CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Evaluating and extending the feature model process: a case study

Bachelor of Science Thesis in Software Engineering and Management

Malik Hannan Ahmed
Malik Waleed Mahboob

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Evaluating and extending the feature model process: a case study

Supervisor: Wardah Mahmood
Examiner: Christian Berger

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023

Evaluating and Extending the Feature Modeling Process via a Case Study

Malik Waleed Mahboob
University of Gothenburg
Gothenburg, Sweden
gusmahboma@student.gu.se

Malik Hannnan Ahmed
University of Gothenburg
Gothenburg, Sweden
gusmalikah@student.gu.se

***Abstract*—The development of software product lines (SPLs) has revolutionized software engineering by enabling efficient creation of diverse software systems through the selection of feature combinations. Feature models serve as a critical tool in the context of SPLs. Andrzej Wasowski and Thorsten Berger recently proposed the process for feature model creation, derived from the principles of feature models proposed by Nešić et al. The process, although detailed and thorough, lacks a concrete and realistic evaluation. This thesis aims to address this gap by evaluating and extending the feature modeling process. Through a collaborative case study with a Swedish company, we assess the comprehensibility and applicability of the process model. Additionally, we propose improvements and extensions to the process in order to enhance its effectiveness in future applications. The study recognizes the importance of the feature modeling process in SPL development and seeks to provide valuable insights into its usability and potential enhancements. By evaluating and extending the process model through a practical case study, this research contributes to the advancement of feature modeling practices and their impact on software product line development.**

I. INTRODUCTION

A software product Line (SPL) is a collection of related software systems that share common features and development processes [1]. The major difference SPL-based development has from traditional software development is that the former involves creating a centralised platform with a core set of features implemented in it, and then selecting different feature combinations to meet the specific needs of different customers or market segments. This allows fast and improved development for new software products or variants. Additionally, as features are implemented in a consolidated platform, developers can put more focus on the quality of their implementation, thereby leading to high-quality products. SPL is highly efficient and desirable for industries where there is a need for a range of products that share common functionality and a range of diverse customers to cater to. Software Product Line Engineering (SPLE) is a development approach that focuses on creating and managing software product lines. It involves identifying and managing commonalities and variabilities across related software and developing reusable code (software) that can be used efficiently to build different products in the line.

Feature models are a critical artefact in SPL engineering and help in guiding the development of the product line. Feature models are the representation of the commonalities and variabilities of software products in a software product line (SPL). Specifically, feature models capture the features of a particular software system along with the dependencies and constraints between those features. Feature models can provide high-level visualisation of a system that is easily understandable for the stakeholders of the system, and as such, they offer a way to communicate requirements to the stakeholders. In the present times, feature models are used to model the common and variable characteristics of products in a software product line [2]. The utilisation of feature models within the industry is not a recent development, as they have been present for nearly three decades [3]. Feature models have proven their usefulness and how they can help in order to understand the core principles of a system along with the creation of new variants of a system. Recently, Andrzej Wasowski and Thorsten Berger [4] presented a process model that companies can follow when creating a feature model. This process model was the first process to provide a concrete set of guidelines for feature model creation. A process model is a mechanism that helps divide work into distinct phases in order to increase productivity and achieve the goal of the process. It offers a clear understanding of the different components of the process, enabling better organisation and management. The process model features both a top-down and bottom-up analysis, enabling developers to create feature models both from scratch as well as later when the company has already created some variants. The purpose of discussing feature models in this study is motivated by the lack of evaluation of the above-mentioned process model, which points to the core purpose of this case study.

The process model defined in the book [4] is a viable solution because it is synthesized from 34 concrete feature modeling principles extracted from literature presented by Nešić et al [8]. The principles of the process were created by analyzing and synthesizing these popular literature references, ensuring a solid foundation for the process. Additionally, the principles were motivated by real-world examples and practices from the industry, validating their applicability and effectiveness. These principles were derived from actual industry cases, demonstrating their relevance and suitability for the adoption of software product lines. They were also triangulated using expert interviews with 10 representatives from nine companies. The creators of the process have significant experience of 15 years in the domain of Feature Modeling, which further strengthens the credibility of the process model. By leveraging these principles and incorporating their expertise, the authors have formulated a set of activities that constitute the process model. This approach increases the likelihood of the process model being an effective starting point in practice and delivering the desired outcomes in the creation of feature models for software product lines.

This study aims to evaluate the above-mentioned process model with a Swedish company. While this is the first model to provide guidelines for creating feature models, it has not yet been evaluated in a real-world case study. The study explains the process with multiple activities divided into four phases: pre-modeling, domain analysis & scoping modeling activities, and maintenance & evolution activities. The core idea of this

study is to thoroughly review and comprehend the software, and then conduct a bottom-up analysis, which involves extracting all the implemented features from the software by carefully analysing and understanding the components that implement them. Our study will extend the bottom-up approach described in the book [4] to suit the unique characteristics of the singular product we are using for the case study. This product is distinct in that it does not have any variants for us to analyse. While the original process supports proactive and extractive methods [5]. Proactive method is when you start from scratch and work with the platform directly without having any existing variants. In this method you perform domain analysis and then you start with the product line. The extractive method is when you have multiple variants, and you perform diffing between them to find variation points where the variation points are the features of the product. You add those features in the feature model and copy code into one big codebase which would be the platform.

We will carefully analyse and understand the product's components to identify and extract the features. This process will require close attention to detail and a solid understanding of the programming language used in the software. Our goal is to extract the software's features by gaining a comprehensive understanding of its functionality, which is described as the bottom-up analysis method.

We will conduct our study side-by-side with another team that is working on a similar topic using a top-down approach defined in the book. We will be following the bottom-up approach with some extension for this study and the other team will focus on the top-down approach. The teams will work in collaboration with each other and the company that will help to implement the process on one of their products or a major part of their product to evaluate the feature model principle defined in the book.

A. Background

A feature model represents the information of all possible products of a software product line in terms of features and relationships among them. Feature models are a special type of information model widely used in software product line engineering. As Benavides, Segura, and Ruiz-Cortés [6] explain, a feature model is represented as a hierarchically arranged set of features composed of:

1. The parent or compound feature in a feature model that represents a higher-level feature that encompasses one or more child features or sub features. These child features are dependent on the parent feature and are typically included in the software product only when the parent feature is selected. The relationship between the parent feature and its child features is typically hierarchical in nature, where the child features inherit the properties of the parent feature.
2. Cross-tree or cross-hierarchy constraints, which are conditions that restrict the selection of certain features based on the selection of other features. These constraints are typically expressed as inclusion or exclusion statements that specify that if a particular feature is selected, then one or more other features must also be selected or excluded. These constraints are used to ensure that the selected feature model is consistent and does not result in conflicting or incompatible configurations. Cross-tree constraints are important for maintaining the validity of the feature model and for avoiding errors in the software product resulting from an inconsistent configuration.

