Master Thesis

Development of an Architecture of an Integrated Analysis Framework for Change-Intensive Systems Based On Industry Needs

David Morais Ferreira
moraisferreira@cs.uni-kl.de
Student Number: 378150

July 1, 2021

First Advisor:	Prof. Dr.-Ing. habil. Peter Liggesmeyer
Second Advisor:	M.Sc. Vasil L. Tenev

Department of Computer Science

TU Kaiserslautern

Abstract

Successfully adopting Product Line Engineering (PLE) approaches in the context of software-intensive systems reduces overall development and maintenance costs, reduces time to market and leads to an overall improvement in product quality. The Software and System Product Line (SPL) community has provided a large number of analysis approaches and corresponding tools that aim to solve different analysis needs. These analysis needs arise throughout a product line life-cycle, ranging from initializing a new software product line from existing products, to improving an existing software product line. The aforementioned analysis tools, which are typically developed in the context of research studies and industrial collaborations, serve a specific purpose and are developed to fulfill a specific goal. Additionally, analysis practitioners gain an extensive understanding regarding the analysis tool capabilities, which goals it fulfills, which data formats are supported, what adaptations must be performed to integrate it with other analysis tools, and how the results should be interpreted. This information, which constitutes holistic knowledge, is rarely explicitly documented, making it hard to reuse existing analysis tools in different contexts to fulfill various goals. Furthermore, guided instructions for performing typical application scenarios are largely missing. Our work provides the following contributions: (i) to understand how organizations in industrial settings define goals in the context of SPL adoption, we study existing literature and provide an overview of our findings; (ii) we perform a mapping study with the aim of identifying and classifying existing analysis tools with respect to their applicability to different industrial settings; (iii) we introduce a classification system for describing existing analysis approaches, and possible combinations of analysis tools; (iv) we present a reference architecture for an integrated analysis framework; and finally, (v) we validate our reference architecture by instantiating a prototypical toolchain from our framework.

Keywords: reference architecture, integrated framework, product-line aware analysis, product line engineering

Zusammenfassung

Der erfolgreiche Einsatz von Product Line Engineering (PLE) Ansätzen im Kontext softwareintensiver Systeme reduziert die Gesamtentwicklungs- und Wartungskosten, verkürzt die Zeit zur Markteinführung und führt insgesamt zu einer Verbesserung der Produktqualität. Die Software- und System-Produktlinien-Community (SPL) hat eine Vielzahl von Analyseansätzen und entsprechenden Werkzeugen bereitgestellt, die darauf abzielen, unterschiedlichste Analysebedürfnisse zu lösen. Diese Analysebedürfnisse entstehen während des gesamten Lebenszyklus einer Produktlinie, von der Initialisierung einer neuen Software-Produktlinie aus bestehenden Produkten bis hin zur Verbesserung einer bestehenden Software-Produktlinie. Die oben genannten Analysewerkzeuge, die typischerweise im Rahmen von Forschungsstudien und Industriekooperationen entwickelt werden, dienen einem bestimmten Zweck und werden zur Erreichung eines bestimmten Ziels entwickelt. Des Weiteren erlangen Analysten ein umfassendes Verständnis über die Fähigkeiten des Analysewerkzeugs, welche Ziele es erfüllt, welche Datenformate unterstützt werden, welche Anpassungen vorgenommen werden müssen, um das Werkzeug mit anderen Analysewerkzeugen zu integrieren, und wie die Ergebnisse zu interpretieren sind. Diese Informationen, die ganzheitliches Wissen darstellen, sind nur selten explizit dokumentiert, was die Wiederverwendung bestehender Analysewerkzeuge in anderen Kontexten zur Erfüllung der jeweiligen Ziele erschwert. Außerdem fehlen weitgehend Anleitungen zur Durchführung typischer Anwendungsszenarien. Unsere Arbeit liefert folgende Beiträge: (i) um zu verstehen, wie Organisationen im industriellen Umfeld Ziele im Zusammenhang mit der Einführung von SPL definieren, untersuchen wir vorhandene Literatur und liefern einen Überblick über unsere Ergebnisse; (ii) wir führen eine Mapping-Studie mit dem Ziel durch, bestehende Analysewerkzeuge hinsichtlich ihrer Anwendbarkeit in verschiedenen industriellen Kontexten zu identifizieren und zu klassifizieren; (iii) wir stellen ein Klassifikationssystem zur Beschreibung bestehender Analyseansätze und möglicher Kombinationen von Analysewerkzeugen vor; (iv) wir stellen eine Referenzarchitektur für ein integriertes Analyseframework vor; und schließlich (v) validieren wir unsere Referenzarchitektur indem wir eine prototypische Toolchain aus unserem Framework instanziieren.

Keywords: reference architecture, integrated framework, product-line aware analysis, product line engineering

Ehrenwörtliche Erklärung

Hiermit erkläre ich, David Morais Ferreira, dass ich die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Luxemburg, 1. Juli 2021 Morais

Contents

List of Figures

List of Tables

List of Acronyms

ARCADIA Architecture Analysis and Design Integrated Approach.

AST Abstract Syntax Tree.

BUT4Reuse Bottom-Up Technologies for Reuse.

CPP C Preprocessor.

DIKW Data, Information, Knowledge and Wisdom.

EC Exclusion Criteria.

EPBS End Product Breakdown Structure.

FAS Functional Architectures for Systems.

FOSD Feature-Oriented Software Development.

GQM Goal-Question-Metric.

IC Inclusion Criteria.

ITEA Information Technology for European Advancement.

LCS Longest Common Subsequence.

PL Product Line.

PLA Product Line Analysis.

PLE Product Line Engineering.

ReCoVar Reverse Engineering Configurations and Variability.

REVaMP^2 Round-trip Engineering for Variability Management Platform and Process.

RQ Research Question.

SOA Service-oriented approach.

SPL Software and System Product Line.

SPLC Software Product Line Conference.

SysML Systems Modeling Language.

UI User Interface.

VA Variant Analysis.

VITAL Variability Improvement Analysis.

VM Variability Management.

1 Introduction

1.1 Motivation

In the context of change-intensive systems, software product line engineering (SPLE) is an approach for systematically building products from a common set of assets, thus reducing costs, decreasing time to market and leading to an overall improvement in product quality (Pohl et al., 2005). Increasing market pressure for mass-customization, i. e. the large-scale production of goods tailored to individual customers' needs (Davis, 1987), shorter product release-cycles, the drive for digitalization and ever-smarter technologies, as encountered in the automotive industry or in manufacturing, are prompting organizations to transition to software product line engineering approaches (Berger et al., 2020). Although there are many publications describing SPL adoptions in industrial settings, such as the SPL adoption case studies from the Software Product Line Hall of Fame[1] (Weiss, 2008), this adoption is far from trivial, as it is fraught with adoption barriers that need to be identified, understood, and overcome (Catal, 2009).

Additionally, strategic reuse, as stated by Northrop (2008), is crucial to achieving organizational business benefits beyond what is possible with single-system development approaches. To this end, both business and engineering strategies must be aligned, allowing for a planned, i. e. strategic reuse approach which leverages cross-organizational synergies and ultimately satisfies the current demand for mass-customization.

Thus, establishing a new product line in a brownfield engineering context, i. e. where existing products are developed independently of each other after an initial cloning, requires reverse-engineering activities to extract all information required to initialize the product line. This scenario seems to be prevalent in industry, as a survey performed by Berger et al. (2013) identified that 50% of the respondents have experience in using extractive strategies, i. e. extracting the commonality from existing products to bootstrap a product line, after which the existing products are incorporated into the product line (Krueger, 2002). Additionally, 47% have experience in using reactive strategies, which initializes the product line from a single product.

Moreover, reverse-engineering activities can also be used for managing product lines as they evolve over time. For example, after deriving a new product variant from a software product line, adaptations and bug fixes are performed, causing that specific variant revision to diverge from the product line (Michelon, 2020). Failure to propagate these changes back into the product line will incur technical debt, eventually requiring significant effort to correct these divergences. Thus, evolution in software product lines can be represented as a two-dimensional problem consisting of both space and time dimensions, where space refers to the variability among the products in the product line, while time refers to the evolution of shared code and

[1] https://splc.net/fame.html

individual products over time (Thao, 2012).

Having described where reverse-engineering activities find application within the context of software product lines, it becomes evident that these are of significant interest to the product line engineering community. As a consequence, the product line engineering community has developed a wide variety of analysis approaches and methods that aim to answer the many problems and questions which arise throughout a product line life-cycle, ranging from initializing a new software product line from existing products, to improving an existing software product line (von Rhein et al., 2013). These analyses often rely on tools to perform manual tasks such as extracting, preparing, and evaluating data, leaving more complex decision-making processes to engineers.

1.2 Challenge

Typically, analysis tools are developed in the context of research studies and industrial collaborations, and are used to answer specific questions that arise during these activities. As the information need increases, these tools are expanded to encompass more features, and thus answer more questions. Then, as the research progresses, the analysis practitioner gains an understanding of the tools capabilities, from the technical knowledge regarding data formats and adaptions that are required to the overarching knowledge of which information needs the tool fulfills, which insights can be gained from it, and what its limitations are.

Therefore, each analysis tool is inherently related to the context in which it is created and used, i. e. which information it processes, which information it produces, which file-types are supported, what interoperability capability it provides, and naturally, which underlying question it answers. Unfortunately, this knowledge, which constitutes a holistic approach describing the applicability of the tool, is rarely documented, thus making it hard to reuse the tool in another context.

1.3 Solution Approach

To address the aforementioned shortcomings, we propose a classification for systematically describing combinations of analysis tools and their applicability in the context of SPL reverse-engineering activities in industrial settings. Furthermore, we aim to establish an initial collection of the aforementioned analysis tool combinations.

The creation of such a collection benefits both academic and industrial SPL practitioners, as it provides a searchable documentation of analysis capabilities, their underlying data requirements and resulting findings. After defining a reference architecture that addresses the aforementioned shortcomings, we will provide a prototypical implementation for a given combination of analysis tools.

1.4 Outline

Chapter 2 provides a brief introduction to the fundamentals of product line engineering and system architectures. Then, Chapter 3 provides an overview of existing related work, and its applicability to this study. We extract goals and underlying needs from well-known case studies in Chapter 4 to understand how organizations describe needs and goals in the context of transitioning to software product lines and refactoring of existing software product lines. Chapter 5 presents a mapping study for identifying and characterizing analysis tools which are within our sphere of interest, namely tools developed at Fraunhofer IESE, and analysis tools which provide similar capabilities. Chapter 6 presents a classification system for describing existing analyses and possible combinations of individual analysis tools which solve goals as identified in Chapter 4. Then, using our newly gained insight into what drives organizations in an industrial context, and which complex analyses can be executed, we introduce a reference architecture of our integrated analysis framework in Chapter 7. To validate our reference architecture, a prototypical implementation of our integrated framework for a chosen analysis is presented in Chapter 8. Finally, Chapter 9 concludes this thesis by summarizing our contributions and provides an outlook on possible future improvements.

2 Foundations

In the following Chapter, we provide a brief introduction to the fundamentals of product line engineering and system architectures.

2.1 Software Product Lines

For the purposes of this thesis, we define change-intensive systems as systems which exist in environments where constant adaptation to new requirements is required (V. L. Tenev & Becker, 2020). Change, as described by Lindvall et al. (2016), is the result of software maintenance, evolving organizations, changing customer needs, and an increased demand for mass-customization. To fulfill these demands, these systems must thus be able to handle efficiently handle variability. Thus, evolving systems can be viewed as software product lines (Peña et al., 2007).

Clements and Northrop (2001) provide the following definition of software product lines:

> A Software Product Line (SPL) is a set of software-intensive systems sharing a common, managed set of features which satisfy the specific needs of a particular market segment or mission and which are developed from a common set of core assets in a prescribed way (Clements and Northrop (2001)).

The first key concept that can be identified from the aforementioned definition, described as "sharing a common, managed set of features", is that of variability. Metzger et al. (2007) describe two kinds of variability, namely *software variability* and *product line variability*. Software variability, refers to the "ability of a software system or artifact to be efficiently extended, changed, customized, or configured for use in a particular context" (Svahnberg et al., 2005). Product line variability refers to the variation between variants from the same product line in terms of properties and qualities, i. e. features and requirements. Thus, product line variability describes what should be common (typically referred to as commonality), and what should vary (variability) between product variants of the same product line.

While there are various opinions as to what constitutes a feature in the scientific community, ranging from abstract to technical, we define a feature as follows:

> A feature is a characteristic or end-user-visible behaviour of a software system. Features [...] specify and communicate commonalities and differences of the products between stakeholders, and [...] guide structure, reuse and variation across all phases of the software life-cycle (Apel et al. (2013b)).

The second key concept introduced in the aforementioned definition, "which are developed from a common set of core assets", is that of reuse. Wijnstra (2002) describes four levels of reuse, namely ad-hoc, systematic, domain-oriented, and strategy-driven reuse. As stated by Northrop (2008), strategic reuse is crucial to achieving organizational business benefits beyond what is possible with single-system development approaches. To this end, both business and engineering strategies must be aligned, allowing for a planned, i. e. strategic reuse approach which leverages cross-organizational synergies and ultimately satisfies the current demand for mass-customization ("which satisfy the specific needs of a particular market segment or mission").

In conclusion, the process for product line development differs significantly from that of a single-system software development process, as the focus lies on a set of product variants, not just on one product. Therefore, activities such as requirements engineering, where traditionally only one product is scoped, are replaced by activities that must address the commonality and variability for a range of different products. Subsequent development efforts must also take strategic reuse into consideration.

2.2 Engineering Process for Product Line Development

As illustrated in Figure 1, the engineering process for software product lines can be divided into two dimensions, namely *domain engineering* and *application engineering*. Furthermore, we distinguish two different perspectives, namely *problem space* and *solution space*.

The first process dimension, domain and application engineering, focuses on the different facets of asset reuse. Domain engineering (DE) is the process of analyzing the domain of a product line and developing reusable artifacts, i. e. core assets (Apel et al., 2013b). Domain engineering targets "development for reuse", as at the end of the process, no product variant is created.

The second dimension, application engineering (AE), is where product variants are actually developed. Thus, it can be said that application engineering targets "development with reuse", as depending on the specific requirements, existing assets are combined, after which the product variant is derived. Ideally, no product variant-specific development should occur during application engineering, as the modifications are not propagated back to the core assets.

As is the case for single-system development processes, software product line processes need to consider the needs and expectations from stakeholders, and as well as consider the developer's perspective (Apel et al., 2013b). Thus, two different perspectives emerge. The problem space takes the perspective of stakeholders and their problems and requirements with respect to a single product. This is important, as customers are indifferent as to whether their product is derived from a software product line or not. Thus it can be said that the problem space describes the superset of possible variability for a given software product line. In contrast to the problem space, the solution space takes the perspective of the developer and product vendor. This perspective focuses on the design, implementation, and validation and verification of the realized variability, i. e. features, from the problem space (Apel et al., 2013b).

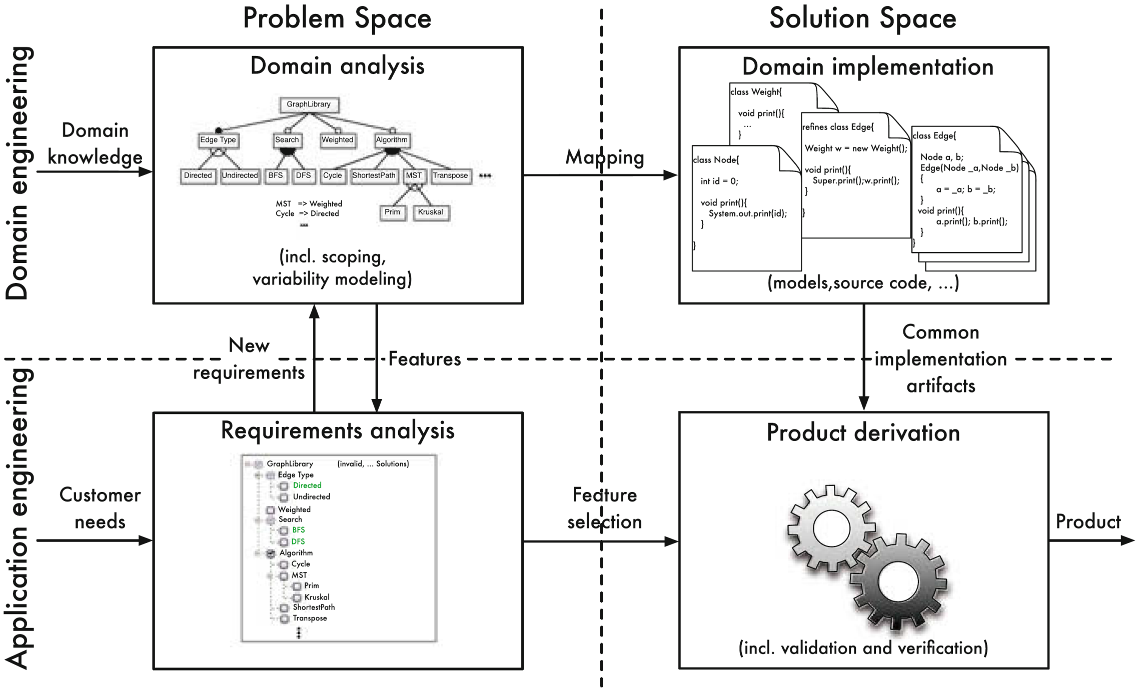


Figure 1: Overview of an engineering process for software product lines (Apel et al., 2013b)

As illustrated in Figure 1, this differentiated view of domain and application engineering, combined with the perspectives of problem space and solution space, yields four clusters of tasks, namely *Domain Analysis*, *Requirements Analysis*, *Domain Implementation*, and finally, *Product Derivation*. In the context of this work, we refer to these tasks as quadrants of the SPL engineering process model. In domain engineering, two tasks are required. The first, domain analysis, is a form of requirements engineering for an entire product line that is performed when initializing a software product line (Apel et al., 2013b). This task defines the scope of the domain, i. e. which features are relevant and should be implemented as core assets. In domain analysis, features are typically documented in a feature model, sometimes also referred to as variability model (Thüm et al., 2014), which captures features of a software product line and their relationships (Berger et al., 2013). This is commonly referred to as Variability management, and is a core concept of software product lines (van der Linden et al., 2007).

Then, during domain implementation, core assets are designed, developed and tested, as is the case in typical software-engineering processes. As previously mentioned, these core assets are “developed for reuse”, thus great care must be taken to ensure that they do not require significant effort to be reused. Furthermore, the features scoped during domain analysis are mapped to these assets. Note that the notion of assets is not limited to implementation artifacts such as source-code, but can encompass many different types of artifacts.

