GUIDED REQUIREMENTS ENGINEERING USING FEATURE ORIENTED SOFTWARE MODELING

Author: Anjali Sreekumar

A dissertation submitted to the faculty at the Open University of Catalonia in partial fulfilment of the requirements for doctoral degree in Network and Information Technologies

UNIVERSITAT OBERTA DE CATALUNYA

May 15, 2022

Contents

List of Figures

List of Tables

Abstract

Ensuring well-defined software requirements yields better results in terms of the software project success and ease of maintenance. Any software requirement specification is said to be well-defined if it uniquely describes a functionality or part of a functionality of the software product, it is consistent with the definitions and descriptions of the other functionalities in the product and it is not redundant. Furthermore, in the real world, most software products do not exist in isolation: they belong to a family of related products sharing common *features* among them. Such families are called *software product lines*.

Engineering a software product line is a complex process. Gathering high quality software requirements and documenting them meticulously is a cumbersome task. It may involve multiple sessions and discussions with various stakeholders of the product and the project development teams. Moreover, consolidating all the outcomes from these meeting minutes and structuring them into identifiable, distinct and measurable software product requirements is also a complex process. Several types of human errors can be introduced and they might even escape repeated peer review exercises. Such errors are proven to cost heavily on the software project success.

Natural language is one of the main challenges in this process. During the project initiation phase, the business analysts and the technology specialists work towards defining clear targets for the software product and possible implementation approaches guided by the organizations process documents and technology stack. The information collected during this time will primarily be in the form of large volumes of textual information spread across multiple mostly unstructured documents. Then, a requirements/business analyst goes through all this content and translates them into software requirements specifications set to the requirements templates used by the organization. The most critical issue is making sense out of such large text corpus. To begin with, there is little guidance for ensuring the correct interpretation of the text-based on the context of the discussions during the meetings. Moreover, there is also no way to check the correspondence between the final requirements and the source documents. This leads to a variety of new problems like misunderstood requirements, mixed up term definitions, incomplete glossary, missed requirements, orphaned requirements, ...

This thesis focuses on techniques and tools for the management of textual documentation in the engineering of a software product line. The contributions of this thesis are threefold. First, we have proposed a method for extracting a *Feature Model* (a formal graphical description of a software product line) from a set of textual documents. This method has been implemented in a tool called FeatureX, taking advantage of the Natural Language Toolkit (NLTK), a popular natural language processing library. Second, we present a method for analyzing and visualizing the correspondence between a Feature Model and a set of textual documents. This approach can help us identify which parts of a Feature Model are supported by the existing documents, which elements from the documentation are missing from Feature Model and the degree of confidence of each element in the Feature Model. Finally, we have studied the use of formal methods for the analysis of the extracted FMs. To this end, we have compared different approaches for encoding FMs using the Alloy language, a popular formal notation that offers powerful analysis. Overall, this thesis aims to support the exploitation of natural language documents in the context of software product lines, providing mechanisms to extract Feature Models and check their quality.

Acknowledgements

This research is guided and supervised by:

- Dr. Robert Clarisó Viladrosa has a BSc in Computer Science from UPC-Barcelona Tech (2000). He has a PhD in Computer Science from UPC-Barcelona Tech (2005).
- Dr. Elena Planas Hortal has a BSc in Computer Science from UPC-Barcelona Tech (2007). She has a PhD in Computer Science from UPC-Barcelona Tech (2013). At present she is the Academic Director of the Computer Engineering Degree at Universitat Oberta de Catalunya (UOC).

There are no words that could describe my true feelings of gratitude and thankfulness for the continued support and guidance provided by my wonderful thesis supervisors, who not only accommodated me through this journey but also remained as the foundational support system that kept encouraging me to ideate in the right direction.

This work is partially funded by the H2020 ECSEL Joint Undertaking Project "MegaM@Rt2: MegaModelling at Runtime" (737494) and the Spanish Ministry of Economy and Competitivity through the project "Open Data for All: an API-based infrastructure for exploiting online data sources" (TIN2016-75944-R).

I am forever indebted to my family for their constant support throughout the challenging times. I could not have completed this work without the wholehearted support of my husband *Sarath*, who not only took care of our little darling daughter *Keya* but was always there to shoulder my efforts during these trying times of COVID. I thank all my friends who had been waiting eagerly to celebrate the successful completion of my thesis work.