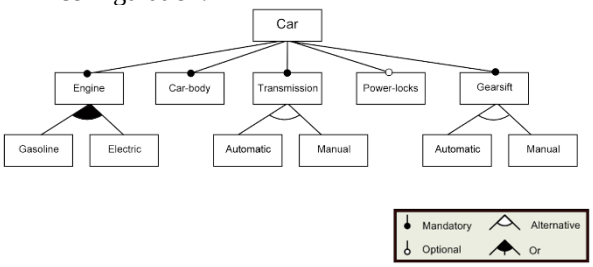

Fig. 1 Feature model for car production, inspired by Czarnecki [7]

A simplified feature model inspired by the car industry is shown in Fig. 1, demonstrating how features are used to specify and build a car. According to the model, the engine, transmission, car body, and gearshift are mandatory features that must be included in all cars. These mandatory features are considered parent features of the feature model. Additionally, all cars must have an engine, which can be either gasoline or electric. The engine is a mandatory feature and is considered a parent feature of the feature model. The choice between gasoline or electric engines depends on the customer's needs and preferences. The choice between automatic and manual gear shift are alternate features from which one of the features can be selected. Moreover, the production of a car may optionally include support for security features such as power locks. These optional features can be selected based on the customer's preferences.

B. Feature Modeling Process

The process defined in the book [4] is classified into four different phases: Pre-modeling, Domain Analysis and Scoping, Modeling, and Maintenance and Evolution. The figure below taken from the book [4] shows the various stages and typical iteration during the last three phases.

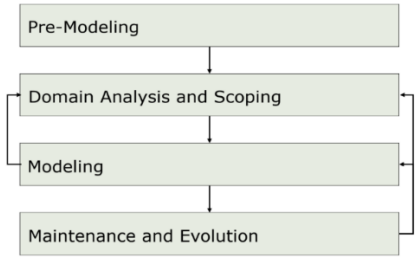

Fig. 2 A high-level illustration of the process presented by Wasowiski and Berger [4]

Pre-Modeling. In the first phase of the modeling process, it is important to plan and train the people involved. This includes clearly defining the purpose of the model, identifying the stakeholders, and creating a plan for managing changes and expectations. To start, we need to determine the specific use case and the important features of the model. This helps us avoid wasting time on irrelevant things. We should involve domain experts, modelers, and users as stakeholders, but it is best to keep the number of modelers low for clarity. The stakeholders need to be trained in using the tools and process for feature modeling. We should also communicate the benefits of feature modeling and explain any necessary

changes in the way we work, our organisational structure, and our architecture. Creating a forum and holding workshops can help maintain and improve the feature model, with a designated group of main modelers providing guidance to others. Additionally, we might consider defining criteria for breaking down the model based on existing hierarchies. It is also helpful to establish a common language by unifying domain terminology with a descriptive hierarchical naming scheme. By following these steps, we can make sure that; i) everyone understands the purpose of the model, ii) the correct people are involved and iii) they know how to manage changes effectively.

Domain Analysis and Scoping Activities. Domain analysis and scoping are important parts of the feature-modeling process. They involve gathering and documenting information about features and their relationships. These activities are performed iteratively to guide the modeling phase. Firstly, features are identified through either a bottom-up or top-down approach. Bottom-up identification focuses on existing and demanded configuration options, while top-down identification involves selecting and defining the domain of focus and collecting relevant domain information. The identification process may include analysing existing system variants or consulting domain experts. Once features are identified, they go through an approval process before being added to the feature model. The next step is modeling the features. This is done by creating a feature diagram that visually represents the feature model using the appropriate notation and concepts of the chosen feature modeling language. In addition, cross-tree constraints need to be identified and modeled in the feature diagram. These constraints span across the feature model and accurately represent the relationships between features.

Modeling activities. During the modeling phase of feature modeling, the focus is on creating a feature model based on the information gathered in the previous phases. The modeling phase involves several key activities. Model modularization is the process of breaking down the feature model into smaller, more manageable models to aid in maintenance, version control, and reducing cross-tree constraints. This activity requires defining the structure of model files and ensuring consistency between them. Next, a coarse feature hierarchy is defined by creating an initial organisation of features within the feature model, including feature groups and sub-trees that logically partition the domain. This improves cohesion, reduces coupling, and minimises the need for cross-tree constraints. Features are then added. Constraints between features are identified and modeled, avoiding complex constraints, and utilising the feature hierarchy and graphical elements. Views or profiles are defined to facilitate model modularization, improve understanding, and enhance navigation of the feature model. The validation of the feature model is crucial, and it can be done through stakeholder reviews, product derivations, and regression testing. Stakeholder reviews involve inviting domain experts and other stakeholders to validate the feature model, while product derivations allow stakeholders to perform configuration and create variants. Regression testing ensures that changes to the feature model do not break existing configurations, maintaining the correctness of the model.

Maintenance and Evolution Activities. Maintenance and evolution activities in feature modeling leverage activities from previous phases, such as domain analysis and scoping, as well as modeling activities. To ensure consistency, a centralised feature model governance approach is crucial. Regular validation activities should be conducted, and additional activities can support the evolution and maintenance of the model. Key activities in maintenance and evolution include model version control, which involves tracking the evolution of the feature model through version control to keep a record of changes and maintain an overview of its evolution. Regularly removing features is important to reduce maintenance overhead and system complexity. The decision to remove features should be made through discussions in established workshop or forum formats. Deprecated features should be flagged, their default values changed to false. Finally, the features should be removed from the model and the corresponding software assets. Additionally, optimizations should be performed as the feature model evolves. The hierarchy and constraints may become more complex over time, so it is essential to optimise them for clarity and maintainability. However, caution should be exercised to avoid invalidating existing variants, and proper tool support for refactoring should be utilised to ensure a smooth optimization process.

II. Related Work

Our field of study falls under the domain of SPL. We will look at feature models which play an essential part in modeling the commonalities and variabilities between the products of a company. Feature models can also be very helpful for organisations as they provide a structured approach to identifying and managing the features of a product or system, which helps in making informed decisions about which features to include, prioritize, or remove. Even though the impact and importance of feature models are quite obvious, and they are very useful to create variants of a system, still there is a lack of process to create feature models.

As previously stated, there are not many studies done previously that propose concrete guidelines on how to perform feature modeling. To identify the related work, we used the paper by Nešić et al [8] as an initial resource for backward snowballing. We chose this paper because it presented a list of 34 feature modeling principles and showed the current modeling practices, which provides a good starting point for related work. We analysed each reference defined in the related work section of the paper by reading the title and the abstract section but in some cases, we read the full papers to get a better understanding which allowed us to identify the relevant papers. Nešić et al‘s work analysed feature modeling practices and synthesised them into principles which were extracted by conducting ten interviews with industrial practitioners and literature reviewing of thirty-one relevant papers using the snowballing method. This paper represents a systematically collected set of principles from a paper on industrial practices, supported by practices reported by feature modeling practitioners. Unlike this paper, we follow the feature modeling process, specifically, the bottom-up modeling process in terms of multiple activities and sub-activities theoretically explained in the book written by Andrzej Wasowski and Thorsten Berger [4], to evaluate the applicability, practicality, and usability of the process model defined in the book [4].