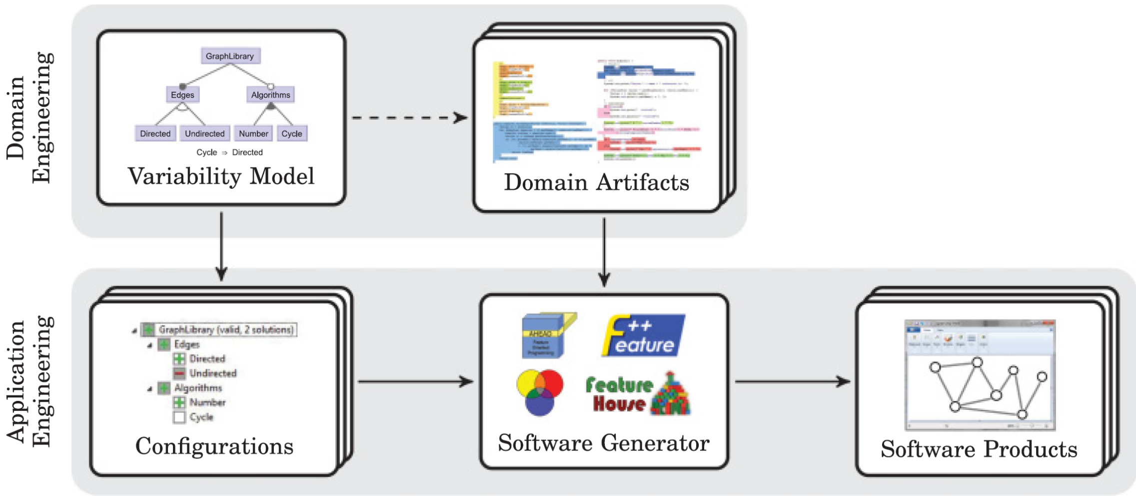


Figure 2: Process for the derivation of a new product variant from a existing software product line (Thüm et al., 2012; Thüm et al., 2014)

With regards to application engineering, we differentiate between requirements analysis and product derivation. During requirements analysis, the needs of a customer is scoped and mapped to the features provided by the software product line. If the customer specifies novel requirements, i. e. features which are not yet realized by the software product line, one can choose to extend the feature model and implement the novel requirements as core assets, or implement them in the solution assets, i. e. in the delivered product variant. During product derivation, the desired features elicited through customer requirements define which core assets are to be combined to deliver a product variant tailored to individual needs. In the simplest case, no further variant-specific development occurs. As previously mentioned, if a customer orders changes which are not to be included into the software product line, this would be where the custom-tailored development process would occur.

Figure 2 illustrates the process required for deriving a new product variant from a software product line. While domain engineering is performed once initially to setup a product line, application engineering is repeated for every product variant that is to be delivered.

In conclusion, software product lines consist of two life-cycles, domain engineering and application engineering, each fulfilling a different purpose (van der Linden et al., 2007). The Software Product Line Engineering Reference Model Framework ISO/IEC 26550:2015(E), as illustrated in Figure 4, provides an overview of the overall structure and processes of software product line engineering and describes the interactions between the components.

2.3 Product Line Adoption Strategies

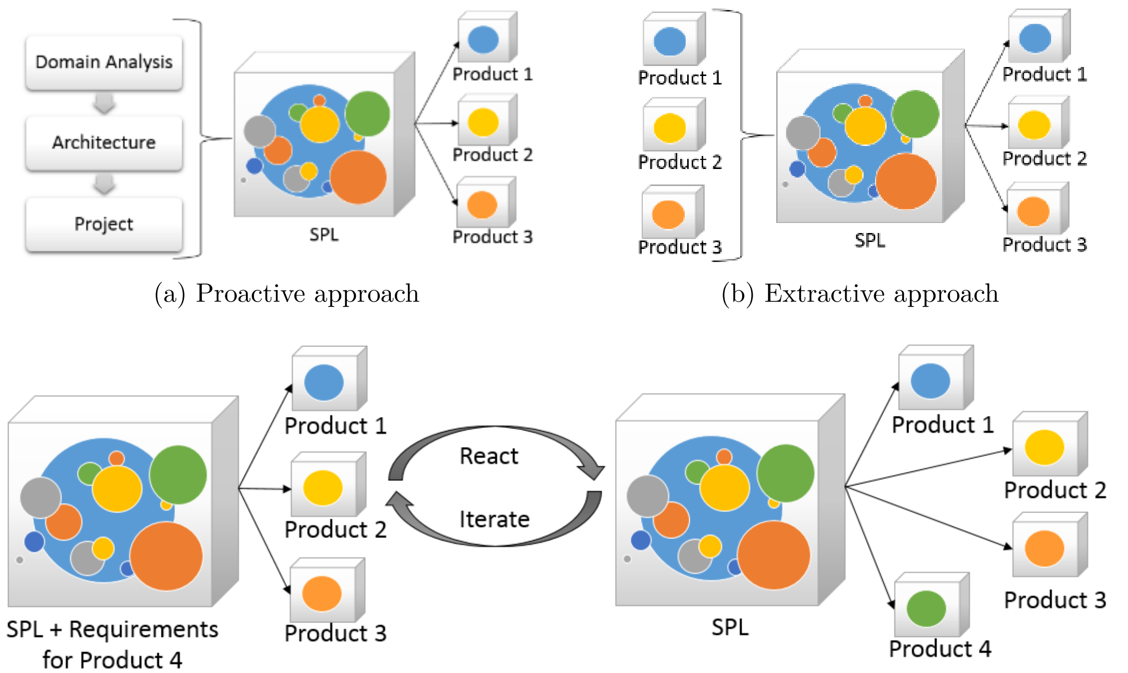
Domain engineering, which concerns itself with defining the scope of the software product line, is the first process to be performed when initializing a new software product line. Krueger (2002) introduces three different adoption models which describe the different scenarios faced by organizations when transitioning to a new software product line. Figure 3 illustrates these adoption strategies, namely the *proactive*, *extractive* and *reactive approach*.

With the proactive approach (Figure 3a), a software product line is initialized from scratch, i. e. no existing products and assets are reused. Thus, an organization first performs the design process consisting of domain analysis, requirements analysis, and domain implementation (cf. Figure 1). This clean-slate approach, as described by Apel et al. (2013b), is the most heavyweight adoption approach and can be seen as idealistic and academic.

On the other hand, the extractive approach (Figure 3b) derives a software product line from an existing portfolio of related products, which are not engineered in a systematic way. In a typical scenario encountered in industry, organizations develop a new product by copying the source-code of another, related, product and performing the required modifications. This approach, which is commonly referred to as ad-hoc clone-and-own, is not systematically managed and is prone to result in overly complex and unmanageable development processes as the products evolve (Antkiewicz et al., 2014). The extractive approach thus requires various reverse-engineering activities, such as identifying and documenting the commonality and variability of existing products, extracting core assets from existing products and tracing them to the realizable variability, and extracting and realizing the variation between individual products (Apel et al., 2013b).

Finally, the reactive approach (Figure 3c) describes an adoption path which is positioned between the proactive and the extractive approach, as the software product line evolves iteratively (Apel et al., 2013b). Starting with a software product line which realizes a single product only, each successive product variant drives the software product line to evolve incrementally, as more and more variability, as defined during domain analysis, is realized. While this approach, driven by market demands for new products and new requirements, is cost-efficient, this same demand can potentially incur significant costs, as core assets may necessitate significant reengineering activities (Krueger, 2002; Apel et al., 2013b).

Note that an organization can follow multiple strategies at once, as they aren't mutually exclusive. For example, to initialize a new software product line, an organization might follow an extractive approach at first, and then transition to a reactive approach for incrementally evolving the software product line over time (Berger et al., 2013; Krueger, 2002). In a survey, Berger et al. (2013) identified that both the extractive and reactive approaches are widely used in industrial settings.



(a) Proactive approach (b) Extractive approach

(c) Reactive approach

Figure 3: Software product line adoption approaches (Martins et al., 2018)

2.4 Analysis of Software Product Lines

In the context of software product lines, variability introduces new analysis challenges. While in traditional single-system engineering each product can be analyzed individually, the sheer amount of possible product variants which can be derived from a software product line is infeasible. In a survey, Berger et al. (2013) identified that in industrial settings 39% of respondents report feature models containing between 100-1000 features, and 26% reported feature models with over 10,000 features. The Linux kernel, for example, has over 10,000 features (Tartler et al., 2011). Therefore, variability-aware analyses, which allow for an efficient analysis of derivable products, are required. Furthermore, evolution, as described in Section 1.1, can be represented as a two-dimensional problem, consisting of both space and time dimensions, where space refers to the variability among the products in the product line, while time refers to the evolution of shared code and individual products over time (Thao, 2012). To differentiate which reverse-engineering goals variability-aware analyses contribute to, Berger et al. (2019) differentiate between platform-centric analysis approaches, and variant-centric analysis approaches. While platform-centric approaches refer to analyses pertaining to existing, i. e. configurable, software platforms, and are thus interested in undocumented variability, variant-centric approaches refer to analyses which only analyse existing product variants.

Summarizing, variability-aware analyses can contribute to initializing software product lines from existing products, i. e. extractive and reactive approaches, and improving, or refactoring, existing software product lines. The latter aims to reduce complexity introduced by evolution in both space and time.

Apel et al. (2013a) introduce a range of different domain-engineering analyses which are of use in software product lines. As such, feature models, feature-to-code mappings, and domain implementations, i. e. core assets, can be analyzed. Analysing feature models, for example, gives insight into what features are mandatory, what is the minimal subset of features which are required to derive a valid product variant, and which features can never be selected due to modelled constraints. When analysing features and their mapping to core assets, represented by the lateral edge connecting domain analysis to domain implementation in Figure 1, gives insight into how features are realized, what interdependencies between features can be detected from the core assets and which code can never be shipped in a product variant, as the corresponding features are not modelled, or cannot be selected. Naturally, this insight lends itself to reverse-engineering a feature model from existing core assets. Finally, analysing domain implementations yields insight into program structures, such as function calls or statements. Typically, these analyses consist of type checking and model checking variability-aware core assets. Furthermore, assets from application engineering can also be analyzed. Analysing the requirements analysis, i. e. the configuration for each product variant, yields valuable insight into which features are always enabled, which combinations of features are typical, or which features are never ordered by customers. The identification of features which are never ordered, for example, could lead to the decision to remove the corresponding core assets from the product line, thus reducing maintenance costs.

In conclusion, variability-aware analyses which support reverse-engineering activities are vital to transitioning to a software product line, and for managing the complexity of existing software product lines. As illustrated in Figure 5, a software product line can evolve in both the employed strategy (independent, platform-based) and methodology (ad-hoc, strategic). In this context, these analyses are also essential, as they directly contribute information which is required for an efficient transition.

2.5 System Architecture

Cloutier et al. (2010) provide a succinct working definition as to what constitutes a reference architecture and what purpose it fulfills:

> Reference Architectures capture the essence of existing architectures, and the vision of future needs and evolution to provide guidance to assist in developing new system architectures (Cloutier et al. (2010)).

As illustrated in Figure 6, reference architectures, which are abstract in nature, serve as a blueprint for actual system architectures, thus guiding the elaboration of future architectures. Furthermore, system architectures provide valuable feedback for updating the reference architecture, ensuring that it reflects the current and future needs of the industry.

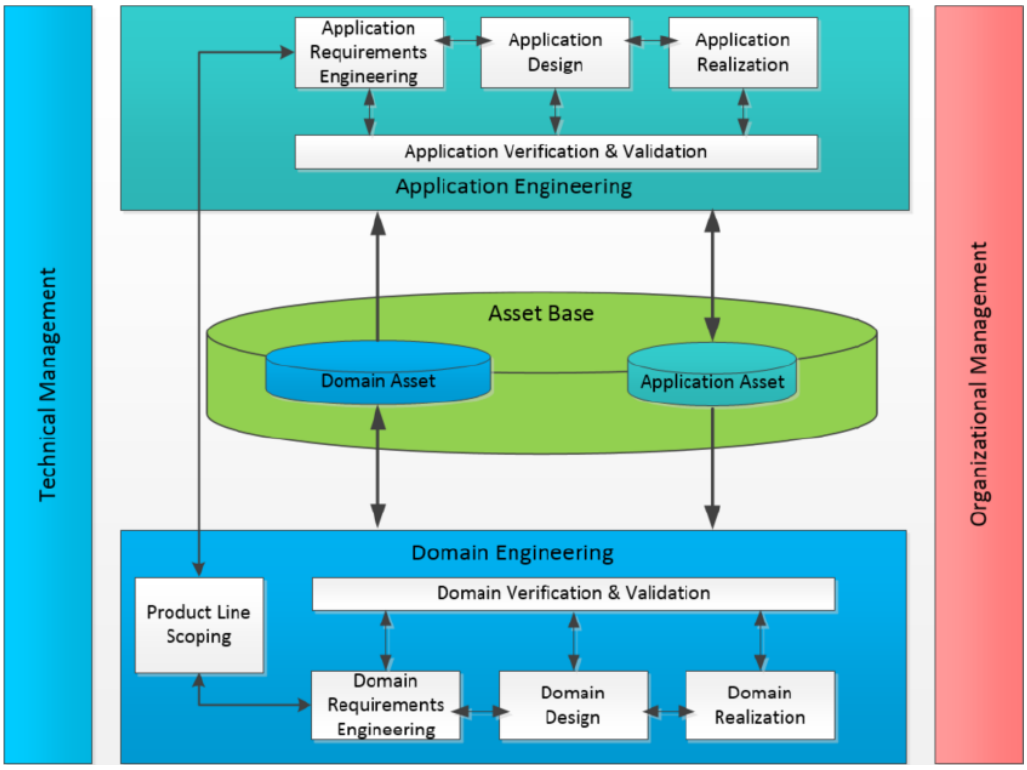


Figure 4: ISO/IEC 26550:2015(E) (2015) Software Product Line Engineering Reference Model Framework (adapted by Becker (2018))

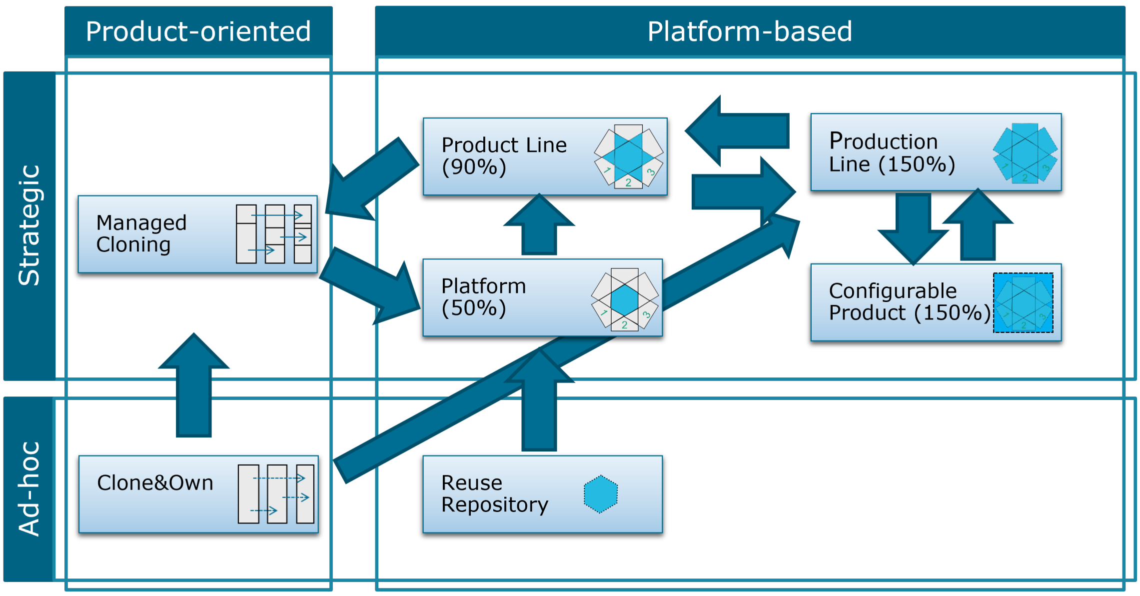


Figure 5: Possible evolution of software product lines (Becker, 2018)

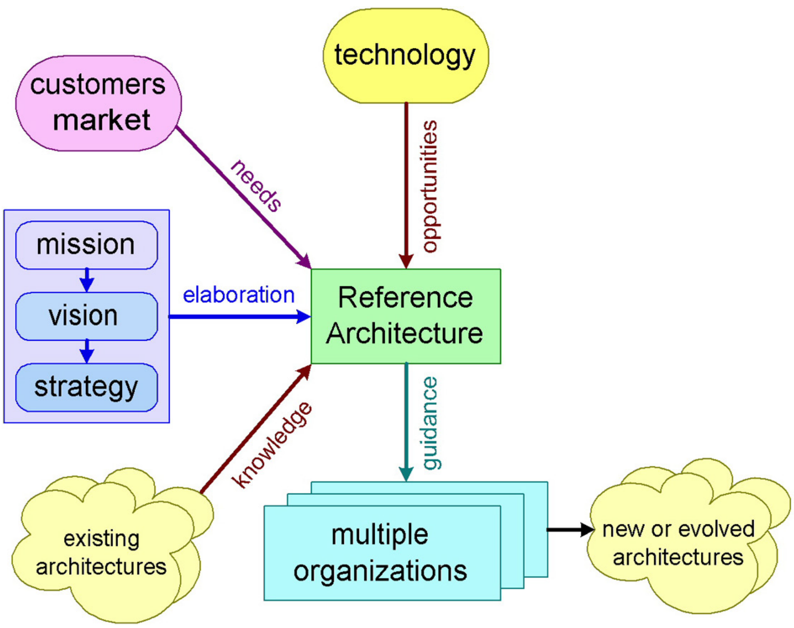


Figure 6: Summary of the role of reference architectures. Feedback loops have been removed for improved clarity (Cloutier et al., 2010)

In accordance with the aforementioned working definition, our reference architecture is derived from prior experiences and efforts associated with integrating multiple analysis tools into one. By defining an initial reference architecture, we lay the foundation for future improvements, subsequently enabling the development of an integrated analysis framework which fulfills our specific needs, namely interoperability between analysis tools. Furthermore, Cloutier et al. (2010) state the following benefits which reference architectures provide from an industrial perspective: (i) reference architectures provide a common understanding of the architecture, thus explicitly documenting domain knowledge; (ii) they enable reuse, potentially reducing costs incurred in later development stages; and (iii) they increase the quality of the end product.

In conclusion, while reference architectures generate additional effort, they provide value to the organization, and are thus useful for the overall achievement of business goals.

While we could have chosen the Functional Architectures for Systems (FAS) method, which is a method for deriving functional architectures from use-cases (Lamm & Weilkiens, 2014) and which uses the Systems Modeling Language (SysML), we ultimately opted for the Architecture Analysis and Design Integrated Approach (ARCADIA) model-based engineering method[2], as it enforces a clear separation between system context, need modeling and solution modeling. Furthermore, excellent tool-support exists for ARCADIA, and the method is thoroughly documented. As

[2]For a detailed overview of the differences between SysML and ARCADIA, please refer to Voirin et al. (2016) and https://www.eclipse.org/capella/arcadia_capella_sysml_tool.html

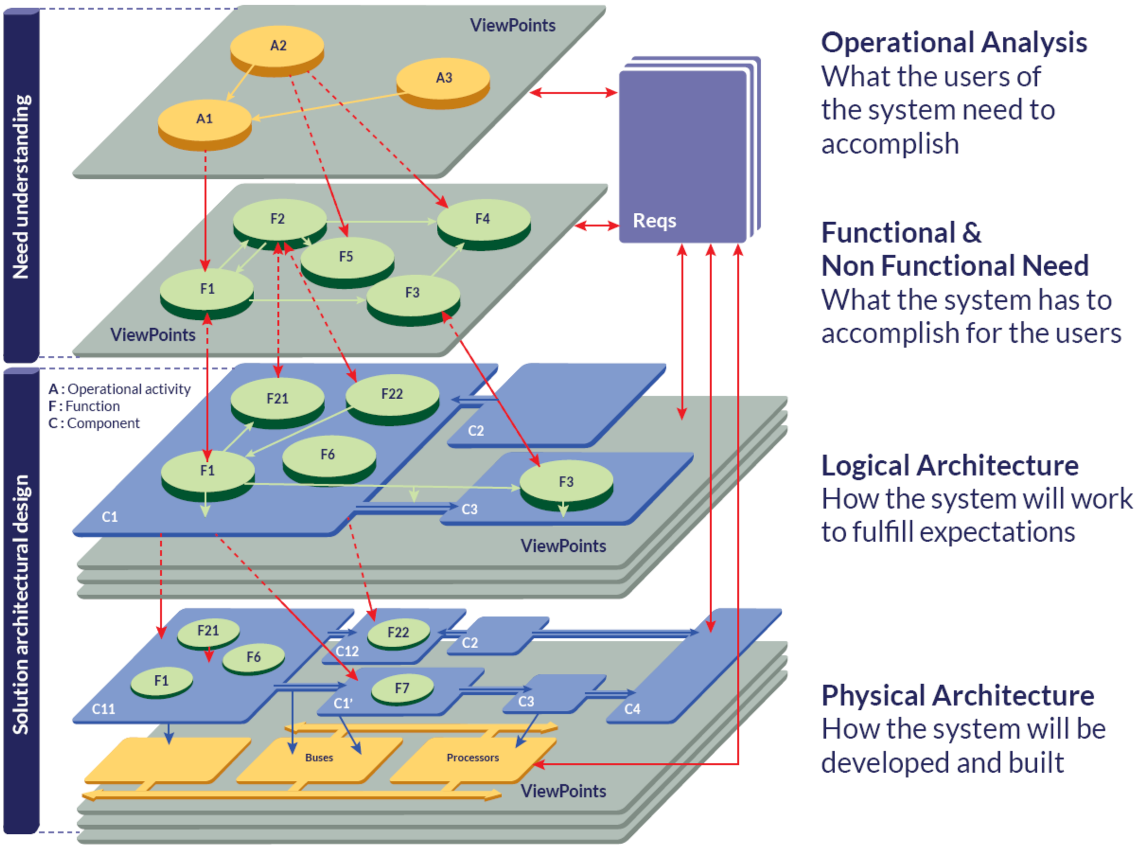


Figure 7: Overview of the main architectural levels of ARCADIA (Roques, 2018)

described by Roques (2018), ARCADIA is a structured engineering method aimed at defining and validating the architecture of complex systems. Through a shared model, all engineering stakeholders share the same information, which enables a common understanding of customer needs, and the product architecture needed to meet that need. Furthermore, different viewpoints and architectural levels enable a decomposition of a complex system into models used by different engineers throughout the process life-cycle. Figure 7 provides an overview of the different architectural levels defined by the ARCADIA method.

