochau, M., Berens, M., Goltz, U., and Schürr, A. (2014). Staged configuration of dynamic software product lines with complex binding time constraints. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, page 16. ACM. (cited on Page 26, 32, 61, 64, 142, 168, 173, 176, and 177)

Bures, T., Hnetynka, P., and Plasil, F. (2014). Strengthening architectures of smart cps by modeling them as runtime product-lines. In *International Symposium on Component-Based Software Engineering (CBSE)*, pages 91–96. ACM. (cited on Page 27, 61, 64, 173, and 176)

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction (UMUAI)*, 12(4):331–370. (cited on Page 105)

Camacho, C., Llana, L., and Nunez, A. (2016). Cost-related interface for software product lines. *Journal of Logical and Algebraic Methods in Programming (JLAMP)*, 85(1):227–244. (cited on Page 26, 32, 35, 38, 43, 172, 174, and 177)

Card, S. K., Mackinlay, J. D., and Shneiderman, B. (1999a). *Readings in information visualization - using vision to think*. Academic Press. accessed December 5, 2017. (cited on Page 163)

Card, S. K., Mackinlay, J. D., and Shneiderman, B. (1999b). *Readings in information visualization: using vision to think*. Morgan Kaufmann. (cited on Page 149)

Cetina, C., Fons, J., and Pelechano, V. (2008). Applying software product lines to build autonomic pervasive systems. In *International Systems and Software Product Line Conference (SPLC)*, pages 117–126. Ieee. (cited on Page 142)

Cetina, C., Giner, P., Fons, J., and Pelechano, V. (2009). Autonomic computing through reuse of variability models at runtime: the case of smart homes. *Computer*, 42(10):37–43. (cited on Page xiii, 6, and 7)

Chavarriaga, J., Noguera, C., Casallas, R., and Jonckers, V. (2014). Propagating decisions to detect and explain conflicts in a multi-step configuration process. In *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 337–352. Springer. (cited on Page 26, 45, 47, 61, 62, 173, and 175)

Chen, J., Cheng, Y., and Nie, D. (2013). Product configuration method based on ontology mapping. In *International Conference on Software Engineering (ICSE)*, pages 97–101. IEEE. (cited on Page 26, 38, 173, and 175)

Chen, L. and Babar, M. A. (2011). A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology (IST)*, 53(4):344–362. (cited on Page 5, 18, and 71)

Chhabra, J. K. and Gupta, V. (2010). A survey of dynamic software metrics. *Journal of Computer Science and Technology (JCST)*, 25(5):1016–1029. (cited on Page 34)

Choi, S., Cha, S., and Tappert, C. C. (2010). A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics (JSCI)*, 8(1):43–48. accessed November 15, 2017. (cited on Page 84)

Clarke, D., Muschevici, R., Proença, J., Schaefer, I., and Schlatte, R. (2010). Variability modelling in the abs language. In *International Symposium on Formal Methods for Components and Objects*, pages 204–224. Springer. (cited on Page 43)

Cleland-Huang, J., Settimi, R., Zou, X., and Solc, P. (2007). Automated classification of non-functional requirements. *Requirements Engineering Journal (REJ)*, 12(2):103–120. (cited on Page 34)

Commission, I. O. F. S. E. et al. (2001). Software engineering–product quality–part 1: Quality model. *ISO/IEC*, 9126:2001. (cited on Page 30, 141, and 162)

Constantino, K., Pereira, J. A., Padilha, J., Vasconcelos, P., and Figueiredo, E. (2016). An empirical study of two software product line tools. In *International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*. Springer. (cited on Page 75, 98, 147, and 168)

Cruz, J., Neto, P. S., Britto, R., Rabelo, R., Ayala, W., Soares, T., and Mota, M. (2013). Toward a hybrid approach to generate software product line portfolios. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 2229–2236. IEEE. (cited on Page 109, 172, and 175)

Cubo, J., Gamez, N., Fuentes, L., and Pimentel, E. (2013). Composition and self-adaptation of service-based systems with feature models. In *International Conference on Software Reuse (ICSR)*, pages 326–342. Springer. (cited on Page 26, 38, 61, 64, 173, and 175)

Czarnecki, K., Eisenecker, U. W., and Czarnecki, K. (2000). *Generative programming: methods, tools, and applications*, volume 16. Addison Wesley Reading. (cited on Page 5 and 6)

Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., and Wąsowski, A. (2012). Cool features and tough decisions: a comparison of variability modeling approaches. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 173–182. ACM. (cited on Page 6)

Czarnecki, K. and Helsen, S. (2003). Classification of model transformation approaches. Available at: http://www.ptidej.net/course/ift6251/fall05/presentations/050914/Czarnecki_Helsen.pdf/. (cited on Page 159)

Czarnecki, K., She, S., and Wasowski, A. (2008). Sample spaces and feature models: there and back again. In *International Systems and Software Product Line Conference (SPLC)*, pages 22–31. IEEE. (cited on Page 163)

Desrosiers, C. and Karypis, G. (2011). A comprehensive survey of neighborhood-based recommendation methods. In *Recommender Systems Handbook*, pages 107–144. Springer. (cited on Page 10, 11, 85, and 111)

Dhungana, D., Grünbacher, P., and Rabiser, R. (2011). The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering (ASE)*, 18(1):77–114. (cited on Page 164)

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 Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, page 8. ACM. (cited on Page 172 and 176)

dos Santos Neto, P. d. A., Britto, R., Rabêlo, R. d. A. L., de Almeida Cruz, J. J., and Lira, W. A. L. (2016). A hybrid approach to suggest software product line portfolios. *Journal Applied Soft Computing (ASC)*, 49:1243–1255. (cited on Page 25, 32, 35, 37, 38, 39, 52, 54, 59, 60, 172, 174, and 177)

Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., and Mirakhorli, M. (2011). On-demand feature recommendations derived from mining public product descriptions. In *International Conference on Software Engineering (ICSE)*, pages 181–190. IEEE. (cited on Page 120)

Dybå, T. and Dingsøyr, T. (2008). Strength of evidence in systematic reviews in software engineering. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 178–187. ACM. (cited on Page 16)

Eichelberger, H., El-Sharkawy, S., Kröher, C., and Schmid, K. (2014). Easy-producer: product line development for variant-rich ecosystems. In *International Systems and Software Product Line Conference (SPLC)*, pages 133–137. ACM. (cited on Page 172 and 175)

Eichelberger, H., Qin, C., Sizonenko, R., and Schmid, K. (2016). Using ivml to model the topology of big data processing pipelines. In *International Systems and Software Product Line Conference (SPLC)*, pages 204–208. ACM. (cited on Page 26, 38, 43, 45, 47, 48, 61, 62, 172, and 175)

Eichelberger, H. and Schmid, K. (2015). Ivml: a dsl for configuration in variability-rich software ecosystems. In *International Systems and Software Product Line Conference (SPLC)*, pages 365–369. ACM. (cited on Page 172 and 175)

El-Sharkawy, S., Kröher, C., Eichelberger, H., and Schmid, K. (2015). Experience from implementing a complex eclipse extension for software product line engineering. In *Eclipse Technology eXchange (ETX)*, pages 13–18. ACM. (cited on Page 172 and 176)

Fang, M., Leyh, G., Doerr, J., Elsner, C., and Zhao, J. (2015). Towards model-based derivation of systems in the industrial automation domain. In *International Systems and Software Product Line Conference (SPLC)*, pages 283–292. ACM. (cited on Page 27, 45, 172, and 175)

Figueiredo, E., Cacho, N., Sant'Anna, C., Monteiro, M., Kulesza, U., Garcia, A., Soares, S., Ferrari, F. C., Khan, S. S., Filho, F. C., and Dantas, F. (2008). Evolving software product lines with aspects: An empirical study. In *International Conference on Software Engineering (ICSE)*, pages 261–270. ACM. (cited on Page 159)