Rabiser et al. [9] have also made significant contributions to the field of feature modeling in the context of large-scale industrial software systems. In their paper, they present a

multi-purpose, multi-level approach to feature modeling that can handle the complexity of such systems. The authors describe a method for systematically identifying and modeling features at different levels of abstraction and demonstrate its effectiveness through case studies in two large-scale industrial systems. The study proposes a modeling approach that aims to support multi-purpose and multi-level feature modeling while addressing the need for defining model dependencies and feature-to-code mappings. To begin the modeling process, the authors used a top-down modeling strategy and data sources such as problem space models, configuration space models, and solution space models to create feature models for KeMotion and KePlast. The nature of the case study is exploratory, as it aimed to investigate the effectiveness of the proposed approach in managing complexity and improving software quality. The modeling process followed by the author is less systematic. In contrast to their approach, we adopt a feature modeling process that involves pre-defined multiple activities and sub-activities, specifically the bottom-up modeling process outlined in the book co-authored by Andrzej Wasowski and Thorsten Berger [4].

Through the snowballing process, we identified another relevant paper [10] that presents an approach to managing the variability in SOA (service-oriented architecture) using the bottom-up approach for feature extraction from different process models. Unlike our study where we analyse the bottom-up approach via a case study, this paper proposes a bottom-up SPL construction approach that relies on formal concept analysis, latent semantic indexing, and DBSCAN.

Krüger et al. [11] presented a case study on recovering feature facets in software systems. The authors of the paper manually identified and located features and their facets in two open-source systems using a combination of static and dynamic analysis techniques. Their method was able to extract both mandatory and optional feature facets, providing insights into the design of the software system. Although their paper presents multiple methods and techniques to extract features from systems, our study will focus solely on extracting features from the codebase and documentation within the repository, such as commit messages, pull requests, and sprint stories. In our study, we will follow the manual feature location method from the paper as one of the core methods, with the repository itself serving as the data source. The product repository is hosted on the Azure cloud service, and additional documentation such as wikis and other text pages within the repository will be beneficial for our feature extraction. The study will follow a precise template to document the extracted features, which we will use to create feature models following the process defined in the book [4].

III. RESEACH METHODOLOGY

In our research, we employed a case study approach to comprehensively evaluate and extend the feature modeling process. By selecting a specific product developed by a Swedish company as our case study, we aimed to look deep into the applicability, feasibility, and comprehensibility of the feature modeling process and its individual components. Choosing a case study methodology allowed us to investigate the phenomenon in its real-world context, gaining rich insights and capturing the complexities associated with the feature modeling process. Through this approach, we were able to consider multiple perspectives, involve relevant stakeholders, and establish a clear governance structure, ensuring a meticulous examination of each phase of the feature modeling process. Moreover, the case study design enabled us to collaborate closely with the company's developers, analyze the product's codebase and UI, and extract features from various sources. Ultimately, the case study approach provided us with a comprehensive understanding of the feature modeling process and its practical implications, contributing to both theory and practice in the field.

The aim of the study is to evaluate the feasibility and effectiveness of the bottom-up approach to feature modeling in the context of a specific product developed by a Swedish company. The study will also aim to identify potential areas for improvement and provide insights into the limitations and effectiveness of the feature modeling process described in the book [4]. In addition, based on the identified areas for improvement, the study will extend the bottom-up approach as needed to ensure its effectiveness for the specific product being analysed. While the original process supports extractive and proactive methods only, we will adapt to the reactive strategy to identify and extract the implemented features [5]. The reactive method is when you have one variant that becomes the platform. You clone it and make changes in the cloned variant and then merge it back into the first variant. Here the first variant keeps growing but the development takes place outside in other variants making the first variant safe.

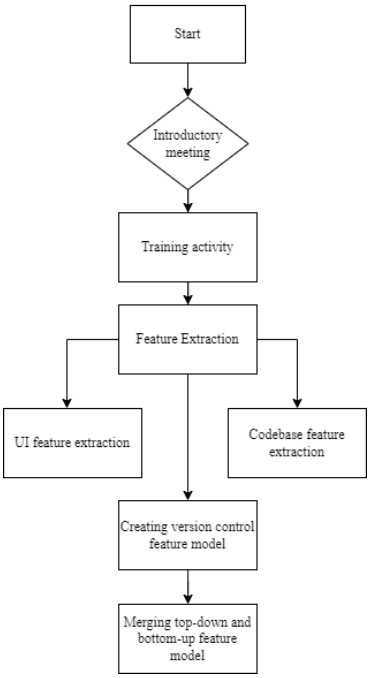

Fig. 3 Research Methodology Flow Diagram.

A. *Research questions and/or hypotheses*

In this case study, we will be triangulating our findings with the other thesis study that is following the top-down methodology to gather more data and gain a comprehensive understanding of the process's applicability.

To this end, we aim to answer the following research questions (RQs):

RQ1 To what extent is the process applicable in modeling features of a real-world industrial case study?

1. How feasible is the implementation of the process for the research team?
2. How comprehensible is the process?

3. How comprehensible are the individual parts of the process?
4. How applicable are the individual parts of the process?

RQ2 How can the process be extended for reactive product-line adoption?

1. What are the improvements and extensions that are missing in the process?
2. How can the existing guidelines about feature information sources help model features?

B. Research methology to be used

For this study, we focused on evaluating and extending the feature modeling process through a case study of a product developed by a Swedish company. Our goal was to answer the research questions regarding the applicability, feasibility, and comprehensibility of the feature modeling process and its individual parts. To achieve this, we took the following steps in our research methodology.

- We comprehensively gathered and documented information about features and their relationships to gradually increase modeling expertise and safely evolve the feature model during each of the four main phases of the feature modeling process described in the book [4]: pre-modeling, domain analysis and scoping, modeling, and maintenance and evolution. We involved relevant stakeholders and established a clear governance structure in each phase to ensure that changes were made carefully and consistently.

- As previously mentioned, we adopted the bottom-up approach for feature extraction as described in the book [4], with necessary extensions tailored to the product under evaluation. Since the product under study did not have any variants and only consisted of a single base model, the team extended the bottom-up approach and worked on the product in isolation while collaborating with the company developers for analysis. This involved extending the process for the reactive product-line adoption method, rather than following the extractive and proactive methods supported by the original process.