To guide the practitioner throughout the ARCADIA method, Eclipse Capella[3], a model workbench software, provides a plenitude of useful features for linking information both between different architectural levels and within different models of the same level. Given that ARCADIA focuses on architectural levels and doesn't enforce a strict sequence of actions, it makes no difference whether a practitioner follows a top-down, bottom-up, or incremental approach.

Another argument which supports our decision to use ARCADIA is that the aforementioned method provides a lot of freedom. Given that each system under study may require a different approach, there is no mandatory level at which a

[3]https://www.eclipse.org/capella/

practitioner must begin modeling. Thus, operational analysis, logical architecture and End Product Breakdown Structure (EPBS) are considered to be optional levels and may be skipped, depending on the project context and complexity.

In conclusion, the freedom provided by the ARCADIA method, coupled with the excellent software-support provided by Capella, makes it an interesting choice for modeling our reference architecture, as subsequent improvement efforts to our reference architecture can be seamlessly integrated into our existing model.

3 Related Work

SPL adoption cases in an industrial context are well-documented (Weiss, 2008; van der Linden et al., 2007). However, we believe that the underlying motivation for transitioning to a SPL is still of current interest, as research publications typically only report “success stories” (Rabiser et al., 2019), and sharing information specific to organizations is carefully considered as to not divulge any sensitive information. This opinion seems to be supported by Berger et al. (2020), which have found that most case studies don’t describe challenges, negative experiences, or shortcomings of tooling faced during the SPL adoption process.

Thus, reported success stories tend to vaguely describe the underlying needs faced by the organization, and almost always mention typical business benefits, i. e. reduced costs, lower time to market and improved quality.

Regarding tools, Rabiser et al. (2018) found that there is a strong increasing trend in both academic and industrial research in recent years. Furthermore, Rabiser et al. (2018) found that reverse-engineering variability from existing assets and reengineering existing products into a product line has attracted more interest from academic research than from industrial research. Nevertheless, in a multiple-case study, Berger et al. (2020) observed that the adoption of software product line engineering concepts is a tool-integration problem, but leave open the question as to how tool-integration can be improved upon.

In an effort to solve tool-integration needs for forward-engineering activities, Horcas et al. (2019) first present an overview of the existing SPL tools, their availability and their usability. Furthermore, 12 roadmaps, consisting of combinations from a subset of the previously identified tools, which allow for a complete SPL engineering process, i. e. from domain analysis to product derivation, are presented. While Horcas et al. (2019) demonstrated that existing tools do support performing a complete SPL process, they are fraught with limitations for non-trivial software product lines. The aforementioned roadmaps are also available online[4]. Thus, even in the “simple” case of performing a complete SPL process, which stands in contrast to the complex reverse-engineering activities required for deriving a SPL from existing products, current tool-support could be improved upon.

So as to obtain an overview of existing product-line aware analysis tools, Thüm et al. (2014) performed a survey and classified existing tools along a given scheme. To facilitate further research and enable analysis practitioners to identify potentially interesting tools with less effort, Thüm et al. (2014) have made the corpus available online[5]. Furthermore, Meinicke et al. (2014) refined the aforementioned corpus, focusing on tools which enable feature-oriented software development (FOSD). Likewise, this corpus is also available online[6]. However, both corpora mainly contain analysis tools which are available online, thus lacking analysis tools which were never

[4]https://github.com/jmhorcas/SPLC2019
[5]http://thomas.thüm.de/pla/
[6]http://www.fosd.net/tools

released publicly, as is the case for various analysis tools developed at Fraunhofer IESE[7]. For tools supporting domain analysis, Benavides et al. (2010) and Lisboa et al. (2010) provide an overview of existing tool support, and analysis of feature models, respectively.

Assunção et al. (2017) present a systematic map of tools in the context of industrial reengineering activities. This systematic map presents tools, which phases of the reengineering process they contribute to, their applicability to the different reengineering process strategies, and which input and output artifact types are supported.

For classifying and comparing both variability-aware and variability-unaware analyses, von Rhein et al. (2013) introduce the Product-Line-Analysis-Model (PLA-Model), a formal model which lays the foundation for systematically describing combinations of individual analyses. In this work, von Rhein et al. (2013) provide a few examples of analysis combinations. This formal approach describes specific analysis tools as a white-box on a logical level.

In the context of the ITEA3[8], which is a Eureka cluster programme for the advancement of software-intensive systems, the ReVaMP^2 consortium[9] aims to conceive and evaluate comprehensive tool-supported round-trip engineering processes for leveraging legacy assets (Martinez & Parsai, 2018). ReVaMP^2 aims to provide a comprehensive toolchain which automates product line extraction, and supports the verification and management of the co-evolution in software product lines (Martinez & Parsai, 2018; Sadovykh et al., 2019). As described by Martinez and Parsai (2018) and Sadovykh et al. (2019), notable results include KernelHaven[10], an experimentation workbench for developing research prototypes in the context of SPL analyses. KernelHaven provides a wide range of wrapper-plugins for common data extractors and analysis tools. Controlled by a central configuration file, KernelHaven can extract information from variability models, build files, and source-code (Kröher et al., 2018b; Kröher et al., 2018a). Depending on how KernelHaven is used, it can be considered a toolchain which fulfills a specific complex analysis need. However, no integrated data-model is used, as the different extractors and analysis tools each use their internal representations. Furthermore, existing open-source tools, such as But4Reuse, were improved upon. But4Reuse is an Eclipse-based tool for mining software artifact variants, and provides capabilities such as, but not limited to, variability and commonality analysis, feature extraction and identifying feature constraints (Martinez et al., 2015).

Also supported by ReVaMP^2, Grüner et al. (2019) introduce four scenarios which motivate the need for an holistic toolchain which enables extraction and visualisation approaches for both variant-centric and platform-centric approaches in an industrial

[7]Fraunhofer Institute for Experimental Software Engineering (IESE), https://www.iese.fraunhofer.de

[8]Information Technology for European Advancement, https://www.itea3.org

[9]Round-trip Engineering for Variability Management Platform and Process, http://www.revamp2-project.eu

[10]https://github.com/KernelHaven/KernelHaven

context. While the former refers to scenarios where the object of analysis is a set of products, i. e. products developed via clone-and-own, the latter refers to a scenario where variants are derived from an existing product line. This work presents first steps towards the creation of this toolchain, which makes use of both existing and newly-developed analysis tools. Unfortunately, the toolchain is not publicly available, and is specifically tailored for feature extraction, as defined by Burger and Grüner (2018) consists of feature identification, feature localization and feature tracing.

Thus, we may conclude that while overviews of existing analysis tools are plentiful, combinations of individual analysis tools, which provide the foundations for integrated toolchains, are still an open issue which are of current interest for both academic and industrial research.

4 Identification of Goals in Industrial Settings

In the following Chapter, we aim to gain an understanding of how organizations define goals when migrating to, or improving existing, PLE approaches. To this end, we will derive goals from well-known cases of industrial product line engineering adoption in industrial settings. This Chapter has been submitted to the 2021 23rd Workshop for Software-Reengineering&-Evolution organized by the *Software-Reengineering* working group[11] of the *German Informatics Society* (GI) under the title "Deriving Goals From Well-Known Industrial Cases of Product Line Engineering Adoption". While the main workshop was cancelled and replaced by a smaller online-only workshop (entitled WSRE light 2021[12]), we still presented our submission to the workshop participants.

4.1 Motivation

Goals describe desirable results whose achievements require careful planning of actionable steps, commitment of resources, and purposeful and productive actions (Locke & Latham, 1990). Moreover, goals are articulations of fundamental needs, and may be related to one or multiple needs, and vice versa. Note that in this thesis, the notion of goals and needs may be used interchangeably, differing only in perspective. For example, while a technical need may be formulated as "reduce configuration complexity", a corresponding goal may be expressed as "introduce tool support for automating the configuration process".

Therefore, defining attainable goals paves the way to a successful adoption of product line engineering approaches. One key aspect of which is strategic reuse, which, as stated by Northrop (2008), is crucial to achieving organizational business benefits beyond what is possible with single-system development approaches. To this end, both business and engineering strategies must be aligned, allowing for a planned, i. e. strategic reuse approach which leverages cross-organizational synergies and ultimately satisfies the current demand for mass-customization.

The needs and expectations of different stakeholders throughout the organizational hierarchy can be systematically described by defining high-level *business* goals and successively decomposing them into more concrete, lower-level *technical* goals. To this end, we analyzed a subset of the available publications, focusing on well-known case studies to elicit a set of goals and attempt a first classification of goals that might typically be encountered in an industrial setting.

[11]Fachgruppe Software-Reengineering, Gesellschaft für Informatik, https://fg-sre.gi.de/
[12]https://fg-sre.gi.de/veranstaltung/workshop-software-reengineering-evolution-light

4.2 Data Collection

First, representative publications for each organization inducted into the SPLC's Hall of Fame[13] were chosen from the list of suggested references. As some did not provide useful insight into goals, the scope was expanded to include additional primary and secondary literature (Bass et al., 2003; Berger et al., 2020; van der Linden et al., 2007). Then, in a manual process, goals were extracted from the available literature. This was performed by reading through the documents, identifying and extracting the relevant passages, and converting them into a tabular representation. As we expect business-level goals to be mentioned repeatedly, we systematically exclude the following *business-level* goals: (i) time to market; (ii) reduce costs; and (iii) improve quality.

After creating this list of relevant passages, containing either explicitly or implicitly defined goals, we proceeded to refine them into goals. If the passage explicitly mentioned a goal, it was copied without modification. Otherwise, we interpreted the passage and formulated one or more explicit goals. Then, this flat list of goals was deduplicated, and we turned our focus to organizing them hierarchically along the aforementioned *business-level* goals.

In a first step, each identified goal was assigned to one of the three business-level goals, depending on what contribution the identified goal provided to the overall picture. As one goal rarely only affects a single business-level goal, we defined that a goal should be assigned to the higher-level goal to which it contributes the most. Furthermore, interdependencies between goals with different parent-goals were not modelled, as the resulting overview would be very cluttered.

This bottom-up approach of iteratively processing goals was useful, as it quickly resulted in a tree-like shallow-depth overview. Unfortunately, the jump in granularity between the assigned goals was sometimes too high, leading us to manually introduce additional goals. This resulted in an overview with a deeper, richer, hierarchy. These manual additions are represented in yellow in the subsequent figures. To improve the hierarchy for "improve quality", we relied on the characteristics described in the ISO/IEC 26550:2015(E) (2015) standard to provide the second level. This choice was made as the aforementioned standard is more extensive compared to its superseded predecessor, namely the ISO/IEC 9216-1 (2001) standard.

4.3 Results

As outlined by van der Linden et al. (2007), the main business advantages can be classified into *product qualities* and *process qualities*. While product qualities describes characteristics that impact product quality and its perception by end-users, such as reliability and maintainability, process qualities describes process characteristics which are affected by product line engineering activities. Naturally, improving the development efficiency will reduce the overall costs and lower the time to mar-

[13] https://splc.net/fame.html

ket. This will enable the company to strengthen their foothold in existing market segments, potentially enter new markets, and allow for a greater competitiveness.

While top-level business goals fundamentally do not change over time, technical goals, i. e. goals whose achievement drives the aforementioned goals, may change over time due to the ever-changing landscape of academic and industrial research. For instance, challenges regarding the user-friendliness of variability modeling have mostly been addressed by the advances of commercial and open-source tools (Berger et al., 2020).

Most, if not all, cases mentioned “decreasing time to market” (Figure 8), “improving quality” (Fig. 9), and “reducing costs” (Figure 10) as motivating goals. These form the top-level goals in our overview. In our set of publications, most goals are either very abstract (e. g. “improve quality”), or very specific, describing solution-oriented goals such as “remove unused code”. Unfortunately, current case studies don’t typically describe actual challenges, instead they focus on successes and lessons learned (Rabiser et al., 2019; Berger et al., 2020). “Reduce possible configuration space” (Figure 10), for example, could have two different interpretations, depending on which higher-level goal is prioritized. By reverse-engineering the dependencies between switches, constraints can be added to the configuration space, therefore minimizing it. This reduces the overall complexity. Alternatively, configuration switches that are always identical for all delivered variants can be removed from the platform, and the associated variability (which is never used) can be stripped from the core assets. This solution-driven mindset misses the underlying root cause, potentially diverting resources from high return on investment activities to lesser efficient ones.

4.4 Conclusion

As the aforementioned goal-derivation process required some level of interpretation, the resulting overview cannot be claimed to be free of biases. When presenting our findings at the *WSRE light* 2021 workshop, participants expressed that quality goals are typical drivers for change, and are rated very highly overall. However, lacking representative data, we cannot make any claim as to whether the three business-level goals are equally important, or whether there is a hierarchy.

In conclusion, we believe that our approach provides useful insight into what goals and underlying needs prompt organizations to adopt or refactor existing software product lines. Thus, explicitly documenting goals provides an overview of the overall problem space in industrial settings, and provides first insights into potential benefits of our analysis cookbook.