Foster, G., Turner, C., Ferguson, S., and Donndelinger, J. (2014). Creating targeted initial populations for genetic product searches in heterogeneous markets. *Journal of Engineering Optimization (JEO)*, 46(12):1729–1747. (cited on Page 26, 38, 52, 58, 173, and 174)

Galindo, J. A., Dhungana, D., Rabiser, R., Benavides, D., Botterweck, G., and Grünbacher, P. (2015a). Supporting distributed product configuration by integrating heterogeneous variability modeling approaches. *Information and Software Technology (IST)*, 62:78–100. (cited on Page 2, 75, 98, 100, and 121)

Galindo, J. A., Dhungana, D., Rabiser, R., Benavides, D., Botterweck, G., and Grünbacher, P. (2015b). Supporting distributed product configuration by integrating heterogeneous variability modeling approaches. *Information and Software Technology (IST)*, 62:78–100. (cited on Page 26, 45, 51, 61, 63, 172, and 174)

Galster, M., Weyns, D., Tofan, D., Michalik, B., and Avgeriou, P. (2014). Variability in software systems a systematic literature review. *IEEE Trans. Softw. Eng.*, 40(3):282–306. (cited on Page 66)

Gamez, N. and Fuentes, L. (2013). Architectural evolution of famiware using cardinality-based feature models. *Information and Software Technology (IST)*, 55(3):563–580. (cited on Page 27, 61, 64, 65, 173, and 174)

Ge, J., Yang, C., Duan, T., and Chen, Y. (2013). Research on method of construction and configuration of product family based on ontology. In *International Conference on Measurement, Information and Control (ICMIC)*, volume 2, pages 1436–1440. IEEE. (cited on Page 26, 38, 173, and 175)

Gençay, E., Schüller, P., and Erdem, E. (2017). Applications of non-monotonic reasoning to automotive product configuration using answer set programming. *Journal of Intelligent Manufacturing (JIM)*, pages 1–16. (cited on Page 27, 61, 65, 171, and 174)

Golub, G. H. and Van Loan, C. F. (2012). *Matrix computations*, volume 3. JHU Press. (cited on Page 130)

Griss, M. L., Favaro, J., and Alessandro, M. D. (1998). Integrating feature modeling with the RSEB. In *International Conference on Software Reuse (ICSR)*, pages 76–85. IEEE. (cited on Page 6)

Guedes, G., Silva, C., and Soares, M. (2017). Comparing configuration approaches for dynamic software product lines. In *Brazilian Symposium on Software Engineering (SBES)*, pages 134–143. ACM. (cited on Page 25, 32, 38, 52, 56, 58, 61, 63, 171, 176, and 177)

Guedes, G., Silva, C., Soares, M., and Castro, J. (2015). Variability management in dynamic software product lines: A systematic mapping. In *Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, pages 90–99. IEEE. (cited on Page 72, 123, and 142)

Guo, J., Zulkoski, E., Olaechea, R., Rayside, D., Czarnecki, K., Apel, S., and Atlee, J. M. (2014). Scaling exact multi-objective combinatorial optimization by parallelization. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 409–420. ACM. (cited on Page 25, 32, 38, 52, 59, 60, 68, 173, 175, and 177)

Hajri, I., Goknil, A., Briand, L. C., and Stephany, T. (2016). Pumconf: a tool to configure product specific use case and domain models in a product line. In *Proceedings of the International Symposium Foundations of Software Engineering (FSE)*, pages 1008–1012. ACM. (cited on Page 26, 45, 46, 172, and 176)

Halchenko, Y. O. and Hanke, M. (2012). Open is not enough. let's take the next step: an integrated, community-driven computing platform for neuroscience. *Frontiers in Neuroinformatics*, 2012. accessed November 15, 2017. (cited on Page 92)

Hamza, M. and Walker, R. J. (2015). Recommending features and feature relationships from requirements documents for software product lines. In *International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pages 25–31. IEEE Press. (cited on Page 120)

Harman, M., Jia, Y., Krinke, J., Langdon, W. B., Petke, J., and Zhang, Y. (2014). Search based software engineering for software product line engineering: a survey and directions for future work. In *Proceedings of the 18th International Software Product Line Conference-Volume 1*, pages 5–18. ACM. (cited on Page 72)

Heider, W., Rabiser, R., Grünbacher, P., and Lettner, D. (2012). Using regression testing to analyze the impact of changes to variability models on products. In *International Systems and Software Product Line Conference (SPLC)*, pages 196–205. ACM. (cited on Page 26, 45, 61, 65, 173, and 175)

Henard, C., Papadakis, M., Harman, M., and Le Traon, Y. (2015a). Combining multi-objective search and constraint solving for configuring large software product lines. In *International Conference on Software Engineering (ICSE)*, volume 1, pages 517–528. IEEE. (cited on Page 25, 32, 38, 52, 59, 68, 172, 175, and 177)

Henard, C., Papadakis, M., Harman, M., and Le Traon, Y. (2015b). Combining multi-objective search and constraint solving for configuring large software product lines. In *International Conference on Software Engineering (ICSE)*, pages 517–528. IEEE. (cited on Page 98 and 142)

Heradio, R., Perez-Morago, H., Fernandez-Amoros, D., Cabrerizo, F. J., and Herrera-Viedma, E. (2016). A bibliometric analysis of 20 years of research on software product lines. *Information and Software Technology*, 72:1–15. (cited on Page 71)

Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *International ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 230–237. ACM. (cited on Page 3, 11, and 85)

Hierons, R. M., Li, M., Liu, X., Segura, S., and Zheng, W. (2016a). Sip: Optimal product selection from feature models using many-objective evolutionary optimization. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(2):17. (cited on Page 25, 32, 38, 52, 54, 55, 59, 60, 69, 172, and 174)

Hierons, R. M., Li, M., Liu, X., Segura, S., and Zheng, W. (2016b). SIP: optimal product selection from feature models using many-objective evolutionary optimization. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(2):17. (cited on Page 98 and 142)

Higgins, J. P. and Green, S. (2005). Cochrane handbook for systematic reviews of interventions. (cited on Page 17)

Hillston, J. (2005). Process algebras for quantitative analysis. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 239–248. IEEE. (cited on Page 43)

Hinchey, M., Park, S., and Schmid, K. (2012). Building dynamic software product lines. *IEEE Computer*, 45(10):22–26. (cited on Page 123)

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association (JASA)*, 58(301):13–30. (cited on Page 11)

Holl, G., Grünbacher, P., Elsner, C., and Klambauer, T. (2012a). Supporting awareness during collaborative and distributed configuration of multi product lines. In *Asia-Pacific Software Engineering Conference (APSEC)*, volume 1, pages 137–147. IEEE. (cited on Page 173 and 175)

Holl, G., Grünbacher, P., Elsner, C., Klambauer, T., and Vierhauser, M. (2013). Constraint checking in distributed product configuration of multi product lines. In *Asia-Pacific Software Engineering Conference (APSEC)*, volume 1, pages 347–354. IEEE. (cited on Page 173 and 175)

Holl, G., Thaller, D., Grünbacher, P., and Elsner, C. (2012b). Managing emerging configuration dependencies in multi product lines. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 3–10. ACM. (cited on Page 173 and 176)

Holzmann, G. (2003). *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional. (cited on Page 44)

Horcas, J.-M., Pinto, M., and Fuentes, L. (2017). Green configurations of functional quality attributes. In *International Systems and Software Product Line Conference (SPLC)*, pages 79–83. ACM. (cited on Page 25, 32, 36, 38, 52, 59, 171, 175, and 177)

Hubaux, A. (2014). What research in software product line engineering is not solving in configuration. In *International Systems and Software Product Line Conference (SPLC)*, page 19. ACM. (cited on Page 9 and 145)

Hubaux, A., Jannach, D., Drescher, C., Murta, L., Mannisto, T., Czarnecki, K., Heymans, P., Nguyen, T., and Zanker, M. (2012). Unifying software and product configuration: A research roadmap. In *Proceedings of the Workshop on Configuration (ECAI)*. (cited on Page 71)