- We comprehensively understood the codebase of the existing product, which did not have any variants, in the feature extraction process through the extended bottom-up approach. Close collaboration with the company's developers was ensured, and since the process lacked sufficient details on data sources to consult for bottom-up analysis we took inspiration from the paper [12]. The paper extracts feature from various sources, including pull requests, commit messages, and manual feature location techniques. These techniques include code-based, text-based, and visualisation-based techniques that enable the analysis of a product capable of accommodating variants and feature toggles to identify implemented features that can be utilised as variants for customers. Through this approach, we extracted the features and constructed the base model.

- We, along with different stakeholders, participated in the training sessions for the activities of the process. This training session was a part of the pre-modeling phase and as such, was performed before the domain analysis and scoping. The training sessions consisted of questionnaires about the phases to understand and analyse the feasibility, applicability, and comprehensibility of the process and its individual activities. These questions helped us to identify any area of improvement for any specific activity in any of the 4 phases of the process.

- We started examining the product UI to extract its features and understand their dependencies. We thoroughly analysed the UI across multiple devices to identify any functional variations. Throughout the evaluation, we encountered three main pages, spending approximately five days on each page to comprehensively extract and understand all the features. This approach proved beneficial as it increased our familiarity with the product, facilitated comprehension of feature dependencies, and provided insights into the product's functionality and design. We focused on key elements such as buttons, menus, input fields, and visual components to identify and categorise features effectively. Starting with the UI was helpful as it allowed us to understand the product from a user's perspective, utilise visuals for feature identification, and expect a wide range of functionalities, including user customization, data representation, visualisation, and navigation controls. Through UI analysis, we aimed to extract and document all the features within the product's UI.

- Once we finished the feature extraction from the UI, we started looking into the product’s code base, which consisted of two repositories. Since the functionality across the three pages was independent, we adopted a sequential approach, focusing on one page at a time. Our goal was to identify the code sections associated with functional logic, rather than the UI representation or styling. This approach allowed us to find multiple features in the codebase that were not prominently visible in the UI but held direct usefulness for users. Extracting features from the code proved to be a time-consuming process, taking approximately seven days to complete for each of the three pages. Once we completed the feature extraction from the code, we scheduled a meeting with a core developer of the product from the company to showcase our findings and gather feedback for potential improvements. During this meeting, we collaborated on reviewing the product documentation in the wiki and other available resources to finalise feature names and establish their hierarchical dependencies. The decision-making process for features involved iterative discussions with the developer and thorough analysis of the codebase increased our expertise and understanding of the product domain. Our expectation in examining the code was to find features related to data processing, logic implementation, business logic, data validation, and other crucial logical operations. Overall, our approach entailed a detailed examination of the codebase, sequential analysis of each page, collaborative decision-making for feature selection and naming, and an iterative process that spanned approximately seven days on each page,

with the aim of identifying functional logic features in the codebase.

- Once we identified all the features, and finalised the feature model, we established a local Git repository. In order to combine both feature models from the bottom-up and top-down approach we initially committed the feature model from the top-down analysis in the local git repository and subsequently made incremental commits by adding our updated feature model to the repository. For each update, we used descriptive commit messages.

This research methodology aimed to identify potential areas of improvement in the described process and to extend the bottom-up approach described in the book. At the end of the study, we were left with a feature model for the product under evaluation, which helped the company identify features that can be considered as toggle features for the variants of that specific product and create a base model of the product.

C. Data collection

Data collection played an essential part in this case study. We strategically organised meetings on a need-to-know basis in order to ensure effective communication and obtain necessary clarifications on specific features or validation from developers. We conducted a total of three meetings with one of the core product developers who has three years of experience working with the product analysed in this study. These meetings were helpful to enhance our understanding of the subject. The primary objective of the first meeting was to provide the research team with a comprehensive introduction to the product, which played a crucial role in our case study. During this session, the developer guided us through the product's user interface and offered a concise overview of its functionalities. The meeting, titled "Introduction to the Project," involved the participation of both the research team and the developer. In the second meeting, our focus shifted towards gaining familiarity with the product's codebase. We aimed to look deeper into the technical aspects and inner workings of the product. This session allowed us to explore the underlying structure and architecture of the product, enabling us to comprehend its functionalities more effectively. Lastly, the final meeting served as a major helpful step in our case study, where we validated the feature model we had developed. Collaborating closely with one of the product developers, we reviewed and assessed the final feature model. This validation process was vital in ensuring the accuracy, reliability, and alignment of our feature model with the product's functionality. These meetings with the product developers gave us a better understanding of the product and identified which features can be considered variant features and what can be the base model.

During the pre-modeling phase of our process, we actively participated in a training session facilitated by the top-down research team. This workshop was specifically designed to equip stakeholders, including ourselves, with the necessary knowledge and skills essential for successful feature modeling. The training primarily focused on providing a comprehensive understanding of the feature-modeling notation, tools, and overall process. We took notes during these meetings to document the study. In addition, we provided stakeholders with questionnaires before each training session to gather their understanding of the training material. During the sessions, we sought feedback and comments on the activities from the stakeholders.

D. Data analysis

This thesis is a case study that aims to evaluate a process, making data analysis a crucial component of our study. We gathered a substantial amount of data in the form of notes, questionnaires, and feedback from the stakeholders of the company after each activity. Our primary focus was on analysing this data and discussing our findings with our supervisor regularly within a specified time frame. We compared our data analysis results with the documentation provided in the book [4] and assessed the consistency of the findings. Additionally, the stakeholders' feedback obtained through questionnaires provided valuable insights into the study's applicability and feasibility, which further contributed to our analysis.

IV. Results

The research involved the evaluation of a product in collaboration with a Swedish company. We found it challenging to comprehend the product's code structure directly due to the complexity of the product and unfamiliarity with the technologies and framework used. To overcome this, we adopted an approach of examining the product's features from the user interface (UI).

As previously mentioned, by analysing the UI, we gained a better understanding of the product and identified dependencies among the features across multiple pages. We started by interacting with the product on multiple screen sizes to see any difference in functionality, but the functionality remained consistent. We also analysed some overlay on the UI on small screen sizes, which emphasised that the product was mainly for large-screen users. The product was divided into three pages on which we spent approximately five days on each screen. This UI interaction helped us understand the user perspective of the functionality and gave insight into the flow of functionality, which was beneficial to understand the feature dependencies. Throughout our interaction with the UI, our focus was to look for components that were bound by some logic instead of the styling of the product. For this reason, we were more interested in how buttons, menus, input fields, and visual components behaved.

Through UI analysis, we extracted and documented all the features within the product's UI. When we documented these features, we also added the source of the feature to help us identify them later easily, along with their respective functionalities. Initially, we classified them as mandatory or optional based on our understanding. We took into consideration when classifying a feature as an optional feature if the functionality can be selected or deselected by different use cases from the user perspective. This helped us understand their use case and model the features into the feature model. In total, we extracted a total of 39 features where 22 were optional and 17 were mandatory features from the UI. These UI-extracted features were mainly domain or business-oriented and less technical features. Some of the features extracted from the UI were mainly representations of data and beneficial for the user experience rather than any technical logic behind them.