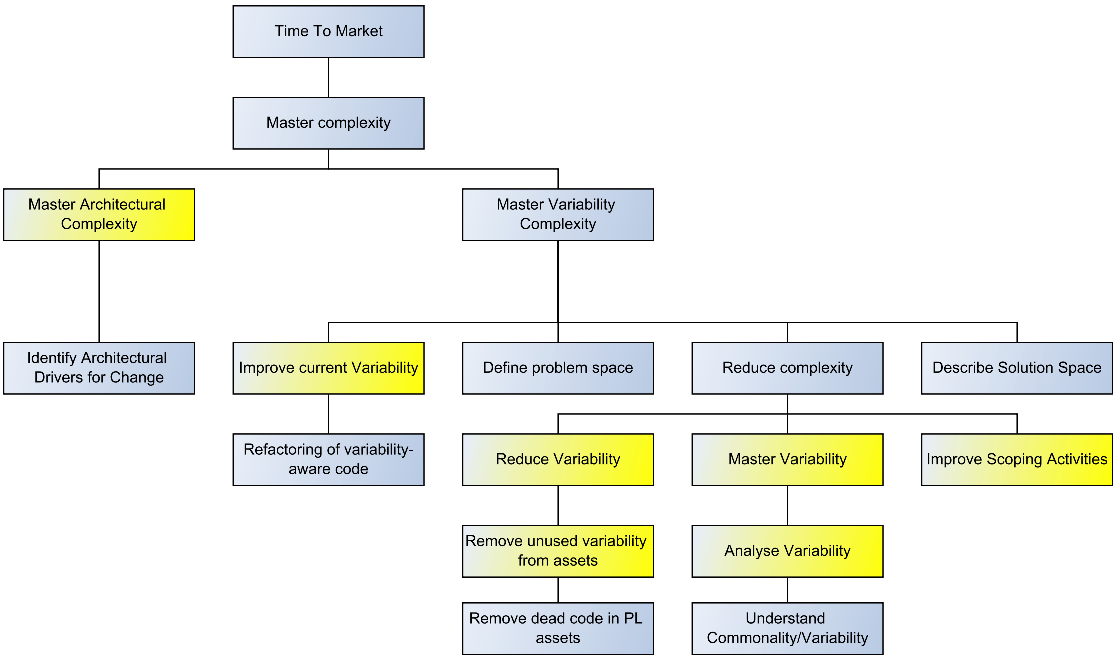


Figure 8: Goals concerning time to market

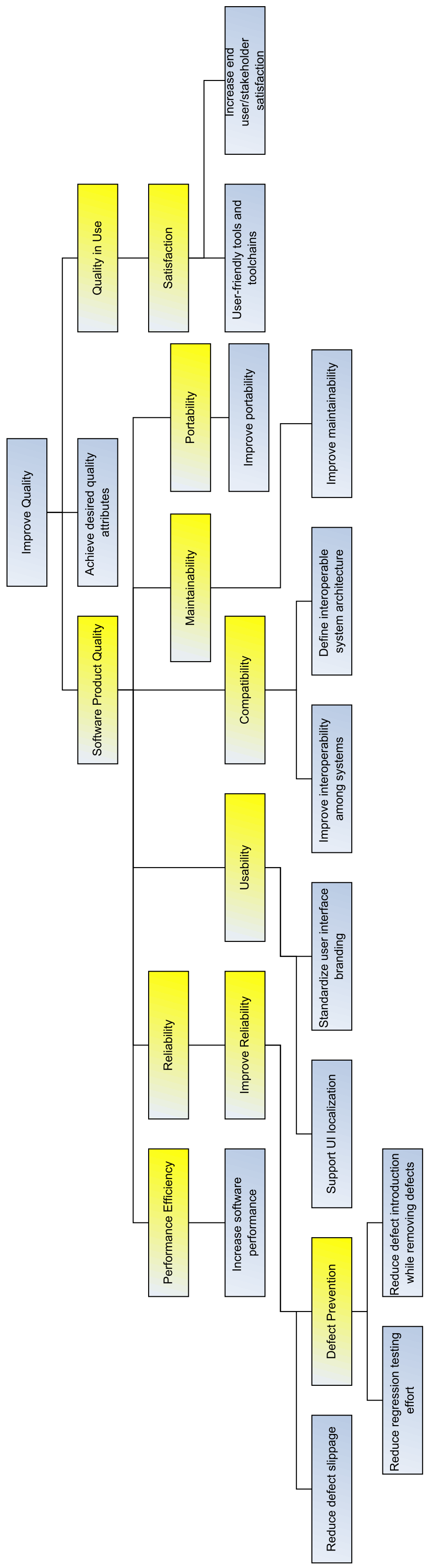


Figure 9: Goals concerning quality

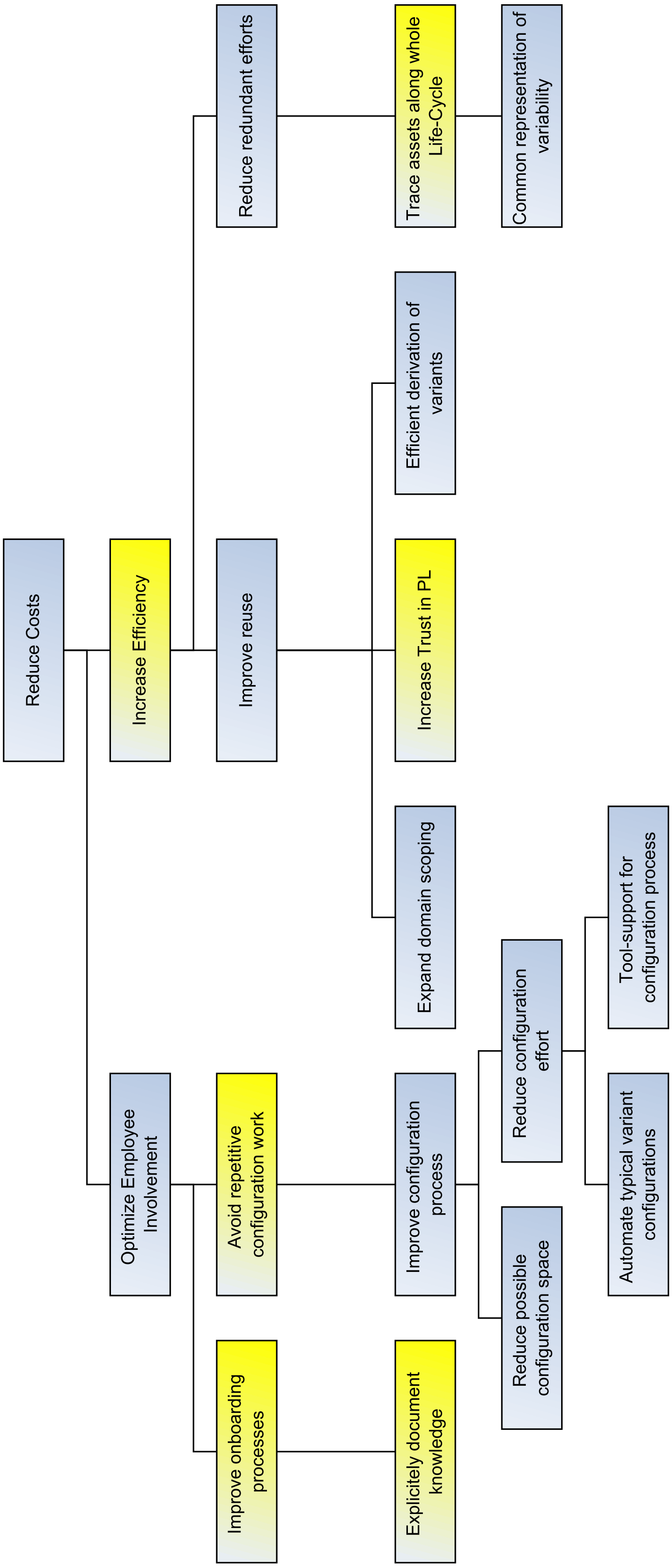


Figure 10: Goals concerning costs

5 Systematic Identification of Analysis Tools

In the following Chapter,we perform a mapping study for identifying and characterizing analysis tools which are within our sphere of interest, namely tools developed at Fraunhofer IESE, and analysis tools which provide similar capabilities.

5.1 Motivation

As previously described, our analysis cookbook contains analysis recipes, which are composed of combinations of different analysis tools, each fulfilling a different capability. While an analysis recipe fulfills a higher-level goal, individual analysis tools typically fulfill technical information needs. Thus, it is important to identify and accurately describe existing analysis tools, focusing on their characteristics, capabilities, and which transitions through the different quadrants of the SPL engineering process they enable.

While classifications of existing analysis tools already exist, as described in Chapter 3, they do not describe the information we require. Furthermore, we will restrict our efforts to analysis tools developed at Fraunhofer IESE, and external analysis tools which provide similar capabilities, as a complete overview is out of scope for this thesis. To systematically identify these analysis tools with respect to the aforementioned facets, we will conduct a systematic mapping study, rather than a systematic literature review. This being due to the fact that, in contrast to the latter, the former provides a broad overview of a particular research field and is used to answer higher-level questions. However, both share a common methodology for searching and including literature (Kitchenham & Charters, 2007; Petersen et al., 2015).

This knowledge, i. e. which analysis tools exist and which transitions they enable, lays the foundation for our analysis cookbook recipes.

5.2 Planning of the Procedure

5.2.1 Research Question

We define the following research questions:

RQMAP1: Are there analysis tools similar to ours that are publicly available?

RQMAP2: Which transitions across the different SPL engineering process quadrants are possible with the identified analysis tools?

5.2.2 Study Selection

To identify relevant publications, Petersen et al. (2015) define three different search strategies, namely snowballing, manual search and database search (ordered from less popular to most popular). For our approach, the snowballing search strategy was chosen due to its efficiency and effectiveness. As described by Wohlin (2014), snowballing procedures can be decomposed two-fold: backward snowballing, which uses the reference list of a publication to identify additional publications, and forward snowballing, which relies on a citation database to identify publications which reference our already-identified publication. For our purposes, we will use the citation tracker provided by Google Scholar[14] to identify forward citations. We automated the retrieval process through the use of Publish or Perish. To enable a reproducible decision-making process, we defined inclusion criteria (IC) and exclusion criteria (EC) for deciding whether a publication, and corresponding analysis tool, should be included, or excluded, in the subsequent iteration. By extensively defining exclusion criteria, we willfully restricted the scope of our mapping study.

We defined the following inclusion criteria:

IC1. Publication presents an analysis tool that fulfills a similar purpose to analysis tools included in the start set.

IC2. Analysis tool provides capabilities which pertain to one or more quadrants of the SPL engineering process as described in Figure 1.

IC3. Analysis tool artifacts (source-code, executable files) are publicly available. For Fraunhofer IESE analysis tools, artifacts are stored in internal repositories.

Furthermore, we defined the following exclusion criteria:

EC1. Publication isn't peer-reviewed.

EC2. Publication is not written in English.

EC3. Publication was published before 2000.

EC4. Publication is not accessible in full-text.

EC5. Publication is a duplicate of another publication, i. e. the same publication was submitted to multiple venues.

EC6. Publication presents results of a systematic study (systematic mapping, systematic literature review).

EC7. Publication presents intermediate results of another, already identified, study.

EC8. Publication does not introduce a new analysis tool, relying instead on one or more existing tools to perform a case study.

EC9. Publication presents an analysis tool which possesses less capabilities than the analysis tools captured in the start set.

EC10. Publication presents an integrated toolchain.

EC11. Analysis tool doesn't process source-code assets.

[14] https://scholar.google.com/

EC12. Publication presents a Linux-centric analysis tool, i. e. an analysis tool which analyses Makefiles and Kconfig configurations.

EC13. Analysis tool does not contribute to the outlined thesis topic.

EC14. Tool is not available online, or is not available to us.

To summarize the aforementioned exclusion criteria: We are interested in analysis tools which fulfill a similar purpose to ours and are available online. To keep the overall effort manageable, we decided to also exclude analysis tools which analyse assets such as models and existing feature models. To further reduce the scope, we explicitly excluded analysis tools which are specifically tailored to the Linux "ecosystem", i. e. analysis tools for analysing Makefiles and Kconfig configurations (EC12).

Thus, given a start set of publications, we iteratively process the reference list and citation list of each publication and, using the aforementioned inclusion and exclusion criteria, expand our list of relevant publications (cf. Figure 11). While one could read every publication in its entirety, this would be very time-consuming and not very efficient. To reduce this effort, Wohlin (2014) recommends making a decision based on reading the title, abstract and only browsing through the publication content. Furthermore, the context of the reference is also useful for deciding whether to include or exclude the publication.

5.2.3 Building a Start Set

As illustrated in Figure 11, the first step of the snowballing procedure consists of identifying a tentative start set. This challenging task should not be underestimated, as the quality of the start set directly impacts the outcome of the procedure. To avoid biases and typical pitfalls, Wohlin (2014) has defined characteristics that a good start set should possess:

Diversity: A good start set provides good coverage of the study area. This avoids the risk of missing publications from independent clusters. Therefore, it should encompass publications from different authors, years and publishers. As our interest lies in identifying analysis tools which offer similar capabilities to those developed at Fraunhofer IESE, we directly added the relevant research publications to our tentative start set. Traditionally, one would define relevant keywords and identify a first set of publications from the search results. As we rely on our prior knowledge to build a start set, we ensure that all, if any, independent clusters are represented.

Size: The ideal start set size depends on the study area, as a focused study area generally requires a smaller, more focused, start set. For our purposes, a smaller start set is sufficient, as we are only interested in specific analysis tools. Furthermore, only one person will be performing the snowballing procedure.

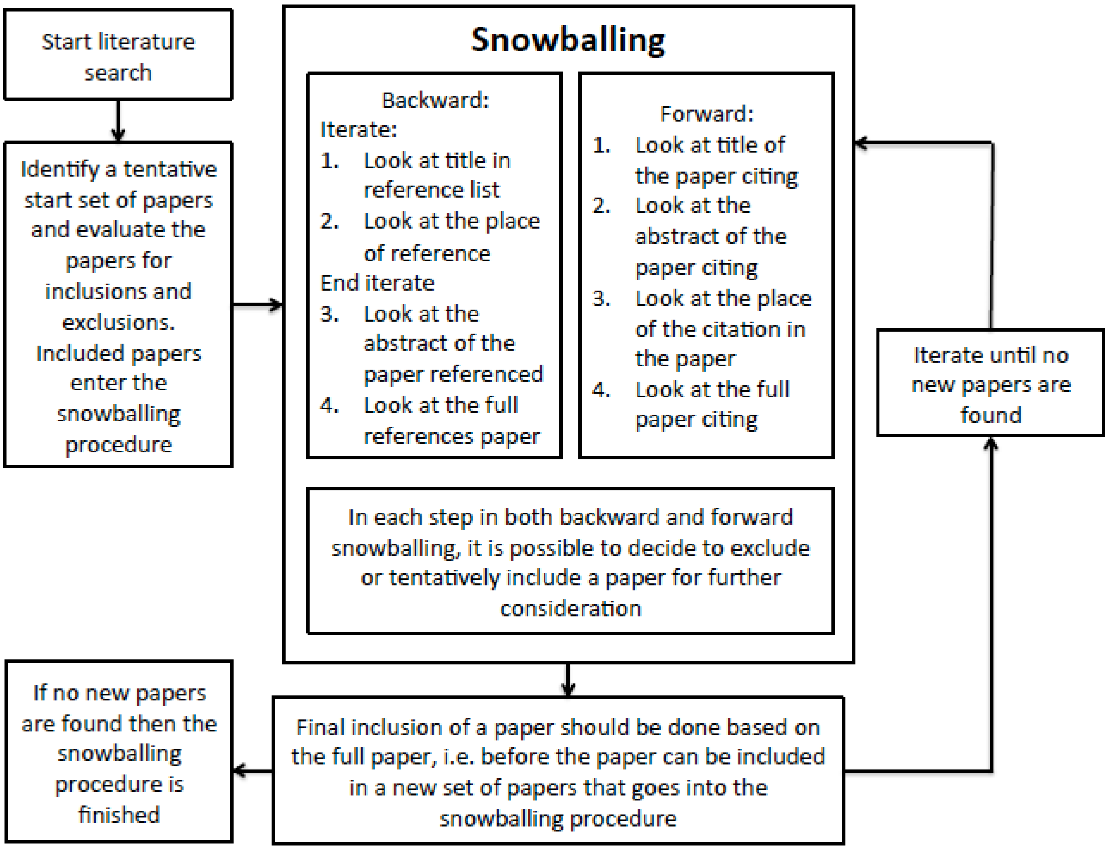


Figure 11: Overview of the snowballing procedure (Wohlin, 2014)

By relying on prior experience and by interviewing researchers familiar with the relevant tools, we identified the following publications for inclusion into our tentative start set. Note that a leading asterisk indicates that a publication has been dropped from the tentative start set.

C1. Zhang, B., & Becker, M. (2012). Code-based variability model extraction for software product line improvement. *Proceedings of the 16th International Software Product Line Conference - Volume 2*, 91–98. https://doi.org/10.1145/2364412.2364428

C2. Zhang, B., & Becker, M. (2014). Variability code analysis using the VITAL tool. *Proceedings of the 6th International Workshop on Feature-Oriented Software Development*, 17–22. https://doi.org/10.1145/2660190.2662113

C3. * Zhang, B. (2015). *Phd theses in experimental software engineering. Vol. 53. VITAL - Reengineering variability specifications and realizations in software product lines* (*Vol. 53*). Fraunhofer IRB Verlag. http://publica.fraunhofer.de/dokumente/N-345252.html

C4. * Morais Ferreira, D., Becker, M., & Tenev, V. L. (2020). Experience report on variability improvement in a product line engineering unaware company. *Proceedings of the 24th ACM International Systems and Software Product Line Conference - Volume B*, 21–28. https://doi.org/10.1145/3382026.3425771

C5. * Nair, S. S., Becker, M., & Tenev, V. (2020). A comparative study on variability code analysis technology. *Proceedings of the 24th ACM International Systems and Software Product Line Conference - Volume B*, 37–43. https://doi.org/10.1145/3382026.3425775

C6. Zhang, B., & Becker, M. (2013b). RECoVar: A solution framework towards reverse engineering variability. *2013 4th International Workshop on Product Line Approaches in Software Engineering (PLEASE)*, 45–48. https://doi.org/10.1109/PLEASE.2013.6608664

C7. * Zhang, B., Tenev, V. L., & Becker, M. (2018). Analyzing software build architectures. *20. Workshop Software-Reengineering und -Evolution (WSRE)*, 13–14. https://fg-sre.gi.de/fileadmin/FG/SRE/wsre2018/WSRE2018_Proceedings.pdf

C8. Duszynski, S., Knodel, J., & Becker, M. (2011). Analyzing the source code of multiple software variants for reuse potential. *2011 18th Working Conference on Reverse Engineering*, 303–307. https://doi.org/10.1109/WCRE.2011.44

C9. * Duszynski, S. (2015). *PhD theses in experimental software engineering. Vol. 51. Analyzing similarity of cloned software variants using hierarchical set models* (*Vol. 51*). Fraunhofer IRB Verlag. http://publica.fraunhofer.de/dokumente/N-332392.html

C10. Tenev, V., Duszynski, S., & Becker, M. (2017). Variant analysis: Set-based similarity visualization for cloned software systems. *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B*, 22–27. https://doi.org/10.1145/3109729.3109753

C11. * Miodonski, P., Forster, T., Knodel, J., Lindvall, M., & Muthig, D. (2004). *Evaluation of software architectures with eclipse* (IESE-Report 107.04/E). Fraunhofer Institute for Experimental Software Engineering (IESE). Germany. http://publica.fraunhofer.de/dokumente/N-25615.html

C12. * Duszynski, S., Knodel, J., & Lindvall, M. (2009). SAVE: Software architecture visualization and evaluation. *Proceedings of the 2009 13th European Conference on Software Maintenance and Reengineering*, 323–324. https://doi.org/10.1109/CSMR.2009.52

C13. Schmid, K., & Schank, M. (2000). PuLSE-BEAT – A decision support tool for scoping product lines: Proceedings of the international workshop on software architectures for product families (iw-sapf-3). In F. van der Linden (Ed.), *Software architectures for product families* (pp. 65–75). Springer Berlin Heidelberg

As most of the aforementioned analysis tools were developed and improved in the context of collaborations between industrial partners and researchers, there exist multiple publications referring to the same analysis tool. For example, early findings during the development of VITAL have been published in C1, analysis capabilities were demonstrated using mock product lines in C2, and finally the complete tool was presented in a PhD dissertation (C3).

To reduce the size of our tentative start set, we dropped C3, C9 and C11, as they are not peer-reviewed. C12 was dropped as it focuses on analysing architectural compliance. C7 was dropped, as MArZ focuses on optimising make files by means of detecting inefficient patterns. Therefore, it is not directly relevant for our purposes. Furthermore, C4 and C5 were dropped, as they are extensions of VITAL (C1).

In conclusion, 7 publications have been removed from our tentative start set, leaving us with a start set containing 6 publications.

5.3 Snowballing Procedure

The following subsections describe how the snowballing procedure was performed, and what data was extracted. Table 1 summarizes the results of the snowballing procedure for each iteration. Additionally, Table 2 presents an overview of the excluded tools and the corresponding exclusion criteria used to support that decision.

5.3.1 Iteration 1

Starting with our start set, we first identified all references contained within the 6 publications. From the 105 references, we identified three relevant publications during backward snowballing:

C14. Liebig, J., Kästner, C., & Apel, S. (2011). Analyzing the discipline of preprocessor annotations in 30 million lines of c code. *Proceedings of the Tenth International Conference on Aspect-Oriented Software Development*, 191–202. https://doi.org/10.1145/1960275.1960299

C15. Sincero, J., Tartler, R., Lohmann, D., & Schröder-Preikschat, W. (2010). Efficient extraction and analysis of preprocessor-based variability. *Proceedings of the Ninth International Conference on Generative Programming and Component Engineering*, 33–42. https://doi.org/10.1145/1868294.1868300

C16. Zhang, B. (2012). Extraction and improvement of conditionally compiled product line code. *Proceedings of the 20th International Conference on Program Comprehension*, 257–258. https://doi.org/10.1109/ICPC.2012.6240498

From the 141 references identified through Google Scholar, we identified two relevant publications:

C17. Ziadi, T., Henard, C., Papadakis, M., Ziane, M., & Le Traon, Y. (2014). Towards a language-independent approach for reverse-engineering of software product lines. *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 1064–1071. https://doi.org/10.1145/2554850.2554874

C18. Duszynski, S., & Becker, M. (2012). Recovering variability information from the source code of similar software products. *Proceedings of the Third International Workshop on Product LinE Approaches in Software Engineering*, 37–40

Thus, in the first iteration, we identified three possibly interesting tools: (i) cpp-stats, a C-preprocessor directive analyser (C14); (ii) undertaker, a static code analyser for preprocessor directives (C15); and (iii) ExtractorPL, a language-independent tool for reverse-engineering software product lines (C17).

Interestingly enough, we did not identify both Zhang (2012) and Duszynski and Becker (2012) for inclusion into our tentative start set. Fortunately, this does not pose a problem, as it makes no difference whether the aforementioned publications are added into the initial start set or during subsequent iterations.

5.3.2 Iteration 2

Starting with our new set, we repeated the snowballing procedure. Backward snowballing identified 114 publications, none of which were relevant. During forward snowballing we identified two relevant publications from a total of 249 publications:

C19. Kuiter, E., Krieter, S., Krüger, J., Ludwig, K., Leich, T., & Saake, G. (2018). Pclocator: A tool suite to automatically identify configurations for code locations. *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1*, 284–288. https://doi.org/10.1145/3233027.3236399

C20. Kenner, A., Kästner, C., Haase, S., & Leich, T. (2010). Typechef: Toward type checking #ifdef variability in c. *Proceedings of the 2nd International Workshop on Feature-Oriented Software Development*, 25–32. https://doi.org/10.1145/1868688.1868693

In conclusion, the second iteration identified two tools which warrant further evaluation: (i) PCLocator, a tool for identifying presence conditions using C-preprocessor directives (C19); and (ii) TypeChef, a C-preprocessor variability analyser (C20).

5.3.3 Iteration 3

We did not identify further relevant publications during both backward (55 publications) and forward snowballing (87 publications). This marks the end of our snowballing procedure.

5.4 Classification

Having identified potentially relevant analysis tools, we can now address RQMAP1, namely whether there are analysis tools which provide capabilities similar to ours. To this end, we must first introduce our tools.