Jannach, D. and Zanker, M. (2013). Modeling and solving distributed configuration problems: A csp-based approach. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 25(3):603–618. (cited on Page 27, 61, 62, 173, and 174)

Janota, M. (2008). Do sat solvers make good configurators? In *International Systems and Software Product Line Conference (SPLC)*, pages 191–195. Springer. (cited on Page 82)

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute. (cited on Page 5, 6, and 13)

Kang, K. C., Kim, S., Lee, J., Kim, K., Kim, G. J., and Shin, E. (1998). FORM: a feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5(1):143–168. (cited on Page 6)

Kang, K. C., Lee, J., and Donohoe, P. (2002). Feature-oriented product line engineering. *IEEE Software*, 19(4):58–65. (cited on Page 5)

Karatzoglou, A., Amatriain, X., Baltrunas, L., and Oliver, N. (2010). Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering. In *ACM Recommender Systems Conference (ACM RecSys)*, pages 79–86. ACM. (cited on Page 3, 12, 124, and 127)

Karimpour, R. and Ruhe, G. (2013). Bi-criteria genetic search for adding new features into an existing product line. In *International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, pages 34–38. IEEE. (cited on Page 26, 38, 52, 54, 55, 58, 173, and 176)

Kästner, C., Trujillo, S., and Apel, S. (2008). Visualizing software product line variabilities in source code. In *International Systems and Software Product Line Conference (SPLC)*, pages 303–312. ACM. (cited on Page 163)

Khoshnevis, S. and Shams, F. (2017). Automating identification of services and their variability for product lines using nsga-ii. *Frontiers of Computer Science*, 11(3):444–464. (cited on Page 26, 32, 35, 45, 52, 54, 58, 171, 174, and 177)

Kifetew, F. M., Muñante, D., Gorroñogoitia, J., Siena, A., Susi, A., and Perini, A. (2017). Grammar based genetic programming for software configuration problem. In *International Symposium on Search-Based Software Engineering (SSBSE)*, pages 130–136. Springer. (cited on Page 25, 32, 38, 45, 52, 59, 171, 176, and 177)

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., and Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. *Information and Software Technology (IST)*, 51(1):7–15. (cited on Page 16)

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. (cited on Page 13, 16, 17, 19, and 72)

Klambauer, T., Holl, G., and Grünbacher, P. (2013). Monitoring system-of-systems requirements in multi product lines. In *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 379–385. Springer. (cited on Page 26, 45, 47, 61, 63, 173, and 175)

Kolesnikov, S. S., Apel, S., Siegmund, N., Sobernig, S., Kästner, C., and Senkaya, S. (2013). Predicting quality attributes of software product lines using software and network measures and sampling. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, page 6. ACM. (cited on Page 26, 32, 35, 173, and 176)

Koren, Y. (2009). Collaborative filtering with temporal dynamics. In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 447–456. ACM. (cited on Page 12)

Koren, Y. and Bell, R. (2015). Advances in collaborative filtering. In *Recommender systems handbook*, pages 77–118. Springer. (cited on Page 12)

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42:30–37. (cited on Page 12 and 124)

Kramer, D., Oussena, S., Komisarczuk, P., and Clark, T. (2013). Using document-oriented guis in dynamic software product lines. In *ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE)*, volume 49, pages 85–94. ACM. (cited on Page 26, 38, 61, 64, 173, and 175)

La Rosa, M., van der Aalst, W. M., Dumas, M., and Ter Hofstede, A. H. (2009). Questionnaire-based variability modeling for system configuration. *Software and Systems Modeling (SoSyM)*, 8(2):251–274. (cited on Page 121)

Lau, S. Q. (2006). *Domain analysis of e-commerce systems using feature-based model templates*. PhD thesis, Master's thesis, Dept. Electrical and Computer Engineering, University of Waterloo, Canada. (cited on Page 159)

Lee, J. (2013). Dynamic feature deployment and composition for dynamic software product lines. In *International Systems and Software Product Line Conference (SPLC)*, pages 114–116. ACM. (cited on Page 27, 61, 64, 173, and 175)

Lee, K., Kang, K. C., and Lee, J. (2002). Concepts and guidelines of feature modeling for product line software engineering. In *International Conference on Software Reuse (ICSR)*, pages 62–77. Springer. (cited on Page 5)

Leite, A. F., Alves, V., Rodrigues, G. N., Tadonki, C., Eisenbeis, C., and de Melo, A. C. M. A. (2015). Automating resource selection and configuration in inter-clouds through a software product line method. In *International Conference on Cloud Computing (CLOUD)*, pages 726–733. IEEE. (cited on Page 25, 32, 38, 52, 59, 61, 63, 172, 174, and 177)

Lettner, D., Vierhauser, M., Rabiser, R., and Grünbacher, P. (2012). Supporting end users with business calculations in product configuration. In *International Systems and Software Product Line Conference (SPLC)*, pages 171–180. ACM. (cited on Page 25, 32, 36, 38, 45, 173, 175, and 177)

Li, J., Liu, X., Wang, Y., and Guo, J. (2012). Formalizing Feature Selection Problem in Software Product Lines Using 0-1 Programming. In Wang, Y. and Li, T., editors, *International Conference on Intelligent Systems and Knowledge Engineering*, volume 124 of *AINSC*, pages 459–465. Springer, Berlin, Heidelberg. (cited on Page 56)

Lian, X. and Zhang, L. (2015a). Optimized feature selection towards functional and non-functional requirements in software product lines. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 191–200. IEEE. (cited on Page 25, 32, 38, 52, 54, 55, 59, 60, 172, 175, and 177)

Lian, X. and Zhang, L. (2015b). Optimized feature selection towards functional and non-functional requirements in software product lines. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 191–200. IEEE. (cited on Page 98 and 142)

Lin, C.-Y. and Kremer, G. E. O. (2014). Strategic decision making for multiple-generation product lines using dynamic state variable models: The cannibalization case. *Journal of Computers in Industry (JCI)*, 65(1):79–90. (cited on Page 27, 52, 57, 58, 173, and 174)

Lin, C.-Y. and Okudan, G. (2012). Application of dynamic state variable models for multiple-generation product lines with cannibalization across generations. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME)*, pages 167–177. American Society of Mechanical Engineers. (cited on Page 173 and 175)

Lisboa, L. B., Garcia, V. C., Lucrédio, D., de Almeida, E. S., de Lemos Meira, S. R., and de Mattos Fortes, R. P. (2010). A systematic review of domain analysis tools. *Information and Software Technology*, 52(1):1–13. (cited on Page 71)

Lopez-Herrejon, R. E. and Egyed, A. (2013). Towards interactive visualization support for pairwise testing software product lines. In *International Working Conference on Software Visualization (VISSOFT)*. IEEE. (cited on Page 163)

Lopez-Herrejon, R. E., Linsbauer, L., and Egyed, A. (2015). A systematic mapping study of search-based software engineering for software product lines. *Information and Software Technology (IST)*, 61:33–51. (cited on Page 72 and 142)

Lops, P., de Gemmis, M., and Semeraro, G. (2011). Content-based recommender systems: state of the art and trends. In *Recommender Systems Handbook*, pages 73–105. Springer. (cited on Page 10)

Lotufo, R., She, S., Berger, T., Czarnecki, K., and Wąsowski, A. (2010). Evolution of the linux kernel variability model. In *SPLC*, pages 136–150. Springer. (cited on Page 6)

Lu, H., Yue, T., Ali, S., Nie, K., and Zhang, L. (2014). Zen-cc: An automated and incremental conformance checking solution to support interactive product configuration. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, pages 13–22. IEEE. (cited on Page 172 and 176)

Lu, H., Yue, T., Ali, S., and Zhang, L. (2016a). Model-based incremental conformance checking to enable interactive product configuration. *Information and Software Technology (IST)*, 72:68–89. (cited on Page 45, 46, 47, 49, 50, 172, and 174)

