Bachelor's Thesis

FEATURE IDENTIFICATION WITH FORMAL CONCEPT ANALYSIS: A CASE STUDY

LUKAS SELVAGGIO

July 14, 2022

Advisor:
Christof Tinnes   Chair of Software Engineering

Examiners:
Prof. Dr. Sven Apel   Chair of Software Engineering
Prof. Dr. Martina Maggio   Department of Computer Science

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,______________________ ______________________________
(Datum/Date) (Unterschrift/Signature)

ABSTRACT

Software product lines offer a way of combining the benefits of mass production with the benefits of customization by providing a form of structured reuse that allows software products to be composed of reusable components. This allows manufacturers to provide tailor-made products at a large scale, while leading to a reduction in cost, with an increasing product portfolio, compared to developing the products from scratch. However, the high upfront investment for the design and implementation deters companies from adopting a product line approach from the get-go. Only in retrospect does the overhead of maintenance and increase in development costs associated with their current approach lead companies to adopt a software product line approach. This, however, is a manual, tedious, time and money-consuming task, which is why many approaches have been proposed by researchers in order to at least semi-automatically perform parts of it. We evaluated a lightweight implementation of such a semi-automated approach to software product line engineering, based on formal concept analysis. For this, we applied the approach to a real-world, production-scale case study from the railway domain and developed a plugin for the modeling tool MagicDraw that visualizes the results to the engineers in a familiar environment and intuitive way. Using this visualization, we gained meaningful insights from a focus group conducted with domain experts, into what can be expected from such an approach, when applied to more than the usual illustrative examples.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AME	Atomic Model Element
EMF	Eclipse Modeling Framework
FCA	Formal Concept Analysis
MOF	Meta Object Facility
PDE	Plugin Development Environment
SPL	Software Product Line
SPLE	Software Product Line Engineering
ST	Structured Text

CHAPTER 1

Introduction

Starting with software being written from scratch for individual use cases and very specific hardware, it soon became apparent that this would be too difficult to maintain, with the steady increase in its complexity and variety of its use case scenarios. Similar to the process of mass producing physical goods the production of software was streamlined to the point that the industry resided to the use of standardized software based on standard platforms. But while this standardization allowed for software to be made readily available to the general public it also came at the cost of being able to target smaller market groups or the requirements of individual customers. Similar to their physical counterparts, software product lines offer a way of reaping the benefits of mass production, while maintaining the possibility of catering to different customer requirements. They achieve this by letting customers combine and configure predefined software components or chose between different alternatives, with the goal of sharing as many components as possible between products. All while leaving leeway for potentially newly created components based on an individual customer's needs. While in an ideal world software engineers would stick to the guidelines given by researches and alike for the development of product portfolios, unfortunately, due to time and money constraints, it is almost common practise to cut corners in favor of short term benefits in exchange for quality and the lack of maintainability in the long run. One of such approaches is appropriately called the 'clown and own' approach where in its most literal sense, source code is copy and pasted from one product to another and from there on developed independently from its original host. In order to overcome the resulting, inevitable overhead of maintenance and increasing costs of development for an increasing product portfolio, many companies reside to retroactively adopting a software product line approach. This, however, is a tedious, time and money consuming task, which is why many approaches have been proposed by researchers in order to at least semi-automatically perform parts of it.

1.1 Goal

Countless approaches to extractive software product line engineering have been proposed for different domains of software artefacts, ranging from source code to different formats of models. Many of them, however, are only evaluated on, at best, synthetic industry examples, where the specifically created controlled environment fails to include artifacts that arise from the natural development of product line-ups. New approaches are being actively developed, without making sure that even the most basic approaches are even feasible if applied in practise. This is why our goal is to evaluate a lightweight, semi-automatic approach to extractive software product line engineering, on a real-world, production-scale case study and to gain some insights on what can be expected from applying such an approach to it.

1.2 Contributions

We evaluate a lightweight, semi-automatic approach to extractive software product line engineering, on a real-world, production-scale case study. The approach is based on formal concept analysis and implemented in an open source framework called But4Reuse. As far as we can tell, at the time of writing, we are one of the first, if not the first, to evaluate such an approach on an industry case study of this size based in the modeling domain. For this we implement a first iteration of a visualization plugin for the modeling tool MagicDraw, which visualizes the results directly in the tool and will be used to evaluate our findings. Future iterations of this plugin, as we will discuss in this thesis, might end up in the hands of the engineers behind our case study. Additionally we implemented some extensions and improvements to the BUT4REUSE framework, but, as of the time of writing, these changes have not been incorporated into the official version, mainly due to time constraints. This will be left as future work.

1.3 Overview

In Chapter 2 we present related work to this thesis. First we take a look at the work related to the main tool BUT4REUSE that we will be using. After that we will present a state-of-the-art approach to extractive software product line engineering (SPLE), also based on formal concept analysis (FCA) like the approach used in this thesis. Next we give a quick overview of two other approaches to extractive SPLE. Finally, we compile a list of case studies used to evaluate SPLE approaches. In Chapter 3 we give the necessary background for the subjects discussed in this thesis. First we introduce the concept of software product lines, the field of software product line engineering, and related terminology. After that, we present the most basic mathematical background on FCA. In Chapter 4 we introduce the methodology used for this thesis. First we present our research questions. Next we discuss our experimental setup, where we first talk about our hardware setups, then about the different implementations of BUT4REUSE that we will be using, and finally we present the case study that is the subject of this thesis. Then we discuss how we plan on answering our research questions. In Chapter 5 we first evaluate our findings and give an answer to our research questions. Afterwards we discuss the threats to the validity of our experiments. Finally, in Chapter 6 we draw final conclusions for this thesis and give an outlook into possible future work.

CHAPTER 2

Related Work

In this chapter we will introduce some of the work related to this thesis. First, we will take a look at the work related to the framework BUT4REUSE, which will be used as the main tool for this thesis. Next, we present a state-of-the-art approach to extractive SPLE that is also based on FCA. After that, we present a selection of other approaches to extractive SPLE. Finally, we compile a list of SPLE case studies that are used by others to evaluate all kinds of different approaches to SPLE.

2.1 But4Reuse - Bottom-Up Technologies For Reuse

The paper 'Bottom-Up Adoption of Software Product Lines - A Generic and Extensible Approach' by Martinez et al. [17] proposes a unifying framework for the extractive adoption of software product lines from existing artefact variants and it provides an implementation BUT4REUSE[1] of said framework based on the Eclipse Plugin Development Environment (PDE)[2], which will be used as the tool of choice for the experiments conducted in this thesis. The authors identify three problems with the already existing approaches to bottom-up adoption of software product lines. Firstly, the lack of abstraction in the sense that existing approaches are often targeted at very specific software artifacts. Secondly, the three main objectives of bottom-up software product line adoption, namely *feature identification and analysis*, *feature location*, and *re-engineering* are subject to different requirements regarding their individual approaches making it hard to consolidate them into a single workflow. Lastly, the

[1]https://github.com/but4reuse/but4reuse
[2]https://www.eclipse.org/pde/

lack of benchmarking, which is partly due to the inherent nature of the individual approaches and their focus on one specific type of software artifact, but also due the lack of availability of implementations to the public. The authors try to tackle these problems by providing a unified, extensible framework, that relies on an intermediate model for the provided artifacts in order to reuse or integrate existing specific approaches for the three aforementioned main objectives of bottom-up software product line adoption. Figure 2.1.1 illustrates the process of said SPL adoption using the provided framework, where the steps with black background indicate the given extension points. The framework itself is built on top of three main principles [17]:

1. A typical software artefact can be decomposed into distinct elements
2. Given a pair of a elements of the same type of artefact, they can be compared using a similarity metric
3. Given a set of elements extracted from existing artefacts, a new artefact can at least be partially constructed from them

In order to accommodate different types of artefacts, adapters are used to decompose provided artefacts into atomic elements. These are wrapped by an intermediate model that is later also used to reconstruct a new artefact of that type using said elements. There already is a fairly large set of implemented adapters for various artefact types: an adapter for graphs given in the GraphML or GML format, one for Java source code and one for C source code, to mention a few. But the one that will be used in this thesis is the one for the Eclipse Modeling Framework[3] (EMF) introduced in detail by Martinez et. al. in [16, 19]. Here a Meta Object Facility[4] (MOF) compliant model is decomposed into three types of *atomic model elements* (AMEs) [19]

- a *class*, which is constituted of an 'owner' class and the class 'object' itself
- an *attribute*, which is constituted of an 'owner' class, an 'attribute identifier', and the 'value' of the attribute

[3]https://www.eclipse.org/modeling/emf/
[4]https://www.omg.org/mof/

- a *reference*, which is constituted of an 'owner' class, a 'reference identifier', and a set of 'referenced' classes

and similarity of these AMEs is computed by comparison methods provided via the EMF DiffMerge[5] API.

2.2 Formal Concept Analysis In Extractive SPLE

The paper 'Extraction of Feature Models from Formal Contexts' by Ryssel et al. [23] proposes an approach of automatically creating feature models that exactly represent a set of given input model variants using formal concept analysis on the incidence matrices describing the different components present on each variant. The approach is exemplified on function-block-based models. Function block models represent systems in the form of blocks that only expose input and output pins and where the underlying algorithms can be further configured using parameters provided to the respective blocks. In order to obtain the aforementioned incidence matrices for a set of product variants, one has to compare all the product variants to find the similar and different components that the individual products are made of. Ryssel et al. developed an approach that is specialized to functional-block-based models [25], where individual models are represented as a hierarchy of subsystems, and where the subsystems are then compared using a similarity metric that uses the type, name, structural and behavioral parameters of blocks and recursive similarity for non-primitive blocks as local criteria and the distance between blocks, the connected ports and graphical direction as neighborhood criteria, in order to produce a similarity value s between 0.0 and 1.0. This similarity value is then used as the distance (1.0 - s) for an agglomerative clustering method, which is restricted by a threshold, in order to avoid clustering all similar subsystems into one cluster. This finally results in the desired incidence matrix of clustered subsystems on their respective product variants. In chapter 4 of his PhD thesis [22], Ryssel gives a more elaborate overview on different clustering and model matching techniques. He concludes that for clustering, hierarchical or density

[5]https://wiki.eclipse.org/EMF_DiffMerge

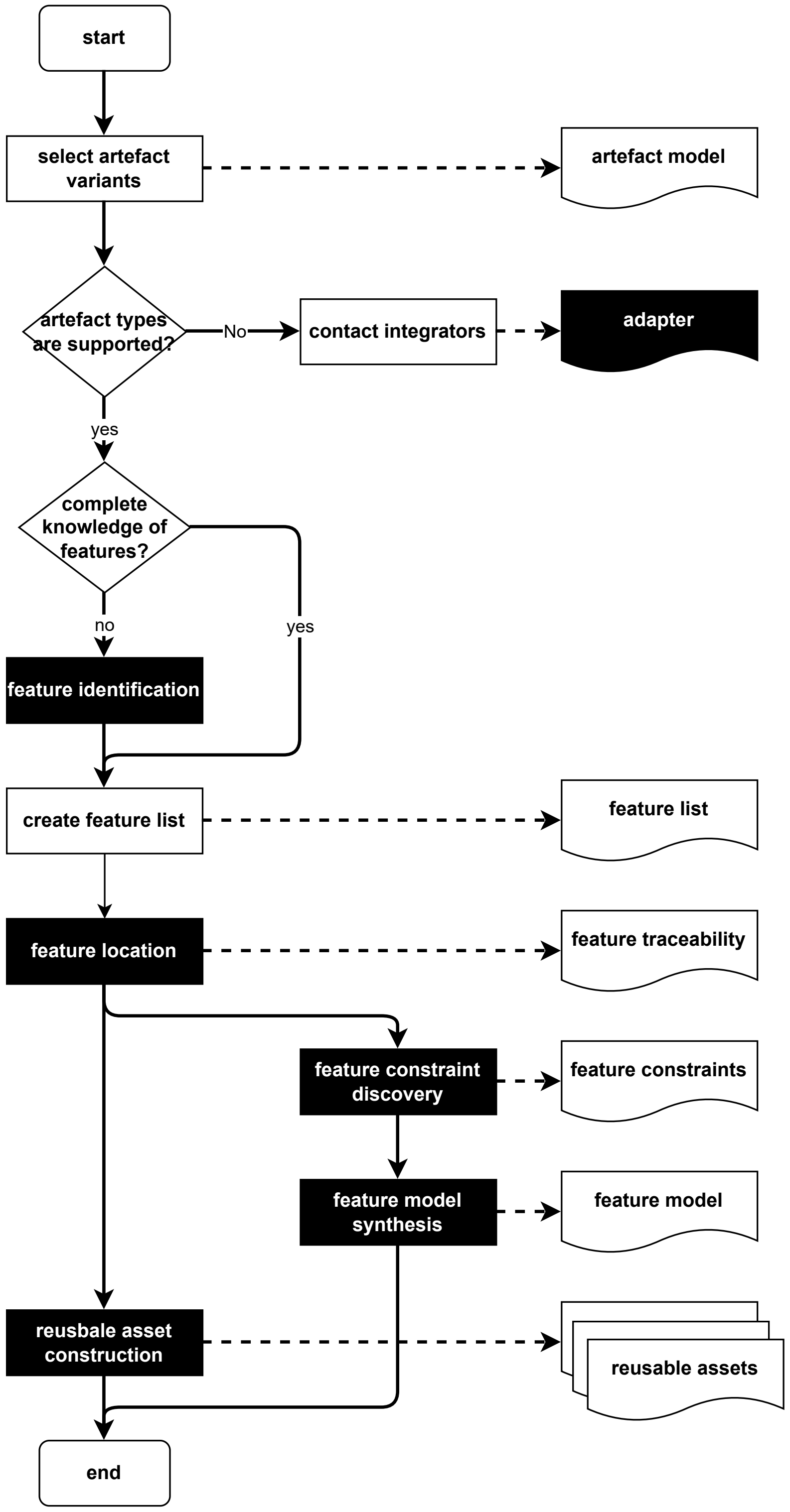


Fig. 2.1.1: Process of SPL adoption with But4Reuse. Adopted from [18].

based techniques are the most suitable for this use-case, and for model matching a custom solution similar to the one described by Ryssel et al. [25] is needed, due to the lack of out-of-the-box suitability or availability of existing approaches. Following the structure of Figure 2.2.1, which summarizes the workflow of chapter 5 of Ryssel's PhD thesis [22], as a first step towards creating a feature model, with the identified clustered subsystems as feature candidates, Ryssel obtains all feature implications from the incidence matrices. For this, the context (incidence matrix) is extended by the negations of all attributes, in order to obtain implications with negations. Then, the (extended) attribute concept graph of the (extended) context is calculated, where the attribute concepts are the nodes and the edges represent the subset relations between them. This way calculating the complete set of concepts, for the size of which Ryssel gives an upperbound as $|\underline{\mathfrak{B}}(O, A, I)| \leq \frac{3}{2} * 2^{\sqrt{|I|+1}} - 1$ (see Section 3.2 for notes on notation) [22] opposed to the linear increase in attribute concepts with the amount of clarified (mutually distinct in their incidence) attributes [23], is avoided. Using the extended attribute concept graph, redundant features are identified by equivalent or reducible attributes and with the help of an implication base, that is calculated in terms of proper premises, these redundant features (or a subset of them depending on mutual redundancy as defined by Ryssel in Section 5.4.4 of his PhD thesis [22]) are removed from the attribute concept graph and replacement terms for them are calculated. Using this reduced attribute concept graph, the set of previously obtained mandatory and optional relationships between features is extended by additionally calculating inclusive and exclusive or-relationships between them, based on all the subconcepts of a specific concept in the concept graph and then a feature tree is derived from these relations. With a minimized version of the previously calculated implication base, this feature tree is then extended by additional cross-tree constraints which forms the final feature model.

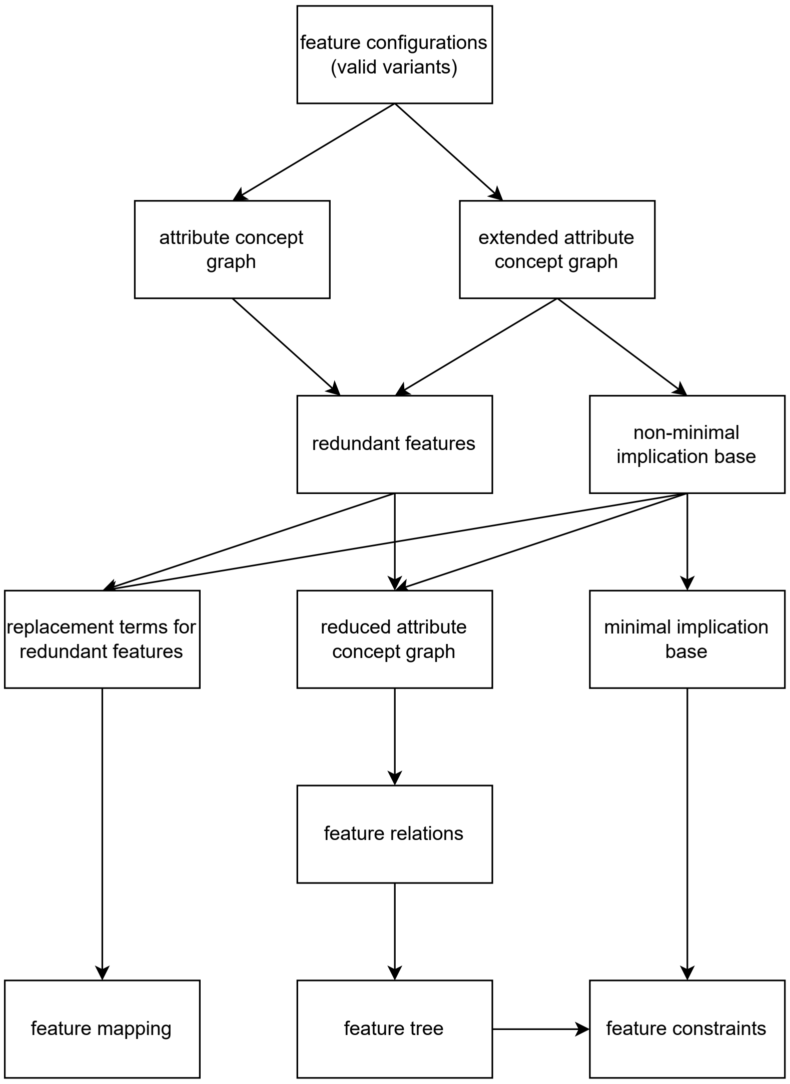

feature configurations (valid variants)
attribute concept graph
extended attribute concept graph
redundant features
non-minimal implication base
replacement terms for redundant features
reduced attribute concept graph
minimal implication base
feature relations
feature mapping
feature tree
feature constraints

Fig. 2.2.1: Extraction of feature models from feature configurations. Adapted from [22].