After successfully extracting the relevant features from the product's user interface (UI), our next step was to look into the code repository. The codebase was divided among two

different repositories, the first repository consisted of 2 pages out of the 3 pages of the product, and the second repository was only for the last page due to the aim of the company to reuse it and due to the size of the page. At that time, due to our interaction with the product through the UI, we knew that the functionality among the three pages was independent. Hence, we adopted a sequential approach, focusing on one page at a time and each page took approximately seven days. The goal was to identify functional logic code blocks rather than looking at the styling components. We divided the product page into smaller components for our own understanding and started looking into the hierarchy of feature implementation to understand the code structure and how the repository was implemented. The repositories were well-structured, with different pages and smaller shared components in their own respective folders, which made it easier for us to find any specific logic.

Our extraction from the code base helped find multiple features that were not prominently visible on the UI but held direct usefulness for the user. This process was time-consuming due to the massive code size in the repository and the fact that the framework used for the code implementation was not familiar to us. The project repository served as the primary source for understanding the relationships between features, as indicated by the commit messages. The user stories, which is a commonly used term in software development to capture requirements or desired features from the perspective of end users, provided insights into the hierarchical implementation of the features.

Interestingly, we discovered 19 distinct features from the codebase that had been missed during the initial extraction from the UI. The features extracted from the UI were mainly the visible prominent features that provided users with some benefit. On the other hand, the features extracted from the codebase were the logical implementations that made the UI-extracted features possible to implement. Furthermore, some of the components on the UI, which appeared to be data visualisations during UI feature extraction, were also considered features once we examined the codebase. We recognized their use case for the benefit of the user once we understood them from the code.

After completing the feature extraction, we scheduled a meeting with a core developer of the product from the company to present our findings and gather feedback for potential improvements. In this meeting, we collaborated on reviewing the product documentation in the wiki and other available resources to finalise the feature names and establish their hierarchical dependencies.

Feature Type	UI (User-Interface)	Code Base (Distinct Features)
Mandatory	17	19
Optional	22	-
Total	39	19
Combined	58	

Table. 1. Number of Extracted Features from Different Sources

We present the number of extracted features in Table 1, which enabled us to enhance the previously established feature model created by another research team using a top-down approach. By comparing the two feature models, we observed that our model provided a deeper level of detail and identified some features that had been missed by the top-down approach. The features extracted through the bottom-up approach were found to be highly technical and implementation-oriented. These features looked into the specific technical aspects of the system, providing a deeper understanding of its inner workings. On the other hand, the top-down feature model focused on user-facing and business-oriented features, highlighting functionalities and capabilities that are more relevant from a user's perspective.

To validate our feature model and gather feedback, we arranged a meeting with one of the core developers from the company. The purpose of the meeting was to assess if any improvements were needed before finalising the feature model. The main objective of the meeting was to validate the feature model we had developed following the feature model process defined in the book [4], before its finalisation. During the session, we examined the feature model alongside one of the core developers. Our primary aim was to verify the accuracy and appropriateness of the feature names utilised within the model, as well as to validate the constraints associated with each feature. This meeting served as a crucial step in ensuring the robustness and integrity of our feature model before its implementation.

RQ1: Assessment of the Feature-Modeling Process

RQ1 To what extent is the process applicable in modeling features of a real-world industrial case study?

The research team found that the process described in the book [4] was highly applicable in modeling features of a real-world industrial case study. The research team successfully implemented the process and effectively modeled the features for the industrial case study under evaluation.

However, it is important to note that the research team encountered certain considerations and challenges during the application of the process. Consequently, they adapted the process and took a reactive strategy to identify and extract the implemented features. While the original process supports extractive and proactive methods, we carefully analysed and understood the product's components to identify and extract the features.
Firstly, the process had some assumptions such as the product under consideration will consist of some variants, on the basis of which the activities were defined. However, in our case, those assumptions were not fulfilled as our product did not have any variants to extract features. Therefore, we needed to adapt and extend the process, which is discussed later in RQ2.

Secondly, the process required some improvements to increase its applicability for implementation with a real-world industrial case study. Some of the improvements are to discuss more on how to extract features in the bottom-up approach and how to combine the feature models from the top-down and bottom-up approaches. These improvements are also discussed in RQ2.

The research team also faced external dependencies that were not within their control and had to update assumptions made in the process based on the specific case study. Despite

these challenges, the team overcame them and achieved successful feature modeling.

Overall, the process provides a comprehensive and iterative approach to feature modeling, addressing various aspects and challenges involved in modeling features for real-world software systems. By following this process, organisations can effectively model and maintain feature models, ensuring the applicability and usefulness of the process in an industrial context.

RQ1.A: How feasible is the implementation of the process for the research team?

The research team found it feasible to implement the process described in the book [4] and successfully modeled the features of the evaluated product. However, there were concerns regarding external dependencies that were beyond the control of the research team, which affected the feasibility of certain activities outlined in the process.

First of all, the company's team went through training based on the process' activities in the first phase. This training was an external dependency, as the team size was not under our control. Secondly, the product under our research was developed using certain frameworks and technologies that were unfamiliar to the research team. This factor was beyond the control of the research team. These factors could affect the feasibility of the process and could impact the allocation of resources for implementing the process.

Additionally, some assumptions made in the process needed to be updated. The process is specific on how the product under evaluation will be, which was not the case in our research as the product did not have any variants which need to be updated in the process as discussed in RQ1.

However, the incremental nature of the process proved to be a significant advantage that enhanced its feasibility for the research team. The step-by-step approach guided the team in determining the sequence of activities, enabling them to organise their work and progress smoothly, regardless of any dependencies encountered.

RQ1.B: How comprehensible is the process?

The book [4] presents a comprehensive overview of the feature modeling process, which covers various phases including optional and mandatory activities. The process provided a step-by-step guide for implementing feature modeling, starting from pre-modeling activities, domain analysis & scoping, modeling, and maintenance & evolution.

The process was presented in a structured manner, with clear explanations of each phase and activity. It defined key concepts related to feature modeling, including the purpose of the model, relevant stakeholders, feature identification, modeling constraints, and model validation. Optional activities were also highlighted, along with recommendations for establishing a forum, conducting workshops, and maintaining the feature model over time.

Overall, the process offered a comprehensive understanding of feature modeling. However, it is important to consider the expertise and familiarity of the research team performing the process. Depending on the team's background and experience, they may require additional training or support to fully understand and implement the process effectively.

RQ1.C: How comprehensible are the individual parts of the process?