VITAL (Variability Improvement Analysis) possesses two major capabilities. First, it mines complex feature correlations from C/C++ source-code containing CPP directives, and, secondly, it extracts a variability reflection model which can be used to detect eroded code and compute other metrics (Zhang, 2015). Zhang (2015) also describes an improvement process which uses the aforementioned information to improve existing software product lines. Furthermore, a source-code editor which can color lines depending on which feature is selected is presented in Zhang and Becker (2013b). Note that RECoVar (Reverse Engineering Configurations and Variability), introduced in Zhang and Becker (2013b), was absorbed into VITAL. Thus, RECoVar should not be considered a standalone tool. In prior work, Morais Ferreira et al. (2020) extended the aforementioned improvement process and introduced a

Iteration	Publication	Backward		Forward	
		Total	Relevant	Total	Relevant
First	C1	24	C14,C15,C16	17	0
	C2	22	0	4	0
	C6	21	0	13	C17
	C8	13	0	74	C18
	C10	14	0	7	0
	C13	11	0	26	0
Second	C14	39	0	137	C19
	C15	29	0	55	C20
	C16	7	0	4	0
	C17	26	0	48	0
	C18	13	0	5	0
Third	C19	17	0	3	0
	C20	38	0	84	0
$\sum$		274	3	477	4

Table 1: Results of snowballing procedure

Microsoft Power BI[15]-based knowledge base for sharing information between product line practitioners and software engineers. While we did not identify a drop-in replacement for VITAL and RECoVar, we did identify some tools whose capabilities overlap with ours. For analysing CPP annotated C/C++ source-code, cppstats provide similar analyses (Liebig et al., 2011). Furthermore, both tools rely on srcML (Collard et al., 2011) annotated source-code. PCLocator identifies presence conditions on a line-by-line basis (Kuiter et al., 2018). Given a feature model, PCLocator can derive all valid configurations. VITAL also identifies presence conditions, albeit in a rudimentary form. TypeChef computes an abstract syntax tree by analysing C/C++ source-code containing CPP macros (Kenner et al., 2010). From the different analyses provided by TypeChef, reasoning about feature expressions is one capability which VITAL also possesses to a certain degree. However, this capability could also be improved on.

Variant Analysis (VA) is an analysis framework based on Eclipse which efficiently computes commonality and variability from existing software variants (Duszynski, 2015). VA supports a multitude of different similarity detection algorithms, supports querying of results and provides insightful visualisations. During our systematic identification of relevant tools, we did not identify a drop-in replacement for VA. However, we identified a number of smaller tools which implement some aspects of VA, namely CloneDetective (Jürgens et al., 2009) and ConQAT (Deissenboeck et

[15] https://powerbi.microsoft.com/

Tool	Publication	Exclusion Criteria					
		EC9	EC10	EC11	EC12	EC13	EC14
ASTEC	McCloskey and Brewer (2005)					•	
But4Reuse	Martinez et al. (2015)		•				
C-CLR	Singh et al. (2007)	•					
CDL Tools	She et al. (2011)				•		
CIDE	Kästner and Apel (2008)					•	
Clone Miner	Basit and Jarzabek (2009)						•
CloneDetective	Jürgens et al. (2009)	•					
Colligens	Medeiros et al. (2018)					•	
CRefactory	Garrido and Johnson (2003)						•
CScout	Spinellis (2003)					•	
DMS/SRT	Aversano et al. (2002)						•
FAMILIAR	Acher et al. (2013)			•			
FARCE	Arcaini et al. (2017)						•
FeatureC	Krüger et al. (2018)					•	
FeatureCommander	Feigenspan et al. (2012)	•				•	
FeatureCoPP	Ludwig et al. (2020)					•	
FeatureHouse	Apel et al. (2009)					•	
FEVER	Dintzner et al. (2016)						•
FLIPPER	Ruprecht et al. (2014)				•	•	
FMDiff	Dintzner et al. (2014)			•	•		
GOLEM	Nadi et al. (2014)				•		
KernelHaven	Kröher et al. (2018a)		•				
MetricHaven	El-Sharkawy et al. (2019)					•	
Morpheus	Liebig et al. (2015)					•	
PRET	Kanda et al. (2013)	•					
R3V3RS3	Adams et al. (2009)						•
srcML	Collard et al. (2011)					•	
SKUNK	Fenske et al. (2015)					•	
SPLCATool	Johansen et al. (2012)			•		•	
SuperC	Gazzillo and Grimm (2012)					•	
ToolDAy	Lisboa et al. (2011)						•
VAMPYR	Tartler et al. (2014)					•	
VariCell	Acher et al. (2012)			•			
VAT	Schlie et al. (2019)			•			
VORM	Grebhahn et al. (2021)			•			
VSat	Young et al. (2020)					•	
Xrefactory	Vittek (2003)					•	

Table 2: Overview of excluded tools

al., 2008), which can detect software clones. Furthermore, CloneDetective is based on ConQAT and relies on calculating edit distances from suffix trees. CCFinder (Kamiya et al., 2002) detects clones by decomposing source-code into tokens and computing a token-by-token similarity measure. PRET computes the evolution history for a given set of variants (Kanda et al., 2013). It relies on a modified similarity metric based on the longest common subsequence (LCS). Other approaches, such as Debbiche et al. (2019), rely on tools used to resolve merge conflicts in version control systems for computing similarities between files. Furthermore, both Undertaker (Tartler et al., 2011) and VA can detect blocks of source-code which cannot be enabled or disabled by comparing the CPP directives structure to configuration models.

It should be noted that while we excluded But4Reuse (Martinez et al., 2015) based on EC10, it provides similar capabilities when compared to VA, namely when computing commonality and variability for given variants.

Finally, PuLSE-BEAT introduces a decision support tool used to efficiently scope a product line project (Schmid & Schank, 2000). PuLSE-BEAT allows for simultaneously scoping different aspects, such as scoping during domain analysis and scoping reusable assets from an existing product line. While we did identify ToolDAy (Lisboa et al., 2011), a tool for performing domain analysis, we ultimately excluded it, as it was not available online[16] (EC14).

To identify which transitions across the SPL engineering process model quadrants are possible with the identified tools (RQMAP2), we decided to visualise the possible transitions along the SPL engineering process model, as illustrated in Figure 12. Figure 12 documents the transitions across the quadrants of the SPL engineering process model that are possible with aforementioned tools. Note that analysis tools that only improve assets in a single phase, i. e. do not enable the transition to another phase, were not represented in our illustration. Chapter 6 presents Figure 12 in greater detail.

5.5 Conclusion

As stated by van der Linden et al. (2007), a large number of product line tools have been developed from a research context, where product quality requirements such as usability and portability are typically not the main drivers. During our mapping study, we identified a number of prototypical tools, many of which were not available online. However, those that are available online typically share the source-code and corresponding executable binaries.

With regards to validity threats, we believe that while we followed most recommendations for a good start set as described by Wohlin (2014), only one person performed the snowballing process. The same person then extracted the relevant information and performed the classification. Ideally, an independent reviewer should verify the mapping study findings, thereby reducing potential validity concerns.

[16]The GitHub repository is empty, https://github.com/RiSE/ToolDAy

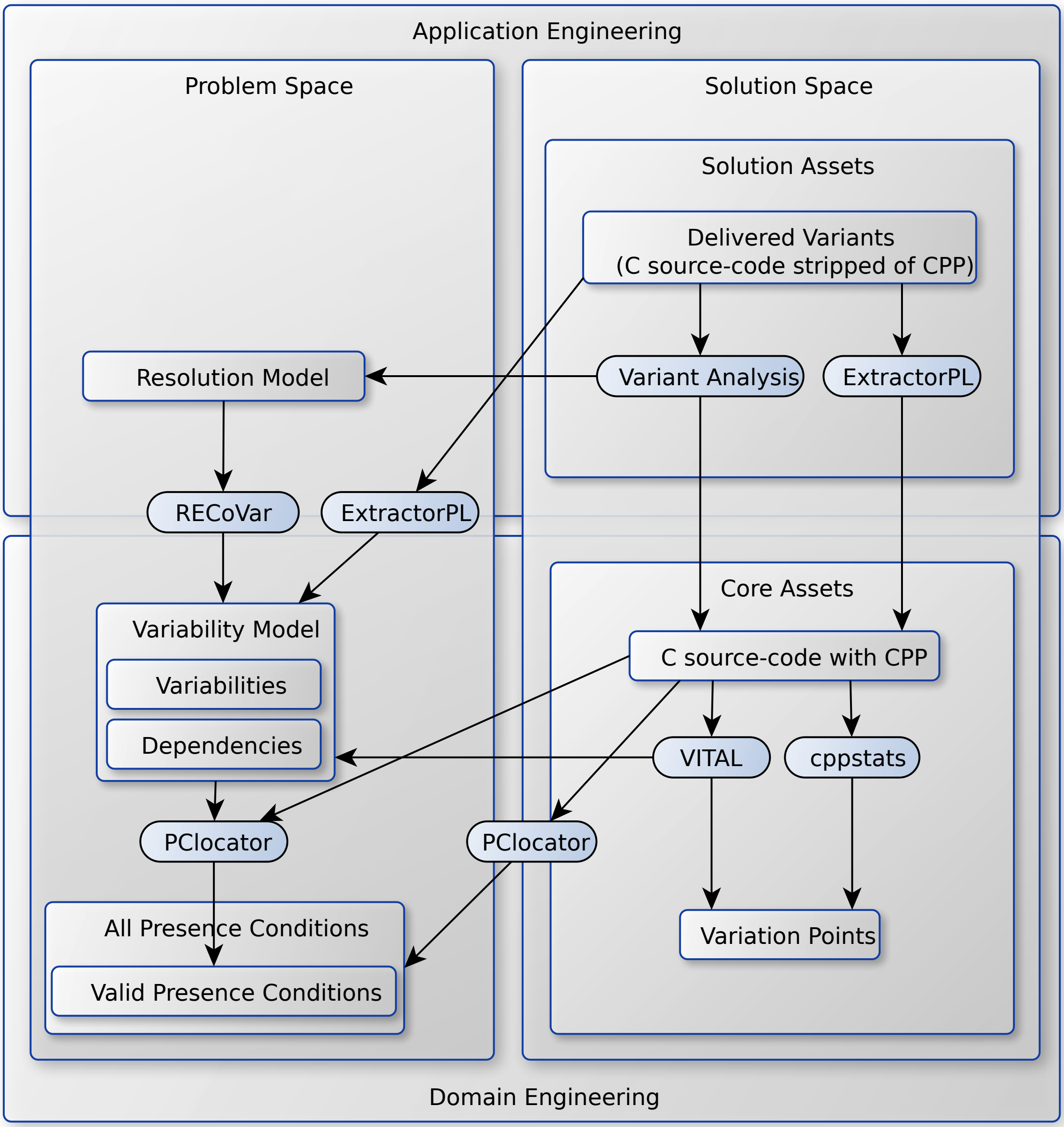


Figure 12: Overview of feasible transitions in the SPL engineering process model

To remedy this potential bias, the complete list of publications and applied inclusion/exclusion criteria is provided in the thesis supplements. In future work, the aforementioned data should provide an adequate starting point.

6 Analysis Cookbook

This chapter introduces a classification system for describing existing analyses and possible combinations of individual analysis tools.Please note that this Chapter has been accepted for publication at the 2021 9th International Workshop on Reverse Variability Engineering, which will be held in conjunction with the SPLC 2021 25th International Software Product Line Conference (Morais Ferreira et al., 2021).

6.1 Motivation

Typically, analysis tools are developed in the context of research studies and industrial collaborations, and are used to answer specific questions that arise throughout this endeavour. As the information need increases, these tools are expanded to encompass more features, and thus answer more questions. As the research progresses, the analysis practitioner gains an understanding of the tools capabilities, from the technical knowledge regarding data formats and adaptions that are required to the overarching knowledge of which information needs the tool fulfills, which insights can be gained from it, and what its limitations are.

Therefore, each analysis tool is inherently related to the context in which it is created and used, i. e. which information it processes, which information it produces, which file-types are supported, what interoperability capability it provides, and naturally, which underlying question it answers. Unfortunately, this knowledge, which constitutes a holistic approach describing the applicability of the tool, is rarely documented, thus making it hard to reuse the tool in another context.

In addition to the aforementioned issue, research studies and industrial collaborations aim to fulfill one or more goals, which vary from the very abstract goals, such as "improve time to market" to the very specific "improve variability" (cf. Chapter 4). As described in Section 4.3, goals require a faceted perspective, as there may be complex goals for which no single analysis is sufficient for its fulfillment, but where a chaining of individual analyses provides the desired result. Additionally, different analysis tools may solve an equivalent analysis need, either by analysing different information, or by using different algorithms.

Thus, we propose a framework for systematically describing analysis tools and their applicability in the context of SPL reverse-engineering activities. We aim to lay the foundations for a collection of systematic analysis instructions, henceforth referred to as analysis cookbook, which guide the practitioner in solving analysis needs potentially spanning across multiple quadrants of the SPL engineering process model (cf. Figure 1). The creation of such a collection benefits both academic and industrial SPL practitioners, as it provides a searchable documentation of analysis capabilities, their underlying data requirements and resulting information. Furthermore, these analysis instructions contribute to the development of our integrated analysis framework. The resulting framework should allow for the integration and execution of these analysis instructions.

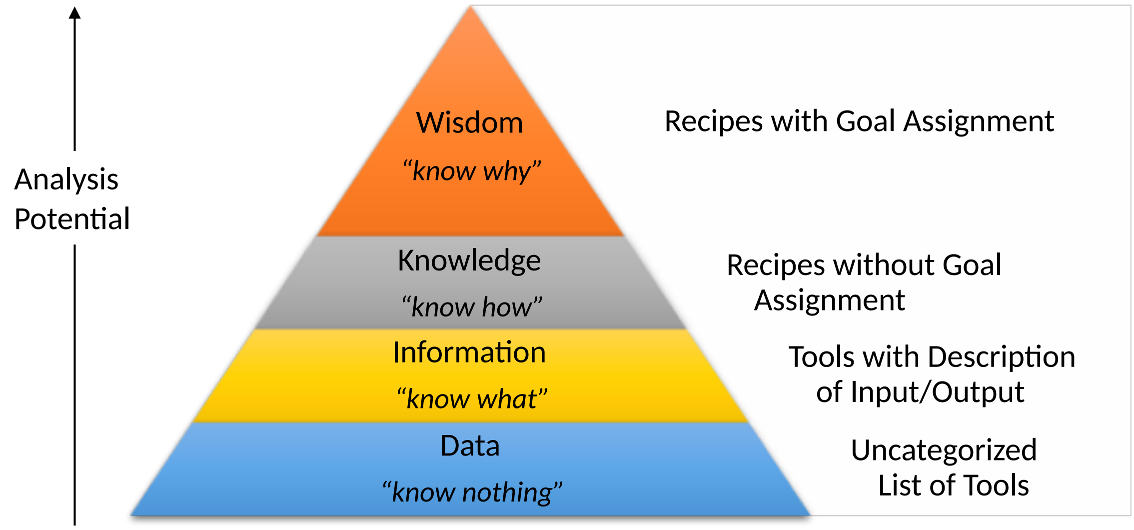


Figure 13: DIKW hierarchy (Zeleny (1987) and Rowley (2007))

Figure 13 provides an hierarchical overview of the aforementioned concepts along the DIKW[17] hierarchy, as described by Zeleny (1987) and Rowley (2007). Situated at the bottom of the DIKW hierarchy, data represents an unprocessed list of analysis tools, as identified during the snowballing procedure presented in Chapter 5. One level higher, information, which is inferred from data, describes the data requirements of the analysis tools, and which transitions along the SPL process model they enable. Knowledge, which builds on information and data, describes the applicability of each analysis tool to different scenarios, as well as which analysis toolchains could be described to solve complex problems. However, at this level, it is unclear which goal, i. e. underlying need, is being fulfilled. Lastly, wisdom describes a deep understanding of industrial needs and goals, and how each analysis, complex or simple, contributes to their fulfillment. Thus, while at the lower-levels we speak of individual analysis tools, higher-levels describe combinations of multiple analysis tools capable of solving complex analysis needs, ultimately leading to the zenith, i. e. insight into which goals are fulfilled by these combinations.

6.2 Analogy

This collection, which we will henceforth refer to as "cookbook", contains a number of different analysis recipes, each contributing to one or more goals. Drawing on an analogy, the analysis recipes can be compared to cake recipes, which, while potentially overlapping to a certain degree, each produce a different cake.

To successfully bake the desired cake, one first needs to obtain all the required ingredients and equipment. Then the provided instructions are followed step by step to first produce intermediate results, which are then combined to form the final cake

[17]Data, Information, Knowledge and Wisdom

batter. For example, after separating the eggs yolks from the egg whites, the egg whites are beaten to stiff peaks, and the egg yolks are combined with sugar and beat until thickened. Then, after mixing the yolk-sugar mixture with the flour, baking powder and salt, the stiff egg whites are folded in to form the cake batter. This last step combines two intermediate products with additional ingredients to form the final product.

To tie the analogy back to our context, assume that the baker is the analysis practitioner. The company context describes the boundaries in which the practitioner must operate. After identifying the underlying needs and defining appropriate goals, the practitioner chooses the adequate analysis recipe (cake recipe), procures the required data (cake ingredients) and tools (kitchen utensils). Then, the practitioner follows the analysis instructions (cake recipe) as described in the cookbook. Depending on the situation, these instructions consist of executing a fully automated pipeline, which only requires effort to adapt the input data. Alternatively, these instructions can describe a fully manual, or a semi-automated procedure. For example, the latter case may require expert knowledge to interpret findings provided by an analysis tool, which then serves as the input for subsequent pipeline steps.

6.3 Cookbook Overview

In the context of forward-engineering software product lines, Horcas et al. (2019) present an overview and characterization of tools which enable the development of a software product line. In addition to characterizing each tool with regards to its availability and applicability, roadmaps, which enable a complete SPL process through combinations of different tools, are presented. Although, the first iteration of our analysis cookbook only contains analysis recipes which combine reverse-engineering analysis tools, nothing impedes the inclusion of recipes consisting of both forward and reverse-engineering analyses, which may both fulfill analysis goals and drive the transition to a SPL. Thus, one could represent SPL roadmaps, as described by Horcas et al. (2019), as cookbook recipes.

Figure 12 presents possible transitions across the quadrants of the SPL engineering process enabled by different reverse-engineering analysis tools. We decided to base our illustration on the SPL engineering process, as it places the focus on the transitions along the different quadrants. In contrast to the representation chosen by Horcas et al. (2019), we are easily able to describe transitions representing both forward and reverse-engineering activities. Furthermore, it is generic enough to visually represent a large number of different analyses, independently of their complexity. Grüner et al. (2019) illustrate the interaction sequences of their toolchain for feature extraction. While this provides a clear overview of their intended purpose, it cannot be reused at a higher abstraction levels for purposes other than feature extraction.

While square nodes represent data artifacts, rounded nodes represent analysis tools. Furthermore, edges depict input and output information flows for each approach. A recipe, i. e. an useful combination of two or more approach tools, can be represented as a path across this graph-like representation of tools and artifacts, which starts and ends in different artifact nodes. Note that while there exists a large number of different reverse-engineering approaches in the context of product lines, Figure 12 only contains tools which we identified in Chapter 5.

For an analysis cookbook to be useful, it is of the utmost importance that the information is systematically documented. First and foremost, the analysis practitioner must be able to find the needed analysis recipe with reasonable effort. Using the mono-hierarchical classification of goals introduced in Chapter 4, recipes are classified according to which goal they help fulfill.

As the interpretation of goals and needs by the practitioner depends on the specific application scenario and prioritization of higher-level goals (cf. Section 4.3), we envision that an individual recipe may fulfill multiple goals. The practitioner must also know which analysis tools to procure, what data these tools read as input, and what data they produce. Finally, the recipe must contain detailed instructions required to reproduce the analysis from start to finish.

6.4 Recipe Template

For an analysis cookbook to be useful, it is of the utmost importance that the information is systematically documented. Table 3 introduces our recipe template. First and foremost, the analysis practitioner must be able to find the needed analysis recipe with reasonable effort. Then, recipes are classified according to which goal they help fulfill. As the interpretation of goals and needs by the practitioner depends on the specific application scenario and prioritization of higher-level goals, we envision that an individual recipe may fulfill multiple goals. The practitioner must also know which analysis tools to procure, what data these tools read as input, and what data they produce. Finally, the recipe must contain detailed instructions required to reproduce the analysis from start to finish.