2.3 Other Approaches To Extractive SPLE

2.3.1 Family Model Mining

The paper ‘Family Model Mining for Function Block Diagrams in Automation Software’ [10] by Wille et al. proposes an improved version of the authors previous work regarding family mining, on the example of function block models. The approach is divided into three phases. The first phase, called the comparing phase, performs a data flow-oriented comparison between a manually selected base model, and all remaining models, referred to as compare models. Starting from the start block of the base model and the current compare model, blocks are compared using a weighted similarity metric. The pair of blocks is stored together with its similarity in so called compare elements and the next pair is obtained by following the flow of the model exhaustively. Termination is ensured by only creating new compare elements if they do not already exist. In the case of unequal model sizes, the remaining blocks are compared with null, in order to include potential optional elements. In the second phase, called matching phase, the list of compared elements for each compare model is iterated to find the best match for each block of the base model. If multiple compare elements with the same block of either the base model or compare model exist, the similarity value is used to decide which match is best. If two or more compare elements have the same similarity value, their matching is deferred until the end, in hope that the algorithm solves the issue automatically by finding matches for the problematic blocks first. If the problems that occurred can not be solved automatically, a user is involved to manually solve them. If a block of the base model is unmatched at the end, it is again compared against null. In the third and final phase, called the merging phase, the obtained matching is used to create a 150% model, where different thresholds are used on the best matching compare elements in order to define mandatory, alternative or optional blocks. In order to show the feasibility of the approach, the authors apply the approach to the pick-and-place unit case study [29, 28].

2.3.2 Extraction And Composition For Clone-And-Own (ECCO)

The paper 'Enhancing clone-and-own with systematic reuse for developing software variant' [4] by Fisher et. al. proposes an approach for semi-automatically aiding the clone-and-own practice of creating new product variants. The authors identify three underlying steps of a clown-and-own approach, which form the basis of the proposed conceptual framework. An implementation based on the conceptual framework is presented by the authors in [5]. The first step, called the extraction step, is concerned with locating and extracting reusable artefacts from the existing set of variants. This step is automated and requires full knowledge of the features and the implementation artefacts that implement them for each variant. The result of this step is an initial version of a database that contains feature traces, an ordering of artifacts and dependencies between traces. In next step, called the composition step, engineers semi-automatically create a new product by selecting desired features from the database. The resulting products may or may not be complete, based on the previous occurrence of the selected feature combination in the input variants. In the final step, called completion, the incomplete products, created in the second step, are completed manually with the help of hints generated in the second step, indicating potentially missing features or feature interactions, or implementation artefacts that the automated approach was not able to separate into features and their interactions. The newly created product variants can be fed back into the first step to incrementally enhance the initial version of the database. The authors evaluated their tool based on five SPL case studies, with a range of 12 to 256 variants each. A random subset of these variants was fed into the tooling and and the remaining variants were automatically created based on their feature selection. The output variants were then compared with the original variants for similarity. For the evaluated case studies the tool was able to almost perfectly re-engineer the remaining variants with around 20 percent of the original variants as input [5].

2.4 SPLE Case Studies

In the paper 'ESPLA: A Catalog of Extractive SPL Adoption Case Studies' [15] Martinez et. al. compiled a catalog of case studies on the extractive approach to software product line adoption, that can be accessed online[6]. As of the time of writing, the catalog contains 135 different case studies, with mainly 4 different origins, namely 'Academic', 'Illustrative', 'Industry', and 'Open Source'. For the type of artefacts we only focus on the case studies working on models of some format, because our case study is also located in the modeling realm, but there are many case studies regarding different kinds of artefacts, e.g. code or requirements, with a part of them already being listed in the catalog. The illustrative case studies include the 'Banking System', which was used by Martinez et el. to illustrate the proposed EMF model adapter for the But4Reuse framework [16]. The size of this case study is rather small, with only 3 variants and an average size of 57 classes, 56 attributes and 25 references (atomic model elements, as defined in Section 2.1). Also used as an example on the But4Reuse framework [16], but originally published by Couto et. al. [3], is the ArgoUML SPL[7], obtained from source code of the open source modeling tool ArgoUML. A total of 9 UML models were extracted from the source code of the Argo UML variants and it is a bigger-sized case study, with an average of 51087 classes, 77519 attributes, and 28885 references. For a case study with academic background, the work of Schulze et. al. [26] evaluates introducing variability into functional safety models, by manually identifying features on an automotive airbag case study. For case studies with industrial background, we have to differentiate between case studies that were synthetically generated, based on industry use-cases or in cooperation with industry partners, and case studies that actually concern real-world, production data. For synthetic, industrial case studies, there are two prominent examples. The first of which is the body comfort system case-study [13, 20], a small-sized case study, set in the automotive domain, with state charts created in IBM Rational Rhapsody[8] and an

[6]https://but4reuse.github.io/espla_catalog/ESPLACatalog.html
[7]https://github.com/but4reuse/argouml-spl-benchmark
[8]https://www.ibm.com/products/uml-tools

average size of 187 elements per model and 18 variants in total. The second prominent example is the pick-and-place unit and its extended version [29, 28], both of which are not listed in the catalog. The original case study consists of 15 evolution scenarios of a pick-and-place unit, with SysML models available for each scenario and with a physical counterpart on site, to test the different scenarios. It is a small-sized case study, with an average of 435 classes, 643 attributes, and 327 references. Its extended version consists of 27 evolution scenarios, also with available SysML models. With an average of 1755 classes, 1770 attributes, and 1285 references for each model, it is a medium-sized case study. For real-world, industrial case studies, the authors of [7, 6] evaluate locating features on models obtained from BSH induction hobs. The models reside in a proprietary DSL called IHDSL, with the first study dealing with 46 variants and the second with 112 variants in total. From the average size given in the catalog, this seems to be a small-sized case study with somewhere between 100 and 500 elements for a given model.

CHAPTER 3

Background

In this chapter, we will take a look at the knowledge prerequisites for the remaining chapters of this thesis. First the concept of a software product line (short SPL) is introduced, alongside some key terminology in the field of software product line engineering (short SPLE), such as the definition of a product, features and the dependencies between them, the different areas of SPL-engineering, namely domain and application engineering, which are further split into two subdomains, domain analysis and domain implementation, and requirements analysis and product derivation, respectively. Afterwards, the notions of formal context and their formal concepts are introduced in addition to some of the basic terminology of formal concept analysis (short FCA).

3.1 Software Product Lines

The mass production of physical goods is often stated to have first been introduced with the advent of the industrial revolution and the introduction of automated machines into the manufacturing process and was driven even further with the introduction of standardized parts into the construction process and by moving production to an assembly line process. It allowed manufacturers to cover broad ranges of consumer markets and make physical goods available to the general public at an affordable price. But while a peak in efficiency was reached, it became apparent that with the ever decreasing possibility for customization, smaller market segments, with more specialized requirements, and the general crave of individual consumers for more individual choices were lost potential at best. It was this exact insight that spawned the idea of a *product line*, a way of reaping the benefits of mass production while maintaining the possibility of catering to different customer requirements by letting customers combine and configure predefined components and chose between different alternative components, with the goal of sharing as many components as possible between products. All while leaving leeway for potentially newly created components based on an individual customer's needs.

Figure 3.1.1 illustrates such a product line on an unusual yet intuitive example of Domino's pizza configurator 'Pizza Chef' that allows you to chose between different predefined sizes, crusts, sauces, and toppings. These ingredients are prepared and ready to be assembled easily once an order comes through, which allows for the use of employees with little to no culinary background to prepare and serve pizzas in a cost effective way, all while giving the customer an incredibly large yet manageable amount of choices.

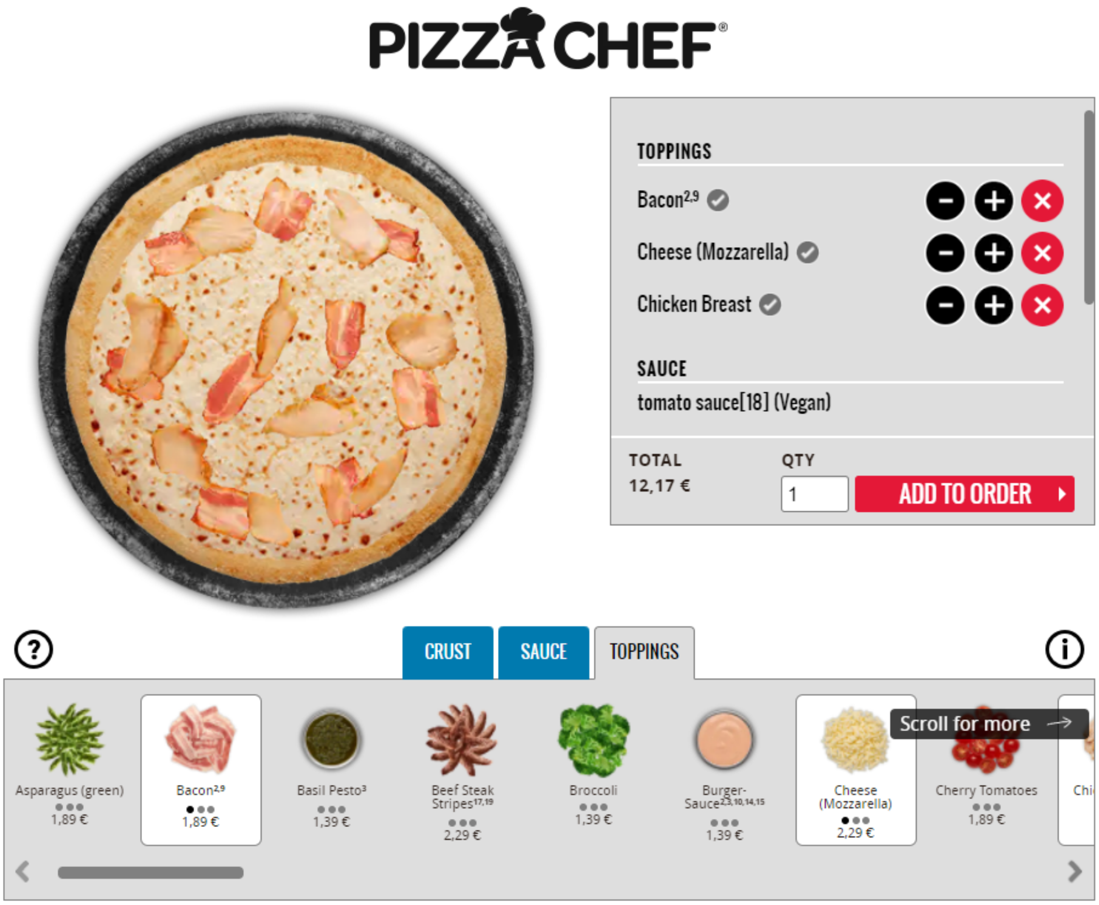
A very similar development as for physical goods can be observed for software products as well, leading to the development of *software product lines*. Adapting the definition of a product line given Apel et al. [1] to software, we define a *software product line* as *a collection of software products from a manufacturer's software product lineup that share major similarities between them and are, in the best case scenario, constructed from a collection of reusable parts.*

3.1.1 Benefits And Costs Of Software Product Lines

Software product lines offer various benefits over other approaches to software engineering but also come at some costs. A list of the most important benefits is given by Apel et al. [1] as:

- The ability to tailor products to individual customers instead of providing, at best, a few different product variants to a wide range of customers

- The reduction of costs that results out of the long run of using reusable components over the development of components or entire products from scratch



TOPPINGS

Bacon2,9

Cheese (Mozzarella)

Chicken Breast

SAUCE

tomato sauce[18] (Vegan)

TOTAL 12,17 €

QTY 1

Fig. 3.1.1: Domino's pizza configurator Pizza Chef

- The improvements in quality that arises from the ability to test individual components on their own and in integration in multiple different products
- The improved time to market that is the result of assembling a set of components that, in the best case scenario, is covered by the reusable components already in existence or is otherwise extended by new components depending on an individual customer's needs instead of developing from scratch

The approach of a software product line, however, comes at the cost of a high upfront investment to identify, design, and develop the reusable components and the parallel development of products, as well as the scale of the configuration space, may come at a further cost of management. In chapter 2.3 of their book 'Software Product Lines: Practices and Patterns' [2] Clements and Northrop give a more comprehensive overview of the costs and benefits of software product lines in general as well as their individual assets.

3.1.2 Feature Orientation

The notion of a feature is a key concept in software product line engineering. They are used as a kind of an intermediary between customers and their individual requirements and the functionalities a software product offers. The orientation towards them means to build the whole infrastructure of your software product line around them and to make them explicit in every process along the line, when possible. With the main goal being *feature traceability*, basically meaning to be able to follow a golden thread all the way from a customer's needs and wishes down to individual software artefacts. Because of this important role, it is hard to find a general definition of a feature that captures the views and knowledge level of different stakeholders of a product line at different stages of engineering. A list of prominent definitions ordered by degree of technicality is given by Apel et al. [1] and a new intermediary definition is given as '*A feature is a characteristic or end-user-visible behaviour of a software system. Features are used in product-line engineering to specify and communicate commonalities and differences of products between stakeholders, and to guide* *structure, reuse, and variation across all phases of the software life-cycle*'.

For the sake of this thesis, however, we will use yet another, more technical definition of a feature, which we will introduce in more detail in section 4.2.2. Our definition is based on formal concepts, which are introduced in more detail in Section 3.2. With the definition of a feature established, we define a *product* of a product line like Apel et al. [1], simply as a *valid selection* of features (more on that in Section 3.1.4).

3.1.3 Software Product Line Engineering (SPLE)

According to Apel et al. [1], the process of software product line engineering is split into two phases, namely *domain engineering* and *application engineering*. On the one hand, the first phase is concerned with scoping the *domain* ('a specialized body of functionality, an area of expertise, or a collection of related functionality' [2]). Basically, reasoning about which features should or should not be included and documenting it in a feature model. On the other hand, the phase is concerned with the creation of the reusable assets, according to those features, that the products of product line are finally constituted of. These two tasks are named *domain analysis* and *implementation* respectively and the goal of them is the '*development for reuse*' [1]. The second phase is concerned with analyzing the requirements of customers, ideally mapping them to the reusable assets constructed in the domain analysis or feeding them back into it if not. And finally, the creation of products using the reusable assets mapped to the identified required features. These two tasks are named *requirements analysis* and *product derivation* and the overall target of them is '*the development with reuse*' [1]. More detail on the individual tasks is given by Apel et al. in chapter 2 of [1], by Ryssel in chapter 2 of his PhD thesis [22], and by Clements and Northrop in chapter 4 of [2].

3.1.4 Feature Modeling

As mentioned earlier, in order to capture the variability of the product line identified during domain analysis, it has to be documented in some shape or form. We will be using a common approach called feature modeling, where the variability of the product line is defined in terms of common and optional features and the relationship and constraints between them are presented in a graphical manner called feature diagrams [1].

3.1.4.1 Feature Diagrams

In a feature diagram, features are presented as nodes of a tree, where in the most basic terms the parent-child relationship expresses the notion that the child feature can only be selected if the corresponding parent feature is selected as well. In order to convey mandatory or optional relationships between parents and their children, small circles on the children end of an edge are used, where an empty circle indicates an optional relationship and conversely, a filled circle indicates a mandatory relationship. Further, in a case where a parent has multiple children, but they can not be chosen freely, additional notation clarifies the relationship. A common case is a disjunctive relationship, where a filled half-circle on the parent end of an edge is used to convey an (inclusive) or-relationship and conversely, an empty half-circle conveys a exclusive or-relationship (alternative). In the case where there are additional constraints between features in different branches of the feature tree, edges across the tree (named accordingly as cross-tree constraints) may be introduced or the constraints are visualized in a textual manner somewhere in the diagram using, for example, propositional logic.

Figure 3.1.2 illustrates a feature diagram again on the example of Domino's pizza line-up that was created using FeatureIDE[1]. Note that for the sake of readability we only included the toppings that are used on the traditional pizzas. Also some constraints were left out for the same reason, notably, each topping can be added

[1]https://featureide.github.io/

up to three times (which could be implemented by splitting each topping into an OR-subtree with the topping as the abstract root and with three children named topping_1, topping_2, topping_3 respectively), and there can be only be a total of 11 toppings, including multiples of the same one (which could be implemented as a big disjunction over conjunctions of all sets of 11 toppings). One could also implement an extension of the notation for the above mentioned (a notation similar to the UML multiplicity notation would suffice in this case [21]).