The process was divided into four phases:

In the first phase of pre-modeling, all the activities and sub-activities were comprehensible for the research team as well as well-structured in this phase. This phase set the foundation of the process and included activities such as defining the model's purpose, identifying stakeholders, providing training, and managing change and expectations. The descriptions of these activities were straightforward, making them easily understandable and actionable. This phase established the overall structure of the process and covered all the prerequisites required to initiate the remaining steps.

In the Domain Analysis and Scoping phase, the process focused on identifying and modeling features, as well as cross-tree constraints. The steps for feature identification, approval, and placement within the hierarchy were clearly explained, facilitating the understanding of how to effectively organise the feature model.

In the Modeling phase, the process introduced activities including model modularization, defining a coarse feature hierarchy, adding features, modeling constraints, and validating the feature model. These activities were explained in detail, outlining the necessary steps and considerations for each one. However, some of these activities, such as modeling constraints, might have required a deeper understanding of constraint modeling techniques to fully comprehend and apply them effectively. One of the optional activities in this phase was the "Define Views," which lacked any examples and made it difficult for us to comprehend as the purpose and application were unclear without an example.

The Maintenance and Evolution phase introduced activities that provided guidance on model version control, removing features, and optimising the hierarchy and constraints. These activities were described in a way that highlighted their importance and provided general recommendations. However, the specific implementation details and techniques for model version control and optimization might have required further exploration and expertise.

RQ1.D: How applicable are the individual parts of the process?

The individual parts of the process were highly applicable in the context of feature modeling. They provided a structured approach and practical guidance for carrying out the necessary activities throughout the feature-modeling process. However, some of the activities of the process were not entirely applicable to our case study.

The pre-modeling phase, which included planning and training stakeholders, defining the model's purpose, and managing change and expectations, played a crucial role in setting the foundation of the feature-modeling process. These activities were highly applicable as they helped establish a

clear understanding of the goals, stakeholders, and scope of the model. This phase ensured that the modeling effort was focused and aligned with the organisation's needs and that the stakeholders were prepared for their roles in the process.

The domain analysis and scoping phase, which included activities such as identifying features, were highly applicable to our case study. We followed the bottom-up approach for the modeling strategy. However, the activity of Bottom-Up Feature Identification was not applicable to our case study as our case study product didn't have any variants. We employed a different strategy of looking at the product UI and identifying features and then looking at the product codebase to identify features that were missed or not covered while UI identification. During this strategy, we were able to extract all the features for the product and validated these features afterward with one of the core company developers. This strategy to identify features for a product with non-existing variants is not mentioned in the process.

The modeling phase, which included activities such as model modularization, defining a feature hierarchy, adding features, and modeling constraints, was directly applicable to constructing the feature model itself. These activities provided practical steps for organising and representing the features, making them highly applicable to our case study as well. One of the optional activities in this phase, Define Views, was not applicable to our case study.

The maintenance and evolution phase, which included activities such as model version control, was applicable to our case study. We iteratively modeled the version control of the feature model from the top-down approach, and we improved it using a local git repository. However, some of the activities, such as removing features and optimizations, were not applicable to our case study as they would be done in future iterations and performed after a certain period.

RQ2: Extension of the Feature-Modeling Process

RQ2 How can the process be extended for reactive product-line adoption?

In order to adapt the process for reactive product-line adoption, where the studied product lacked variants, we made modifications to the bottom-up feature identification approach described in the process. Instead of considering existing variants or performing pairwise diffing among variants, we chose an alternative approach.

To gain a comprehensive understanding of the functionality of the process, we extracted features directly from the product's UI and documented them. This approach allowed us to improve our comprehension of the product and familiarise ourselves with the feature dependencies within it. Subsequently, we looked into the codebase of the product and identified additional features by examining the commit messages, stories, and product wiki within the repository. These resources provided us with valuable insights into the feature dependencies written within the product's codebase.

RQ2.A: What are the improvements and extensions that are missing in the process?

While reviewing the feature model process defined in the book, we identified potential extensions that could enhance its applicability. The extensions we proposed are as follows:

Extending the process model for one variant (i.e., reactive adoption): The process proposed in the book relied on the analysis of products with multiple variants and a base model. However, the product in our case study was a base model without any variants for analysis. This required an extension of the process to accommodate this specific scenario.

We propose to extend the process in the phase of domain analysis and scoping activities where we have the activity called bottom-up feature identification. The process can be extended and further divided into two activities: bottom-up feature identification with variants and bottom-up feature identification without variants. The process only covers the activity with multiple variants, and we provided the results based on the activity without variants. The major difference these activities had was that the product without variants couldn't be assessed or compared with multiple existing variants as described in the process. Therefore, we needed to create a different approach to provide the base model.

In our study, we identified the features from the product UI, which gave us an understanding of the product and feature dependencies across the product. Then we identified additional features from the codebase while using additional sources such as commit messages, user stories, and product wiki to identify the feature dependencies and feature implementation hierarchy at the codebase level. This approach gave us a better understanding of the product and provided features from both the user’s perspective and the technical perspective. Along with feature extraction, we held meetings with the product developers to gather feedback from them and enhance our understanding of the product. This approach proved beneficial in extracting features for a non-existing variant product as it did in this study.

Proposed activity: Incremental Analysis of Multiple Sources. If you have the product UI, you start interacting with it, documenting features where you, as a user, are getting any benefit from the product. Another approach is to create use cases from the user's perspective about the product and interact with it accordingly. Extracting features this way can be difficult if you are not familiar with the product. Once you have identified the features, you can document them individually for better traceability in the future. For feature documentation feature name, source, description, and feature type is recommended to be added in the documentation. To ensure high-level data protection, the documentation of the feature can be performed locally on the devices, or a protected shared drive can be used. This will give you a list of features to start with. Repeat the same process with all the pages of the product to ensure you don't miss any features on any page. Make sure to indicate for each feature if it is optional or mandatory, as it will become relevant when modeling the features. You can also add the source for each feature in the documentation to help identify feature dependencies later. One optional activity you can perform is interacting with all the pages using different screen sizes, as this might reveal additional features for some of the pages.

Once you have extracted all the features from the UI, you can move to the codebase of the product. If you are not experienced with the frameworks used in the product, it will be useful to do some research and understand the basic syntax of the framework before looking into the code. When identifying features from the code, start by separating the code for similar pages and interact with one page at a time. Your familiarity with the product gained through UI interaction will be beneficial in tracing the right code snippets in the repository. Once you have identified the relevant code

component, you will notice that the coding files are populated with multiple methods. At this point, focus mainly on the logical part of the code rather than the visual representation of the HTML files. The logical part will be mainly in languages such as JavaScript, TypeScript, Java, Python, or C#, although this can vary depending on the product. You will find some parts of the implementation that you are looking for, and to proceed further, go to the definition of the methods and perform a detailed analysis of the feature hierarchy. When you have identified part of the feature in the code, it is beneficial to look at the commit messages to find any feature dependencies from the codebase perspective. User stories in the product repository can also be helpful in identifying feature dependencies on different components, providing you with a hierarchy of how the features are implemented. If you have identified all the features from the codebase, it is recommended to document them as well, including writing the source and indicating if they are optional or mandatory for traceability and modeling purposes in the future.