Chapter 1

Introduction

Today in the twenty-first century, almost every thing in our world uses software. A software is very different from hardware in the sense that it is not a tangible object. Software is always associated with a computer which is a device that can store and process data. A computer software is a collection of logical instructions encoded in a format understood by the computer's hardware. Such instructions are executed by the computer and an output or result is generated. This output can be perceived either physically in the form of a light bulb turning on or it can be just computational as in simulating artificial human brain neurons. Due to this ease of access to fast and accurate computing abilities, software has evolved into a necessary area of study and an engineering discipline.

According to the Institute of Electrical and Electronics Engineers (IEEE) Systems and software engineering vocabulary, *the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software is called Software Engineering.* It is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the software system after it has gone into use [175]. In software engineering, software requirements are the description of product functionalities and services that a software system must provide under a set of constraints [175]. The process of establishing the set of requirements for a software system is called *Requirements Engineering (RE)*.

Requirements engineering is the first stage of envisioning the to-be created software system and thus it becomes the cornerstone for the development, management and extension of the software system in the future. The **requirements engineering** phase involves unearthing the complete collection of requirements from stakeholders who are interested in the software system. Requirements can be defined as the purpose for which a software system must come into existence: the expected behavior of the system and its interaction with the users in order to solve a business problem [175]. The need for making the requirements engineering phase as complete and error free as possible is an open problem in the RE domain [194] with no single solution which is better than the others. Di Vito et. al. [194] have identified the need for formalizing requirements as early as in the 1990's, mentioning the below problems of not having formal requirements specification:

- Lack of structured requirements analysis methodology
- Inability to decision the completion criteria
  - for evaluating the thoroughness and
  - for coverage of the requirements
- Inability to reuse the assets generated from requirements analysis
- Lack of support for effective requirements change management

There have been several research efforts since then, to explore the efficient use of formal methods in the field of requirements analysis with promising results [17, 35, 59, 74, 77, 95, 151, 191, 211] that gave way to establishing engineering disciplines like requirements engineering, computational linguistics and information science. As the world of digital data started exploding, the need for faster data analysis became more prominent. Due to this, the study of the application of formal methods in information analysis became diversified and segmented. Various research publications like [48, 116, 148, 172, 207] have evaluated the trends in the application of existing and upcoming machine learning techniques, including the use of blockchain technology on formalizing requirements specifications. It is observed that the diversification of information analysis, propelled the creation of advanced techniques which found its application in cross functional domains including software requirements specification and analysis. Requirements engineering is a time consuming and manual effort. The tasks performed in this phase could potentially benefit from having some level of re-usability. There were three strategies recognized from research publications related to efficient and effective approaches for identifying requirements during the RE phase: goal-based [149], scenario-based [215] and feature-based [63]. In goal-oriented requirements engineering, a goal model graphically represents relationships between the required goals (functional requirements), tasks (realizations of goals), and optional goals (non-functional properties) involved in designing a system. It may however, be impossible to find a design that fulfills all required goals and all optional goals. A scenario is a set of situations of common characteristics that might reasonably occur in the use of a system. However, for a complicated software system, there may be a great number of scenarios. Synthesizing a requirements definition from a set of scenarios and managing the consistency and the completeness of such scenarios can be very difficult. Finally, Feature Models take the best from both the above approaches and make the problem space more specific. A Feature Model (FM) recognizes the most important functionality/goals of the software, combining it with the most common scenario in which it is used. This approach thereby comes with an inherent indication of functionality prioritization, defines the specific constraints for its usability and decomposes the requirements engineering problem into multiple smaller problems each limited to a feature. Figure 1.1 shows some information regarding the number of research papers published for each of the above strategies over the past three decades. Even though the data collected is not meant to be exhaustive, it can be used to observe research trends over the years. To gather this data we have systematically analyzed the research publications published in the IEEE RE Conference, an important scientific venue in the RE research community. The search criteria used to assign each paper to the mentioned categories were: Publication Title, Keywords and Abstract. As can be seen from the trends in the plot, Feature-based RE started relatively smaller than the other two methods, but had been doubling ever since. Since goal-based RE also partially relates to Feature Models, a growing trend has been observed for it too.