Lu, H., Yue, T., Ali, S., and Zhang, L. (2016b). Nonconformity resolving recommendations for product line configuration. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 57–68. IEEE. (cited on Page 27, 172, and 175)

Ma, H., King, I., and Lyu, M. R. (2007). Effective missing data prediction for collaborative filtering. In Kraaij, W., de Vries, A. P., Clarke, C. L. A., Fuhr, N., and Kando, N., editors, *International ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 39–46. ACM. accessed November 15, 2017. (cited on Page 85)

Machado, L., Pereira, J., Garcia, L., and Figueiredo, E. (2014a). Splconfig: Product configuration in software product line. In *Brazilian Congress on Software (CBSoft)*, pages 1–8. ACM. (cited on Page 108, 171, and 175)

Machado, L., Pereira, J., Garcia, L., and Figueiredo, E. (2014b). SPLConfig: product configuration in software product line. In *Brazilian Congress on Software (CBSoft)*, pages 1–8. ACM. (cited on Page 142)

MahdaviHezavehi, S., Galster, M., and Avgeriou, P. (2013). Variability in quality attributes of service-based software systems: A systematic literature review. *Information and Software Technology*, 55(2):320–343. (cited on Page 66)

Mairiza, D., Zowghi, D., and Nurmuliani, N. (2010). An investigation into the notion of non-functional requirements. In *ACM Symposium on Applied Computing (SAC)*, pages 311–317. ACM. (cited on Page 141, 161, and 162)

Mannion, M. (2002). Using first-order logic for product line model validation. In *International Systems and Software Product Line Conference (SPLC)*, pages 176–187. Springer. (cited on Page 54)

Marcén, A. C., Font, J., Pastor, Ó., and Cetina, C. (2017). Towards feature location in models through a learning to rank approach. In *International Systems and Software Product Line Conference (SPLC)*, pages 57–64. ACM. (cited on Page 120)

Martinez, J., Rossi, G., Ziadi, T., Bissyandé, T. F. D. A., Klein, J., and Le Traon, Y. (2015a). Estimating and predicting average likability on computer-generated artwork variants. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 1431–1432. ACM. (cited on Page 26, 37, 38, 40, 45, 51, 172, and 175)

Martinez, J., Rossi, G., Ziadi, T., Bissyandé, T. F. D. A., Klein, J., and Le Traon, Y. (2015b). Estimating and predicting average likability on computer-generated artwork variants. In *GECCO*, pages 1431–1432. ACM. (cited on Page 75, 98, and 100)

Martinez, J., Ziadi, T., Mazo, R., Bissyandé, T. F., Klein, J., and Le Traon, Y. (2014). Feature relations graphs: A visualisation paradigm for feature constraints in software product lines. In *International Working Conference on Software Visualization (VISSOFT)*, pages 50–59. IEEE. (cited on Page 27, 28, 45, 67, 75, 98, 100, 153, 155, 163, 173, and 175)

Matuszyk, P. and Spiliopoulou, M. (2014). Hoeffding-CF: neighbourhood-based recommendations on reliably similar users. In *International Conference on User Modeling, Adaptation, and Personalization (UMAP)*, pages 146–157. Springer. (cited on Page 11 and 86)

Mauro, J., Nieke, M., Seidl, C., and Yu, I. C. (2016). Context aware reconfiguration in software product lines. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 41–48. ACM. (cited on Page 26, 32, 38, 61, 63, 69, 126, 127, 172, 176, and 177)

Mazo, R., Dumitrescu, C., Salinesi, C., and Diaz, D. (2014a). Recommendation heuristics for improving product line configuration processes. In *Recommendation Systems in Software Engineering (RSSE)*, pages 511–537. Springer. (cited on Page 26, 32, 45, 47, 50, 173, 174, and 177)

Mazo, R., Dumitrescu, C., Salinesi, C., and Diaz, D. (2014b). Recommendation heuristics for improving product line configuration processes. In *Recommendation Systems in Software Engineering (RSSE)*, pages 511–537. Springer. (cited on Page 75, 76, and 98)

Mazo, R., Muñoz-Fernández, J. C., Rincón, L., Salinesi, C., and Tamura, G. (2015). Variamos: an extensible tool for engineering (dynamic) product lines. In *International Systems and Software Product Line Conference (SPLC)*, pages 374–379. ACM. (cited on Page 26, 38, 45, 61, 64, 172, and 175)

Mazo, R., Salinesi, C., and Diaz, D. (2012a). VariaMos: a tool for product line driven systems engineering with a constraint based approach. In *International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 147–154. CEUR-WS.org. (cited on Page 98, 100, and 121)

Mazo, R., Salinesi, C., and Diaz, D. (2012b). Variamos: a tool for product line driven systems engineering with a constraint based approach. In *International Conference on Advanced Information Systems Engineering (CAiSE)*. (cited on Page 173 and 175)

Mazo, R., Salinesi, C., Djebbi, O., Diaz, D., and Lora-Michiels, A. (2012c). Constraints: The heart of domain and application engineering in the product lines engineering strategy. *International Journal of Information System Modeling and Design (IJISMD)*, 3(2):50. (cited on Page 25, 27, 32, 38, 45, 52, 59, 61, 62, 69, 173, and 174)

McCall, J. A., Richards, P. K., and Walters, G. F. (1977). *Factors in software quality: Vol. 1: Concepts and definitions of software quality*. General Electric. (cited on Page 30)

Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., Leich, T., and Saake, G. (2017). *Mastering Software Variability with FeatureIDE*. Springer. (cited on Page 46, 77, 132, 146, and 147)

Mendonça, M., Bartolomei, T. T., and Cowan, D. D. (2008). Decision-making coordination in collaborative product configuration. In *ACM Symposium on Applied Computing (SAC)*, pages 108–113. ACM. (cited on Page 159)

Mendonça, M., Branco, M., and Cowan, D. (2009). S.P.L.O.T.: software product lines online tools. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, pages 761–762. ACM. (cited on Page xvii, 41, 42, 98, 100, 121, 132, 147, 151, 157, 159, and 164)

Mitchell, S. (2012). Efficiently managing product baseline configurations in the model-based system development of a combat system product family. In *European Systems Engineering Conference (EuSEC)*, pages 622–632. INCOSE. (cited on Page 27, 61, 65, 173, and 175)

Mizouni, R., Matar, M. A., Al Mahmoud, Z., Alzahmi, S., and Salah, A. (2014). A framework for context-aware self-adaptive mobile applications spl. *Expert Systems with Applications*, 41(16):7549–7564. (cited on Page 142)

Munoz, D.-J. (2017). Achieving energy efficiency using a software product line approach. In *International Systems and Software Product Line Conference (SPLC)*, pages 131–138. ACM. (cited on Page 36, 171, and 175)

Munoz, D.-J., Pinto, M., and Fuentes, L. (2017). Green software development and research with the hadas toolkit. In *European Conference on Software Architecture: Companion Proceedings (ECSA)*, pages 205–211. ACM. (cited on Page 171 and 175)

Murashkin, A., Antkiewicz, M., Rayside, D., and Czarnecki, K. (2013). Visualization and exploration of optimal variants in product line engineering. In *International Systems and Software Product Line Conference (SPLC)*, pages 111–115. ACM. (cited on Page 164, 172, and 175)

Murguzur, A., Capilla, R., Trujillo, S., Ortiz, Ó., and Lopez-Herrejon, R. E. (2014). Context variability modeling for runtime configuration of service-based dynamic software product lines. In *International Systems and Software Product Line Conference (SPLC)*, pages 2–9. ACM. (cited on Page 26, 38, 45, 61, 64, 173, and 175)

Murwantara, I., Bordbar, B., and Minku, L. L. (2014). Measuring energy consumption for web service product configuration. In *International Conference on Information Integration and Web-Based Applications and Services (iiWAS)*, pages 224–228. ACM. (cited on Page 26, 32, 35, 173, 175, and 177)