Once all the features are identified, you can visit the product wiki to see the right naming convention for the features, or you can conduct a detailed meeting with one of the core developers of the product to finalise the feature naming in collaboration. This process will provide you with all the features of the product, ensuring accurate naming and feature dependency. One important aspect of UI interaction and codebase analysis is that you may identify some of the same features again, which is fine, but you need to merge the similar features at the end of feature identification to ensure that you don't repeat the same feature again. However, always look for more logical and technical features in the code compared to the UI, where more user-beneficial features will be extracted. You can identify feature relationships from the UI by interacting with the feature itself. If you can toggle one of the features by interacting with the other feature, then there is a parent-child relationship. To identify feature relationships from the codebase it is recommended to look for recursive programing, which is when a method is called within another method. This will help you identify parent-child relationships at the codebase level. You can also gather feature groups from the code base by looking at conditional rendering at the codebase level. Some of the examples of conditional rendering can be switch statements If statements or ternary operators. You can also identify feature groups for features while UI interaction by looking at possible toggle gestures that can be enabled or disabled by the user.

Improving validation and data sources in bottom-up feature identification: One of the phases in the process was domain analysis and scoping, which involved extracting relevant information about features and their relationships to guide the subsequent modeling phase. The objective of this phase was to collect and document information, particularly a comprehensive list of features and their relationships, in a manner that would support the modeling activities.

In this phase, two approaches were defined for feature extraction: the top-down approach and the bottom-up approach. However, the bottom-up approach lacked detailed information on how to extract features from the system. It primarily emphasised the use of a tool to differentiate and locate features by analysing the existing system and its variants. As a result, there was room for improvement in the process by providing more details about validating the features and about different data sources to consider.

Proposed improvement: Parallel validation in bottom-up feature identification. At the time when you are extracting the features in the bottom-up feature extraction activity, you will validate the features parallelly at the time of extraction. You will validate the features from both the code of the product and the UI of the product. You will be able to minimise the use of resources with the help of parallel validation and you will not be spending time on unnecessary components in the UI or the codebase. You can do parallel validation for the UI components by examining if the component under consideration fulfils any specific requirement for the user you consider it as a feature, if the UI component only represent something visually you can skip that component. In the codebase if you find a code snippet that provides any technical logic or something useful for the user you can consider it as a feature otherwise you may skip the code if it is only for visualisation.

Proposed improvement: Multiple data sources in bottom-up feature identification. When extracting features from the codebase during the bottom-up feature extraction activity, you will utilize multiple additional sources such as commit messages, the product wiki, and user stories to extract the desired features. Through the use of these additional data sources, you will be able to gain insights into the feature dependencies and their constraints. Commit messages will provide information about the hierarchy in which the features are implemented, making it easier for you to extract feature dependencies. The product wiki will assist you in naming the features, as it serves as the primary source of documentation for the product description. By examining the user stories, you can identify the hierarchy of feature dependencies across multiple pages of the product.

Extending the process model for combining the feature models: During the domain analysis and scoping phase, we extracted features and established their relevant relationships based on the given instructions. This phase was further divided into two feature-identifying approaches: top-down and bottom-up. These approaches were implemented separately in two different case studies, but the process did not provide any guidance on how to merge the feature models obtained from the bottom-up and top-down approaches throughout the four phases of the process.

Furthermore, the process did not provide any guidance on resolving conflicts that might arise from changes in the feature models or determining the prioritisation of these changes. Additionally, it remained unclear whether updates to the feature model should be performed sequentially or iteratively. Below is the activity to add at the end of the modeling phase.

Proposed activity: Combining top-down and bottom-up feature models. Once you have the final versions of the top-down and bottom-up feature models, you can start by combining them iteratively. You should version the feature model and add the top-down feature model as the first version. It is recommended to start with the top-down model first because the bottom-up model tends to be more detailed, reducing the chances of missing any features when starting with the top-down approach.

Start by identifying the common and different features between the two feature models to update the feature model accurately. Update the feature model one page at a time, or if the product consists of a single page, update it iteratively to minimise errors. With each update, append a new version for

future traceability. Only update the changes present in the two feature models and maintain the common features. You will have some conflicts between the two feature models which should be resolved with the help of product developers. Book a meeting with one or more product developers from the company and discuss the conflicts with them. One of the common conflicts you will face will be the naming of the features. The best practice to solve this conflict is to look over the product wiki page for any documentation about that specific feature. It is also possible that if you don't find that feature on the wiki page then the company developer can be helpful in this case to provide you with additional documentation if they have any. Conflicts about the features and feature groups should be discussed with the developer and analysed which feature model (top-down or bottom-up) has modeled the feature more accurately.

For each update, provide a description for future reference and for the company to understand the different versions of the feature model. Once all the features are combined, add the cross-tree constraints at the end of the feature model to make it more descriptive for viewers.

By following these steps, you can effectively combine the top-down and bottom-up feature models and create a comprehensive feature model with traceability and clear cross-tree constraints.

RQ2.B: How can the existing guidelines about feature information sources help model features?

We utilised a paper authored by Krüger et al. [11] as one of our primary sources to model the features of our product in the context of reactive product-line adoption, as described in RQ2. This paper provided valuable insights on how to identify and locate features and their facets in two open-source systems using a combination of static and dynamic analysis techniques. Their method enabled the extraction of both mandatory and optional features. By implementing the guidelines outlined in the paper [11], we successfully modeled features by extracting relevant information from the UI, codebase, and documentation sources within the repository.

V. Threats to Validity

A. Internal Validity

This thesis, being a case study and a collaboration with a company, had its own set of limitations and potential threats to validity. One potential threat to the validity of this study was the relatively small sample size of participants from the company. Due to the limited number of participants, there is a possibility that the results of the study might differ if more individuals had been included. However, it is important to note that although the sample size was not large, the few individuals who did participate in the company were highly relevant to the study and had undergone training in the relevant concepts during the initial phase of the process.

B. Transferability Validity

It is important to emphasise that a comprehensive understanding of the languages and frameworks utilised in the codebase is crucial for accurately evaluating the process. However, a potential threat to the validity of the study arises from our limited knowledge and expertise regarding the specific framework and technologies employed in the product. During the study, we had an iterative learning process to familiarize ourselves with the framework. We also sought the assistance of a developer to validate our work. It is worth noting that this iterative learning process may have influenced the results of the study. Nevertheless, it is important to highlight that while our expertise in the specific framework was limited, we possessed previous experience with different frameworks, which contributed to our understanding of software development processes in general. Additionally, to ensure the accuracy and reliability of the results, the work was validated by a developer from the company at the conclusion of the study.