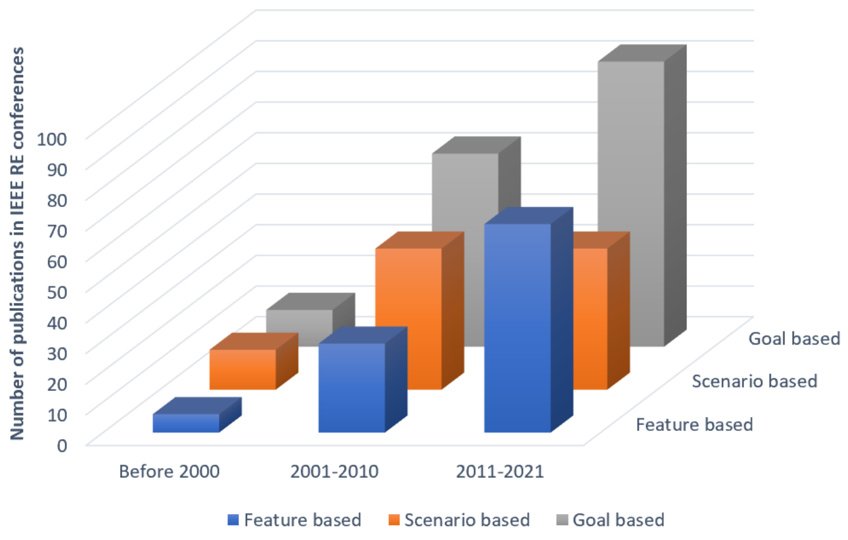


Figure 1.1: Analysis of research trends in Software Requirements engineering.

After surveying the existing state-of-the-art research on FM-based RE, **Feature Oriented Design and Analysis (FODA)** [101] was found to be the most promising approach to be investigated and explored further for efficient requirements engineering.

For making the RE phase more efficient, it was evident that we needed to explore avenues for automating at least some parts of the RE process. From the existing open research questions in the field of RE, we chose to work in depth on the capabilities of automating the process of establishing a complete and correct set of software product features and constraints using Feature-based requirements engineering. Therefore, this thesis provides a detailed account of the research work that went towards building an integrated solution for automation in RE that considers:

- The automated extraction of a Feature Model from the original informal requirements (specified in Natural Language).
- The semi-automated validation of the extracted Feature Model against the text specifications in order to assess its quality and detect potential defects as well as to guide the extraction process to improve the quality of the generated FM.
- Using existing FM analysis techniques for formal verification of the model that can be supported by the previous validation results.

The contributions of this thesis is built upon existing work in **Feature Model Extraction**, **Feature Model Validation** and **Feature Model Analysis**, and describe a novel and innovative method for the automation of the same. The methods and tools developed as part of this work was then evaluated against various case studies for validating its effectiveness. The progressive research outcome has been published in the following 3 international conferences where top research in this area are discussed:

1. 7th International Workshop on Formal Methods and Analysis in Software Product Line Engineering (FMSPLE at ETAPS 2016) [179]
2. 22nd International Systems and Software Product Line Conference (SPLC 2018) [177]
3. 15th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS 2021) [178]

1.1 Natural Language Processing

Natural language processing (NLP) refers to any kind of programmatic manipulation on language used by humans to communicate in their daily lives [40]. NLP techniques range from simple mechanisms for counting the word frequency in a given text to algorithms for automatically extracting the facts and concepts described in a complex paragraph. The later would require intelligent contextual processing of textual data similar to how human brain processes messy communication and is still able to make sense of the information thus communicated. This is one of the main open problems in NLP research which has, in recent times, started finding reasonable solutions with the application of deep learning [100, 111, 127]. In this thesis, we focus on the application of machine learning techniques attempting to solve for classification problems in the natural language context. The majority of the work revolves around the use of regression algorithms applied for supervised learning. Regression is a machine learning technique used for predictive modeling where the algorithm is used to generate the predicted outcome.

Although NLP emerged in the 1950s, it is still a very active research field and it is employed in a wide variety of application domains, from spam filters to intelligent personal assistants. In the software engineering domain, NLP has been used to gather information from requirements documents, specifications and stakeholder feedback in a (semi)automatic way [11, 60, 136, 174, 205]. One particular area of interest has been the description of *Software Product Lines* (SPL), i.e. families of related software products sharing a set of features.