Recipe Name	
Contributes to Goal(s)	Provide list of goals for which the recipe contributes to. Enables practitioners to find recipes, depending on which goal they are following.
Short Description	Provide context information for the scenario usage and details on the contribution to the aforementioned goals.
Required Data	Which assets are required to perform the analysis.
Required Tools	Provides insight into which tools must be procured.
Instructions	Step-by-step description of how to perform the analysis. The individual analysis approaches are considered an atomic black-box.

Table 3: Recipe template

Recipe Name	
Contributes to Goal(s)	Reduce Costs→Optimize Employee Involvement→Avoid Repetitive Configuration Work→Improve Configuration Process→**Reduce Possible Configuration Space**
Short Description	By constraining the feature model, we reduce the total amount of possible configurations to those which are plausible.
Required Data	Resolution Model
Required Tools	RECoVar
Instructions	(i) Given a resolution model, perform a feature-correlation mining analysis using RECoVar (arrow 1 in Figure 14a). (ii) Then, a domain expert must manually evaluate the findings and decide whether the identified relationships are relevant or not (Zhang & Becker, 2013b). (iii) Once the feature correlations have been identified and assessed, configuration processes can be improved upon with respect to efficiency and correctness (Zhang & Becker, 2013a). These correlations also describe the smallest possible feature model, as it only contains the correlations which are present in the delivered variants.

Table 4: Recipe for constraining a feature model / configuration space

6.5 Recipe Examples

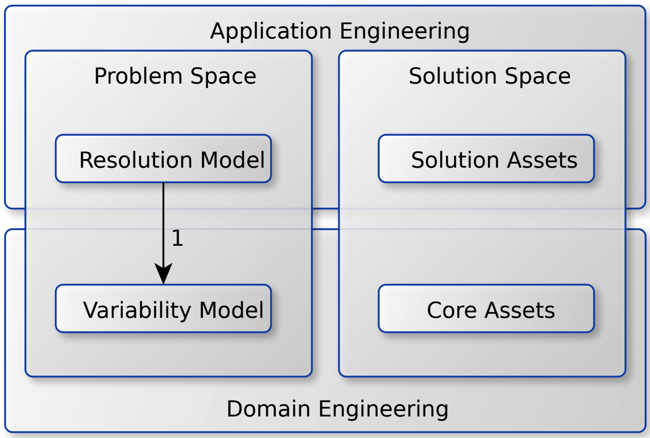
Recipe 1, as outlined in Table 4, introduces a "simple" recipe for reducing the possible configuration space by analyzing resolution models and extracting interdependencies between features so as to improve the feature model (cf. Section 4.3). Furthermore, Recipe 2 describes the case study introduced by Grüner et al. (2019) to motivate the need for a holistic toolchain which allows a transition from clone-and-own product development to a systematic PLE approach (Table 5). This non-trivial case study describes the need for feature extraction, namely feature identification, feature localization and feature tracing (Burger & Grüner, 2018). Note that the visualisation use-case described by Grüner et al. (2019) is not captured within the boundaries of our cookbook overview.

6.6 Conclusion

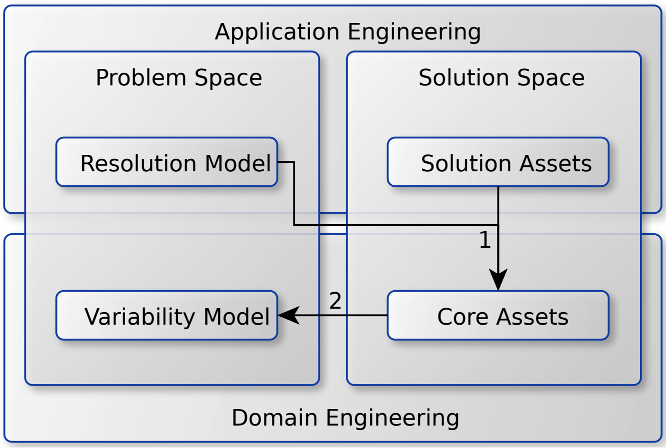
In this section, we introduced a classification system for documenting existing analysis approaches in the form of a cookbook. This cookbook reveals the possible combinations of analysis approaches, and addresses complex analysis needs in industrial settings. The individual cookbook recipes thus provide answers to these industrial

Recipe Name	
Contributes to Goal(s)	Reduce Costs, Time To Market, Improve Quality
Short Description	Variants are developed by means of ad-hoc clone-and-own, i. e. the most basic mechanism for variability realization. Thus, features are not identified and are not traced to their implementation. This typical scenario, as described by Grüner et al. (2019), requires significant reengineering effort to successfully perform a migration to systematic product line engineering approach.
Required Data	Unannotated source-code files of all relevant variants, product documentation (Scenario 1 from Grüner et al. (2019))
Required Tools	FINALIST^2, But4Reuse, KernelHaven, pure::variants, and other unnamed tools.
Instructions	(i) Beginning with unannotated source-code and build traces, FINALIST^2 is able to detect and embed feature annotations within the source-code assets. (ii) Then, after attempting to build all the possible variants out of the possible configuration space, But4Reuse is used to detect commonality and variability from the product variants. Once the new insight has been embedded into the relevant assets, the first transition in the SPL process model has been performed (Arrow 1 in Figure 14b). (iii) To derive a variability model (Arrow 2), KernelHaven, pure::variants, and other unnamed tools are used to identify presence conditions and dependencies for features, after which a feature model can be created

Table 5: Recipe for performing a feature extraction from products derived via clone-and-own



(a) Describes recovery of dependencies between features from existing configurations



(b) Describes the transition from clone-and-own to a PLE approach (Burger & Grüner, 2018; Grüner et al., 2019)

Figure 14: Overview of analysis recipes

use-cases in a structured manner. Furthermore, we provided first exemplary recipes which illustrate the method for two settings of different complexity.

Given that an empirical study regarding the ease of use and practicality is not envisaged within the scope of this thesis, it would be of interest to conduct one in future work.

Furthermore, crowd sourcing additional recipes from PLE practitioners and sharing them openly with the wider community will result in (i) a freely accessible analysis cookbook which documents typical needs for analyses in an industrial context; and (ii) corresponding multi-analysis approaches which address these needs. This open platform will promote the usage of analysis approaches and encourage collaboration between researchers.

7 Reference Architecture

This Chapter presents our reference architecture of our integrated analysis framework.

7.1 Overview

Figure 15 describes our reference architecture, as modelled on the logical level in ARCADIA. The proposed reference architecture has only been specified insofar as to allow for the description of our main use-cases, which are represented as logical functions assigned to a logical actor, namely an analyst. Thus, the reference architecture only captures the essential functions which our integrated framework should implement, providing guidance towards an integrated approach, and relies on previous experience and industrial needs. Additionally, these functions are assigned to components and exchange information via functional exchanges.

We differentiate between actor-allocated functions and system-allocated functions. In the following Sections, we will provide a brief description of our reference architecture. Figure 6 provides a simplified overview of our reference architecture which does not include the logical actor and logical components. In this figure, actor-allocated functions are colored in blue, and system-allocated functions are colored in green.

7.1.1 Actor-Allocated Functions

Actor-allocated functions describe *what* functions an analyst must be able to perform. As such, the analyst must be able to prepare assets, which are provided in a multitude of different file formats, for analysis. Then, an analysis recipe containing one or more analyses must be chosen. Once the analysis results are available, the analyst must then be able to browse and export findings, including its underlying analysis data, for further processing. These four main activities are modelled as logical functions.

7.1.2 System-Allocated Functions

On the other hand, the framework must be capable of realizing the functionality required to fulfill the analysis needs.

Thus, an analyst relies on the *Workflow Editor* component to create and configure analysis projects for individual projects. Then, assets are imported by a *Raw Data Importer*, which reads assets in a variety of different file formats. Given that an integrated framework must be able to process a wide variety of asset types, the *Model Builder* transforms these assets into a representation which allows the integrated analysis tools to process them. Furthermore, the *Filter* component enables an analyst to remove superfluous information, thus only keeping a relevant subset of assets. Thus, both transformation and filtering components provide normalization and generalization capabilities to our framework. Having selected the desired analysis recipe (cf. Chapter 6), which can consist of one or combinations of analysis tools, the *Analyser* executes the analysis as specified in the cookbook recipe.

Finally, the *Exporter* and *Results Visualisation* components provide access to both the raw analysis results, and visual representations.

7.2 Architectural Decision

As described in Figure 6, knowledge gained from existing architectures and customer needs, as identified in Chapter 4, provide a valuable contribution to the definition of reference architectures. For example, re-engineering existing analysis tools to support new data formats is not desirable, as that effort could be invested otherwise.

For modeling our reference architecture, as illustrated in Figure 15, we adopted a Service-oriented approach (SOA). Thus, following the key principles of SOA, as described by Erl (2005), each service, which maps to a logical component, describes an independent unit which encapsulates the required logic, promoting loose coupling, autonomy and abstraction. Furthermore, the workflow editor enables the discovery and reuse of existing components, and allows for composing and configuring each complete analysis process, i. e. from importing assets to exporting and visualising analysis results, to individual analysis needs. Furthermore, the data storage component and its underlying logical data model, as illustrated in Figure 17, ensures components adhere to a standardized schema for communicating with other services through a central data repository (service contract). The introduction of a middleware component decouples the application logic from the underlying data storage, thus enabling a standardized interface to the data storage layer.

For interacting with the framework, we propose a flow-based approach which enables an analyst to plan and execute an analysis from end-to-end. In this workflow editor, an analyst could select nodes, which describe system-allocated functions, and connect them together, providing the second node with input data from the output of the first node. Thus, an end-to-end execution of our toolchain would be modelled as a flow, beginning with import nodes and ending with visualisation and/or extraction nodes. While these nodes may describe fully-automated activities, activities requiring manual effort should also be supported. The workflow editor would then also provide an overview as to the current status of the overall activity.

The proposed approach thus enables an analyst to tailor a complete analysis from end-to-end with great flexibility, which is not possible with a pipeline-based approach, as each component of the pipeline is logically coupled to its predecessor and successor. Furthermore, existing analysis tools do not need to be modified, as our data model, in combination with adapters, provides the required data in a supported format.

7.3 Logical Data Model

Figure 17 describes a logical data model for our integrated framework. By adopting a model-based approach for describing the required information, we distinguish between two perspectives, represented as distinct data packages. On the left-hand side, the Metadata package describes the relation between analysis projects, and workflows, i. e. recipes. As previously described, the proposed flow-based approach uses Workflows, which contain Jobs, consisting of Functions (nodes) and Exchanges (edges). We explicitly differentiate between Workflows and Jobs, as a Workflow may contain multiple, parallelizable, Jobs.

On the right-hand side, the Primary Data package describes how data, which occurs throughout the analysis process, is stored. As we propose a flow-based approach, which consists of chaining individual components in a manner akin that of a pipeline, intermediate and final results must be stored. We refer to this information as Assets. Additionally, to enable traceability between assets, such as which module created it, and which input assets were processed, we use Relations, i. e. Connections. The combination of both Assets and Connections enables us to store all the required information in a graph-like representation. Note that additional Information pertaining both Assets and Connections are stored as Attributes.

Thus, our logical data model, which describes a model-based approach for uniformly representing data, enables the derivation of physical data models tailored to individual needs.