Murwantara, I. M. and Bordbar, B. (2014). A simplified method of measurement of energy consumption in cloud and virtualized environment. In *International Conference on Big Data and Cloud Computing (ICBDCC)*, pages 654–661. IEEE. (cited on Page 173 and 175)

Mussbacher, G., Araújo, J., Moreira, A., and Amyot, D. (2012). Aourn-based modeling and analysis of software product lines. *Software Quality Journal (SQJ)*, 20(3-4):645–687. (cited on Page 26, 32, 38, 52, 56, 58, 173, 174, and 177)

Myllärniemi, V., Raatikainen, M., and Männistö, T. (2015). Representing and configuring security variability in software product lines. In *International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*, pages 1–10. IEEE. (cited on Page 25, 32, 33, 38, 43, 45, 46, 172, 175, and 177)

Nestor, D., Thiel, S., Botterweck, G., Cawley, C., and Healy, P. (2008a). Applying visualisation techniques in software product lines. In *International Symposium on Software Visualization (SOFTVIS)*, pages 175–184. ACM. (cited on Page 149 and 163)

Nestor, D., Thiel, S., Botterweck, G., Cawley, C., and Healy, P. (2008b). Applying Visualisation Techniques in Software Product Lines. In *International Symposium on Software Visualization (SOFTVIS)*, pages 175–184. ACM. (cited on Page 151 and 164)

Neves, L., Teixeira, L., Sena, D., Alves, V., Kulezsa, U., and Borba, P. (2012). Investigating the safe evolution of software product lines. *ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE)*, 47(3):33–42. (cited on Page 27, 61, 64, 65, 173, and 175)

Nieke, M., Engel, G., and Seidl, C. (2017). Darwinspl: an integrated tool suite for modeling evolving context-aware software product lines. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 92–99. ACM. (cited on Page 26, 38, 45, 61, 63, 65, 171, 176, and 177)

Nieke, M., Seidl, C., and Schuster, S. (2016). Guaranteeing configuration validity in evolving software product lines. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 73–80. ACM. (cited on Page 27, 61, 65, 172, and 176)

Nöhrer, A., Biere, A., and Egyed, A. (2012a). A comparison of strategies for tolerating inconsistencies during decision-making. In *International Systems and Software Product Line Conference (SPLC)*, pages 11–20. ACM. (cited on Page 27, 45, 47, 49, 173, and 175)

Nöhrer, A., Biere, A., and Egyed, A. (2012b). Managing sat inconsistencies with humus. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 83–91. ACM. (cited on Page 173 and 176)

Noir, J. L., Madelénat, S., Gailliard, G., Labreuche, C., Acher, M., Barais, O., and Constant, O. (2016). A decision-making process for exploring architectural variants in systems engineering. In *International Systems and Software Product Line Conference (SPLC)*, pages 277–286. ACM. (cited on Page 25, 32, 37, 38, 40, 45, 51, 52, 58, 172, 175, and 177)

Noorian, M., Asadi, M., Bagheri, E., and Du, W. (2014). Addressing non-functional properties in feature models: a goal-oriented approach. *International Journal of Software Engineering and Knowledge Engineering (JSEKE)*, 24(10):1439–1487. (cited on Page 171 and 174)

Noorian, M., Bagheri, E., and Du, W. (2016). Quality-centric feature model configuration using goal models. In *ACM Symposium on Applied Computing (SAC)*, pages 1296–1299. ACM. (cited on Page 171 and 176)

Noorian, M., Bagheri, E., and Du, W. (2017). Toward automated quality-centric product line configuration using intentional variability. *Journal of Software: Evolution and Process (JSEP)*, 29(9). (cited on Page 25, 32, 37, 38, 39, 45, 52, 55, 56, 59, 60, 69, 171, 174, and 177)

Ochoa, L., González-Rojas, O., and Thüm, T. (2015). Using decision rules for solving conflicts in extended feature models. In *International Conference on Software Language Engineering (SLE)*, pages 149–160. ACM. (cited on Page 25, 32, 38, 41, 43, 45, 47, 48, 52, 54, 59, 69, 108, 172, 175, and 177)

Ochoa, L., Gonzalez-Rojasa, O., Pereira, J. A., Castro, H., and Saake, G. (2018). A systematic literature review on the semi-automatic configuration of extended product lines. *Journal of Systems and Software*. Accepted. (cited on Page 2, 13, 103, 142, and 164)

Ochoa, L., Pereira, J. A., González-Rojas, O., Castro, H., and Saake, G. (2017). A survey on scalability and performance concerns in extended product lines configuration. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 5–12. ACM. (cited on Page 2, 13, 18, 70, 103, 142, 146, and 164)

Ognjanovic, I., Mohabbati, B., Gaevic, D., Bagheri, E., and Bokovic, M. (2012). A metaheuristic approach for the configuration of business process families. In *SCC*, pages 25–32. IEEE. (cited on Page 26, 32, 37, 38, 39, 52, 58, 173, 175, and 177)

Oh, J., Batory, D., Myers, M., and Siegmund, N. (2017). Finding near-optimal configurations in product lines by random sampling. In *Proceedings of the International Symposium Foundations of Software Engineering (FSE)*, pages 61–71. ACM. (cited on Page 25, 32, 38, 52, 58, 171, and 176)

Olaechea, R., Rayside, D., Guo, J., and Czarnecki, K. (2014). Comparison of exact and approximate multi-objective optimization for software product lines. In *International Systems and Software Product Line Conference (SPLC)*, pages 92–101. ACM. (cited on Page 25, 32, 38, 52, 53, 54, 55, 59, 60, 173, 175, and 177)

Olaechea, R., Stewart, S., Czarnecki, K., and Rayside, D. (2012). Modelling and multi-objective optimization of quality attributes in variability-rich software. In *International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages (NFPinDSML)*, page 2. ACM. (cited on Page xvii, 41, 42, 172, and 176)

Ostrosi, E., Fougères, A.-J., Ferney, M., and Klein, D. (2012). A fuzzy configuration multi-agent approach for product family modelling in conceptual design. *Journal of Intelligent Manufacturing (JIM)*, 23(6):2565–2586. (cited on Page 25, 32, 38, 45, 47, 48, 52, 56, 58, 173, and 174)

Parra, C., Romero, D., Mosser, S., Rouvoy, R., Duchien, L., and Seinturier, L. (2012). Using constraint-based optimization and variability to support continuous self-adaptation. In *ACM Symposium on Applied Computing (SAC)*, pages 486–491. ACM. (cited on Page 25, 32, 37, 38, 40, 52, 59, 61, 63, 69, 173, 176, and 177)

Pascual, G. G., Lopez-Herrejon, R. E., Pinto, M., Fuentes, L., and Egyed, A. (2015a). Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. *Journal of Systems and Software (JSS)*, 103:392–411. (cited on Page 25, 32, 38, 52, 54, 55, 59, 60, 61, 63, 69, 143, 172, 174, and 177)

Pascual, G. G., Lopez-Herrejon, R. E., Pinto, M., Fuentes, L., and Egyed, A. (2015b). Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. *Journal of Systems and Software (JSS)*, 103:392–411. (cited on Page 98 and 142)

Pascual, G. G., Pinto, M., and Fuentes, L. (2013). Run-time adaptation of mobile applications using genetic algorithms. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 73–82. IEEE Press. (cited on Page 172 and 176)

Payne, J. W., Bettman, J. R., and Johnson, E. J. (1993). *The Aadaptive Decision Maker*. Cambridge University Press. (cited on Page 75)

Pereira, J. A. (2017). Runtime collaborative-based configuration of software product lines. In *International Conference on Software Engineering (ICSE)*, pages 94–96. IEEE Press. (cited on Page 123)

Pereira, J. A., Constantino, K., and Figueiredo, E. (2015). A systematic literature review of software product line management tools. In *International Conference on Software Reuse (ICSR)*, pages 73–89. Springer. (cited on Page 1, 9, 13, 46, 71, 75, 147, and 164)