1.2 Software Product Line

Software requirements that can be reused in the design and implementation of several closely related software systems are called **Software Product Line (SPL)** [19] specifications. The use of assembly lines in the automobile industry for mass production of automobiles based on a single main design inspired its comparison to software requirements reusability in facilitating diverse software product development [42, 109] from a single base product template. Thus originated the concept of product line engineering where similar products from a software domain can be engineered from one base model called the Feature Model.

In product line engineering, domain analysis consists of identifying commonalities and variabilities (differences) among the members of a family of products [42]. Products in an SPL differ from each other in terms of the presence and absence of certain product features. For example, consider a product line of mobile phones where we are specifically interested only in the products which have a high definition camera. Assuming that we have identified 2 such products, these will not be exactly the same products. It might happen that one also comes with additional SD cards. This essentially means that a product is nothing but a configuration of the product line specifications. One way to reason about these assets is to describe them in terms of features, which are domain abstractions relevant to stakeholders and are typically increments in program functionality. These features are organised hierarchically in Feature Models (FMs). Feature Models are a means to specify product commonalities and variabilities in a compact way, and to provide automated support to the domain analysis process [1]. With this understanding, we went ahead to create a reliable framework for requirements engineering with an automated tool support which enables the domain expert to keep adding new requirements and editing the existing ones without worrying about the consistency and correctness of the overall requirements specifications. In the coming sections we will elaborate on the motivation behind this research and the research question that drove this effort.

1.3 Motivation

RE is a crucial phase in the software development process. Some studies [39] conclude that the success or failure of a software project highly depends on the success of this activity. RE has to deal with several challenges related to the format in which the requirements are documented. For example, if the requirements have been written down in the form of English language text, dealing with potentially unclear or ambiguous sentences, ensuring the completeness of the set of sentences or paragraphs, detecting inconsistencies etc. becomes very difficult to track and maintain manually. This is why it is necessary to have tools and automated support for managing textual information in order to be able to reliably use the software requirements specifications for guiding the development of the software.

The RE activity is divided into several phases: *elicitation, analysis, specification and management.* The elicitation phase involves unearthing the requirements from the stakeholders. Typical approaches for requirements elicitation includes discussions, interviews, focus groups, literature survey, and document analysis [175]. Usually the proceeds of these activities are documented in natural language text. Such specifications tend to be ambiguous and, at some point, formal notations needs to be introduced in order to eliminate ambiguity and allow the *analysis* of the *specification.* When formalism increases, the specifications become discrete non-redundant definitions of various product features that could exist in one or more products created from these definitions. These feature definitions that describe various software product features pertaining to a domain is called the *Software Product Line (SPL)* [19] specification. Such specifications become easy to maintain as the process for *managing* them can now be automated. We will now look at some of the approaches, proposed and discussed in literature for automating this requirements engineering process.

Feature Oriented Domain Analysis (FODA) [101] aids in the creation and validation of SPL, which is based on the notion of a *"feature"*. A feature is an increment in software product functionality that can be used to specify and distinguish products in product lines. Each feature communicates product functions in an easy-to-understand way by capturing functionalities concisely, and helps distinguish the commonalities and variability of a domain [27]. Features are organized hierarchically in diagrams called *Feature Models (FM)*. Feature Model aim to represent the constraints under which features occur together in a product configurations [63], e.g. a feature may require or depend on other features. A literature survey of variability modelling showed that FMs are by far the most frequently reported notation in the software industry [33] for capturing software product feature requirements and their relationships. However, without automatic support, the quality of the resulting FMs heavily depends on the domain expert's personal experience, knowledge and understanding [54], who is responsible for building out the FM. This is not very reliable and will not be a concrete representation of the features and its relationships. This becomes very obvious when we task a set of domain engineers to create an FM for a specific domain using the same set of textual documentations. When we compared their created FM's it was evident that no two FMs were the same and hardly had anything in common. There has been several research works attempting to find a solution for minimizing these differences and standardize the FM generation process [12, 13, 20, 21, 44, 63, 73, 82, 90, 105, 106, 120, 128, 130, 137, 145, 154, 177, 192, 200]. But its use and adaptation in the software industry was very minimal. This was certainly attributed to the lack of automated tool support which can provide guidance to the domain engineers for using some of these approaches. Providing automated tools support for guiding the Feature Model creation and maintenance in a reliable and effective way irrespective of the input types (i.e. textual documents, meeting minutes, discussion summary etc.) is a challenging problem. The problem itself is very interesting in the sense that it has several sub-problems. The expected ideal situation would be to have a tool:

1. that could automatically identify the different features and its relationships which can then be reviewed and corrected by the domain engineer.
2. that would automatically reorganize the FM when a new feature is introduced or an existing feature is modified, in order to ensure that the FM is still consistent and complete without introducing any conflicting or dead features.
3. that would allow the domain engineer to detect potential issues with the requirements specifications and provide guidance on the corrective actions.
4. that would generate an FM which is accepted by all domain engineers and can account for any existing product configuration from that domain.

The opportunity for applying various kinds of techniques for devising a solution for this problem was the driving factor that motivated us to pursue this research. The ability to standardize and have a mechanism for continually improving the quality of Feature Models is a rewarding capability which can potentially impact the software industry at large and can drastically improve the speed at which we create software products and solutions today. Ensuring unbiased automated identification of product features and their relationships will normalize the domain engineering efforts and will make Feature Models (FMs) as the standard industrial format for a Software Product Line (SPL). Given the complexity and cost of requirement engineering and the potential consequences of having wrong/incomplete requirements [39], the results of this thesis are relevant to any organization developing complex software belonging to diverse domains.

This thesis work includes an elaborate and detailed study of all existing work in the field of automated FM extraction and analysis. Based on the findings and shortcomings identified during the literature survey, the thesis aims to bridge the gaps in the practical use of FMs in RE. As part of that, an integrated framework for FM extraction and validation is presented with results of its evaluation conducted on various software product domains.

1.4 Research Methodology

The primary objective of this research work is to improve the performance of software requirements engineering process in industrial practice. For this, it was essential to follow a methodology that can systematically explore each contributing research objective and have the ability to evaluate it for its practical application. This has led to the creation of a novel and innovative FODA-based RE approach as this research outcome. This approach is supported with a fully implemented and tested design model that was evaluated and proven to show evident improvement in the practice of software requirements engineering process. Therefore this research work has used the *Design Science Research* [195] paradigm for finding the solutions to the identified research questions. Design Science Research (DSR) is a problem-solving paradigm that seeks to enhance human knowledge via the creation of artifacts

[195] by following a process that simultaneously generates knowledge about the method used for designing the artefact and the artefact itself. It seeks to enhance technology and science knowledge bases via the creation of innovative artifacts that solve problems and improve the environment in which they are instantiated [195].

For this research work, rather than working on a well known open research question, we performed exploratory studies to piece in the concrete definitions of the problem statement relevant to the research subject. The idea behind guided RE process is to identify and eliminate fundamental specification defects early on which could cost heavily at a later stage in the software development life-cycle. But the main problem here is that all the artifacts available during the early phases of a software business case analysis, are mostly unstructured textual data. Processing this data requires it to be converted into a structured format on which data processing and analysis algorithms can be applied. We took this as the starting point of this research work and selected Alloy [93] as the requirements specification language. The very reason for selecting *Alloy* over other popular specification languages like *Z notation*, and Object Constraint Language (OCL) is that, Alloy uses first-order logic and can be analysed automatically. From that point on, a new sub-problem is defined and solved every time, as part of which various techniques were used such as prototyping, case study evaluation and refining the research scope. This led to incremental development of this research where the problem definition and its best possible solution, obtained within the constraints of the techniques applied, were constantly evaluated and validated at every stage of the research progress.

During the course of this work we have consistently revised the problem definition and its scope based on the feedback obtained from the implementation and practical assessment of the various parts of our proposed solution. This has heavily contributed towards the quality of the overall solution design. It not only helped with guiding the research direction but also unearthed new opportunities for the application of this research outcome. We have used the following activities of the DSR process [195] in an iterative loop to build the knowledge base and document the experiments and evaluation results:

1. problem identification and motivation which has led to a problem-centered solution initiation
2. define objectives of a candidate solution
3. design and development of the candidate solution
4. evaluation and feedback communication

In the next section we will describe the research questions in detail and relate them back to the problem definition and scope.