Feature diagrams graphically represent logical relationships between features and can therefore be expressed in terms of propositional logic. Formally, given a set of propositional variables F defined as the set of all feature names, a parent feature $p \in F$, and child features $f, f_1, \ldots, f_n \in F$, the following relationships are defined by the notation introduced earlier [1]

$$\begin{aligned}
mandatory(p, f) &\mapsto p \Leftrightarrow f \\
optional(p, f) &\mapsto f \Rightarrow p \\
xor(p, \{f_1, \ldots, f_n\}) &\mapsto ((f_1 \vee \cdots \vee f_n) \Leftrightarrow p) \wedge \bigwedge_{i<j} \neg(f_i \wedge f_j) \\
or(p, \{f_1, \ldots, f_n\}) &\mapsto (f_1 \vee \cdots \vee f_n) \Leftrightarrow p
\end{aligned}$$

3.1.5 Adoption Of Software Product Lines

The specifics of adopting a software product line approach heavily depend on the current status-quo for an individual company regarding the products they offer and therefore there are many different approaches that might fit your specific use-case. Despite their individual differences the different approaches can generally be grouped into three main categories [1, 14]. The first and probably most straight-forward category is the *proactive* approach to SPLE, meaning to proactively develop the entire product line from scratch, following the steps introduced earlier. While this path will most likely result in the highest quality and most maintainable end-result [1], it also has some potentially insuperable drawbacks as mentioned in section 3.1.1. The

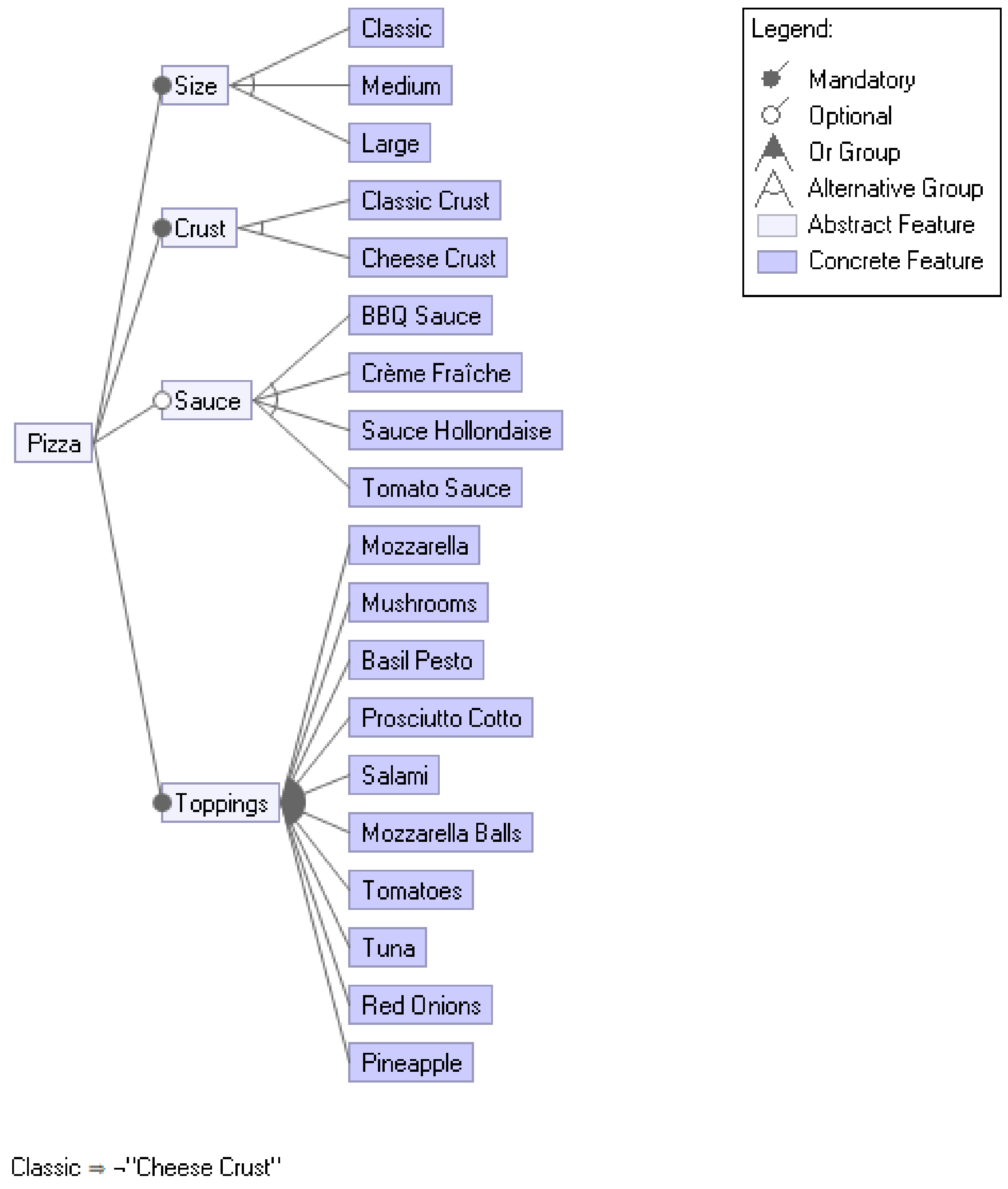


Fig. 3.1.2: Feature diagram for Domino's pizza line-up (only including traditional toppings)

second group of approaches is concerned with dynamically developing the product line by iteratively building upon an initial version, adapting it to products that were originally envisioned, but are not yet implemented, or entirely new products that meet additional customer requirements that have sprung up along the line. It is this adaptive behaviour that leads them to be categorized as *reactive* approaches. While the reactive approach can be used to develop a product line for the first time, it can also be used to increment upon an already existing product line created using another approach. The third and final category is concerned with extracting a product line out of an already existing set of products that share commonalities but were developed more or less independently and inconsistently and these approaches are therefore called *extractive* (or bottom-up) approaches. As it is the goal of this thesis to investigate the use of formal concept analysis for the tasks involved in these approaches we will give more detail on this type of adoption path.

3.1.5.1 Extractive Approach To SPLE

While in an ideal world software engineers would stick to the guidelines given by researches and alike in literature, unfortunately due to time and money constraints it is almost common practise to cut corners in favor of short term benefits in exchange for quality and the lack of maintainability in the long run. One of such approaches is appropriately called the ‘clown and own’ approach where in its most literal sense, source code is copy and pasted from one product to another and from there on developed independently from its original host. This approach makes sense at first, if the generated variants are supposed to be developed independently in the future. But practise has shown that even in these cases, implementation artefacts are shared and development effort is repeated needlessly in all variants. For a small product portfolio the approach is still feasible, but with an increase in product variants comes a point where the overhead of maintenance and increasing development costs outweigh the benefits of continuing unabated. The different tasks associated with an extractive approach to SPLE are outlined by Martinez in Section 2.2.2 of [14] and the ones we will focus on are defined as follows. As a first step into the direction of a product line, to which we will refer to as *feature identification*, the commonalities and variability between all the product variants of the current product portfolio - if undocumented - have to be found or if there is already some documentation for them, the implementation artefacts associated with them have to be located in the product variants (which is referred to as *feature location*). Note that the typical definition for feature identification only refers to, as the name suggests, the act of identifying the features present in the product port-folio overall. The feature identification step of But4Reuse as described in Section 4.2.2, however, is a mixture between feature identification and feature location, in the sense that it finds and locates the features in the variants at the same time. Somewhat confusingly there is also a separate extension point in the tool for feature location, which is used only if the features are already known in advance, as mentioned above. After the identification step, in order to ensure the validity and to restrain the configuration space, the constraints between the features have to be identified, in a task we will refer to as *constraints discovery.* Finally, together with the features, these constraints have to be documented in a feature model, as mentioned in section 3.1.4, which can be done manually or automatically, to which we will refer to as *feature model synthesis.* The specific implementation of these individual tasks is subject of section 4.2.2.

3.2 Formal Concept Analysis

Formal concept analysis (FCA) is a field of applied mathematics that originated from the attempt to restructure order and lattice theory. Only later the connections to concepts of human thought were discovered with the aim of the field today being '[...] *to support the rational communication of humans by mathematically developing appropriate conceptual structures which can be logically activated.*' [8]. The field deals with two basic notions, formal concepts and formal contexts. The adjective 'formal' is supposed to help to make the mathematical notions stand out from their meaning in every day language [9].

A *formal context* is a triplet defined as (O, A, I) consisting of two sets O, called the (formal) *object set*, and A, called the (formal) *attribute set*, and a binary relation $I \subseteq O \times A$ between them, called the *incidence relation*. In order to define a *formal concept* of a formal context (O, A, I) we define the following two derivation operators for any $X \subseteq O$, $Y \subseteq A$:

$$X \mapsto X^I := \{a \in A \mid (o, a) \in I \ \forall o \in X\}$$
$$Y \mapsto Y^I := \{o \in O \mid (o, a) \in I \ \forall a \in Y\}$$

Now a formal concept of formal context (O, A, I) is defined as a pair (X, Y) with $X \subseteq O$, $Y \subseteq A$, $X = Y^I$, and $Y = X^I$. The sets X and Y are called the *extent* and *intent* of the formal concept (X, Y), respectively. Given a formal context (O, A, I), and $X, X_1, X_2 \subseteq O$, and $Y, Y_1, Y_2 \subseteq A$ the following propositions hold

$$\begin{aligned} &X_1 \subseteq X_2 \Rightarrow X_2^I \subseteq X_1^I \qquad && Y_1 \subseteq Y_2 \Rightarrow Y_2^I \subseteq Y_1^I \\ &X \subseteq X^{II} && Y \subseteq Y^{II} \\ &X^I = X^{III} && Y^I = Y^{III} \end{aligned}$$

for which the proof is trivial and is given by Ganter et al. [9]. Further given two concepts $C_1 := (X_1, Y_1)$, $C_2 := (X_2, Y_2)$, we call C_1 a *subconcept* of C_2 if $X_1 \subseteq X_2$ (note that this is equivalent to $Y_2 \subseteq Y_1$ which follows from $X_1 \subseteq X_2 \Rightarrow X_2^I \subseteq X_1^I$ where $X_2^I = Y_2$ and $X_1^I = Y_1$). Subsequently C_2 is a *superconcept* of C_1 which we denote as $C_1 \leq C_2$, the *hierarchical order* of concepts. The partially ordered set of all formal concepts of (O, A, I) is denoted as $\underline{\mathfrak{B}}(O, A, I)$ and is called the *(formal) concept lattice* of the context. For any object $o \in O$ the smallest concept in $\underline{\mathfrak{B}}(O, A, I)$ whose extent contains o is called the *object concept* $\gamma o := (\{o\}^{II}, \{o\}^I)$ of o and for any attribute $a \in A$ the largest concept in $\underline{\mathfrak{B}}(O, A, I)$ whose intent contains a is called the *attribute concept* $\mu a := (\{a\}^I, \{a\}^{II})$ of a. According to the basic theorem on concept lattices, for which the definiton and a proof is given by Wille and Ganter [9], the concept lattice $\underline{\mathfrak{B}}(O, A, I)$ is a complete lattice for which the infimum and

pizza	mozzarella	mushrooms	basil pesto	prosciutto cotto	salami	mozzarella balls	tomatoes	tuna	red onions	pineapple
Salami	X				X					
Margherita	X									
Funghi	X	X	X							
Caprese	X		X			X	X			
Prosciutto e Funghi	X	X		X						
Tuna	X							X	X	
Waikiki	X			X						X

Table 3.2.1: Formal context for Domino's traditional pizzas

supremum are given by

$$\inf = \bigwedge_{t\in T}(X_t, Y_t) = (\bigcap_{t\in T} X_t, (\bigcup_{t\in T} Y_t)^{II})$$
$$\sup = \bigvee_{t\in T}(X_t, Y_t) = ((\bigcup_{t\in T} X_t)^{II}, \bigcap_{t\in T} Y_t)$$

for an index set T where $\forall t \in T, X_t \subseteq O$ ($Y_t \subseteq A$). The best way to understand the notions of a formal context and its formal concepts is to visualize them. The most intuitive way to depict a formal context on the one hand is to use a cross table.

Table 3.2.1 illustrates a formal context for Domino's traditional pizza line-up, where the object set $O :=$ {Salami, Margherita, Funghi, Caprese, Prosciutto e Funghi, Tuna, Waikiki} is represented as the rows, and the attribute set $A :=$ {mozzarella, mushrooms, basil pesto, prosciutto cotto, salami, mozzarella balls, tomatoes, tuna, red onions} (only considering toppings) is represented by the columns, and where each 'X' indicates that $(o, a) \in I, o \in O, a \in A$.

On the other hand for visualizing formal concept lattices it is common practice

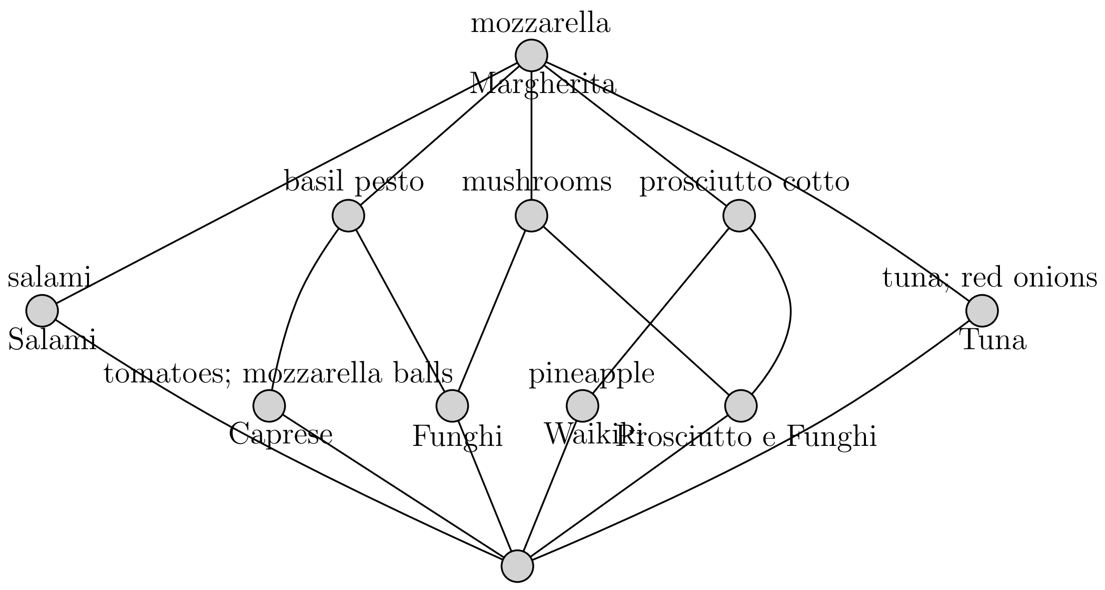


Fig. 3.2.1: Formal concept lattice for the formal context of Table 3.2.1

to use a labeled line graph. The one for the formal concept lattice of Table 3.2.1 can be found in Figure 3.2.1. Here each circle represents a formal concept of the formal context, each object is written below its object concept and each attribute is written above its attribute concept. The intents of each concept can be read from the diagram and consist of all attributes whose attribute concept is reachable by ascending a path from the circle corresponding to the concept. The same applies to the extent, consisting of all the objects whose object concept can be reached by descending a path from the corresponding circle.

An important aspect of the concept lattice is that we can obtain all attribute implications from it, where $A_1 \Rightarrow A_2$ for two sets of attributes $A_1, A_2 \subseteq A$ if $A_2 \subseteq A_1^{II}$ or rather $(A_1^I, A_1^{II}) \leq (A_2^I, A_2^{II})$. Visually we can read this off the diagram by checking whether or not for all $a_2 \in A_2$ the attribute concept μa_2 is above the infimum of all attribute concepts μa_1 for $a_1 \in A_1$.

CHAPTER 4

Methodology

In this chapter we will present the methodology used for our thesis. First, we present our research questions. Next, we discuss our experimental setup, where we first talk about our hardware setups, then about the different implementations of But4Reuse that we will be using, and finally we present the case study that is the subject of this thesis. Finally, we discuss how we plan on answering our research questions.

4.1 Research Questions

Our research questions are aimed at evaluating the usability of a lightweight, semi-automated approach to extractive software product line engineering on a production-scale, real-world set of product variants, in our case specifically, an approach based on formal concept analysis as discussed in Chapter 4.2.2. Contrary to tailor-made case studies that are often used to evaluate similar approaches such as the (extended) pick and place unit [29] (see Chapter 2.4). The first research question **RQ.1** is concerned with reasoning about the overall usability of the FCA based approach as an out-of-the-box first step in the incremental extractive adoption of a software product line. As noted by Apel et al. [1] 'the quality of the extracted product line relies on the quality of the tools supporting the extraction' and it is the goal of this research question to at least partially evaluate said quality. The second research question **RQ.2** is concerned with reasoning about the feasibility of an automated approach applied to a set of real-world products that grow beyond the scale of usual toy examples as mentioned above. The third and final research question **RQ.3** is concerned with finding ways to optimally present the results of such an semi-automated approach to the engineers, in order to evaluate similar approaches or maximize the usefulness of the results, when user decisions are wanted in field application. Summarizing, our list of research questions is as follows

RQ.1 Is the semi-automated approach based on FCA able to assist as a starting point in the different tasks associated with an extractive approach to SPLE?

RQ.2 Is the semi-automated approach based on FCA feasible, when applied to a production scale, real-world set of product variants?

RQ.3 How can we represent the results of the semi-automated approach in a way that maximizes their usefulness to the engineers?

4.2 Experimental Setup

4.2.1 Hardware Setup

For our experiments we ran two different hardware setups, the first one was used to run the But4Reuse tool on the data obtained from our case study and the second one carries the license for the modeling tool MagicDraw[1] in which the case study resides and was used to evaluate the results. The first hardware setup consists of a MSI B550-A Pro motherboard equipped with an AMD Ryzen 5600X processor, a six-core processor capable of handling up to 12 threads and that runs at a base clock of 3,7 GHz with its peak clock at 4,65GHz. The setup also runs 32 GB of DDR4-3600 RAM with 16-18-18-38 timings and a Samsung 970 Evo Plus 1 TB SSD connected via NVMe (PCIe 4x 8.0 GT/s). The second hardware setup is a HP Pro X2 equipped with an Intel Core i5-7Y57, a dual-core processor with a base clock of 1.10 GHz and peak clock at 1,61 GHz, 8 GB of LPDDR3-1866 RAM and a 512 GB Toshiba XG5 connected via NVMe.

[1]https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/

4.2.2 Feature Identification With Formal Concept Analysis

In this subsection we will take a look at the specific implementations of three extension points of But4Reuse as given in Figure 2.1.1, namely feature identification, feature constraint discovery, and feature mode synthesis respectively, that were used in this thesis. We used the latest available version of the tool at the time (commit *b39799271ca562b400118b45883088c84ac689c5*) for our work.