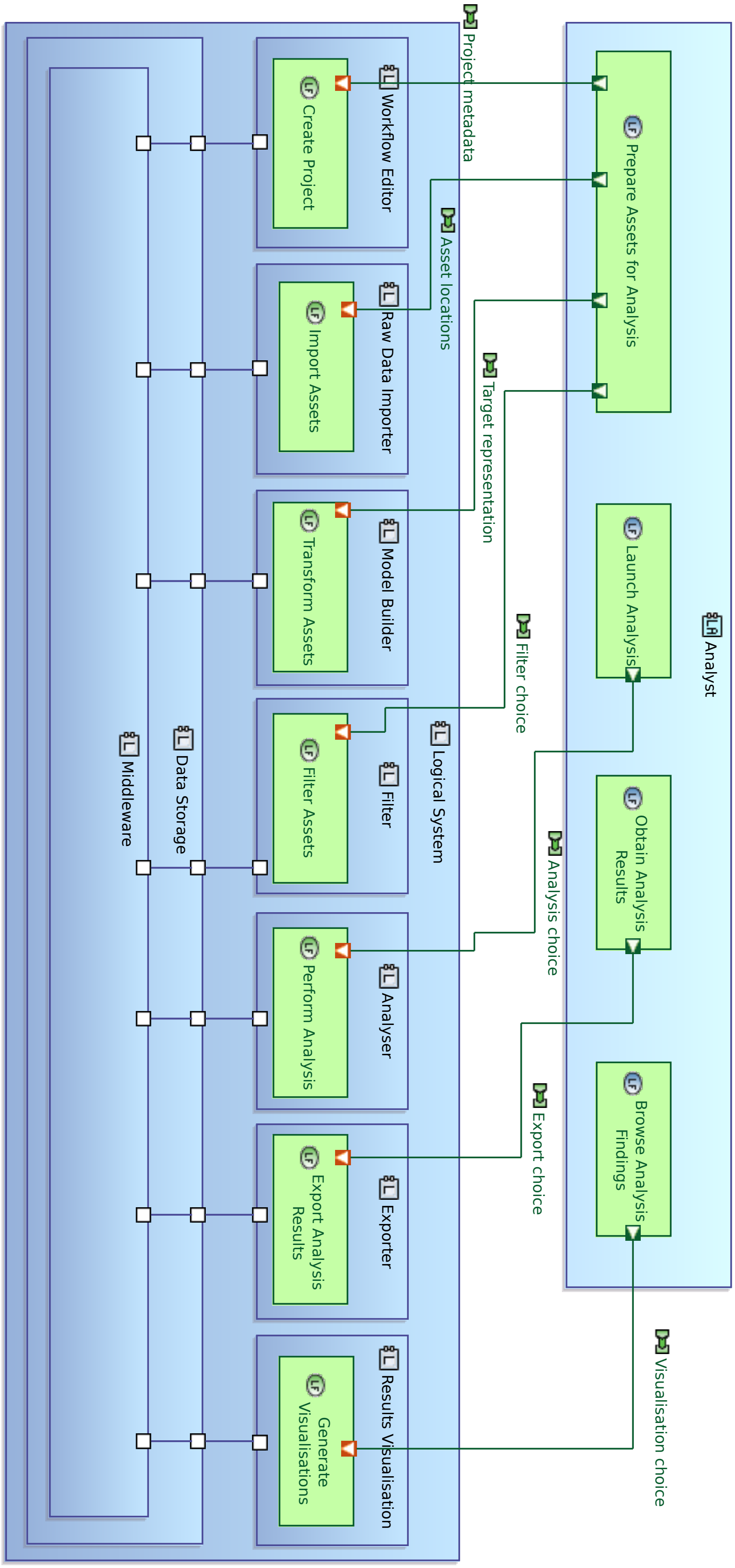


Figure 15: Reference architecture with logical components

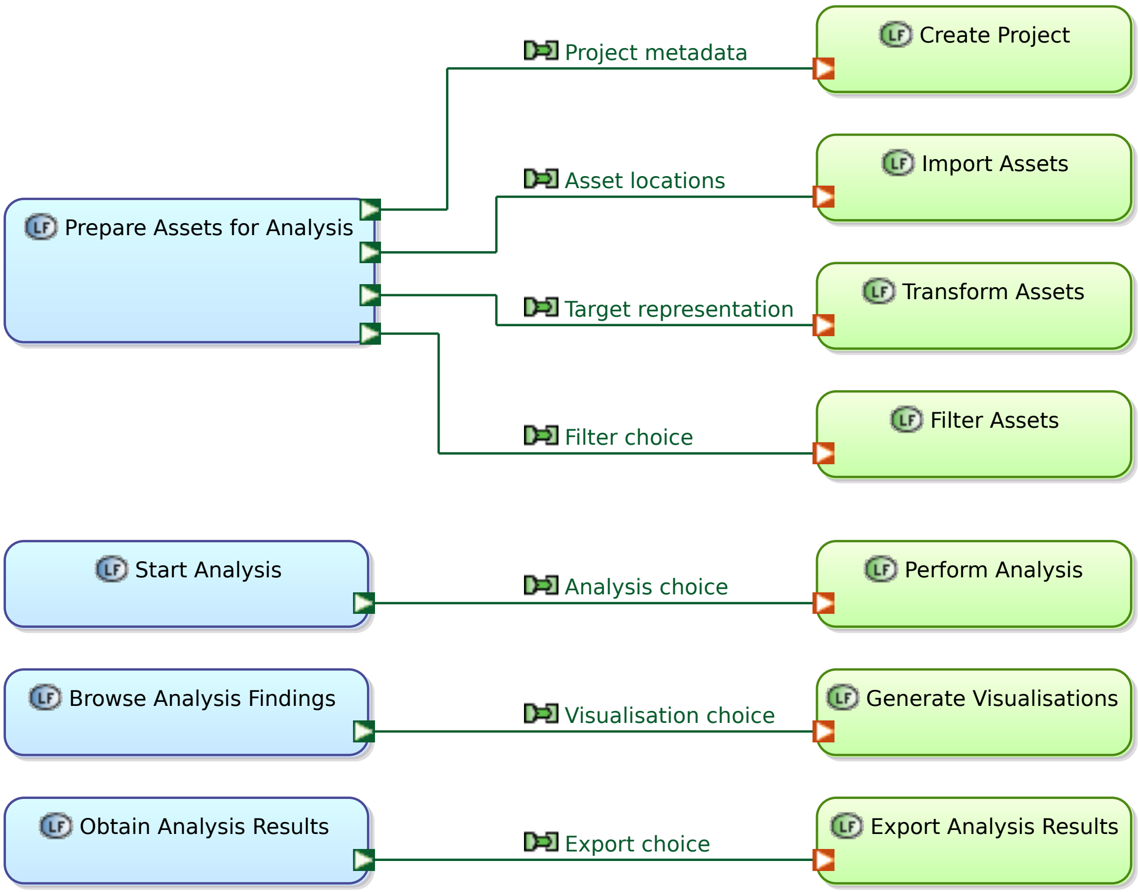


Figure 16: Overview of Logical Functions

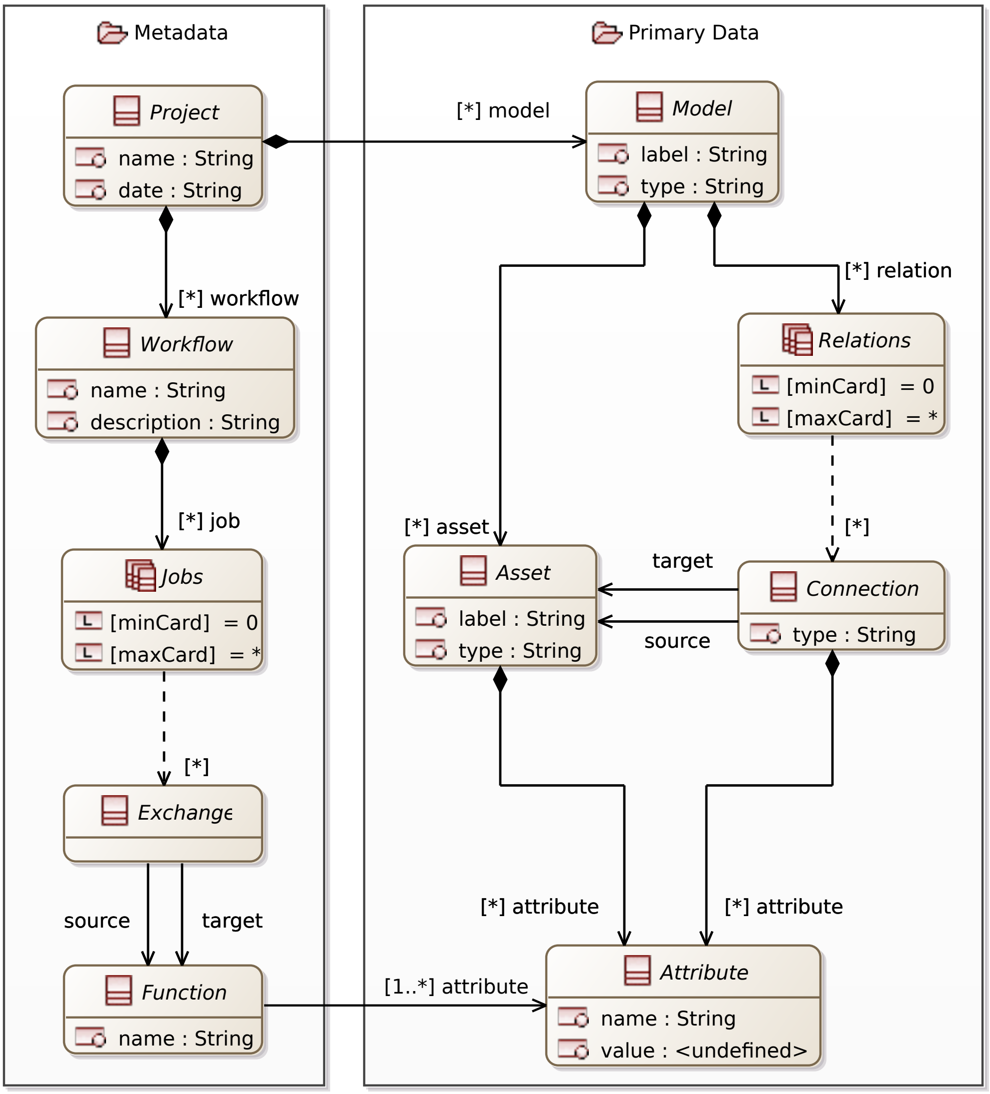


Figure 17: Logical data model

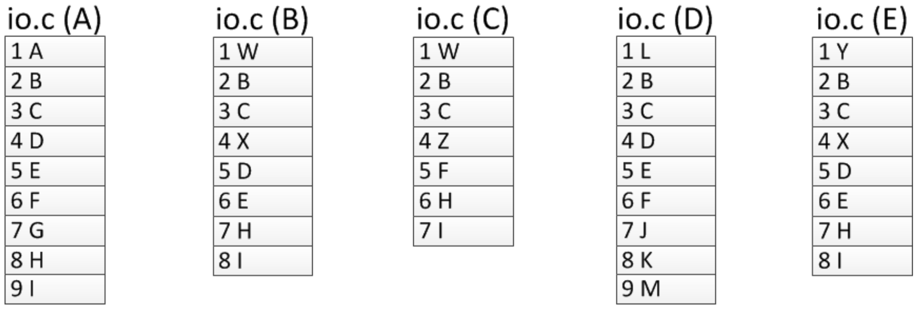


Figure 18: Five example file variants (Duszynski et al., 2020)).

8 Architecture Validation

8.1 Prototype Scenario

To validate our reference architecture of an integrated framework, we present a prototypical implementation of a complex cookbook recipe that aims to transition an existing clone-and-own approach to a systematic platform approach, as demonstrated in Figure 14b. However, our cookbook recipe, as described in Table 5, differs from the toolchain introduced by Grüner et al. (2019), as we use different analysis tools.

In this scenario, an organization derives product variants via a copy-and-paste mechanism, i. e. by cloning an existing product codebase, and modifying it to fulfill new requirements. As the changes, such as bug-fixes, new features, and improvements, are not propagated between products, one refers to this approach as ad-hoc, i. e. not systematically managed. This results in significant effort, worsening not only with each introduction of a new product, but also with every bug-fix and change to the code-base which require propagation to other related products. Figure 18 illustrates a file which exists in 5 variants, labelled A to E. To simplify our scenario, each line contains a single letter.

8.2 Scope

As we only aim to validate the aforementioned reference architecture and logical data model, we intentionally defined a reduced scope of our prototypical implementation. Therefore, as illustrated in Figure 19, we only implemented select components of our reference architecture. The scope of the prototype is thus represented as a functional chain, i. e. as a blue path spanning across the functional exchanges. Furthermore, Figure 20 presents an illustrative example for a workflow. Note that we did not implement a prototypical workflow editor, as it would not directly contribute to validating the reference architecture and logical data model. However, one possible solution approach would be to implement individual components in Node-RED[18], a

[18] https://nodered.org/

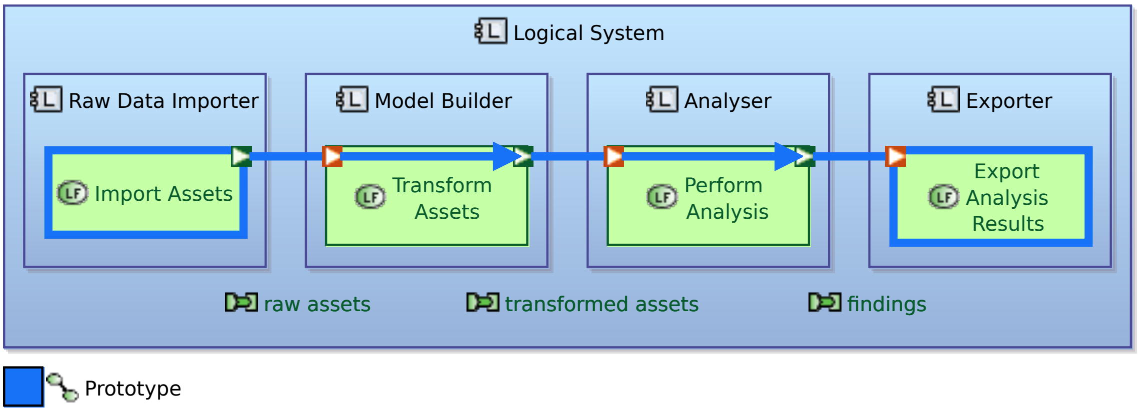


Figure 19: Scope of framework prototype

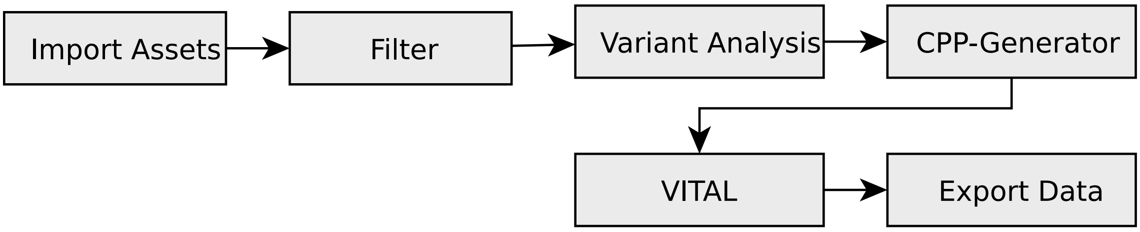


Figure 20: Flow-Based representation of a toolchain for performing a complete analysis workflow

widely used flow-based editor which is very easily extensible. Thus, the individual components could be implemented as Node-RED nodes, after which building recipes would be as trivial as selecting the relevant subset of nodes, and connecting them together. With regards to the data storage, and its underlying data model, we opted not to implement a middleware, as the benefits gained don't contribute to validating the relevant aspects of the proposed reference architecture.

8.3 Implementation

Figure 20 illustrates the workflow and functions required for performing the analysis. In our prototypical implementation, each component has either been adapted from previous integration efforts, as is the case for importing and filtering assets developed from the ground up. Thus, importing and filtering file-based assets, as described in Figure 18, already exist as executable JAR files. Variant Analysis and VITAL are also existing analysis tools, and thus require Adapters to both read data and to store results into our data model. From the results of VA, CPP-Generator generates C-based source-code assets which rely on the CPP for realizing variability, thus enabling the migration to a platform-based software product line. However, the macro definitions are still unaware of features, only relying on variant names instead. Thus, the transition to macro definitions which contain features is a manual task in the context of our prototype. Once this transition has been performed, the new assets are imported again, which VITAL then uses to reverse-engineer a feature model. Finally, the findings are extracted from the database.

For storing information, we relied on ArangoDB[19], a multi-model database system. ArangoDB natively supports key/value stores, document stores and graph data, and allows complex querying which leverages the benefits of the aforementioned data storage paradigms.

8.4 Validation

By developing a prototypical toolchain to validate key aspects of our reference architecture, we found that one of the main properties of SOA, namely separation of concerns through independent units which encapsulate the business logic, greatly reduced the effort required to both implement new components, and integrate existing analysis tools. Furthermore, centrally managing the underlying data model enables the integration required of our framework, as simply storing analysis results without any transformation directly goes against the principle of what we aim to accomplish. Thus, a model-based representation of analysis artifacts which relies on a graph-like representation allows for efficient querying of data enables an improved understanding of the underlying information contained within these analysis artifacts. This efficient querying allows an analyst to easily explore the analysis artifacts, perhaps resulting in the development of analyses of greater complexity.

Furthermore, previous experience has shown that the physical data model diverges from the logical model as the framework evolves with regards to the number of integrated components, i. e. capabilities. Thus, in addition to defining clear guidelines which document how the integration of new components is to be performed, model validation features of Capella should be used extensively. This confirms our choice of using Arcadia to model both the reference architecture, the logical data model, and their derived counterparts.

Regarding the usability of an integrated framework, we found that manually configuring components through configuration files or command-line arguments is cumbersome and tedious, as it hinders fast reconfigurations and changes to the overall workflow. This need for flexibility can be encountered in workshop sessions with industrial partners, where an analysis might first be defined collaboratively as an experiment, which is then successively refined. Thus, the analyst may want to quickly reconfigure an existing analysis, or perhaps enable or disable filters, amongst other possible activities. Thus, as previously described, a visual approach which relies on a flow-based editor for configuring and performing an end-to-end analysis is desirable.

In conclusion, the key architectural decisions of our reference architecture, which are motivated by real-world experiences, proved to be useful for our prototypical implementation.

[19] https://www.arangodb.com/

9 Summary

This chapter concludes this thesis by summarizing our contributions and provides an outlook on possible future improvements.

9.1 Conclusion

After identifying typical goals from industrial publications and identifying analysis tools pertaining to our analysis needs, we demonstrated the need for complex analysis recipes through the use of scenarios we have encountered in prior work. Then, after presenting a recipe classification which enables the description of transitions through the different quadrants of the SPL process, we provided first recipes which solve complex analysis needs. Finally, we described a reference architecture for an integrated framework, which enables the integration of the aforementioned recipes (also referred to as toolchain), and provided a prototypical implementation for a selected recipe.

9.2 Future Work

Eliciting goals and needs from the SPLC's Hall of Fame proved to be difficult, as many study cases described solution-driven goals, but failed to describe the actual underlying need. The reason for this may be that organizations typically only publish "success stories" (Rabiser et al., 2019). Thus in future work, goals should be elicited from industrial partners to fully understand what motivates reengineering and reverse-engineering efforts in the context of product line engineering.

Additionally, it may be worthwhile to crowdsource analysis recipes from the academic community (for example at the Software Product Line Conference), given that previous publications already describe what might be considered analysis recipes, i. e. useful combination of individual analyses (von Rhein et al., 2013; Horcas et al., 2019; Grüner et al., 2019). To this end, the relevant subset of all analysis tools identified in Chapter 5 should be expanded to encompass all common analysis tools, allowing for the largest possible number of analysis recipes, functionally equivalent or unique. This collection could be made available online, benefiting the wider software product line community.

Regarding the reference architecture, we opted for an abstract representation of the core functionality an integrated framework should provide. This decision was willfully made as we aimed to provide a first reference architecture, which was used for our prototypical implementation for gaining additional insight. Thus, architectural drivers such as the ability to perform incremental analyses on systems which are constantly growing in time and space were left out. However, as systems grow in time and space, being able to perform incremental analyses becomes crucial to mastering the overall complexity. V. L. Tenev and Becker (2020) describe this need in the context of the Variant Analysis approach. As a result, this delta-analysis capability should be considered an architectural driver for subsequent efforts.

In conclusion, an integrated framework is of current value to both research and industry, and warrants further investigation. As stated by Berger et al. (2020), practitioners would rather realize new functionality instead of recovering information from existing code.

Thus, our work not only provides a broad overview of the problem to be addressed and the current state of the practice, but also punctually delves into detail and depth, providing first findings and laying the foundation for further explorations into the topic.

References

Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., & Lahire, P. (2012). On extracting feature models from product descriptions. *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, 45–54. https://doi.org/10.1145/2110147.2110153

Acher, M., Collet, P., Lahire, P., & France, R. B. (2013). FAMILIAR: A domain-specific language for large scale management of feature models. *Science of Computer Programming (SCP)*, *78*(6), 657–681.

Adams, B., De Meuter, W., Tromp, H., & Hassan, A. E. (2009). Can we refactor conditional compilation into aspects? *Proceedings of the 8th ACM International Conference on Aspect-Oriented Software Development*, 243–254. https://doi.org/10.1145/1509239.1509274

Antkiewicz, M., Ji, W., Berger, T., Czarnecki, K., Schmorleiz, T., Lämmel, R., Stănciulescu, Ş., Wąsowski, A., & Schaefer, I. (2014). Flexible product line engineering with a virtual platform. *Companion Proceedings of the 36th International Conference on Software Engineering*, 532–535. https://doi.org/10.1145/2591062.2591126

Apel, S., Batory, D., Kästner, C., & Saake, G. (2013a). Analysis of software product line. *Feature-oriented software product lines: Concepts and implementation* (pp. 243–282). Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-642-37521-7_10

Apel, S., Batory, D., Kästner, C., & Saake, G. (2013b). A development process for feature-oriented product lines. *Feature-oriented software product lines: Concepts and implementation* (pp. 17–44). Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-642-37521-7_2

Apel, S., Kastner, C., & Lengauer, C. (2009). Featurehouse: Language-independent, automated software composition. *Proceedings of the 31st International Conference on Software Engineering*, 221–231. https://doi.org/10.1109/ICSE.2009.5070523

Arcaini, P., Gargantini, A., & Vavassori, P. (2017). Automated repairing of variability models. *Proceedings of the 21st International Systems and Software Product Line Conference - Volume A*, 9–18. https://doi.org/10.1145/3106195.3106206

Assunção, W. K. G., Lopez-Herrejon, R. E., Linsbauer, L., Vergilio, S. R., & Egyed, A. (2017). Reengineering legacy applications into software product lines: A systematic mapping. *Empirical Software Engineering*, *22*(6), 2972–3016. https://doi.org/10.1007/s10664-017-9499-z

Aversano, L., Di Penta, M., & Baxter, I. D. (2002). Handling preprocessor-conditioned declarations. *Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation*, 83.

Basit, H. A., & Jarzabek, S. (2009). A data mining approach for detecting higher-level clones in software. *IEEE Transactions on Software Engineering*, *35*(4), 497–514. https://doi.org/10.1109/TSE.2009.16

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice* (2nd ed.). Addison-Wesley Professional.

Becker, M. (2018). Product line engineering: PLE principles & experiences [Lecture notes, TU Kaiserslautern, INF-31-52-M-6]. http://wwwagse.informatik.uni-kl.de/teaching/ple/

Benavides, D., Segura, S., & Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*, *35*(6), 615–636. https://doi.org/https://doi.org/10.1016/j.is.2010.01.001

Berger, T., Chechik, M., Kehrer, T., & Wimmer, M. (2019). Software evolution in time and space: Unifying version and variability management (Dagstuhl Seminar 19191) (T. Berger, M. Chechik, T. Kehrer, & M. Wimmer, Eds.). *Dagstuhl Reports*, *9*(5), 1–30. https://doi.org/10.4230/DagRep.9.5.1 Keywords: software configuration management, versioning, variability management, software product lines, empirical evaluation

Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., & Wąsowski, A. (2013). A survey of variability modeling in industrial practice. *Proceedings of the Seventh International Workshop on Variability Modelling of Software-Intensive Systems*. https://doi.org/10.1145/2430502.2430513

Berger, T., Steghöfer, J.-P., Ziadi, T., Robin, J., & Martinez, J. (2020). The state of adoption and the challenges of systematic variability management in industry. *Empirical Software Engineering*, *25*(3), 1755–1797. https://doi.org/10.1007/s10664-019-09787-6

Burger, A., & Grüner, S. (2018). FINALIsT 2: Feature identification, localization, and tracing tool. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 532–537. https://doi.org/10.1109/SANER.2018.8330254

Catal, C. (2009). Barriers to the adoption of software product line engineering. *SIGSOFT Software Engineering Notes*, *34*(6), 1–4. https://doi.org/10.1145/1640162.1640164

Clements, P. C., & Northrop, L. (2001). *Software product lines: Practices and patterns*. Addison-Wesley Longman Publishing Co., Inc.

Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E., & Bone, M. (2010). The concept of reference architectures. *Systems Engineering*, *13*(1), 14–27. https://doi.org/https://doi.org/10.1002/sys.20129

Collard, M. L., Decker, M. J., & Maletic, J. I. (2011). Lightweight transformation and fact extraction with the srcml toolkit. *Proceedings of the 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation*, 173–184. https://doi.org/10.1109/SCAM.2011.19

Davis, S. (1987). *Future perfect*. Addison-Wesley.

Debbiche, J., Lignell, O., Krüger, J., & Berger, T. (2019). Migrating java-based apo-games into a composition-based software product line. *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, 98–102. https://doi.org/10.1145/3336294.3342361

Deissenboeck, F., Juergens, E., Hummel, B., Wagner, S., Parareda, B. M. y., & Pizka, M. (2008). Tool support for continuous quality control. *IEEE Software*, *25*(5), 60–67. https://doi.org/10.1109/MS.2008.129

Dintzner, N., Van Deursen, A., & Pinzger, M. (2014). Extracting feature model changes from the linux kernel using FMDiff. *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*. https://doi.org/10.1145/2556624.2556631

Dintzner, N., van Deursen, A., & Pinzger, M. (2016). FEVER: Extracting feature-oriented changes from commits. *Proceedings of the 13th International Conference on Mining Software Repositories*, 85–96. https://doi.org/10.1145/2901739.2901755

Duszynski, S. (2015). *PhD theses in experimental software engineering. Vol. 51. Analyzing similarity of cloned software variants using hierarchical set models* (*Vol. 51*). Fraunhofer IRB Verlag. http://publica.fraunhofer.de/dokumente/N-332392.html

Duszynski, S., & Becker, M. (2012). Recovering variability information from the source code of similar software products. *Proceedings of the Third International Workshop on Product LinE Approaches in Software Engineering*, 37–40.

Duszynski, S., Knodel, J., & Becker, M. (2011). Analyzing the source code of multiple software variants for reuse potential. *2011 18th Working Conference on Reverse Engineering*, 303–307. https://doi.org/10.1109/WCRE.2011.44

Duszynski, S., Knodel, J., & Lindvall, M. (2009). SAVE: Software architecture visualization and evaluation. *Proceedings of the 2009 13th European Conference on Software Maintenance and Reengineering*, 323–324. https://doi.org/10.1109/CSMR.2009.52

Duszynski, S., Tenev, V. L., & Becker, M. (2020). N-way Diff: Set-based comparison of software variants. *2020 Working Conference on Software Visualization (VISSOFT)*, 72–83. https://doi.org/10.1109/VISSOFT51673.2020.00012

El-Sharkawy, S., Krafczyk, A., & Schmid, K. (2019). MetricHaven: More than 23,000 metrics for measuring quality attributes of software product lines. *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B*, 25–28. https://doi.org/10.1145/3307630.3342384

Erl, T. (2005). *Service-oriented architecture: Concepts, technology, and design*. Prentice Hall PTR.