Pereira, J. A., Constantino, K., Figueiredo, E., and Saake, G. (2016a). Quantitative and qualitative empirical analysis of three feature modeling tools. In *International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pages 66–88. Springer. (cited on Page 9, 72, 75, and 145)

Pereira, J. A., Krieter, S., Meinicke, J., Schröter, R., Saake, G., and Leich, T. (2016b). FeatureIDE: scalable product configuration of variable systems. In *International Conference on Software Reuse (ICSR)*, pages 397–401. Springer. (cited on Page 1, 27, 28, 43, 45, 46, 47, 76, 82, 145, 172, and 175)

Pereira, J. A., Maciel, L., Noronha, T. F., and Figueiredo, E. (2017). Heuristic and exact algorithms for product configuration in software product lines. *International Transactions in Operational Research (ITOR)*, 24(6):1285–1306. (cited on Page 25, 28, 32, 37, 38, 40, 41, 44, 45, 52, 53, 59, 60, 69, 103, 161, 171, 174, and 177)

Pereira, J. A., Martinez, J., Gurudu, H. K., Krieter, S., and Saake, G. (2018a). Visual guidance for product line configuration using recommendations and non-functional properties. (cited on Page 145)

Pereira, J. A., Matuszyk, P., Krieter, S., Spiliopoulou, M., and Saake, G. (2016c). A feature-based personalized recommender system for product-line configuration. In *ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE)*, pages 120–131. ACM. (cited on Page 27, 28, 45, 46, 50, 67, 69, 75, 103, 123, 145, 172, and 175)

Pereira, J. A., Matuszyk, P., Krieter, S., Spiliopoulou, M., and Saake, G. (2018b). Personalized recommender systems for product-line configuration processes. *Computer Languages, Systems & Structures (COMLAN)*. (cited on Page 75, 103, 123, and 145)

Pereira, J. A., Schulze, S., Figueiredo, E., and Saake, G. (2018c). N-dimensional tensor factorization for self-configuration of software product lines at runtime. In *International Systems and Software Product Line Conference (SPLC)*. ACM. to appear. (cited on Page 123)

Pereira, J. A., Schulze, S., Krieter, S., Ribeiro, M., and Saake, G. (2018d). A context-aware recommender system for extended software product line configurations. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 1–8. ACM. (cited on Page 103, 123, and 145)

Pereira, J. A., Souza, C., Figueiredo, E., Abilio, R., Vale, G., and Costa, H. A. X. (2013). Software variability management: an exploratory study with two feature modeling tools. In *Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, pages 20–29. IEEE. (cited on Page 75, 98, 147, and 168)

Pleuss, A. and Botterweck, G. (2012). Visualization of variability and configuration options. *International Journal on Software Tools for Technology Transfer (JSTTT)*, 14(5):497–510. (cited on Page 26, 32, 45, 47, 61, 62, 173, and 174)

Pohl, K., Böckle, G., and van der Linden, F. J. (2005). *Software product line engineering: foundations, principles and techniques.* Springer, Berlin Heidelberg. (cited on Page 5, 6, 8, and 147)

Qin, Y. and Wei, G. (2012). Product configuration flow from obtaining customer requirement to providing the final customized product. *Journal of Software (JS)*, 7(2):308–315. (cited on Page 26, 32, 38, 173, 174, and 177)

Rabiser, D., Prähofer, H., Grünbacher, P., Petruzelka, M., Eder, K., Angerer, F., Kromoser, M., and Grimmer, A. (2016). Multi-purpose, multi-level feature modeling of large-scale industrial software systems. *Software & Systems Modeling (SSM)*, pages 1–26. (cited on Page 26, 28, 45, 46, 61, 62, 172, and 174)

Rabiser, R., Grünbacher, P., and Dhungana, D. (2010). Requirements for product derivation support: Results from a systematic literature review and an expert survey. *Information and Software Technology*, 52(3):324–346. (cited on Page 72)

Rabiser, R., Grünbacher, P., and Lehofer, M. (2012a). A qualitative study on user guidance capabilities in product configuration tools. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 110–119. ACM. (cited on Page 107)

Rabiser, R., Grünbacher, P., and Lehofer, M. (2012b). A qualitative study on user guidance capabilities in product configuration tools. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 110–119. ACM. (cited on Page 172 and 175)

Rezapour, S., Hassani, A., and Farahani, R. Z. (2015). Concurrent design of product family and supply chain network considering quality and price. *Transportation Research Part E: Logistics and Transportation Review*, 81:18–35. (cited on Page 25, 32, 37, 38, 40, 52, 55, 59, 60, 172, 174, and 177)

Ricci, F., Rokach, L., and Shapira, B. (2011). *Introduction to recommender systems handbook*. Springer. (cited on Page 9 and 120)

Roos-Frantz, F., Benavides, D., Ruiz-Cortés, A., Heuer, A., and Lauenroth, K. (2012). Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal (SQJ)*, 20(3-4):519–565. (cited on Page 25, 32, 38, 43, 52, 59, 173, 174, and 177)

Ruiz, C., Duran-Limon, H. A., and Parlavantzas, N. (2016). Towards a software product line-based approach to adapt iaas cloud configurations. In *International Conference on Utility and Cloud Computing (UCC)*, pages 398–403. ACM. (cited on Page 25, 32, 38, 52, 58, 61, 63, 143, 172, 175, and 177)

Saaty, R. (1987). The analytic hierarchy process ? what it is and how it is used. *Mathematical Modelling*, 9(3):161–176. (cited on Page 39)

Safdar, S. A., Lu, H., Yue, T., and Ali, S. (2017). Mining cross product line rules with multi-objective search and machine learning. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 1319–1326. ACM. (cited on Page 26, 45, 47, 61, 62, 63, 172, and 175)

Salinesi, C., Mazo, R., Diaz, D., and Djebbi, O. (2010). Using integer constraint solving in reuse based requirements engineering. In *International Requirements Engineering Conference*, pages 243–251. IEEE. (cited on Page 54)

Saller, K., Lochau, M., and Reimund, I. (2013). Context-aware dspls: model-based runtime adaptation for resource-constrained systems. In *International Systems and Software Product Line Conference (SPLC)*, pages 106–113. ACM. (cited on Page 26, 38, 61, 63, 127, 173, and 175)

Sánchez, L. E., Diaz-Pace, J. A., Zunino, A., Moisan, S., and Rigault, J.-P. (2014). An approach for managing quality attributes at runtime using feature models. In *Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, pages 11–20. IEEE. (cited on Page 25, 32, 35, 37, 38, 40, 52, 53, 58, 59, 61, 63, 173, 176, and 177)

Sanchez, L. E., Moisan, S., and Rigault, J.-P. (2013). Metrics on feature models to optimize configuration adaptation at run time. In *International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, pages 39–44. IEEE. (cited on Page 173 and 176)

Santos, I. S., Rocha, L. S., Neto, P. A. S., and Andrade, R. (2016). Model verification of dynamic software product lines. In *Brazilian Symposium on Software Engineering (SBES)*, pages 113–122. ACM. (cited on Page 26, 32, 38, 44, 61, 64, 172, 176, and 177)

Sawyer, P., Mazo, R., Diaz, D., Salinesi, C., and Hughes, D. (2012). Constraint programming as a means to manage configurations in self-adaptive systems. *EEE Computer Dynamic Software Product Lines (CDSPL)*, pages 1–12. (cited on Page 172 and 174)

Sayyad, A. S., Ingram, J., Menzies, T., and Ammar, H. (2013a). Optimum feature selection in software product lines: Let your model and values guide your search. In *International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, pages 22–27. IEEE. (cited on Page 173 and 176)

Sayyad, A. S., Ingram, J., Menzies, T., and Ammar, H. (2013b). Scalable product line configuration: A straw to break the camel's back. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 465–474. IEEE. (cited on Page 25, 32, 38, 52, 173, 175, and 177)