4.2.2.1 Feature Identification

The input for feature identification are the AMEs produced by the chosen artefact adapter as mentioned in Section 2.1. For our case we chose the EMF model adapter, which produces three different kinds of atomic model elements, namely class, attribute and reference elements, also as described in Section 2.1. For the specific feature identification implementation we chose the one using formal concept analysis. Here, all atomic model elements of the adapted input variant models are iterated and grouped according to a chosen hashing algorithm, for which we chose the one based on XML identifiers. Here, class elements are hashed based on their own XML identifier and attribute and reference elements are hashed based on their owners XML identifier. If two elements end up with the same XML identifier, they are further compared for similarity using a comparison function provided via the EMF DiffMerge[2] API. In our case we use the default boolean similarity metric provided by the framework as described by Martinez in section 5.3.3 of [14]. Now a formal context using a rudimentary external FCA implementation[3] is created, where each group of elements is considered as an attribute, the objects are the input artefact variants, and the incidence relation is created using the union of the source artefacts of each element in a respective group. Then a pruned version of the formal concept lattice is computed using the Ceres algorithm [12], that only contains the attribute and object concepts (also called AOC-poset). Now our features are defined as the union of the groups of elements that ended up in the same attribute concept. (As noted by Martinez [14]

[2]https://wiki.eclipse.org/EMF_DiffMerge
[3]https://github.com/jrfaller/galatea

this seems to be equivalent to the definition of a feature based on the interdependence relation on the atomic model elements given by Martinez et al. [19]).

4.2.2.2 Feature Constraint Discovery

The input to the feature constraint discovery are the features identified in the step introduced earlier. For this step we will again chose the implementation based on formal concept analysis. As of now the framework does not rely on the specifics of implementations of prior steps, so another formal context is created using the features identified earlier as attributes, the input variants as objects, and the incidence relation is created using the location information of features on the different variants calculated earlier. As previously a pruned version of the concept lattice is calculated from which parent-child feature implications of the form $F_1 \Rightarrow F_2$ are obtained as described in Section 3.2. Additionally mutual exclusion relations between features are found using a brute-force approach that checks for every pair of attributes, if they are both part of the intent of a common attribute/object concept or not.

4.2.2.3 Feature Model Synthesis

The input to the feature model synthesis step are the identified features alongside the constraints identified the previously described step. For this step we chose the implementation called 'Alternatives before Hierarchy' which implements the feature models as feature diagrams in FeatureIDE and is described in the source code as '*First we identify alternative groups, then we create the hierarchy. From the possible parent candidates, we discard those that are already containing other parent candidates. The parent candidates of an alternative group are the intersection of the parent candidates of the features integrating the group. In case of several parent candidates, we select parent candidates belonging to alternative groups and in the case of any, we select the parent candidate with the higher number of reasons in the requires constraint description. Finally, the features without parent are added to the root. The common features are set as mandatory and redundant constraints are removed.*', where parent candidates are defined from the requires constraints by the mapping $optional(p, f) \mapsto f \Rightarrow p$ introduced in Section 3.1.4.1 and alternative groups are calculated by brute-force from the mutual exclusion constraints by the mapping $xor(p, \{f_1, \ldots, f_n\}) \mapsto ((f_1 \vee \cdots \vee f_n) \Leftrightarrow p) \wedge \bigwedge_{i<j} \neg(f_i \wedge f_j)$. Note that because from the previous FCA-based constraint discovery step there is only ever one reason for a parent-child relationship, the first suitable parent candidate is used as a parent for a child feature. Additionally, the given implementation did not terminate within a feasible time frame (after three hours we aborted the run), therefore we optimized the implementation by pre-calculating look-up tables for calculations that were needlessly performed with every iteration of the previous implementation. We validated our edits against the results obtained from the pick-and-place case study [29] and achieved a substantial increase in performance.

4.2.3 Case Study

In this subsection we will describe the case study that was used for our experiments.

Our case study is a real-world, industrial, clone-and-own product line, located in the railway domain. The product line-up includes a variety of different trains for different application scenarios, including trams, high-speed trains, locomotives, and commuter rails. The different kind of trains are composed of many different software artefacts, which mainly consist of Structured Text (ST) code. The engineering team decided to migrate to a model-driven engineering approach, in order to overcome the increasing size and complexity of the code bases and to aid in automating parts of the documentation for the highly regulated train domain. As the modeling tool of choice, they use MagicDraw. Using MagicDraw, large parts of the code base are generated out of the SysML models. For this, a custom plugin is used to export the models into a custom EMF meta model, in order to make use of the existing EMF ecosystem. From there the models are transformed into ST code. The software components are grouped into common part systems and kept as projects in a version control system, where there is one branch for each trainset, for example., there is one project for components related to interior lighting, with a total of 12 variants

part system	# variants	min	max	mean	median	std. deviation
0	10	422039	619521	545825	579658	63752.31
1	11	441285	630962	552769.64	594327	63451.78
2	11	431647	627891	546709.27	579569	67479.49
3	12	389469	608655	541736	582720	68476.49
4	11	461525	649136	567489.36	604272	64004.62

Table 4.2.1: Statistics for the AMEs of the five part systems we chose

in separate branches. One of the variants for each part system, referred to as the 'common' or 'trunk' variant, is an in-development software product line, following a manual, extractive SPLE approach, where features are identified in the other variants and merged into the common variant on a regular basis. There is a total of 59 part systems, from which we randomly chose five for our study. Table 4.2.1 shows the amount of variants (including the trunk variant) and statistics for the amount of atomic model elements found in the part systems we chose. To keep consistency between part systems, variants belonging to the same trainset will be referred to with the same number (starting with 0), in particular, the trunk variant will be referred to as the 11th variant (even though, e.g., part system 0 only has 10 variants).

The part systems comprise the following functionalities. The 0th part system comprises functionality related to the sanitary facilities (Note that we chose the index as 0, because we did not have time to incorporate this part system fully into our study, in particular RQ.1). The part system 1 comprises the functionality of the drive and brake controller. The part system 2 contains the functionality of the fire alarm system. The part system 3 comprises functionality regarding the interior lighting and the final part system 4 includes the functionality regarding the HVAC unit. As mentioned above, a custom EMF meta model is used to extract the code from the SysML. Unfortunately, trying to use the exporter for the part systems we chose resulted in model errors that caused unresolvable problems when trying to import the models into BUT4REUSE. We assume that these error do not pose a problem for the engineers at this point in time, or that they are resolved manually. For this reason we chose the UML exporter of MagicDraw, exporting into the Eclipse UML 2.5 meta model[4].

4.3 Operationalization

In this section we will go into more detail on each individual research question and show how we plan to answer each of them respectively.

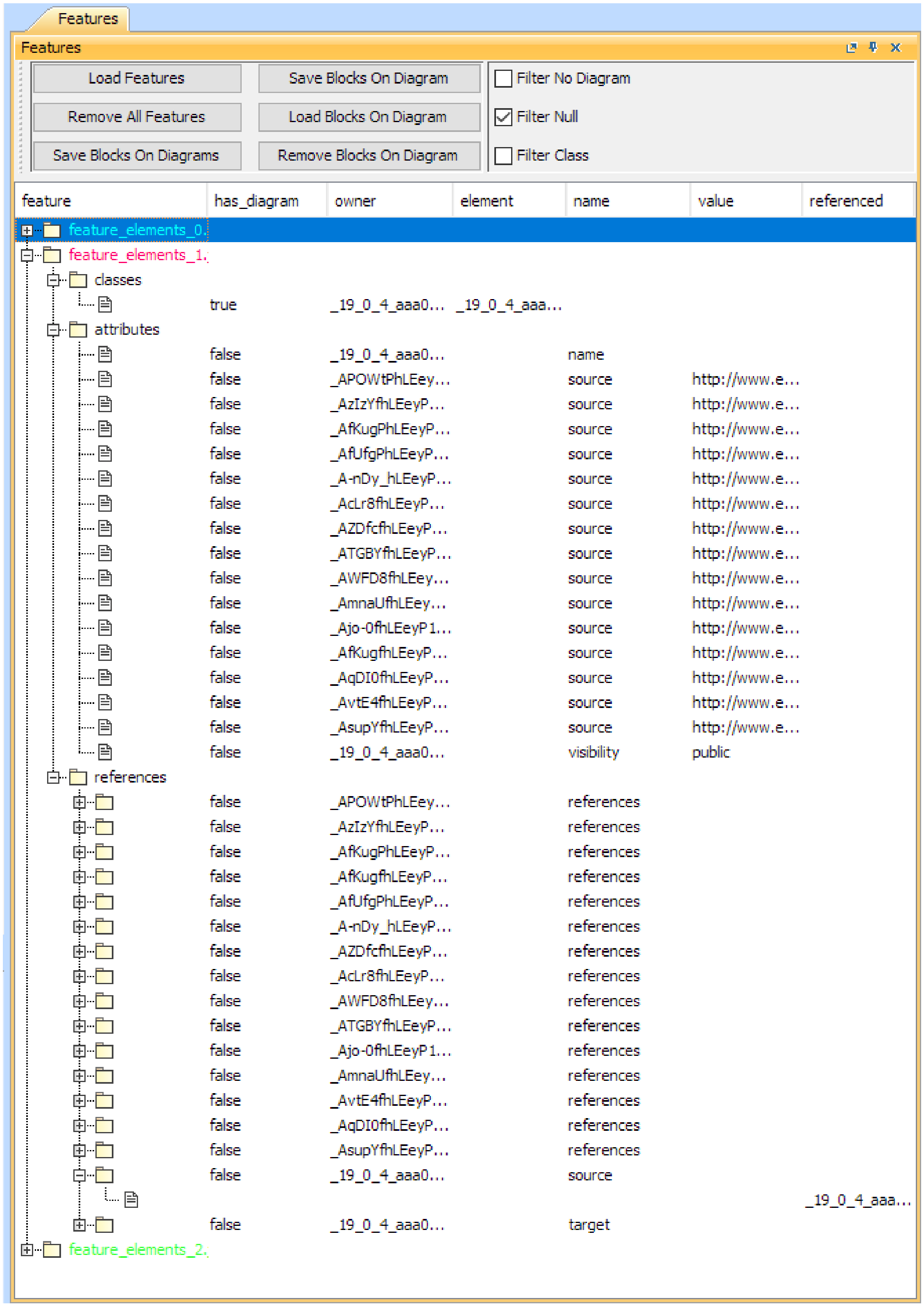
4.3.1 RQ.1 Is the semi-automated approach based on FCA able to assist as a starting point in the different tasks associated with an extractive approach to SPLE?

In order to assess the quality of the features identified by the approach as described in Section 4.2.2 we conducted a focus group together with domain experts in the domain of our case study. The focus group consisted of us, two lead architects, with 10 years of job experience and 6 years of experience within the projects, and the head of tool development, with 15 years of job experience and 6 years of experience within the projects. For assessment we first needed a way of visualizing the enormous amounts of data obtained from applying the approach to our case study. Therefore we extended But4Reuse with a way of serializing the identified features into JSON files. Figure 4.3.1 illustrates such a JSON file, where for each type of AME there is a separate JSON object representation constituted of the properties as described in Section 2.1. Using this external representation for the AME we created a plugin for MagicDraw that loads this representation in form of a tree table as illustrated in Figure 4.3.2. If a class element has a representation in a diagram, the respective element is also highlighted in the diagram with a pre-selected color that can be changed at will as shown in Figure 4.3.3. Hovering over a highlighted element also shows the variants the element can be found in and which feature it is part of. The functionality of the different UI elements are as follows. The buttons 'Load Features' and 'Remove All

[4]https://wiki.eclipse.org/MDT/UML2

```json
{
 "elements": [
  {
   "class": {
    "owner": "_18_0beta_903028d_1388650190935_398212_121847",
    "eObject": "_18_0beta_903028d_1388650183729_356902_85834"
   }
  },
  {
   "attribute": {
    "owner": "_18_5_1_a560294_1504789141594_489453_279391",
    "name": "aggregation",
    "value": "composite"
   }
  },
  {
   "reference": {
    "owner": "_18_0beta_903028d_1388650189740_615232_115906",
    "name": "instance",
    "referenced": [
     "_18_0beta_903028d_1388650183133_769902_82280"
    ]
   }
  }
 ]
}
```

Fig. 4.3.1: Exported feature elements from But4Reuse in JSON form



Filter No Diagram
[✓ Filter Null]
Filter Class

feature	has_diagram	owner	element	name	value	referenced
feature_elements_0.						
feature_elements_1.						
classes						
	true	_19_0_4_aaa0...	_19_0_4_aaa...			
attributes						
	false	_19_0_4_aaa0...		name		
	false	_APOWtPhLEey...		source	http://www.e...	
	false	_AzIzYfhLEeyP...		source	http://www.e...	
	false	_AfKugPhLEeyP...		source	http://www.e...	
	false	_AfUfgPhLEeyP...		source	http://www.e...	
	false	_A-nDy_hLEeyP...		source	http://www.e...	
	false	_AcLr8fhLEeyP...		source	http://www.e...	
	false	_AZDfcfhLEeyP...		source	http://www.e...	
	false	_ATGBYfhLEeyP...		source	http://www.e...	
	false	_AWFD8fhLEey...		source	http://www.e...	
	false	_AmnaUfhLEey...		source	http://www.e...	
	false	_Ajo-0fhLEeyP1...		source	http://www.e...	
	false	_AfKugfhLEeyP...		source	http://www.e...	
	false	_AqDI0fhLEeyP...		source	http://www.e...	
	false	_AvtE4fhLEeyP...		source	http://www.e...	
	false	_AsupYfhLEeyP...		source	http://www.e...	
	false	_19_0_4_aaa0...		visibility	public	
references						
	false	_APOWtPhLEey...		references		
	false	_AzIzYfhLEeyP...		references		
	false	_AfKugPhLEeyP...		references		
	false	_AfKugfhLEeyP...		references		
	false	_AfUfgPhLEeyP...		references		
	false	_A-nDy_hLEeyP...		references		
	false	_AZDfcfhLEeyP...		references		
	false	_AcLr8fhLEeyP...		references		
	false	_AWFD8fhLEey...		references		
	false	_ATGBYfhLEeyP...		references		
	false	_Ajo-0fhLEeyP1...		references		
	false	_AmnaUfhLEey...		references		
	false	_AvtE4fhLEeyP...		references		
	false	_AqDI0fhLEeyP...		references		
	false	_AsupYfhLEeyP...		references		
	false	_19_0_4_aaa0...		source		
						_19_0_4_aaa...
	false	_19_0_4_aaa0...		target		
feature_elements_2.						

Fig. 4.3.2: Screenshot of the MagicDraw Plugin

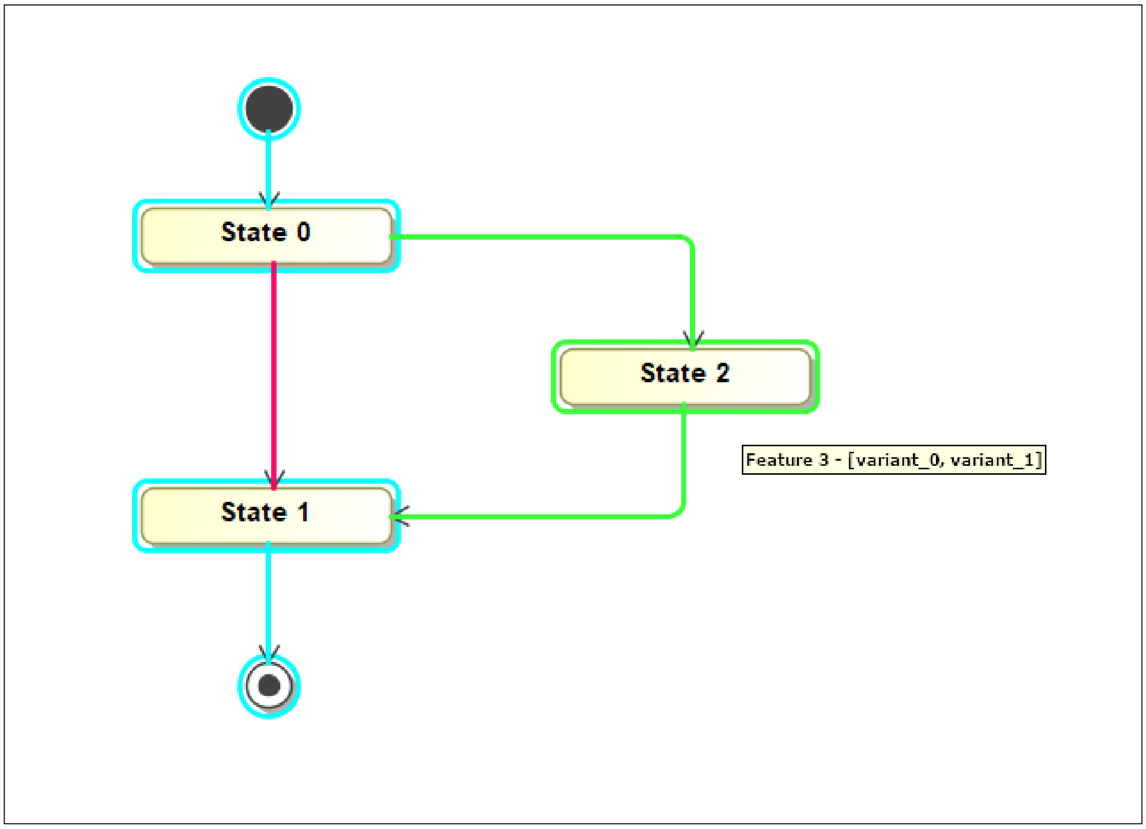


Fig. 4.3.3: Model Highlighting and Tool-Tip

```
{
 "diagrams": [{
  "name": "State Machine",
  "id": "_18_0_4_a560294_1473410836967_542901_131716",
  "blocks": [352, 0, 35, 71, 137, 76],
  "type": "SysML Internal Block Diagram"
 }]
}
```

Fig. 4.3.4: Diagram Info as JSON

Features' are used to load and remove the JSON file representations of the features outputted from But4Reuse. Loading a set of files creates a tree table representation of them. Here the root node displays the color of the highlighting in the diagrams and is labeled by default with the file name it belongs to, but the name can be changed by double clicking the node in case an appropriate name for a feature can already be found. The columns 'owner', 'element', 'name', 'value', and 'referenced' contain the properties of the AMEs as mentioned above, where 'owner', 'element', and 'referenced' additionally act as a hyperlink into the internal containment browser of MagicDraw. The column 'has_diagram' indicates if an element is contained in a diagram, calculated by checking each diagram of a project for whether or not the respective element is contained in it. The check boxes 'Filter No Diagram', 'Filter Null', and 'Filter Class' enable different filter option when loading features. The first one only loads all elements that are contained in a diagram (note that attributes and references are never part of a diagram). The second one only loads elements for which the MagicDraw API returns an actual element when queried with the id. The API might return null in the case you load a feature that is not part of the current product variant or for noise and/or garbage introduced by the UML exporter. The third and final filter enables only loading class elements. The button 'Save Blocks On Diagrams' creates a JSON representation of each diagram of the current project that contains the name, id, type, and list of all blocks (parts of features) present in the diagram from the features previously loaded as shown in Figure 4.3.4. The buttons 'Save Blocks On Diagram' is used to create a separate JSON representation of each block

```json
{
 "projectName": "Important Project",
 "projectId": "_19_0_4_aaa027b_1653918633032_178354_263888",
 "diagramName": "State Machine",
 "diagramId": "_18_5_1_a560294_1523271634782_481899_531745",
 "feature": 26,
 "elements": [
  "_18_5_1_a560294_1511970033163_774696_319757",
  "_18_0_4_a560294_1487851924818_839411_220151"
 ]
}
```

Fig. 4.3.5: Exported block of diagram as JSON

on the currently active diagram where the ids of each element are exported alongside the project name and id, the diagram name and id, and the feature id as depicted in Figure 4.3.5. Finally the buttons 'Load Blocks on Diagram' and 'Remove Blocks On Diagram' are used to load and remove the previously exported blocks per diagram using the project and diagram info to check if the correct diagram is opened.