C. Construction Validity

One significant threat to the study was the possibility of missing some feature extractions from the product. The company developers were primarily responsible for considering and providing input on the extracted features. However, there was no specific mechanism in place to identify any missed features that may have been overlooked by the study which can influence the results of this study. To address this threat, the research team employed two different approaches to extract features. By utilising multiple methods, they aimed to ensure that no features were missed during the extraction process. Additionally, to further validate the extracted features, the team took input from one of the core developers of the product from the company. This step helped to verify the accuracy and completeness of the extracted features.

Additionally, certain activities in the process, such as "Define Views," were perceived as ambiguous and required a deeper understanding of the subject matter. This ambiguity could have affected the participants' motivation and engagement, as well as the accuracy and effectiveness of the outcomes. It is important to acknowledge that clearer explanations or additional support in understanding these activities could have mitigated this issue.

D. External Validity

Another potential threat to the validity of this study was the lack of motivation among the individuals from the company who participated in the process. Since the research team of the process was external to the company, there was a perceived lack of ownership and motivation from the company's employees. If the process had been conducted by individuals within the company itself, it is possible that there would have been greater motivation and engagement from the employees, leading to potentially different outcomes.

Both the top-down and bottom-up approaches defined in the book [4] complement each other in the process model, providing a comprehensive perspective on feature modeling. These approaches do not contradict each other but rather offer different viewpoints and strategies for creating feature models. By combining these approaches, the process model aims to capture both the overarching commonalities and the fine-grained variabilities, ensuring a holistic representation of the software product line. This integration of approaches contributes to a more robust and thorough feature modeling process. Additionally, both strategies defined in the book can be performed independently.

These potential threats to validity should be considered when considering the findings and conclusions of this study.

VI. Conclusion

Software Product-Line Engineering involves identifying and managing commonalities and variabilities across related software, and developing reusable code that can be used to build products in the product line. Feature models are a critical artefact in SPL engineering and help in guiding the development of the product line. While feature models have gained popularity, there has been a noticeable lack of concrete guidelines on their practical implementation. However, this issue has been addressed by the recent publication of a comprehensive book [4] written by Andrzej Wasowski and Thorsten Berger which proposed a feature model process with a concrete set of guidelines. Despite the availability of the book [4], there was still a need to validate the proposed feature model process in a realistic case study. Our study fills this gap by conducting the first comprehensive evaluation of the process and assessing its applicability within a real-world context, focusing on a Swedish company's product. We demonstrate the evaluation and the extension of the feature modeling process through a case study.

In this study, we have thoroughly evaluated the process by performing the process activities described in the 4 phases of the process in the book [4]. To analyse and identify features, we adopted a bottom-up approach, as prescribed in the book [4]. This involved identifying features from multiple sources, including the product code base, user interface, wiki documentation, commit messages, and other relevant resources. By extensively documenting these features, we were able to develop a detailed feature model that accurately represented the software product's functionality and capabilities.

In conclusion of the results, the process described in [4] was highly applicable for modeling features in a real-world industrial case study. The research team successfully implemented the process, overcoming challenges and extending it for reactive feature identification. External dependencies, such as team size and unfamiliar technologies, affected certain activities but the incremental nature of the process mitigated their impact. The process provided a comprehensive overview and step-by-step guide for feature modeling, with each phase serving a distinct purpose. However, some activities like removing features, optimization, and Define Views were not applicable to the case study. The team followed the process by modifying the bottom-up feature identification approach and extracting features from the product's UI and codebase. For improvements, an incremental analysis of multiple sources, including the UI and codebase, could enhance the process. Overall, the process offers a comprehensive and iterative approach for modeling features in real-world software systems, ensuring applicability and usefulness in an industrial context.

To ensure the validity of our findings, we sought feedback from one of the product developers and conducted an iterative process to improve the feature model based on their insights. This collaborative validation and refinement process enhanced the accuracy and completeness of the feature model, making it more robust and representative of the actual software product.

Our research findings not only contribute to the understanding of the software product line engineering process but also have practical implications. The validated feature model can serve as a solid foundation for extending the feature set of the software product and can be leveraged in future research endeavors about feature modeling.

Our research results will contribute to the applicability of the process in future evaluations. We have identified concrete improvements for the process activities and addressed any potential gaps within its phases that hinder its applicability to a particular product. Furthermore, our research has introduced process extensions to address any potential lack of activities in the current process and help overcome any blockers for upcoming iterations of process evaluation.

References

[1] Frank J. Van der Linden, Klaus Schmid, and Eelco Rommes. Software product lines in action: the best industrial practice in product line engineering. Springer Science & Business Media, 2007.

[2] Thorsten Berger, Jan-Philipp Steghöfer, Tewfik Ziadi, Jacques Robin, and Jabier Martinez. "The state of adoption and the challenges of systematic variability management in industry." Empirical Software Engineering 25 (2020): 1755-1797.

[3] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-oriented domain analysis (FODA) feasibility study. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

[4] Andrzej Wąsowski, and Thorsten Berger. "Domain-Specific Languages: Effective modeling, automation, and reuse." (2023).

[5] Jacob Krüger, Wardah Mahmood, and Thorsten Berger. "Promote-pl: a round-trip engineering process model for adopting and evolving product lines." In Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A, pp. 1-12. 2020.

[6] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. "Automated analysis of feature models 20 years later: A literature review." Information systems 35, no. 6 (2010): 615-636.

[7] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. "Formalizing cardinality‐based feature models and their specialization." Software process: Improvement and practice 10, no. 1 (2005): 7-29.

[8] Damir Nešić, Jacob Krüger, Ștefan Stănciulescu, and Thorsten Berger. "Principles of feature modeling." In Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp. 62-73. 2019.

[9] Daniela Rabiser, Herbert Prähofer, Paul Grünbacher, Michael Petruzelka, Klaus Eder, Florian Angerer, Mario Kromoser, and Andreas Grimmer. "Multi-purpose, multi-level feature modeling of large-scale industrial software systems." Software & Systems Modeling 17 (2018): 913-938.

[10] Jihen Maâzoun, Mariem Mefteh, Nadia Bouassida, and Mouna Belmabrouk. "A bottom-up approach for feature model extraction from business process models." In International Conference on Intelligent Systems Design and Applications, pp. 1209-1218. Cham: Springer International Publishing, 2020.

[11] Jacob Krüger, Mukelabai Mukelabai, Wanzi Gu, Hui Shen, Regina Hebig, and Thorsten Berger. "Where is my feature and what is it about? a case study on recovering feature facets." Journal of Systems and Software 152 (2019): 239-253.

[12] Jacob Krüger, Thorsten Berger, and Thomas Leich. "Features and how to find them: a survey of manual feature location." Software Engineering for Variability Intensive Systems (2019): 153-172.