Sayyad, A. S., Menzies, T., and Ammar, H. (2013c). On the value of user preferences in search-based software engineering: a case study in software product lines. In *International Conference on Software Engineering (ICSE)*, pages 492–501. IEEE. (cited on Page 54, 59, 60, 68, 173, and 175)

Schmid, K. and Eichelberger, H. (2015). Easy-producer: from product lines to variability-rich software ecosystems. In *International Systems and Software Product Line Conference (SPLC)*, pages 390–391. ACM. (cited on Page 172 and 175)

Schneeweiss, D. and Botterweck, G. (2010). Using flow maps to visualize product attributes during feature configuration. In *International Systems and Software Product Line Conference (SPLC)*, pages 219–228. IEEE. (cited on Page 164)

Schroeter, J., Mucha, P., Muth, M., Jugel, K., and Lochau, M. (2012). Dynamic configuration management of cloud-based applications. In *International Systems and Software Product Line Conference (SPLC)*, pages 171–178. ACM. (cited on Page 25, 32, 38, 45, 47, 48, 61, 62, 63, 173, and 175)

Schwäger, F. and Westfechtel, B. (2016). Supermod: tool support for collaborative filtered model-driven software product line engineering. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 822–827. IEEE. (cited on Page 27, 45, 47, 48, 172, and 175)

Seinturier, L., Merle, P., Rouvoy, R., Romero, D.and Schiavoni, V., and Stefani, J. (2012). A component-based middleware platform for reconfigurable service-oriented architectures. *Software Practice and Experience*, 42(5):559–583. (cited on Page 159)

Shani, G. and Gunawardana, A. (2011). Evaluating recommendation systems. In *Recommender Systems Handbook*. Springer. (cited on Page 91 and 115)

Sharifloo, A. M., Metzger, A., Quinton, C., Baresi, L., and Pohl, K. (2016). Learning and evolution in dynamic software product lines. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 158–164. IEEE. (cited on Page 26, 38, 61, 63, 64, 65, 143, 172, and 176)

Shi, K. (2017). Combining evolutionary algorithms with constraint solving for configuration optimization. pages 665–669. IEEE. (cited on Page 142)

Siegmund, N., Grebhahn, A., Apel, S., and Kästner, C. (2015). Performance-influence models for highly configurable systems. In *Proceedings of the International Symposium Foundations of Software Engineering (FSE)*, pages 284–294. ACM. (cited on Page 25, 27, 32, 34, 36, 38, 45, 52, 172, 176, and 177)

Siegmund, N., Kolesnikov, S. S., Kästner, C., Apel, S., Batory, D., Rosenmüller, M., and Saake, G. (2012a). Predicting performance via automated feature-interaction detection. In *International Conference on Software Engineering (ICSE)*, pages 167–177. IEEE Press. (cited on Page 53, 59, 172, and 175)

Siegmund, N., Mory, M., Feigenspan, J., Saake, G., Nykolaychuk, M., and Schumann, M. (2012b). Interoperability of non-functional requirements in complex systems. In *International Workshop on Software Engineering for Embedded Systems (SEES)*, pages 2–8. IEEE Press. (cited on Page 8, 172, and 176)

Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., and Saake, G. (2012c). Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal (SQJ)*, 20(3-4):487–517. (cited on Page 172 and 174)

Simons, P., Niemelä, I., and Soininen, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234. (cited on Page 43)

Sincero, J., Tartler, R., Egger, C., Schröder-Preikschat, W., and Lohmann, D. (2010). Facing the linux 8000 feature nightmare. In *European Conference on Computer Systems (EuroSys)*. (cited on Page 75 and 98)

Sion, L., Van Landuyt, D., Joosen, W., and de Jong, G. (2016). Systematic quality trade-off support in the software product-line configuration process. In *International Systems and Software Product Line Conference (SPLC)*, pages 164–173. ACM. (cited on Page 25, 28, 32, 38, 45, 46, 172, and 175)

Soares, L. R., do Carmo Machado, I., and de Almeida, E. S. (2015). Non-functional properties in software product lines: A reuse approach. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, page 67. ACM. (cited on Page 26, 32, 36, 172, 176, and 177)

Soares, L. R., Potena, P., do Carmo Machado, I., Crnkovic, I., and de Almeida, E. S. (2014). Analysis of non-functional properties in software product lines: a systematic review. In *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 328–335. IEEE. (cited on Page 30)

Soltani, S., Asadi, M., Gašević, D., Hatala, M., and Bagheri, E. (2012). Automated planning for feature model configuration based on functional and non-functional requirements. In *International Systems and Software Product Line Conference (SPLC)*, pages 56–65. ACM. (cited on Page 172 and 175)

Sommerville, I. and Sawyer, P. (1997). Viewpoints: principles, problems and a practical approach to requirements engineering. *Annals of software engineering*, 3(1):101–130. (cited on Page 141 and 161)

Spinczyk, O. and Beuche, D. (2004). Modeling and building software product lines with eclipse. pages 18–19. ACM. (cited on Page 98, 100, and 121)

Svee, E.-O. and Zdravkovic, J. (2015). Towards a consumer preference-based taxonomy for information systems development. In *International Conference on Perspectives in Business Informatics Research (BIR)*, pages 213–227. Springer. (cited on Page 172 and 175)

Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2009). Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research (JMLR)*, 10:623–656. accessed November 15, 2017. (cited on Page 3, 12, 87, 88, and 137)

Tan, L., Lin, Y., and Liu, L. (2014a). Quality ranking of features in software product line engineering. In *Asia-Pacific Software Engineering Conference (APSEC)*, volume 2, pages 57–62. IEEE. (cited on Page 2, 75, 98, 99, and 121)

Tan, L., Lin, Y., and Liu, L. (2014b). Quality ranking of features in software product line engineering. In *Asia-Pacific Software Engineering Conference (APSEC)*, volume 2, pages 57–62. IEEE. (cited on Page 25, 32, 37, 38, 39, 45, 50, 173, 175, and 177)

Tan, L., Lin, Y., Ye, H., and Zhang, G. (2013). Improving product configuration in software product line engineering. In *Australasian Computer Science Conference (ACSC)*, pages 125–133. Australian Computer Society, Inc. (cited on Page 27, 45, 47, 50, 132, 133, 173, and 175)

Tan, T. H., Xue, Y., Chen, M., Sun, J., Liu, Y., and Dong, J. S. (2015a). Optimizing selection of competing features via feedback-directed evolutionary algorithms. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 246–256. ACM. (cited on Page 98 and 142)

Tan, T. H., Xue, Y., Chen, M., Sun, J., Liu, Y., and Dong, J. S. (2015b). Optimizing selection of competing features via feedback-directed evolutionary algorithms. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 246–256. ACM. (cited on Page 172 and 176)

Tanhaei, M., Habibi, J., and Mirian-Hosseinabadi, S.-H. (2016a). Automating feature model refactoring: A model transformation approach. *Information and Software Technology (IST)*, 80:138–157. (cited on Page 172 and 174)

Tanhaei, M., Habibi, J., and Mirian-Hosseinabadi, S.-H. (2016b). A feature model based framework for refactoring software product line architecture. *Journal of Computer Science and Technology (JCST)*, 31(5):951–986. (cited on Page 27, 61, 65, 172, and 174)

Temple, P., Acher, M., Jézéquel, J.-M. A., Noel-Baron, L. A., and Galindo, J. A. (2017). Learning-based performance specialization of configurable systems. Research report, IRISA, Inria Rennes ; University of Rennes 1. accessed December 5, 2017. (cited on Page 164)

Ter Beek, M. H., Lafuente, A. L., and Petrocchi, M. (2013). Combining declarative and procedural views in the specification and analysis of product families. In *International Systems and Software Product Line Conference (SPLC)*, pages 10–17. ACM. (cited on Page 172 and 175)

ter Beek, M. H., Legay, A., Lafuente, A. L., and Vandin, A. (2015a). Quantitative analysis of probabilistic models of software product lines with statistical model checking. *Electronic Proceedings in Theoretical Computer Science (EPTCS)*. (cited on Page 172 and 176)

ter Beek, M. H., Legay, A., Lafuente, A. L., and Vandin, A. (2015b). Statistical analysis of probabilistic models of software product lines with quantitative constraints. In *International Systems and Software Product Line Conference (SPLC)*, pages 11–15. ACM. (cited on Page 172 and 175)