The tree table representation of the AMEs does not allow for a feasible navigation for a large amount of data and because we could not come up with another way of visualizing elements that are not part of a diagram, we decided to only focus on elements that are part of diagrams for the sake of this thesis, which completely excludes all attributes and references. We conceived three primary dependent variables that will be used to evaluate three different aspects of a block (part of a feature present) on a diagram

1.1 Coherency - With this variable we want to test how coherent the block in question is in terms of functionality from the point of view of a domain expert. That is, does the block only contain functionality that should be considered as one or does it contain elements that belong to different functionalities.

1.2 Completeness - With this variable we want to test how complete the block in question is in terms of functionality from the point of view of a domain expert. That is, does the block contain all the elements that belong to the functionality(ies) it represents or are there elements missing.

1.3 Variant Specificity - With this variable we want to test how specific the block in question, or rather the functionality it represents, is to the variants it was identified to belong to. That is, does the block only belong to the the identified variants or are there too many or too few variants.

With the list of dependent variables in mind we created a list of independent variables that can be checked against the dependent variables as follows

1. The size of a block in terms of elements
2. The number of blocks in a single diagram
3. The number of variants the blocks of a diagram appear in
4. The size of the diagram in terms of elements
5. The type of diagram
6. The sparsity of the formal context

Unfortunately, due to the strict time limitation of the focus group and the hardware limitations of the second hardware setup (it takes around five to ten minutes to load a single MagicDraw project of that size and only around five projects can be open at a time with the limited amount of RAM), we had to chose the diagrams first, in a way that limits the amount of different projects we need to have open at a time and the amount of diagrams we can incorporate. Therefore our control over the different attributes of individual blocks was fairly limited. We compiled a list of additional independent variables that we did not find the time to incorporate and that rely more on the attributes of individual blocks in Section A.1. In order to chose the diagrams that were presented to the domain experts, we first used the block information obtained from the diagrams, as shown in Figure 4.3.4, to group the diagrams of each part systems into fifteen bins, according to the number of blocks it is constituted of, as shown in Figure 4.3.6. Afterwards, we manually tried to pick the

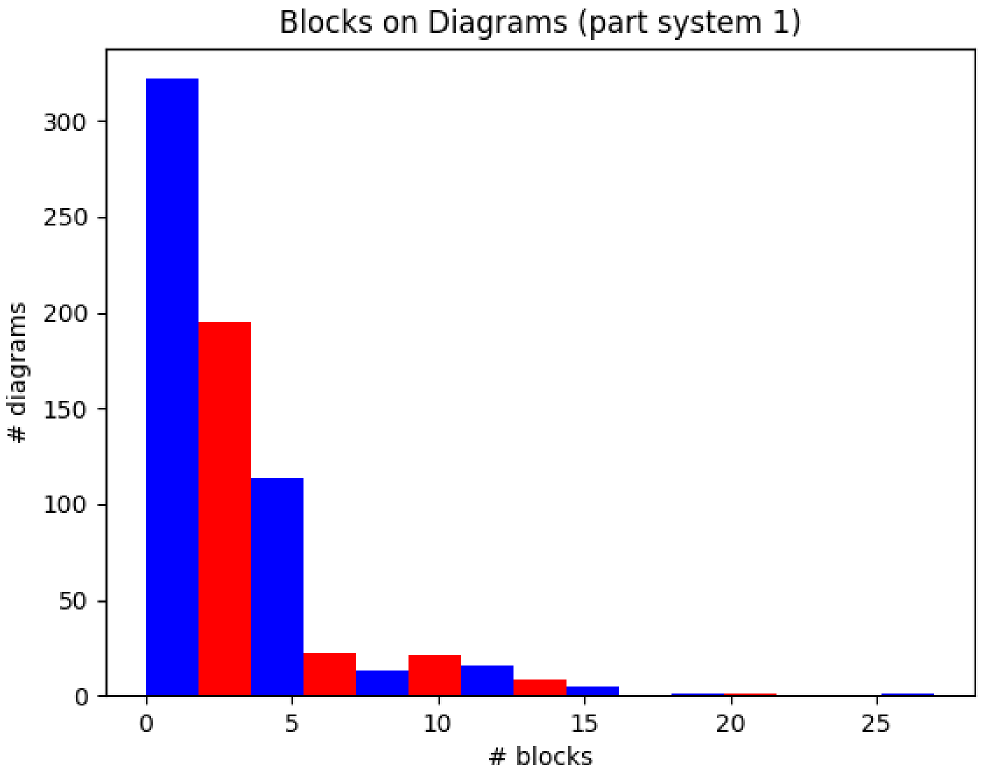


Fig. 4.3.6: Histogram for diagrams binned by number of blocks

diagram	# blocks	# elements	type	part system	# variants
1	14	254	SysML Internal Block	1	11
2	8	95	SysML Internal Block	1	11
3	13	317	SysML Internal Block	1	10
4	14	109	SysML Block Definition	1	11
5	3	146	SysML Activity	1	6
6	15	303	SysML Internal Block	1	11
7	4	179	SysML Activity	2	7
8	15	128	SysML Internal Block	2	11
9	8	127	SysML Internal Block	2	9
10	4	94	SysML Block Definition	3	10
11	5	45	SysML Use Case	3	12
12	11	136	SysML Internal Block	4	11
13	5	67	SysML Activity	4	9
14	20	232	SysML Internal Block	4	11
15	4	64	SysML State Machine	4	9

Table 4.3.1: Diagrams chosen for the focus group

diagrams from each part system and bin respectively, so that the spectrum of possible diagrams is sufficiently covered. With this approach we ended up with fifteen different diagrams as shown in Table 4.3.1.

With the diagrams set, we continued picking the blocks that were presented to the domain experts. For this we decided to chose blocks in three different ways. Firstly, manually selecting a block based on our gut feeling with no domain knowledge. Secondly, randomly selecting block from the set of available blocks. And lastly, making up a block, that consists of one or more merged blocks, where elements were randomly added or removed, purely based on the ids of the elements of a block gathered, as shown in Figure 4.3.5. The idea here was that the latter two act as a control group each, where ideally in the end the manually and randomly selected blocks perform statistically equally well and the made-up blocks perform significantly worse than the other two types. We ended up with the assortment of blocks visible in Table 4.3.2.

block	type	# elements	diagram	variants
1	manual	105	1	0, 3, 4, 5, 6, 7, 8
2	made-up	36	1	0, 1, 2, 3, 4, 5
3	random	4	1	0, 1, 3, 4, 5, 6, 7, 9
4	manual	35	1	0, 1, 3, 4, 5, 6, 7, 8, 9
5	manual	28	2	0, 2, 3, 4, 5, 6, 7, 8
6	manual	32	2	0, 3, 4, 5, 6, 7, 8
7	random	6	2	3, 4, 5, 6, 7, 8
8	made-up	11	2	4, 5, 6, 7
9	manual	28	3	0, 2, 3, 4, 5, 6, 7, 8
10	manual	133	3	0, 2, 3, 4, 5, 6, 7
11	random	92	3	0, 2, 3, 4, 5
12	made-up	22	3	3, 4, 5, 6, 7, 8, 9
13	manual	45	4	0, 2, 3, 4, 5, 6, 7, 8
14	random	27	4	0, 3, 4, 5, 6, 7, 8

15	made-up	9	4	0, 2, 3, 4, 5, 6, 7
16	random	90	5	0, 4, 5, 6, 7
17	manual	50	5	0, 4, 5
18	manual	26	6	0, 1, 3, 4, 5, 6, 7, 8, 9
19	manual	88	6	0, 2, 3, 4, 5, 6, 7, 8
20	random	63	6	0, 3, 4, 5, 6, 7, 8
21	made-up	7	6	0, 2, 3, 6, 7, 8, 9
22	manual	48	7	0, 3, 4, 5, 6, 7
23	random	84	7	0, 3, 4, 5
24	manual	35	8	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
25	made-up	13	8	0, 1, 9
26	manual	74	8	0
27	random	3	8	1, 2, 3, 4, 5, 8, 9
28	random	12	9	0, 2, 3, 4, 5, 6, 7, 8
29	manual	50	9	2, 3, 4, 5
30	made-up	21	9	0, 1, 2, 3, 4, 5
31	random	2	10	0, 2, 3, 4, 5, 6, 7, 8, 10

32	manual	81	10	8
33	random	5	11	3, 4, 8, 10
34	manual	35	11	8
35	manual	32	12	0, 3, 4, 5, 6, 7, 8
36	random	13	12	3, 4, 5, 6, 7, 8
37	made-up	45	12	0, 2, 3, 4, 5, 6, 7, 8
38	manual	35	13	0, 2, 3, 4, 5, 6, 7, 8
39	random	6	13	0, 3, 4, 5, 6, 7, 8
40	made-up	19	14	0, 1, 2, 3, 4, 5, 8, 9
41	manual	60	14	0, 2, 3, 4, 5, 6, 7, 8
42	random	32	14	2, 3, 4, 5, 6, 7, 8
43	manual	54	15	0, 2, 3, 4, 5, 6, 7, 8
44	random	6	15	0, 2, 3, 4, 5, 6, 7

Table 4.3.2: Blocks chosen for the focus group

For each block we asked the participants of the focus group to rate the three different dependent variables on a Likert scale from one to five. Additionally, after concluding the presentation of the individual blocks for each diagram, we asked three additional questions related to tasks other than feature identification. While presenting all blocks of a diagram we asked the participants to rate the following questions on a Likert scale from one to five:

How helpful is the partitioning of elements ...

2.1 ... as a first insight into the current state of variability management?

2.2 ... as an indicator for issues regarding the current state of variability management?

2.3 ... as an indicator for the current state of the modeling quality?

4.3.2 RQ.2 Is the semi-automated approach based on FCA feasible, when applied to a production scale, real-world set of product variants?

Because the first research question is the main focus of this thesis, we allocated all the time we had to spent with the domain experts to the first and partly third question. Therefore this research question is focused on quantitative and qualitative measures that do not overly rely on domain knowledge. For a quantitative measure we investigated the overall time it took to run the tooling. For a qualitative measure we evaluated the overall distribution of elements over the concept lattice and tried to categorize the identified features into groups, similar to [Tinnes and Martinez]. Here Tinnes et al. differentiate between two types of results. Firstly, the ideal clean cases one might expect when designing a product line top-down, mainly variant cores, feature cores, feature interactions and shared elements between features. Secondly, the results expected from an analysis perspective, such as merged features and noise introduced during the independent development of variants.

4.3.3 RQ.3 How can we represent the results of the semi-automated approach in a way that maximizes their usefulness to the engineers?

To answer the third and final research question, we first asked the participants to evaluate the already implemented features of our MagicDraw plugin, that were showcased during the evaluation of our first research question. For this, we had the participants evaluate the following questions on a Likert scale from one to five:

How helpful is the functionality ...

3.1 ... to see in which variant an element is present via a tool-tip?

3.2 ... to highlight groups of elements in one color?

3.3 ... to navigate imported elements via hyperlink into the internal containment browser?

After that, we asked the participants to come up with potential missing features they would like to see implemented and would deem useful from their own perspective. Because the trunk variants are an already work-in-progress adoption of a product line, we presented the formal context table to the focus group and mentioned the ability to use it as a sort of progress report on how far the adoption of the product line has come along. For this we computed the five biggest identified features, that are already reused in some variants, but are not part of the trunk variant, and asked the participants if this would be a useful information to them. Finally, we computed a dendrogram from the formal context of each part system, as show in Figure 4.3.7, with the distance value $1.0 - s$, where s is the Jaccard index computed from the sets of features from two variants. We presented these diagrams to the participants and asked if the shown similarities coincide with their understanding of the development history of the variants.

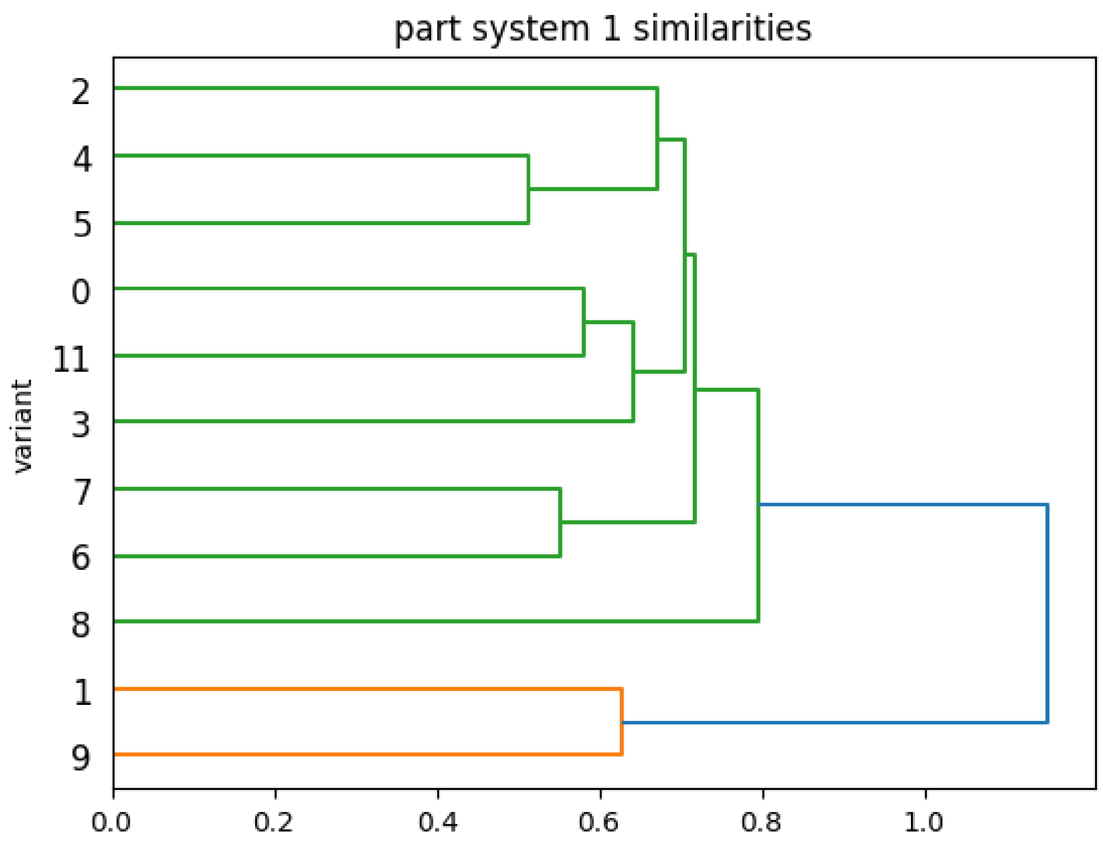


Fig. 4.3.7: Computed variant similarities for part system 1

CHAPTER 5

Evaluation

5.1 Results

5.1.1 RQ.1 Is the semi-automated approach based on FCA able to assist as a starting point in the different tasks associated with an extractive approach to SPLE?