Feigenspan, J., Schulze, M., Papendieck, M., Kästner, C., Dachselt, R., Köppen, V., Frisch, M., & Saake, G. (2012). Supporting program comprehension in large preprocessor-based software product lines. *IET Software*, *6*, 488–501. https://digital-library.theiet.org/content/journals/10.1049/iet-sen.2011.0172

Fenske, W., Schulze, S., Meyer, D., & Saake, G. (2015). When code smells twice as much: Metric-based detection of variability-aware code smells. *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 171–180. https://doi.org/10.1109/SCAM.2015.7335413

Garrido, A., & Johnson, R. (2003). Refactoring c with conditional compilation. *Proceedings of the 18th IEEE International Conference on Automated Software Engineering*, 323–326. https://doi.org/10.1109/ASE.2003.1240330

Gazzillo, P., & Grimm, R. (2012). SuperC: Parsing all of c by taming the preprocessor. *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, 323–334. https://doi.org/10.1145/2254064.2254103

Grebhahn, A., Kaltenecker, C., Engwer, C., Siegmund, N., & Apel, S. (2021). Lightweight, semi-automatic variability extraction: A case study on scientific computing. *Empirical Software Engineering*, *26*(2), 23. https://doi.org/10.1007/s10664-020-09922-8

Grüner, S., Burger, A., Abukwaik, H., El-Sharkawy, S., Schmid, K., Ziadi, T., Paule, A., Suda, F., & Viehl, A. (2019). Demonstration of a toolchain for feature extraction, analysis and visualization on an industrial case study. *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, *1*, 459–465. https://doi.org/10.1109/INDIN41052.2019.8972141

Harzing, A.-W. (2007). *Publish or perish*. https://harzing.com/resources/publish-or-perish

Horcas, J.-M., Pinto, M., & Fuentes, L. (2019). Software product line engineering: A practical experience. *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, 164–176. https://doi.org/10.1145/3336294.3336304

ISO/IEC 26550:2015(E). (2015). *Software and systems engineering – reference model for product line engineering and management* (Standard). International Organization for Standardization. Geneva, Switzerland.

ISO/IEC 9216-1. (2001). *Software engineering – Product quality, part 1* (Standard). International Organization for Standardization. Geneva, Switzerland.

Johansen, M. F., Haugen, Ø., & Fleurey, F. (2012). An algorithm for generating t-wise covering arrays from large feature models. *Proceedings of the 16th International Software Product Line Conference - Volume 1*, 46–55. https://doi.org/10.1145/2362536.2362547

Jürgens, E., Deissenboeck, F., & Hummel, B. (2009). Clonedetective - a workbench for clone detection research. *Proceedings of the 31st International Conference on Software Engineering*, 603–606. https://doi.org/10.1109/ICSE.2009.5070566

Kamiya, T., Kusumoto, S., & Inoue, K. (2002). CCFinder: A multi-linguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, *28*(7), 654–670. https://doi.org/10.1109/TSE.2002.1019480

Kanda, T., Ishio, T., & Inoue, K. (2013). Extraction of product evolution tree from source code of product variants. *Proceedings of the 17th International Software Product Line Conference*, 141–150. https://doi.org/10.1145/2491627.2491637

Kästner, C., & Apel, S. (2008). Type-checking software product lines - a formal approach. *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, 258–267. https://doi.org/10.1109/ASE.2008.36

Kenner, A., Kästner, C., Haase, S., & Leich, T. (2010). Typechef: Toward type checking #ifdef variability in c. *Proceedings of the 2nd International Workshop on Feature-Oriented Software Development*, 25–32. https://doi.org/10.1145/1868688.1868693

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering* (tech. rep. EBSE-2007-01). https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.471&rep=rep1&type=pdf

Kröher, C., El-Sharkawy, S., & Schmid, K. (2018a). KernelHaven: An experimentation workbench for analyzing software product lines. *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 73–76. https://doi.org/10.1145/3183440.3183480

Kröher, C., El-Sharkawy, S., & Schmid, K. (2018b). KernelHaven: An open infrastructure for product line analysis. *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 2*, 5–10. https://doi.org/10.1145/3236405.3236410

Krueger, C. W. (2002). Easing the transition to software mass customization. In F. van der Linden (Ed.), *Software product-family engineering* (pp. 282–293). Springer Berlin Heidelberg.

Krüger, J., Pinnecke, M., Kenner, A., Kruczek, C., Benduhn, F., Leich, T., & Saake, G. (2018). Composing annotations without regret? practical experiences using FeatureC. *Software: Practice and Experience*, *48*(3), 402–427. https://doi.org/https://doi.org/10.1002/spe.2525

Kuiter, E., Krieter, S., Krüger, J., Ludwig, K., Leich, T., & Saake, G. (2018). Pclocator: A tool suite to automatically identify configurations for code locations. *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1*, 284–288. https://doi.org/10.1145/3233027.3236399

Lamm, J. G., & Weilkiens, T. (2014). Method for deriving functional architectures from use cases. *Systems Engineering*, *17*(2), 225–236. https://doi.org/https://doi.org/10.1002/sys.21265

Liebig, J., Janker, A., Garbe, F., Apel, S., & Lengauer, C. (2015). Morpheus: Variability-aware refactoring in the wild. *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, 380–391.

Liebig, J., Kästner, C., & Apel, S. (2011). Analyzing the discipline of preprocessor annotations in 30 million lines of c code. *Proceedings of the Tenth Interna-*

tional Conference on Aspect-Oriented Software Development, 191–202. https://doi.org/10.1145/1960275.1960299

Lindvall, M., Becker, M., Tenev, V., Duszynski, S., & Hinchey, M. (2016). Good change and bad change: An analysis perspective on software evolution. In B. Steffen (Ed.), *Transactions on foundations for mastering change i* (pp. 90–112). Springer International Publishing. https://doi.org/10.1007/978-3-319-46508-1_6

Lisboa, L. B., Garcia, V. C., Almeida, E. S., & Meira, S. R. (2011). ToolDAy: A tool for domain analysis. *International Journal on Software Tools for Technology Transfer*, *13*(4), 337–353.

Lisboa, L. B., Garcia, V. C., Lucrédio, D., de Almeida, E. S., de Lemos Meira, S. R., & de Mattos Fortes, R. P. (2010). A systematic review of domain analysis tools. *Information and Software Technology*, *52*(1), 1–13. https://doi.org/https://doi.org/10.1016/j.infsof.2009.05.001

Locke, E. A., & Latham, G. P. (1990). *A theory of goal setting & task performance*. Prentice-Hall, Inc.

Ludwig, K., Krüger, J., & Leich, T. (2020). FeatureCoPP: Unfolding preprocessor variability. *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*. https://doi.org/10.1145/3377024.3377039

Martinez, J., & Parsai, A. (2018). *REVaMP2 deliverable 3.1: Identification of relevant state of the art* [Alessandro Cornaglia and Ali Parsai and Andrey Sadovykh and Anton Paule and Antonin Abherve and Borja López and Carlos Cetina and Christian Kröher and Damir Nesic and Geert Vanstraelen and Gilles Malfreyt and Jabier Martinez and Jacques Robin and Jaime Font and Jan-Philipp Steghöfer and Klaus Schmid and Marc Bollen and Matthieu Pfeiffer and Michael Benkel and Mukelabai and Olivier Biot and Raúl Mazo and Sascha El-Sharkawy and Sebastian Herold and Sebastian Reiter and Serge Demeyer and Tewfik Ziadi and Thorsten Berger and Tobias Beichter and Uwe Ryssel]. http://www.revamp2-project.eu/images/documentation/REVAMP2_D3.1_v05.pdf

Martinez, J., Ziadi, T., Bissyandé, T. F., Klein, J., & Traon, Y. L. (2015). Bottom-up adoption of software product lines: A generic and extensible approach. In D. C. Schmidt (Ed.), *Proceedings of the 19th international conference on software product line* (pp. 101–110). Association for Computing Machinery. https://doi.org/10.1145/2791060.2791086

Martins, L. A., Parreira, P. A., Freire, A. P., & Costa, H. (2018). Exploratory study on the use of software product lines in the development of quality assistive technology software. *Proceedings of the 8th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-Exclusion*, 262–269. https://doi.org/10.1145/3218585.3218670

McCloskey, B., & Brewer, E. (2005). Astec: A new approach to refactoring c. *Proceedings of the 10th European Software Engineering Conference Held Jointly* *with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 21–30. https://doi.org/10.1145/1081706.1081712

Medeiros, F., Ribeiro, M., Gheyi, R., Apel, S., Kästner, C., Ferreira, B., Carvalho, L., & Fonseca, B. (2018). Discipline matters: Refactoring of preprocessor directives in the #ifdef hell. *IEEE Transactions on Software Engineering*, *44*(5), 453–469. https://doi.org/10.1109/TSE.2017.2688333

Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., & Saake, G. (2014). An overview on analysis tools for software product lines. *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, 94–101. https://doi.org/10.1145/2647908.2655972

Metzger, A., Pohl, K., Heymans, P., Schobbens, P.-Y., & Saval, G. (2007). Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. *15th IEEE International Requirements Engineering Conference (RE 2007)*, 243–253. https://doi.org/10.1109/RE.2007.61

Michelon, G. K. (2020). Evolving system families in space and time. *Proceedings of the 24th ACM International Systems and Software Product Line Conference - Volume B*, 104–111. https://doi.org/10.1145/3382026.3431252

Miodonski, P., Forster, T., Knodel, J., Lindvall, M., & Muthig, D. (2004). *Evaluation of software architectures with eclipse* (IESE-Report 107.04/E). Fraunhofer Institute for Experimental Software Engineering (IESE). Germany. http://publica.fraunhofer.de/dokumente/N-25615.html

Morais Ferreira, D., Becker, M., & Tenev, V. L. (2020). Experience report on variability improvement in a product line engineering unaware company. *Proceedings of the 24th ACM International Systems and Software Product Line Conference - Volume B*, 21–28. https://doi.org/10.1145/3382026.3425771

Morais Ferreira, D., Tenev, V. L., & Becker, M. (2021). Product-line analysis cookbook: A classification system for complex analysis toolchains [Accepted for publication]. *Proceedings of the 25th ACM International Systems and Software Product Line Conference*.

Nadi, S., Berger, T., Kästner, C., & Czarnecki, K. (2014). Mining configuration constraints: Static analyses and empirical results. *Proceedings of the 36th International Conference on Software Engineering*, 140–151. https://doi.org/10.1145/2568225.2568283

Nair, S. S., Becker, M., & Tenev, V. (2020). A comparative study on variability code analysis technology. *Proceedings of the 24th ACM International Systems and Software Product Line Conference - Volume B*, 37–43. https://doi.org/10.1145/3382026.3425775

Northrop, L. M. (2008). *Software product lines essentials* (tech. rep.). Carnegie Mellon University, Pittsburgh. https://apps.dtic.mil/dtic/tr/fulltext/u2/a632553.pdf

Peña, J., Hinchey, M. G., Resinas, M., Sterritt, R., & Rash, J. L. (2007). Designing and managing evolving systems using a MAS product line approach [Special Issue on the 5th International Workshop on System/Software Architectures (IWSSA'06)]. *Science of Computer Programming*, *66*(1), 71–86. https://doi.org/https://doi.org/10.1016/j.scico.2006.10.007

Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, *64*, 1–18. https://doi.org/10.1016/j.infsof.2015.03.007

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

Rabiser, R., Schmid, K., Becker, M., Botterweck, G., Galster, M., Groher, I., & Weyns, D. (2018). A study and comparison of industrial vs. academic software product line research published at splc. *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1*, 14–24. https://doi.org/10.1145/3233027.3233028

Rabiser, R., Schmid, K., Becker, M., Botterweck, G., Galster, M., Groher, I., & Weyns, D. (2019). Industrial and academic software product line research at splc: Perceptions of the community. *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, 189–194. https://doi.org/10.1145/3336294.3336310

Roques, P. (2018). Reminders for the arcadia method. *Systems architecture modeling with the arcadia method* (pp. 1–23). Elsevier. https://doi.org/https://doi.org/10.1016/B978-1-78548-168-0.50001-3

Rowley, J. (2007). The wisdom hierarchy: Representations of the DIKW hierarchy. *Journal of Information Science*, *33*(2), 163–180. https://doi.org/10.1177/0165551506070706

Ruprecht, A., Heinloth, B., & Lohmann, D. (2014). Automatic feature selection in large-scale system-software product lines. *Proceedings of the 2014 International Conference on Generative Programming: Concepts and Experiences (GPCE)*, 39–48. https://doi.org/10.1145/2658761.2658767

Sadovykh, A., Ziadi, T., Bagnato, A., Berger, T., Steghöfer, J.-P., Robin, J., Mazo, R., & Gallego, E. (2019). REVaMP 2 project: Towards round-trip engineering of software product lines-approach, intermediate results and challenges. In M. Mazzara, J.-M. Bruel, B. Meyer, & A. Petrenko (Eds.), *Software technology: Methods and tools* (pp. 406–417). Springer International Publishing. https://doi.org/10.1007/978-3-030-29852-4_34

Schlie, A., Rosiak, K., Urbaniak, O., Schaefer, I., & Vogel-Heuser, B. (2019). Analyzing variability in automation software with the variability analysis toolkit. *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B*, 191–198. https://doi.org/10.1145/3307630.3342408

Schmid, K., & Schank, M. (2000). PuLSE-BEAT – A decision support tool for scoping product lines: Proceedings of the international workshop on software architectures for product families (iw-sapf-3). In F. van der Linden (Ed.), *Software architectures for product families* (pp. 65–75). Springer Berlin Heidelberg.

She, S., Lotufo, R., Berger, T., Wąsowski, A., & Czarnecki, K. (2011). Reverse engineering feature models. *Proceedings of the 33rd International Conference on Software Engineering*, 461–470. https://doi.org/10.1145/1985793.1985856

Sincero, J., Tartler, R., Lohmann, D., & Schröder-Preikschat, W. (2010). Efficient extraction and analysis of preprocessor-based variability. *Proceedings of the Ninth International Conference on Generative Programming and Component Engineering*, 33–42. https://doi.org/10.1145/1868294.1868300

Singh, N., Gibbs, C., & Coady, Y. (2007). C-clr: A tool for navigating highly configurable system software. *Proceedings of the 6th Workshop on Aspects, Components, and Patterns for Infrastructure Software*, 9–es. https://doi.org/10.1145/1233901.1233910

Spinellis, D. (2003). Global analysis and transformations in preprocessed languages. *IEEE Transactions on Software Engineering*, *29*(11), 1019–1030. https://doi.org/10.1109/TSE.2003.1245303

Svahnberg, M., van Gurp, J., & Bosch, J. (2005). A taxonomy of variability realization techniques: Research articles. *Software: Practice and Experience*, *35*(8), 705–754.

Tartler, R., Dietrich, C., Sincero, J., Schröder-Preikschat, W., & Lohmann, D. (2014). Static analysis of variability in system software: The 90,000 #ifdefs issue. *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, 421–432.

Tartler, R., Lohmann, D., Sincero, J., & Schröder-Preikschat, W. (2011). Feature consistency in compile-time-configurable system software: Facing the linux 10,000 feature problem. *Proceedings of the Sixth Conference on Computer Systems*, 47–60. https://doi.org/10.1145/1966445.1966451

Tenev, V., Duszynski, S., & Becker, M. (2017). Variant analysis: Set-based similarity visualization for cloned software systems. *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B*, 22–27. https://doi.org/10.1145/3109729.3109753

Tenev, V. L., & Becker, M. (2020). Variant analysis in changing system landscapes. *22. Workshop Software-Reengineering und -Evolution (WSRE)*, 41–42. https://fg-sre.gi.de/fileadmin/FG/SRE/wsre2020/WSRE2020_Proceedings.pdf

Thao, C. (2012). Managing evolution of software product line. *Proceedings of the 34th International Conference on Software Engineering*, 1619–1621.

Thüm, T., Apel, S., Kästner, C., Kuhlemann, M., Schaefer, I., & Saake, G. (2012). Analysis strategies for software product lines. *Technical report (Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg. Internet)*. https://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen/ps/auto/TAK%2012.pdf

Thüm, T., Apel, S., Kästner, C., Schaefer, I., & Saake, G. (2014). A classification and survey of analysis strategies for software product lines. *ACM Computing Surveys*, *47*(1). https://doi.org/10.1145/2580950

van der Linden, F. J., Schmid, K., & Rommes, E. (2007). *Software product lines in action: The best industrial practice in product line engineering* (1st ed.). Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-540-71437-8

Vittek, M. (2003). Refactoring browser with preprocessor. *Proceedings of the Seventh European Conference on Software Maintenance and Reengineering*, 101.

Voirin, J.-L., Bonnet, S., Exertier, D., & Normand, V. (2016). Simplifying (and enriching) SysML to perform functional analysis and model instances. *INCOSE International Symposium*, *26*(1), 253–268. https://doi.org/https://doi.org/10.1002/j.2334-5837.2016.00158.x

von Rhein, A., Apel, S., Kästner, C., Thüm, T., & Schaefer, I. (2013). The PLA model: On the combination of product-line analyses. *Proceedings of the Seventh International Workshop on Variability Modelling of Software-Intensive Systems*. https://doi.org/10.1145/2430502.2430522

Weiss, D. M. (2008). The product line hall of fame. *Proceedings of the 2008 12th International Software Product Line Conference*, 395–395. https://doi.org/10.1109/SPLC.2008.56

Wijnstra, J. G. (2002). Critical factors for a successful platform-based product family approach. *Proceedings of the Second International Conference on Software Product Lines*, 68–89.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. https://doi.org/10.1145/2601248.2601268

Young, J. M., Walkingshaw, E., & Thüm, T. (2020). Variational satisfiability solving. *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*. https://doi.org/10.1145/3382025.3414965

Zeleny, M. (1987). Management support systems: Towards integrated knowledge management [1]. *Human Systems Management*, *7*, 59–70. https://doi.org/10.3233/HSM-1987-7108

Zhang, B. (2012). Extraction and improvement of conditionally compiled product line code. *Proceedings of the 20th International Conference on Program Comprehension*, 257–258. https://doi.org/10.1109/ICPC.2012.6240498

Zhang, B. (2015). *Phd theses in experimental software engineering. Vol. 53. VITAL - Reengineering variability specifications and realizations in software product lines* (*Vol. 53*). Fraunhofer IRB Verlag. http://publica.fraunhofer.de/dokumente/N-345252.html

Zhang, B., & Becker, M. (2012). Code-based variability model extraction for software product line improvement. *Proceedings of the 16th International Software*

Product Line Conference - Volume 2, 91–98. https://doi.org/10.1145/2364412.2364428

Zhang, B., & Becker, M. (2013a). Mining complex feature correlations from software product line configurations. *Proceedings of the Seventh International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'13)*. https://doi.org/10.1145/2430502.2430529

Zhang, B., & Becker, M. (2013b). RECoVar: A solution framework towards reverse engineering variability. *2013 4th International Workshop on Product Line Approaches in Software Engineering (PLEASE)*, 45–48. https://doi.org/10.1109/PLEASE.2013.6608664

Zhang, B., & Becker, M. (2014). Variability code analysis using the VITAL tool. *Proceedings of the 6th International Workshop on Feature-Oriented Software Development*, 17–22. https://doi.org/10.1145/2660190.2662113

Zhang, B., Tenev, V. L., & Becker, M. (2018). Analyzing software build architectures. *20. Workshop Software-Reengineering und -Evolution (WSRE)*, 13–14. https://fg-sre.gi.de/fileadmin/FG/SRE/wsre2018/WSRE2018_Proceedings.pdf

Ziadi, T., Henard, C., Papadakis, M., Ziane, M., & Le Traon, Y. (2014). Towards a language-independent approach for reverse-engineering of software product lines. *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 1064–1071. https://doi.org/10.1145/2554850.2554874