Ter Beek, M. H., Legay, A., Lafuente, A. L., and Vandin, A. (2016). Statistical model checking for product lines. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, pages 114–133. Springer. (cited on Page 26, 32, 38, 43, 61, 64, 172, 176, and 177)

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., and Leich, T. (2014). FeatureIDE: an extensible framework for feature-oriented software development. *Science of Computer Programming (SCP)*, 79(0):70–85. (cited on Page 100, 121, 159, 172, and 174)

Thurimella, A. K. and Bruegge, B. (2012). Issue-based variability management. *Information and Software Technology (IST)*, 54(9):933–950. (cited on Page 27, 45, 46, 47, 48, 67, 173, and 174)

Travassos, G. and Biolchini, J. (2007). Systematic review applied to software engineering. In *Brazilian Symposium on Software Engineering (SBES)*, page 436. ACM. (cited on Page 18)

Triando, Muschevici, R., and Azurat, A. (2016). Incremental product configuration in software product line engineering. In *International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 597–604. Scopus. (cited on Page 26, 32, 38, 43, 172, and 175)

Trinidad, P., Cortés, A. R., Benavides, D., and Segura, S. (2008). Three-dimensional feature diagrams visualization. In *International Systems and Software Product Line Conference (SPLC)*, pages 295–302. Springer. (cited on Page 151 and 164)

Umpfenbach, E. L., Dalkiran, E., Chinnam, R. B., and Murat, A. E. (2017). Optimization of strategic planning processes for configurable products. *Journal of the Operational Research Society (JORS)*, pages 1–20. (cited on Page 25, 32, 38, 52, 55, 58, 172, 174, and 177)

Urli, S., Blay-Fornarino, M., and Collet, P. (2014). Handling complex configurations in software product lines: a tooled approach. In *International Systems and Software Product Line Conference (SPLC)*, pages 112–121. ACM. (cited on Page 26, 45, 61, 63, 173, and 175)

Vale, G., Abílio, R., Pereira, J., Figueiredo, E., Afonso, P., and Costa, H. (2016). Identification and relationship between notation and tool for feature models with graphic representation. In *International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–12. IEEE. (cited on Page 5, 13, 33, and 71)

Valov, P., Guo, J., and Czarnecki, K. (2015). Empirical comparison of regression methods for variability-aware performance prediction. In *International Systems and Software Product Line Conference (SPLC)*, pages 186–190. ACM. (cited on Page 26, 32, 35, 172, 175, and 177)

Villela, K., Arif, T., and Zanardini, D. (2012). Towards product configuration taking into account quality concerns. In *International Systems and Software Product Line Conference (SPLC)*, pages 82–90. ACM. (cited on Page 172 and 175)

Wang, L. and Ng, W.-K. (2012). Hybrid solving algorithms for an extended dynamic constraint satisfaction problem based configuration system. *Concurrent Engineering Research and Applications (CERA)*, 20(3):223–236. (cited on Page 26, 38, 61, 63, 173, and 174)

Wang, L., Zhong, S.-S., and Zhang, Y.-J. (2017). Process configuration based on generative constraint satisfaction problem. *Journal of Intelligent Manufacturing (JIM)*, 28(4):945–957. (cited on Page 26, 32, 38, 172, and 174)

Wang, Y.-l. and Pang, J.-w. (2014). Ant colony optimization for feature selection in software product lines. *Journal of Shanghai Jiaotong University (Science) (JSJU)*, 19:50–58. (cited on Page 25, 32, 38, 52, 59, 69, 173, 174, and 177)

White, J., Dougherty, B., and Schmidt, D. C. (2009). Selecting highly optimal architectural feature sets with filtered cartesian flattening. *Journal of Systems and Software (JSS)*, 82(8):1268–1284. (cited on Page 161)

White, J., Galindo, J. A., Saxena, T., Dougherty, B., Benavides, D., and Schmidt, D. C. (2014). Evolving feature model configurations in software product lines. *Journal of Systems and Software (JSS)*, 87:119–136. (cited on Page 26, 38, 52, 54, 57, 58, 59, 69, 173, and 174)

Winkelmann, T., Koscielny, J., Seidl, C., Schuster, S., Damiani, F., and Schaefer, I. (2016). Parametric deltaj 1.5: Propagating feature attributes into implementation artifacts. In *Gemeinsamer Tagungsband der Workshops der Tagung Software Engineering*, pages 40–54. (cited on Page 26, 32, 38, 43, 172, and 174)

Wittern, E., Kuhlenkamp, J., and Menzel, M. (2012). Cloud service selection based on variability modeling. In *International Conference on Service-Oriented Computing (ICSOC)*, pages 127–141. Springer. (cited on Page 25, 32, 37, 38, 40, 52, 58, 61, 63, 173, 175, and 177)

Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B., and Wesslen, A. (2000). Experimentation in software engineering: an introduction. (cited on Page 18, 70, 97, 98, 120, and 141)

Xiang, Y., Zhou, Y., Zheng, Z., and Li, M. (2018). Configuring software product lines by combining many-objective optimization and sat solvers. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 26(4):14. (cited on Page 142)

Xue, Y., Zhong, J., Tan, T. H., Liu, Y., Cai, W., Chen, M., and Sun, J. (2016). Ibed: Combining ibea and de for optimal feature selection in software product line engineering. *Journal Applied Soft Computing (ASC)*. (cited on Page 26, 32, 38, 172, 174, and 177)

Zanardini, D., Albert, E., and Villela, K. (2016). Resource–usage–aware configuration in software product lines. *Journal of Logical and Algebraic Methods in Programming (JLAMP)*, 85(1):173–199. (cited on Page 9, 25, 27, 28, 32, 35, 37, 38, 39, 43, 45, 52, 54, 58, 109, 172, 174, and 177)

Zdravkovic, J., Svee, E.-O., and Giannoulis, C. (2015). Capturing consumer preferences as requirements for software product lines. *Requirements Engineering Journal (REJ)*, 20(1):71–90. (cited on Page 26, 37, 38, 39, 40, 172, and 174)

Zhang, B. and Becker, M. (2016). Supporting product configuration in application engineering using exconfig. In *International Systems and Software Product Line Conference (SPLC)*, pages 324–327. ACM. (cited on Page 27, 45, 172, and 175)

Zhang, G., Ye, H., and Lin, Y. (2014). Quality attribute modeling and quality aware product configuration in software product lines. *Software Quality Journal (SQJ)*, 22(3):365–401. (cited on Page 25, 32, 37, 38, 39, 45, 46, 47, 50, 121, 173, 174, and 177)

Zhao, Y., Wang, H., Hong, H., and Chen, J. (2012). Cased-based reasoning based on extension theory for conflict resolution in cooperative design. In *International Conference on Cooperative Design, Visualization, and Engineering (CDVE)*, pages 134–142. Springer. (cited on Page 26, 32, 45, 47, 48, 49, 173, 175, and 177)

Zheng, P., Xu, X., Yu, S., and Liu, C. (2017a). Personalized product configuration framework in an adaptable open architecture product platform. *Journal of Manufacturing Systems (JMSY)*, 43:422–435. (cited on Page 26, 37, 38, 39, 40, 45, 50, 61, 62, 172, 174, and 177)

Zheng, Y.-j., Yang, Y., Su, J.-f., Zhang, N., and Jiao, Y. (2017b). Dynamic optimization method for configuration change in complex product design. *Journal of Advanced Manufacturing Technology (JAMT)*, 92(9-12):4323–4336. (cited on Page 26, 38, 61, 64, 172, and 174)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den 25. June 2018