For the questions 1.1, 1.2, and 1.3 regarding the individual blocks, that we presented to the participants of the focus group, Table A.2.1 shows the results. Note that we did not have time to fit all the prepared blocks into the time frame of two hours. We were not able to present the diagrams 7, 8, 9, and 10. Therefore the prepared blocks 22 - 32 are not part of the results. Figures 5.1.1, 5.1.2, and 5.1.3 illustrate the Likert scores for questions 1.1, 1.2, 1.3 from Table A.2.1 as violin plots, where

variable	mean 1.1/block	mean 1.2/block	mean 1.3/block
1	0,50	0,22	0,41
2	-0,27	-0,51	-0,40
3	-0,30	-0,46	-0,34
4	-0,11	-0,43	-0,27
6	0,11	0,45	0,35

Table 5.1.1: Correlation matrix for means of questions 1.1, 1.2, 1.3 per block and the independent variables

variable	mean 1.1/diagram	mean 1.2/diagram	mean 1.3/diagram
2	-0,40	-0,63	-0,61
3	-0,63	-0,64	-0,60
4	-0,03	-0,51	-0,36
6	-0,04	0,48	0,45

Table 5.1.2: Correlation matrix for means of question 1.1, 1.2, 1.3 per diagram and the independent variables

diagram type	mean 1.1	mean 1.2	mean 1.3
SysML Internal Block Diagram	2,75	1,95	2,75
SysML Block Definition Diagram	$2{,}\overline{33}$	$1{,}8\overline{3}$	2
SysML Activity Diagram	$3{,}41\overline{6}$	3	4,25
SysML Use Case Diagram	2,5	$2{,}\overline{33}$	$3{,}1\overline{6}$
SysML State Machine Diagram	$3{,}1\overline{6}$	$4{,}\overline{33}$	3

Table 5.1.3: Means of questions 1.1, 1.2, 1.3 per diagram type

the median, mean and extrema are represented as vertical lines. Performing Welch's t-test for the manual selected blocks paired with the randomly selected and made-up blocks of questions 1.1, 1.2, 1.3 respectively, we got the p scores as listed in Table 5.1.4. For evaluating the correlation between dependent and independent variables, we will consider coefficients smaller than 0.3 as no correlation, coefficients between 0.3 and 0.5 as small correlation, coefficients between 0.5 and 0.6 as medium correlation and anything above 0.6 will be considered as big correlation. Table 5.1.1 shows the correlation coefficients calculated between the means of the Likert scores per block for question 1.1, 1.2, and 1.3 and our independent variables (excluding the diagram type). Table 5.1.3 shows the means of question 1.1, 1.2, and 1.3 per diagram type. Table 5.1.2 shows the correlation coefficients calculated between the means of the Likert scores per diagram for question 1.1, 1.2, and 1.3 and our independent variables.

5.1.1.1 Coherency

Taking the standard threshold for p-scores of 0.05, we see from Table 5.1.4, that in terms of coherency, the manual blocks performed statistically better than the random and made-up blocks, but there is no significant difference between the random and made-up blocks. With a coefficient of 0.50 we observe a medium positive correlation between the coherency of a block and the size of it in terms of elements, where the coherency of a block seems to increase with its size. We also see a small negative correlation between the number of variants the blocks of a diagram appear in and the coherency of the blocks, where the coherency slighty decreases with an increase in variants. From Table 5.1.2 we observe that there is a small negative correlation between the overall coherency of the blocks of a diagram and the amount of blocks in total, where the coherency seems to slightly decrease with an increase in blocks. Additionally, we observe a strong negative correlation between the overall coherency of the blocks of a diagram and the total amount of variants the blocks appear in, where the coherency seems to strongly decrease with an increase in variants. In terms of overall coherency, we see from Table 5.1.3 that the blocks of the diagrams of type 'SysML Acticity Diagram' performed the best and the blocks of the diagrams of type

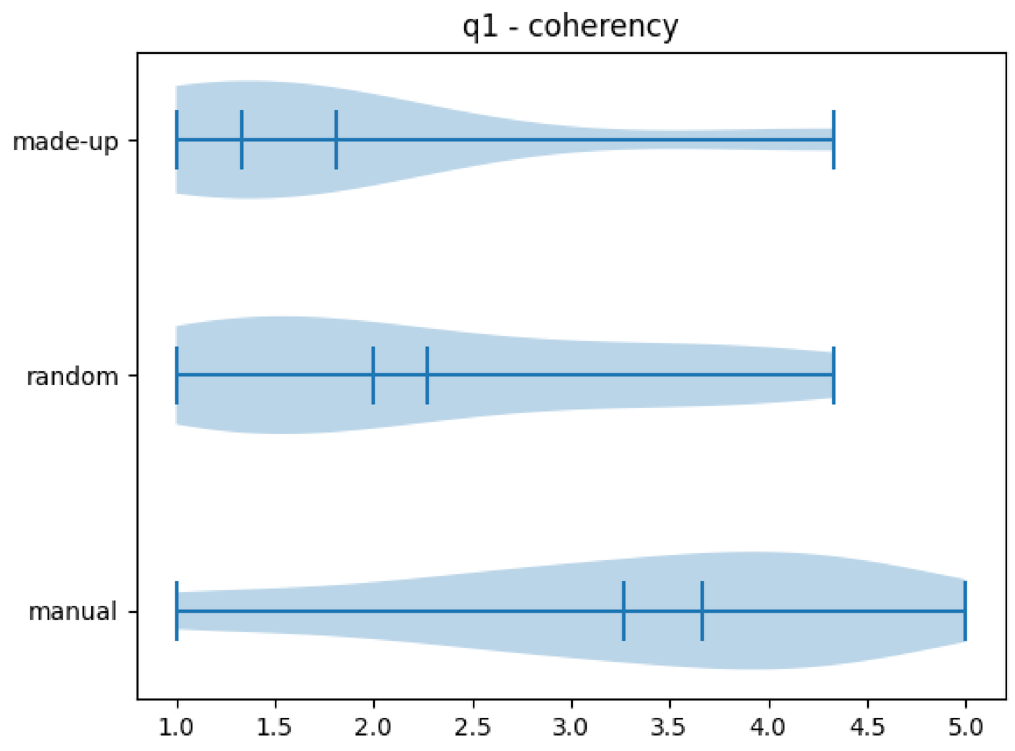


Fig. 5.1.1: Violin plot for the means of question 1.1

‘SysML Block Definition Diagram’ performed the worst.

5.1.1.2 Completeness

In terms of completeness we see from Table 5.1.4 that there is no statistically significant difference for the three categories of blocks when applying a threshold for p of 0.05. From Table 5.1.1 we observe that there is a medium negative correlation between the completeness of a block and the number of blocks in a single diagram, where the completeness of a block seems to decrease with an increase in total blocks. Additionally we observe that there is a small negative correlation between both the number of variants the blocks of a diagram appear and the size of a diagram in terms of elements and the completeness per block. There is also a small positive correlation between the completeness of a block and the sparsity of the formal context of the corresponding part system, where the completeness seems to slightly increase with an

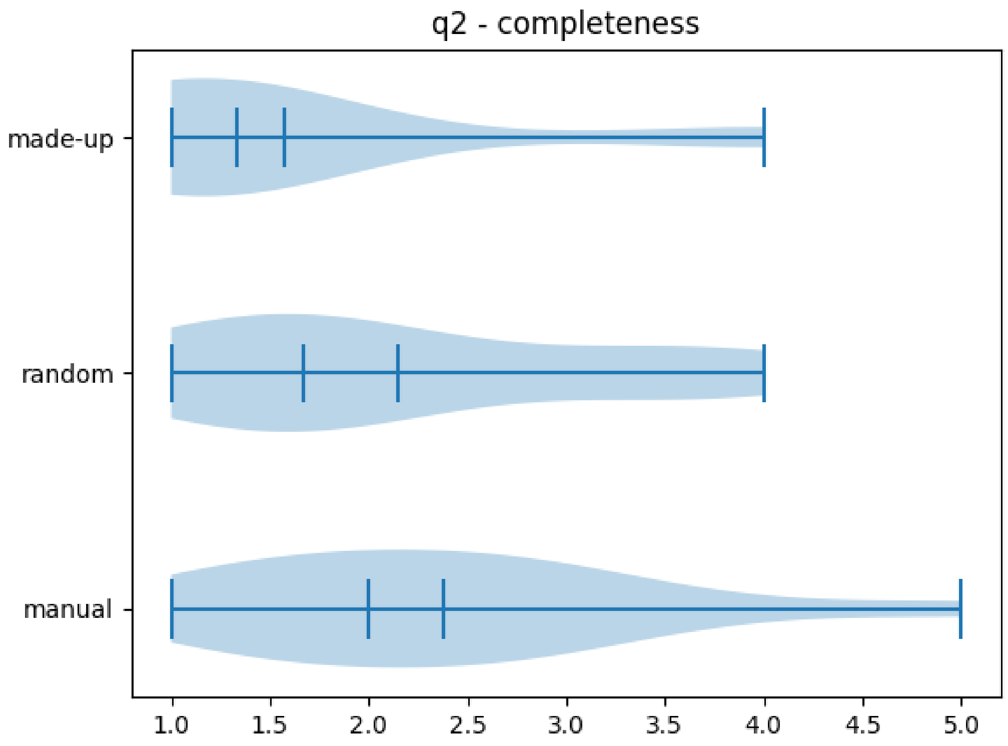


Fig. 5.1.2: Violin plot for the means of question 1.2

increase in sparsity. From Table 5.1.2 we see that there is a big negative correlation between both the number of blocks in a single a diagram and the number of variants these blocks appear in and the overall completeness of the blocks per diagram. Additionally there is a medium correlation between the size of a diagram in terms of elements and the overall completeness of the blocks. Also there is a small positive correlation between the overall completeness of blocks in a diagram and sparsity of the formal context, where the overall completeness seems to slightly increase with an increase in sparsity. In terms of overall completeness, we see from Table 5.1.3 that the blocks of the diagrams of type ‘SysML State Machine Diagram’ are the most complete and the blocks of the diagrams of type ‘SysML Block Definition Diagram’ are the least complete.

question	1.1	1.2	1.3
manual $\leftrightarrow$ random	0.047	0.594	0.107
manual $\leftrightarrow$ made-up	0.019	0.127	0.023
random $\leftrightarrow$ made-up	0.426	0.285	0.31

Table 5.1.4: Results of Welch's t-test for questions regarding individual blocks

5.1.1.3 Variant Specificity

For the third and final question 1.3, regarding the individual blocks, we see that, in terms of variant specificity, there is a significant difference between the made-up blocks and the manually selected blocks, but no significant difference for the remaining pairs, when selecting a threshold for p of 0.05. From Table 5.1.1 we observe that there is a small positive correlation between both the size of a block in terms of elements and the sparsity of the corresponding formal context and the variant specificity per block. Additionally there is a small negative correlation between both the total number of blocks in a diagram and the number of variants these blocks appear in and the variant specificity per block. From Table 5.1.2 we see that there is a big negative correlation between both the size of a block in terms of and the sparsity of the corresponding formal context and the overall variant specificity per diagram. Additionally there is a small negative correlation between the size of the diagram in terms of elements and the overall variant specificity per diagram. Also there is a small positive correlation between the overall variant specificity per diagram and the sparsity of the corresponding formal context. In terms of overall variant specificity, we see from Table 5.1.3 that the blocks of the diagrams of type 'SysML Activity Diagram' performed the best and the blocks of the diagrams of type 'SysML Block Definition Diagram' performed the worst.

5.1.1.4 Additional questions

For the questions 2.1, 2.2, and 2.3 regarding all the blocks of a respective diagram we obtained the results as shown in Table A.2.2. Figure 5.1.4 illustrates the Likert

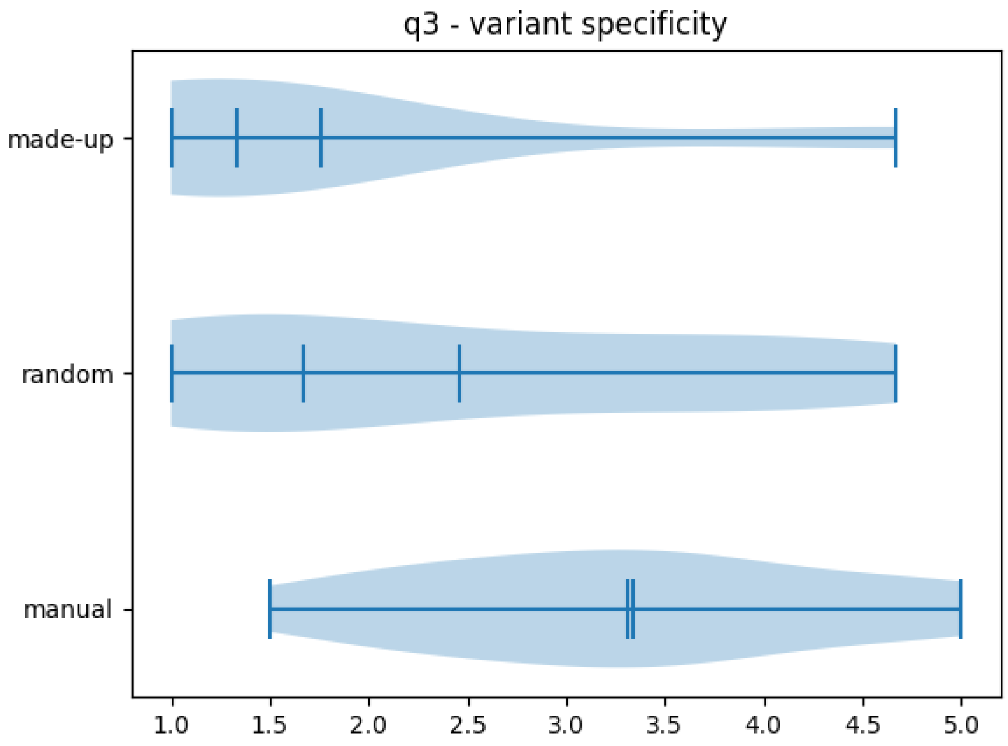


Fig. 5.1.3: Violin plot for the means of question 1.3

diagram	mean 1.1	mean 1.2	mean 1.3
1	$1,\overline{88}$	$1,\overline{22}$	$1,\overline{77}$
2	$3,\overline{22}$	$2,\overline{11}$	$2,\overline{77}$
3	4	$1,\overline{88}$	$3,\overline{22}$
4	$2,\overline{33}$	$1,8\overline{3}$	2
5	$3,\overline{66}$	3,5	4
6	$2,\overline{22}$	$1,\overline{66}$	2,5
11	2,5	$2,\overline{33}$	$3,1\overline{6}$
12	$2,\overline{33}$	$2,\overline{33}$	$3,41\overline{6}$
13	$3,1\overline{6}$	2,5	4,5
14	$2,8\overline{3}$	2,5	$2,8\overline{3}$
15	$3,\overline{16}$	$4,\overline{33}$	3

Table 5.1.5: Means of the questions 1.1, 1.2, 1.3 regarding individual blocks per diagram

variable	mean 2.1	mean 2.2	mean 2.3
2	-0,76	-0,59	-0,21
3	-0,65	-0,53	-0,26
4	-0,64	-0,53	-0,13
6	0,43	0,55	-0,37

Table 5.1.6: Correlation matrix for overall means of question 2.1, 2.2, 2.3 and the independent variables

diagram type	mean 2.1	mean 2.2	mean 2.3
SysML Internal Block Diagram	$1,\overline{33}$	$1,8\overline{3}$	$3,\overline{55}$
SysML Block Definition Diagram	2	2	$4,\overline{66}$
SysML Activity Diagram	$3,\overline{66}$	$3,1\overline{6}$	4,5
SysML Use Case Diagram	$2,\overline{33}$	$2,\overline{33}$	$2,\overline{33}$
SysML State Machine Diagram	5	5	3

Table 5.1.7: Means of questions 2.1, 2.2, 2.3 per diagram type

scores from Table A.2.2 as a violin plot, where the median, mean and extrema are represented as vertical lines. Table 5.1.6 shows the correlation matrix for the overall means of question 2.1, 2.2, and 2.3 and the independent variables.

From Table 5.1.6 we observe that there is a strong negative correlation for the first question regarding variability management and the number of blocks in a single diagram, the number of variants these blocks appear in, and the size of the diagram in terms of elements. There is also a small positive correlation with the sparsity of the corresponding formal context. From Table 5.1.7 we see that for question 2.1 diagrams of the type 'SysML State Machine' performed the best and diagrams of type 'SysML Internal Block Diagram' performed the worst. For the second question regarding variability management we observe from Table 5.1.6 that there is a medium negative correlation with the number of blocks in a single diagram, the number of variants these blocks appear in, and the size of the diagram in terms of elements. There is also a medium positive correlation for question 2.1 and the sparsity of the corresponding formal context. From Table 5.1.7 we see that for question 2.2 diagrams of the type 'SysML State Machine' performed the best and diagrams of type 'SysML Internal Block Diagram' performed the worst. For the third question 2.3, regarding modeling quality, we see from Table 5.1.6 that there is a small negative correlation with the sparsity of the corresponding formal context. From Table 5.1.7 we see that for question 2.3 diagrams of the type 'SysML Block Definition Diagram' performed the best and diagrams of type 'SysML Use Case Diagram' performed the worst.

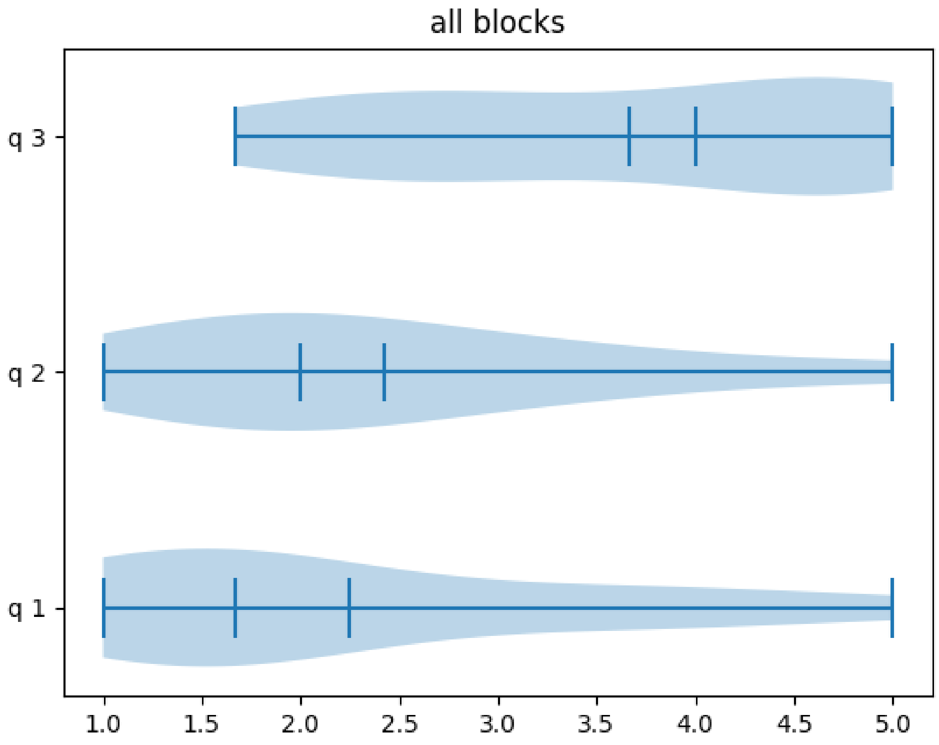


Fig. 5.1.4: Violin plot for the averages of questions 2.1, 2.2, 2.3 for all blocks

5.1.1.5 Summary

Taking the overall mean for question 1.1 of $2,8\overline{48}$ into account, we infer that the approach was able to identify coherent blocks in a diagram with a mediocre result, where a user had to be involved in order to pick the most promising ones. Although the margin of error for the rejection of the assumption that there is no difference between manual and random blocks is higher than the one for manual and made-up blocks, we infer that the tooling was not able to consistently identify complete blocks on a diagram, given that the mean for all diagrams is only $2,\overline{38}$. With a mean of $3,01\overline{76}$ for all manually and randomly selected blocks for all diagrams, we infer that the tooling is able to assign functionality to the correct variants with a reasonable accuracy. With a mean of $2,\overline{24}$ for the first question 2.1 and a mean of $2,\overline{42}$ for the second question 2.2, we see, that the partitioning of elements as presented on a diagram basis did not allow for proper judgement for the current state of variability management. As a whole, we see that the number of blocks in a single diagram and the number of variants these blocks appear in seem to have a great negative effect on our results, with the participants of the focus group openly voicing their concerns over the diagrams with more blocks, summarized by a quote '[...] one cannot see the forest for the trees [...]'. This consensus is reflected in the third and final question regarding the state of modeling quality in the current set of variants. With a mean of $3,\overline{66}$ for all diagrams, we can see that the partitioning helped, instead, to evaluate the state of modeling quality for an existing set of variants. Here the biggest problem with our case study seems to lie in the fact that, while the internal id of the 'bigger building blocks' of the block diagrams seem to have been preserved with the development of new variants, their ports and connectors seem to have been re-implemented for almost every variant. This would also explains why the diagrams 'SysML Internal Block Diagram' and 'SysML Block Definition Diagram' perform generally less favorably compared to the other types of diagrams and performed the best in question 2.3 regarding modeling quality. In the focus group this let to the following conclusion. Either the variants of our case study have to cleaned up, especially in terms of the mentioned ports and their connectors, before applying a semi-automated approach like the one in this thesis. Or that the comparison and similarity metric have to be made more sensitive to these issues, which led to more open problems regarding the implementation of such a comparison method, which can no longer rely on the internal id for initial binning of the elements and needs a new way of finding ports and connectors that have to be compared. The only consensus, given the very limited time, was that the comparison for ports would have to rely on the name and type of the port instead of the internal id. Potentially a similarity metric akin to the one used by Ryssel et al. that we introduced in Chapter 2.2 could be applied, using a mix of weighted local and neighborhood criteria. Additionally the participants voiced their concerns over the fact, that we included every variant of the part systems we chose. Two of the variants, 4 and 5, are no longer maintained and three variants, 2 and 8 and 9, are intentionally treated as clone and own, with no intent of re-integrating them into the envisioned product line. This seems to be reflected with the correlation coefficients obtained from the third independent variable, regarding the number of variants the blocks of a diagram appear. The more variants were present in a diagram, the worse the results were. Together with the issues regarding the modeling quality this led to the conclusion that the tooling did what it was supposed to do, but given the quality of the input data the result is mediocre at best, summarized by a quote '[...] shit in, shit out [...]'. The dendrograms computed from the formal contexts, seem to reflect the history of the development of the variants, with the participants scoring their resemblance with the actual development history unanimously as five, on a scale of one to five.

In summary we see that the semi-automated approach based on FCA is not sufficiently able to support as a starting point in the task of feature identification, given the state of the input data. The approach is able to locate some coherent functionality and assign said functionality to the correct variants and provides some meaningful insights. But in terms of completeness the approach is not properly able to identify full sets of functionality, which is particularly noticeable in diagrams with more development artefacts and noise, such as the re-implementation of ports and connectors.

Additionally it has to be noted that we only investigated blocks of features from a diagram perspective, which excludes the fact, that the blocks in a diagram potentially belong to a big, merged feature with a lot of other functionality, as we will discuss in RQ.2. For judging the state of variability management the results were not good enough to be useful, but instead they helped visualize the problems with the quality of the models. Inferring from the results, the amount of blocks identified on a diagram can be used as an indicator for judging the modeling quality of the respective diagram, with a rule of thumb for our data-set being, that the number of blocks identified should be lower than the variants the elements of diagram appear in. The approach is also able to automatically infer the development history and therefore broadly identify the similarities between the variants. Although the results were less favorably than we hoped for, we and the participants of the focus group seem optimistic about the results from a diagram perspective, after the issues regarding the input data as mentioned above have been resolved.

5.1.2 RQ.2 Is the semi-automated approach based on FCA feasible, when applied to a production scale, real-world set of product variants?

5.1.2.1 Quantitive Measures

The Table 5.1.8 shows the timings for the individual steps of the approach that we applied to the variants of the respective part systems, in milliseconds. Note that for exporting the models out of MagicDraw, we did not measure exact timings, but it took around five to ten minutes for each variant of each part system, and given the hardware limitations, we chose the lower end of that range for the sake of this argument. Taking the mean time it took for the five part systems, around 126,84 minutes or roughly two hours, and the amount of part systems in our case study overall, 59, we can extrapolate that a full initial run of the approach would take around 118 hours, nearly five days. Note that we did not take into consideration the time it would take to import the results back into MagicDraw, iterate on the adoption of the product line and run

part system	0	1	2	3	4
exporting	3000000	3300000	3300000	3600000	3300000
adapting (importing)	50210	65367	40220	43874	58383
feature identification	2105681	4598157	3698170	5331343	4265369
constraints discovery	734	1527	1840	1766	1659
feature model synthesis	226423	266010	256959	283456	253813
total(ms)	2683048	5231061	4297189	5960439	4879224
total(min)	89,72	137,18	121,62	154,34	131,32

Table 5.1.8: Timing for the individual steps (ms)

the approach again. The time for importing/exporting out of MagicDraw could be eliminated by implementing the framework directly into a plugin, but this would go against the idea of a generally applicable framework, the idea behind But4Reuse. The individual runs could also be run in parallel, although depending on the amount of variants in an individual part system, each run needed between 19 and 27 gigabytes of RAM, over the course of the different steps. A major boost in speed can probably be obtained by optimizing the current implementations of the different steps. None of them make use of multi-threading, which is why the main limiting factor for our hardware is the peak single-core clock speed of our CPU and the speed of our different kind memories. Also the tool relies on a very rudimentary implementation of FCA, as mentioned in Section 4.2.2. Although trying to replace said implementation with an implementation[1] offering alternative algorithms for calculating the pruned concept lattice, resulted in an immediate termination of the tool caused by a lack of memory, even with the provided RAM maxed out to the 32 gigabytes that we had available, suggesting that the other implementation does not scale well for the size of our case study. Additionally if we were to implement the solutions for the issues identified in the previous research question, the time for the feature identification step would probably explode, because now we can no longer rely on only the internal ids of

[1]https://www.lirmm.fr/aoc-poset-builder/

elements, arguably the fastest way to group and compare elements (by hashing their ids). The constraints and feature model synthesis steps are also very rudimentary, a state-of-the-art approach like the one described in Section 2.2 would be more desirable, which would probably manifold the, as of now, rather little amount of time spent in those steps. On an up note - if we were to incorporate the information about which variants the participants of the focus group would like us to exclude, we could gain a significant amount of speed up, because the tooling runs an order of magnitude faster, for each variant that we can exclude, partly illustrated by part system 0 in Table 5.1.8, which only has 10 variants.

Summarizing, the current state of the tool could probably be applied to production scale, real-world set of product variants in a reasonable time frame. When we want to incorporate more elaborate algorithms for the different steps however, we can only imagine the time span it would take to apply such an approach to drift into the unreasonable. In the end the time it takes heavily depends on the amount of variants a product line-up contains and the feasibility depends on what the definition of a reasonable time frame for an individual company would be.

5.1.2.2 Qualitative Measures

Table 5.1.9 shows the results of the feature identification step for each individual part system, where min, max, median, and standard deviation refer to the amount of AMEs found in the respective part system. Taking the information about the size of the individual part systems, found in Table 4.2.1, into account, we see that, on the one side, the biggest block is made up of more than 50 percent of the total elements. This is the base block for all part systems, the block, that is present in every variant. It is made up of mostly reference documents, definitions for SI units, model library elements, such as icons and other images and elements used in the different types of diagrams. Here we can already see the drawback of an approach purely based on co-occurrence. Figure 5.1.5 illustrates this on the example of the concept lattice of part system 1 that we scaled (logarithmically to make it more visible on paper) by the size of each identified feature. The attribute concepts are highlighted in red and the object concepts are highlighted in blue, with the disproportions of the features clearly visible. While the approach is able to identify groups of elements that occur in the same variants, it cannot differentiate between elements that have the same occurrence, for example, you can hardly define the individual reference documents together as a feature, it would be more desirable to have them grouped with the functionality that actually references them. On the other side, going off of the median, we see that at least 50 percent of the identified features are tiny compared to the rest, with the smallest being constituted of only one element. Again these groups can hardly be classified as features in a standard fashion, such as mentioned in Section 3.1.2. These are most likely implementation noise that accumulated over the course of development (e.g., different languages for different variants), given that they are mostly made of attribute elements, as shown in Table 5.1.11. Additionally, the second group of bigger features, are the object concepts, visible in blue in Figure 5.1.5 for part system 1. These are the features that were only identified to belong to a single variant. Again these are made up of multiple different groups, such as parts of diagrams, that actually only belong to this variant, the re-implemented ports and connectors as mentioned earlier, and entirely new diagrams, that are considered as only part of this variant, but seem to applicable to more than one variant, seen from a non-domain-expert perspective. Some of these groups could potentially be split up into individual features each, but are not classifiable as a feature as a whole. As seen in Table 5.1.12, for the bigger blocks, the distribution of AME types seems to be more in line with the total distribution per part system, as shown in Table 5.1.10. The reason we got at least mediocre results for our first research question, is that we presented a very limited and localized view on the elements of a feature, by inspecting only blocks of them part of a single diagram at once. For elements that are part of diagrams, this seems to be a more reasonable fine-grained subdivision of the overall feature, they were identified to belong to.

Summarizing, an approach purely based on co-occurrence, like our approach based on FCA, is capable of identifying functionality that is part of the same variants. But the biggest drawback of such an approach is that functionality that occurs in the same

part system	# features	min	max	mean	median	std. deviation
0	333	1	335921	4298	51	25031.98
1	488	1	335445	3157.84	34	20923.54
2	477	1	335109	3275.22	29	21529.89
3	481	1	335116	3134.3	32	21161.83
4	497	1	335172	3343.22	36	21658.07

Table 5.1.9: Results of the feature identification step

part system	class	attribute	reference
0	26.49%	47.91%	25.61%
1	26.30%	48.01%	25.69%
2	26.27%	47.96%	25.77%
3	26.07%	48.05%	25.89%
4	26.36%	47.88%	25.77%

Table 5.1.10: Total percentage of AME types per part system

variants, but has no relationship other than that, is identified as a single feature. For elements that are part of diagrams, the diagram perspective yields a more localized view on the identified features, and seems to be a feasible choice, to start dividing up the big clumps of elements, if applicable. For elements that are not part of diagrams, the approach needs to be extended, in order to yield more fine-grained results and is not feasible.

5.1.3 RQ.3 How can we represent the results of the semi-automated approach in a way that maximizes their usefulness to the engineers?

Table 5.1.13 shows the results for the questions 3.1, 3.2, and 3.3 regarding the individual features of the plugin, that were showcased during the focus group. The two

part system	mean			median		
	class	attribute	reference	class	attribute	reference
0	12.26%	78.84%	8.90%	0.00%	88.76%	0.00%
1	11.39%	76.82%	11.79%	0.00%	100.00%	0.00%
2	10.96%	79.90%	9.14%	0.00%	100.00%	0.00%
3	11.39%	79.66%	8.95%	0.00%	100.00%	0.00%
4	10.92%	80.26%	8.83%	0.00%	100.00%	0.00%

Table 5.1.11: Percentage of AME types for small features ($\leq$ median)

part system	mean			median		
	class	attribute	reference	class	attribute	reference
0	20.42%	62.25%	17.34%	22.35%	56.41%	17.98%
1	20.11%	63.83%	16.06%	22.92%	59.38%	16.35%
2	19.97%	63.92%	16.12%	22.90%	59.38%	17.01%
3	20.17%	63.54%	16.28%	23.08%	57.64%	17.11%
4	20.01%	62.37%	17.62%	22.53%	57.39%	18.16%

Table 5.1.12: Percentage of AME types for big features ($>$ median)

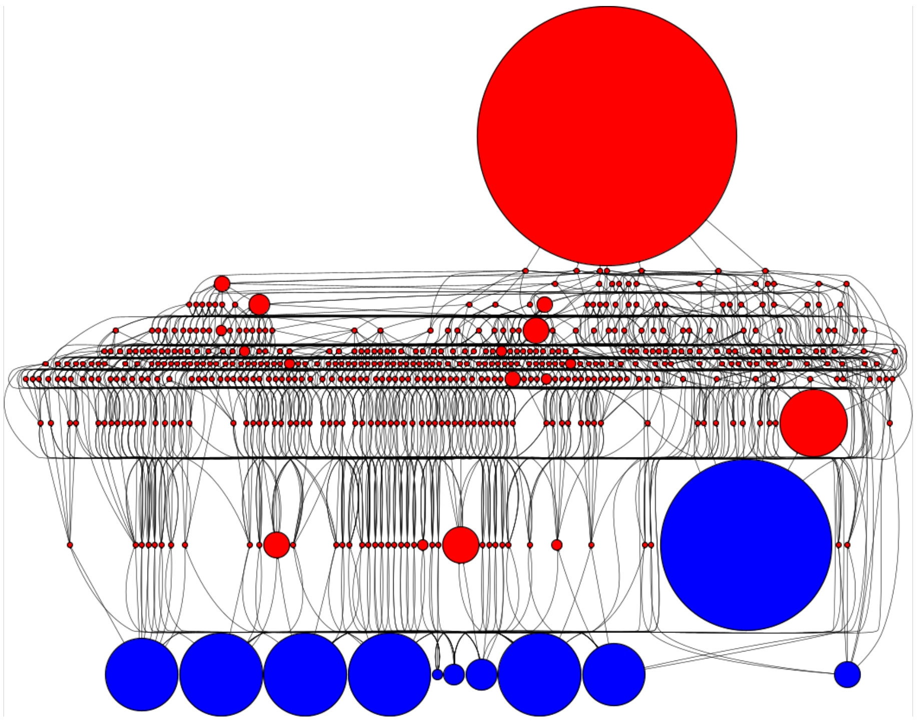


Fig. 5.1.5: Concept lattice for part system 1 scaled logarithmically by the size of each feature

main features, namely the colored highlighting directly in a diagram and the tool-tip that shows in which variants an element was identified to belong to, were received very positively, with both scoring a mean of five on a scale of one to five. The ability to navigate imported elements via a hyperlink into the internal containment browser was received less favorably, with a mean score of $2,\overline{33}$. The general consensus on this question was, that the participants would much rather navigate the elements directly from a diagram. Although it has to be noted, that this can only be done for elements, that actually are contained in diagram. As mentioned in Section 4.3.1 there are also class elements that are not contained in a diagram, and attribute and reference elements are never contained in a diagram, they can only be accessed via a property view for an individual element. The score for the third question re-ensures us in our methodology of ignoring said elements for the focus group, as this way of navigating elements is not suitable for large amounts of data. Although we did not have the time to take a look into the biggest features, that are already reused in some variants, but are not part of the trunk variant, the notion of using those as a sort of progress report

question	mean	median
3.1	5	5
3.2	5	5
3.3	2,333333333	2

Table 5.1.13: Results of the questions regarding plugin features

and an indicator, on what is still missing in the trunk variant, received very positive feedback with a mean score of five, on a scale of one to five. Finally, the focus group came up with the following features, that would be desirable for the next iteration of our MagicDraw plugin.

1. The ability to navigate to other diagrams containing elements of the same feature on demand
2. The ability to navigate to the same feature in the same diagram, if existent, in other variants
3. The ability to select one variant and highlight every element in a diagram that is contained in said variant
4. A better automated way of choosing visually distinct colors for highlighting
5. The ability to dynamically merge (split) blocks on a diagram on demand and matching (unmatching) the colors used for highlighting

The first feature request should be straight-forward to implement, with the information we already extracted from the variants, mainly the information as shown in Figure 4.3.4. The second feature request could form a much greater challenge. For this we would potentially have to open every variant of a part system in parallel which would require a more elaborate hardware setup. Additionally it is unclear if the internal ids of the diagrams are consistent throughout the different variants, so another matching for diagrams might be necessary. The third feature request should also pose no problem to implement. With the information from the formal context table for each part system and the information that was already gathered, as shown in Figure 4.3.5, all that is left to implement, is an user interface for the picking of the variants. For the fourth feature request we already incorporated an automatic way of generating a list of visually distinct colors, based on the YUV color space, adapted from an answer on Stack Overflow[2]. We generated a list of 100 colors and used the feature number to index that list and this was used to load all features of a part system at once in order to find appropriate diagrams for RQ.1. But we still ended up with similar colors on the same diagram. The possibly best way to implement this, would be a hand-picked, predefined list, that is as big as the largest amount of blocks found on a diagram. The fifth and final feature request would require some new internal format for features. At the moment the elements are just represented as nodes in a feature tree, where there is no direct way of accessing the node to which an element belongs to, without iterating the entire feature tree. Additionally there needs to be a way to serialize the newly edited features, possibly overwriting the existing files, that were loaded to begin with.

Summarizing, the biggest limiting factor for the engineers seems to be the lack of dynamical navigation for the enormous amount of data present in the part system variants. With the fact that you have to change projects in order to compare variants (e.g. on a diagram basis) having the biggest impact on productivity. A possible solution for this would be the creation of a project with every variant merged into it and filtering by variant, but there is no out-of-the-box support for this in MagicDraw. The ability to highlight groups of elements in a diagram, combined with the information to which variant these elements belong to, without having to swap projects, sparked general excitement in the focus group, based on the impressions we got. With the additional feature requests implemented, we conclude that the resulting tooling would be an appropriate representation to maximize the usefulness of the results gathered from our approach to the engineers. A similar tooling might find its way into the hands of engineers, although potentially with data obtained from a different approach.

[2]https://stackoverflow.com/a/30881059/7508046

5.2 Threats To Validity

In this section, we will discuss the threats to internal and external validity.

5.2.1 Internal Validity

With regards to internal validity potential problems arise from the tooling itself. Although we ourselves took care to validate the correctness of the implementations in BUT4REUSE by manually inspecting the code of the extensions points we used, no extensive testing was done and we did not inspect the code for the general framework itself. As far as we can tell, the framework is also not in wide-spread use and has therefore not been subject to the many-eyes-principle, potentially skewing the results through implementation bugs. The same applies to the MAGICDRAW plugin we developed. While great care was taken in its development, a tight time schedule and lack of testing might have resulted in bugs that might have affected the presentation of the results and therefore the results obtained from them. Additionally the UML exporter of MAGICDRAW might have introduced potential problems, for example, comments in the models went missing during export. Also, when querying the MAGICDRAW API for an element by id, it returns the first element with that id that it can find, potentially leaving room for errors, if multiple elements have the same id. But we only encountered a handful of elements that share the same id for the part systems we evaluated .Finally, because we conducted an empirical field study, we did not posses much control over experiment, making it hard to ensure internal validity overall. While we tried to tackle this by choosing the blocks and diagrams we presented in a way that allowed us to investigate multiple independent variables, we can not exclude the possibility of other confounding factors.

5.2.2 External Validity

Our study possesses a generally desirable amount of external validity. We applied the approach to a production scale, real-world set of data, that we did not have any control over. Potential threats may arise from only applying the approach to a single case study, but getting a hold of multiple real-world case studies of this size, would be an infeasible, if not impossible, task. Nonetheless, we are confident that our results are generalizable and hold true for other projects as well, particularly because the evaluated approach mainly relied on the internal identifiers of the elements in our models, which should be present and consistent in any modeling tool worth its share. Additional problems may arise from the limited sampling sets regarding our case study. While we tried to tackle this by trying to incorporate a large spectrum into the sample sets, we only evaluated five randomly chosen part systems of a total of 59. For those five part systems we only used fifteen diagrams out of hundreds in total to evaluate the approach. Additionally the focus group only consisted of three participants excluding us, which might have skewed the results in an unwanted direction. For the time measurements in our second research question, we only ran the tooling once for each part system, yielding five samples of measurements, and in no controlled environment, meaning background tasks running on the CPU and similar influences might have affected the results.

CHAPTER 6

Conclusion And Future Work

6.1 Conclusion

In this thesis, we evaluated a lightweight implementation of a semi-automated approach to feature identification/location, based on formal concept analysis. For this we applied the approach to a real-world, production-scale case study from the railway domain and developed a plugin for the modeling tool MagicDraw that visualizes the results to the engineers in a familiar environment and intuitive way.

First we investigated to what extent the approach is able to aid the engineers as a first step in the different tasks associated with the bottom-up adoption of a product line. For this we conducted a focus group, where we presented parts of identified features to the participants, by highlighting them in diagrams using the plugin we developed. We asked the participants to evaluate three different aspects of the features based on the highlighting they were shown. While we saw that for feature identification/location the approach is able to identify coherent functionality and locate it in the correct variants more or less consistently, it cannot identify complete sets of functionalities. We concluded that this is mostly due quality state of the input data, which was partly evidenced by our results and partly by the feedback given vocally in the focus group. On the one hand, we identified that the most common problem for our case study, seems to lie in the fact, that the ports and their connectors in diagrams were mostly re-implemented in every variant. The id based matching was therefore not able to match them accordingly. On the other hand, however, we also saw that for remaining types for diagram elements, the ids seem to have been more or less preserved during the development of the different variants. This lead us and the participants of the focus group to the assumption that if the identified problems are alleviated, either through a more sensitive matching algorithm or by cleaning up the models manually beforehand, the approach would fare much better. Although the drawbacks of an approach based purely on co-occurrence, would still persist, even with quality input data. Drawing from this, the approach helped the participants of the focus group to judge the current state of modeling quality, and could be directly used as a quality metric by taking the amount of identified features present in a single diagram into account. Because of the state of the input data and the resulting convoluted diagrams, the approach was not able to help the participants get a deeper look into the current state of variability management.

Next we investigated if the approach would be feasible if applied in practise. As a quantitative metric, we estimated the time it would take to apply the approach to the entire case study and concluded that the current state of the tooling could probably be applied within a reasonable time frame. However, if we were to incorporate the fixes for the problems identified and would implement more elaborate algorithms, especially for the feature model synthesis step, the time investment would grow rapidly, though heavily depending on the amount of variants fed into the tooling. We concluded that the feasibility depends on the definition of a reasonable time frame for an individual company and would have to be weighted against the resulting benefits. As a qualitative measure we took a look at the overall distribution of elements over the calculated concept lattices and tried to group the identified features into different categories. Here we saw the main drawback of an approach based purely on co-occurrence. On the one hand a big part of the features are in fact merged groups of different categories, for example, parts of multiple diagrams that have no relation to one another but the fact that they occurred in the same variants. These merged groups can hardly be classified as features as a whole. On the other hand we identified that a large part of the identified features are in fact implementation noise, which we concluded from their overall size and the distribution of the atomic model elements that they are constituted of. Finally, all elements of a variant that could not be matched with any other element of another variant, were grouped together into one feature. These groupings can again hardly be classified as a feature in the standard fashion, which we concluded by broadly taking a look at what they are constituted of. We concluded that they are again, a mixture of groups of different categories, for example, parts of diagrams that actually belong to only this variant, but also the before-mentioned re-implemented versions of ports and connectors and the like.

Finally we investigated how we can optimally present the results of our evaluated approach to the engineers, in order to maximize their usability. For this we asked the participants to evaluate the different functionalities of the plugin that we had already implemented. Here we saw a genuine, positive feedback from the participants in regards to the highlighting in the diagrams and the ability to directly see in which variants an element is located without having to swap projects. Afterwards, we asked the participants of the focus group to come-up with additional features for the plugin that they would like to have. Together with our already implemented features, we concluded that these features, which were mostly related to more dynamic ways of interacting with both the models themselves and the data obtained from them, would form an ideal representation for the results of not only our approach, but possibly others as well.

6.2 Future Work

There is a variety of different areas that can be tackled as future work. First, we were not able to fully evaluate all the independent variables for our case study that we set out to. This work can be finished with the already existing data, although some potentially some additional scripts would have to be added to the already fairly large base of existing scripts created for this thesis, in order to access the data from the concept lattice and feature models, without having to do it manually for dozens of blocks. Next, while we added some extensions and improvements to the But4Reuse framework, these are in no state to be merged into the official repository (mainly due to time constraints) and will need to be cleaned-up before-hand. Finally, and probably most importantly, as we have discussed, a second iteration of our case study would probably yield promising results. For this we would have to implement the requested features into our MagicDraw plugin. Additionally the identified issues with the But4Reuse framework, mainly implementation optimizations and more elaborate algorithms for the different extension points, and the issues with our input data would have to be resolved.

APPENDIX A

Apendix

A.1 Additional Independent Variables

This section includes a list of independent variables that we were not able to evaluate due to time constraints. They are left as future work.

1. The total number of diagrams a feature appears in
2. The hierarchy level of the feature in the concept lattice (starting from the top or bottom)
3. The number of features on the same hierarchy level in the concept lattice
4. The distance between the features contained in a diagram in the concept lattice, calculated as the average distance to the lowest common ancestor of the DAG obtained by treating the edges of the lattice as directed into the intent direction
5. The number of children (parents) of a feature in the concept lattice
6. The overall similarity of the variants a feature appears in
7. The depth of the feature in the feature model

A.2 Tables With The Results Of The Focus Group

block	mean 1.1	mean 1.2	mean 1.3	median 1.1	median 1.2	median 1.3
1	2	$1,\overline{33}$	$3,\overline{33}$	2	1	3
2	2	$1,\overline{33}$	2	2	1	2
3	1	1	1	1	1	1
4	$2,\overline{66}$	$1,\overline{33}$	2	3	1	2
5	$4,\overline{33}$	$2,\overline{66}$	$4,\overline{66}$	4	3	4
6	4	$2,\overline{66}$	$4,\overline{66}$	4	3	4
7	$1,\overline{33}$	1	1	1	1	1
8	1	1	1	1	1	1
9	$4,\overline{33}$	2	$3,\overline{33}$	4	2	3
10	4	2	$3,\overline{33}$	4	2	3
11	$4,\overline{66}$	$1,\overline{66}$	3	4	2	3
12	1	1	1	1	1	1
13	$2,\overline{66}$	2	$3,\overline{33}$	3	2	2
14	2	$1,\overline{66}$	$1,\overline{66}$	2	2	2

15	$1,\overline{66}$	$1,\overline{33}$	$1,\overline{33}$	1	1	1
16	$4,\overline{33}$	4	$4,\overline{66}$	4	4	5
17	3	3	$3,\overline{33}$	3	3	3
18	1	1	1,5	1	1	1,5
19	3	2	3	3	2	3
20	$2,\overline{66}$	2	3	3	2	3
21	$1,\overline{33}$	$1,\overline{33}$	1	1	1	1
33	$1,\overline{33}$	$1,\overline{66}$	$1,\overline{66}$	1	1	1
34	$4,\overline{66}$	3	$4,\overline{66}$	4	3	5
35	1	$1,\overline{33}$	2,5	1	1	2,5
36	$4,\overline{66}$	$3,\overline{33}$	$4,\overline{33}$	4	3	4
37	$4,\overline{33}$	4	$4,\overline{66}$	4	4	5
38	4	$2,\overline{66}$	5	4	3	5
39	$3,\overline{33}$	$3,\overline{33}$	4	2	2	4
40	$1,\overline{33}$	1	$1,\overline{33}$	1	1	1
41	$4,\overline{33}$	$4,\overline{66}$	4	4	4	4
42	$1,\overline{33}$	$1,\overline{33}$	$1,\overline{66}$	1	1	2

43	5	5	5	5	5	5
44	$1,\overline{33}$	$4,\overline{66}$	1	1	3	1

Table A.2.1: Results for questions 1.1, 1.2, 1.3 regarding the individual blocks

diagram	mean 2.1	mean 2.2	mean 2.3	median 2.1	median 2.2	median 2.3
1	$1,\overline{33}$	$1,\overline{33}$	$4,\overline{66}$	1	1	5
2	$1,\overline{33}$	$1,\overline{33}$	5	1	1	5
3	1	1	$4,\overline{66}$	1	1	5
4	2	2	$4,\overline{66}$	2	2	5
5	$3,\overline{66}$	$3,\overline{66}$	4	4	4	4
6	$1,\overline{33}$	2	$1,\overline{66}$	1	2	2
11	$2,\overline{33}$	$2,\overline{33}$	$2,\overline{33}$	2	2	2
12	$1,\overline{66}$	$3,\overline{33}$	3	2	4	3
13	$3,\overline{66}$	$2,\overline{66}$	5	4	3	5
14	$1,\overline{33}$	2	$2,\overline{33}$	1	2	3
15	5	5	3	5	5	3

Table A.2.2: Results for questions 2.1, 2.2, 2.3 regarding all blocks of a diagram

A.3 List Of Accompanying Files

In this section, we will give an overview of the files this thesis is accompanied by.

- thesis.pdf - This thesis as a PDF document.
- contexts/ - A folder containing the formal contexts obtained from our case study as HTML tables
- concepts/ - A folder containing the formal concept lattices obtained from our case study as DOT files
- feature_models/ - A folder containing the feature models obtained from our case study as XML files, viewable in FeatureIDE
- figures/ - A folder containing additional figures for the different part system that we did not include in this thesis
- plugin/ - A folder containing the (redacted) code for the visualization plugin
- scripts/ - A folder containing various undocumented (redacted) python scripts that were used for gathering the data and visualizations used in this thesis

REFERENCES

[1] Apel, S., Batory, D., Kstner, C., and Saake, G. (2013). *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated.

[2] Clements, P. C. and Northrop, L. (2001). *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley.

[3] Couto, M. V., Valente, M. T., and Figueiredo, E. (2011). Extracting software product lines: A case study using conditional compilation. In *2011 15th European Conference on Software Maintenance and Reengineering*, pages 191–200.

[4] Fischer, S., Linsbauer, L., Lopez-Herrejon, R. E., and Egyed, A. (2014). Enhancing clone-and-own with systematic reuse for developing software variants. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 391–400.

[5] Fischer, S., Linsbauer, L., Lopez-Herrejon, R. E., and Egyed, A. (2015). The ecco tool: Extraction and composition for clone-and-own. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 665–668.

[6] Font, J., Arcega, L., Haugen, Ø., and Cetina, C. (2016a). Feature location in model-based software product lines through a genetic algorithm. In Kapitsaki, G. M. and Santana de Almeida, E., editors, *Software Reuse: Bridging with Social-Awareness*, pages 39–54, Cham. Springer International Publishing.

[7] Font, J., Arcega, L., Haugen, O., and Cetina, C. (2016b). Feature location in models through a genetic algorithm driven by information retrieval techniques. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, MODELS '16, page 272–282, New York, NY, USA. Association for Computing Machinery.

[8] Ganter, B. and Stumme, G. (2005). *Formal Concept Analysis - Foundations and Applications*. Springer Science Business Media, Berlin Heidelberg.

[9] Ganter, B. and Wille, R. (1999). *Formal concept analysis : mathematical foundations*. Springer, Berlin; New York.

[10] Holthusen, S., Wille, D., Legat, C., Beddig, S., Schaefer, I., and Vogel-Heuser, B. (2014). Family model mining for function block diagrams in automation software. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, SPLC '14, page 36–43, New York, NY, USA. Association for Computing Machinery.

[11] Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A. (1990). Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[12] Leblanc, H. (2000). *Sous-hiérarchie de Galois : un modèle pour la construction et l'évolution des hiérarchies d'objets*. PhD thesis. Thèse de doctorat dirigée par Ducournau, Roland Informatique Montpellier 2 2000.

[13] Lity, S., Lachmann, R., Lochau, M., and Schaefer, I. (2013). Delta-oriented software product line test models - the body comfort system case study. Technical report.

[14] Martinez, J. (2016). *Mining software artefact variants for product line migration and analysis*. Theses, Université Pierre et Marie Curie - Paris VI ; Université du Luxembourg.

[15] Martinez, J., Assunção, W. K. G., and Ziadi, T. (2017a). Espla: A catalog of extractive spl adoption case studies. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B*, SPLC '17, page 38–41, New York, NY, USA. Association for Computing Machinery.

[16] Martinez, J., Ziadi, T., Bissyandé, T. F., Klein, J., and Traon, Y. L. (2015a). Automating the extraction of model-based software product lines from model variants (T). In Cohen, M. B., Grunske, L., and Whalen, M., editors, *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pages 396–406. IEEE Computer Society.

[17] Martinez, J., Ziadi, T., Bissyandé, T. F., Klein, J., and Traon, Y. L. (2015b). Bottom-up adoption of software product lines: a generic and extensible approach. In Schmidt, D. C., editor, *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*, pages 101–110. ACM.

[18] Martinez, J., Ziadi, T., Bissyandé, T. F., Klein, J., and Traon, Y. L. (2017b). Bottom-up technologies for reuse: automated extractive adoption of software product lines. In Uchitel, S., Orso, A., and Robillard, M. P., editors, *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume*, pages 67–70. IEEE Computer Society.

[19] Martinez, J., Ziadi, T., Klein, J., and Traon, Y. L. (2014). Identifying and visualising commonality and variability in model variants. In Cabot, J. and Rubin, J., editors, *Modelling Foundations and Applications - 10th European Conference, ECMFA@STAF 2014, York, UK, July 21-25, 2014. Proceedings*, volume 8569 of *Lecture Notes in Computer Science*, pages 117–131. Springer.

[20] Müller, T., Lochau, M., Detering, S., F, S., Garbers, H., Märtin, L., Form, T., and Goltz, U. (2009). A comprehensive description of a model-based, continuous

development process for autosar systems with integrated quality assurance. Technical report.

[21] Riebisch, M., Böllert, K., Streitferdt, D., and Philippow, I. (2002). Extending feature diagrams with uml multiplicities.

[22] Ryssel, U. (2014). *Automatische Generierung von feature-orientierten Produktlinien aus Varianten von funktionsblockorientierten Modellen*. PhD thesis, Technische Universität Dresden.

[23] Ryssel, U., Ploennigs, J., and Kabitzsch, K. (2011a). Extraction of feature models from formal contexts. page 4.

[24] Ryssel, U., Ploennigs, J., and Kabitzsch, K. (2011b). Extraction of feature models from formal contexts. page 4.

[25] Ryssel, U., Ploennigs, J., and Kabitzsch, K. (2012). Automatic library migration for the generation of hardware-in-the-loop models. *Science of Computer Programming*, 77(2):83–95. Special Issue on Automatic Program Generation for Embedded Systems.

[26] Schulze, M., Mauersberger, J., and Beuche, D. (2013). Functional safety and variability: Can it be brought together? In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, page 236–243, New York, NY, USA. Association for Computing Machinery.

[Tinnes and Martinez] Tinnes, C. and Martinez, J. Feature mining heuristics: Blocks analysis for feature identification and localization.

[28] Vogel-Heuser, B., Bougouffa, S., and Sollfrank, M. (2018). Researching evolution in industrial plant automation: Scenarios and documentation of the extended pick and place unit. Technical report, Institute of Automation and Information Systems, Technische Universität München.

[29] Vogel-Heuser, B., Legat, C., Folmer, J., and Feldmann, S. (2014). Researching evolution in industrial plant automation: Scenarios and documentation of the pick and place unit. Technical report, Institute of Automation and Information Systems, Technische Universität München.