1.5 Research Questions

This thesis explores the automation potential of the requirements engineering process in the software development life-cycle. The vision is to be able to improve the efficiency of RE in practice. The major hurdle in this vision is attributed to the very nature of the RE phase which is manual, procedural documentation of the requirements behind a software that varies from company-to-company and RE practitioners. This lack of consistency and the non-existence of standard methods, leads to unclear, ambiguous and incomplete feature requirements of the software product. Industrial survey on software engineering projects have shown that 80-90% of software projects fail due to software defects that went undetected during the RE phase [39, 194]. Ensuring concise and concrete product requirements specifications with well-defined constraints and dependencies can curtail the large financial losses that software projects can cause to the sponsoring companies [39]. Innovations and improvements in the RE phase for enhancing its efficiency will also contribute to faster software development iterations with stable quality and low defects. Therefore the key research questions (**RQ**) we aim to answer with this work are as follows:

RQ 1 : *What tools are available for automatic analysis of feature models generated from requirements specifications and how can it be used for encoding and analysing various specification data?*

RQ 2 : *What Natural language processing (NLP)-based heuristics can be used for extracting features and relationships from textual specifications?*

RQ 3 : *How can the reliability of the automated FM extraction results be validated, analyzed and quantified?*

RQ 4 : *What are the evaluation results from the application of automated FM extraction, validation and analysis on various case studies?*

To achieve the research objective and for finding the answers to the research questions, we have systematically broken down the problem statement into 4 main goals which are: implementing automated FM extraction using NLP, validating FM against the source text specification using NLP and machine learning algorithms, performing FM analysis using Alloy to identify various defects in the specifications and finally, applying and validating the effectiveness of this solution on various case studies.

Most of the initial literature survey resulted in the identification of the gaps that existed in the current solutions. This helped us to evaluate our research motivation against these findings and thus formulate the methodology that we would apply for this research work. The thesis covers details about the various investigations on Feature-based RE, since it was our chosen method for finding a feasible solution.

This research work aims to target all the RE practitioners in the software engineering field who actively contribute towards the development of business case and analysis of requirements specifications for supporting the project prioritization and resourcing efforts. In today's times where agile processes and extreme programming rule the software development arena, it is ever so important to have an engineering process that supports changing requirements, business priorities and re-platforming needs. The thesis explains the overall approach behind the automated RE framework and thoroughly investigates its impact on real industrial grade RE effort, by collecting feedback from practitioners on its ease of use and benefits.

1.6 Summary

The thesis attempts to investigate the larger problem of finding solutions for the inefficiencies detected in the requirements engineering phase of software engineering. The systematic approach towards conducting this research helped in the evaluation of the identified problem statement and validating its relevance, in not just the research community but mainly for the industry practitioners. It was found that feature oriented design and analysis (FODA) is the best fit approach for modeling the solution for automated RE framework. Based on FODA, Feature Models (FM) are the most used visual notations for product line specifications in the field of software product line engineering. Generating an FM is an arduous task in itself and having a capability to automate it and thus provide tool support for the domain engineers will help improve the efficiency of the various processes involved in the requirements engineering phase. Thus a combination of **automated** *Feature Model extraction* and **semi-automated** *Feature Model validation and analysis* leads to a framework that propels the requirements engineering phase by guiding the Business analyst or practitioners with a constructive feedback. Such feedback is capable of directing the analyst to update the baseline requirements in order to accommodate either a missing product feature or include constraints that will make these requirements consistent and correct. This is a semi-automatic feedback loop, wherein the feedback will directly translate into a semiautomatic update to the *Feature Model* and the baseline requirements from which the features are being extracted. This way it is possible to ensure a significantly minimal role for an Analyst's intuition to be responsible for determining the correctness and completeness of the gathered requirements. Thus establishing an automatic guidance in place during the requirements engineering process. We have evaluated this framework against real world product requirement specifications in order to check its efficiency and reliability over the conventional requirements engineering practice. This solution would thus become the first of its kind which demonstrates a single framework for automatically analyzing the requirements at an early stage of RE phase in order to help mitigate errors, as the errors made at this stage could cost exponentially [39] if discovered at later stages of the software development life cycle.

After the literature survey, we understood that there are issues that stem from the problem of not having an automated, robust and reliable approach for requirements engineering. Though there were several research works focusing on finding a common framework that could work for software development across multiple product domains, there wasn't a concrete implementation for any one of them. Also there were no studies that compared these research findings and evaluated them against each other. Thus in this thesis we survey all major works in

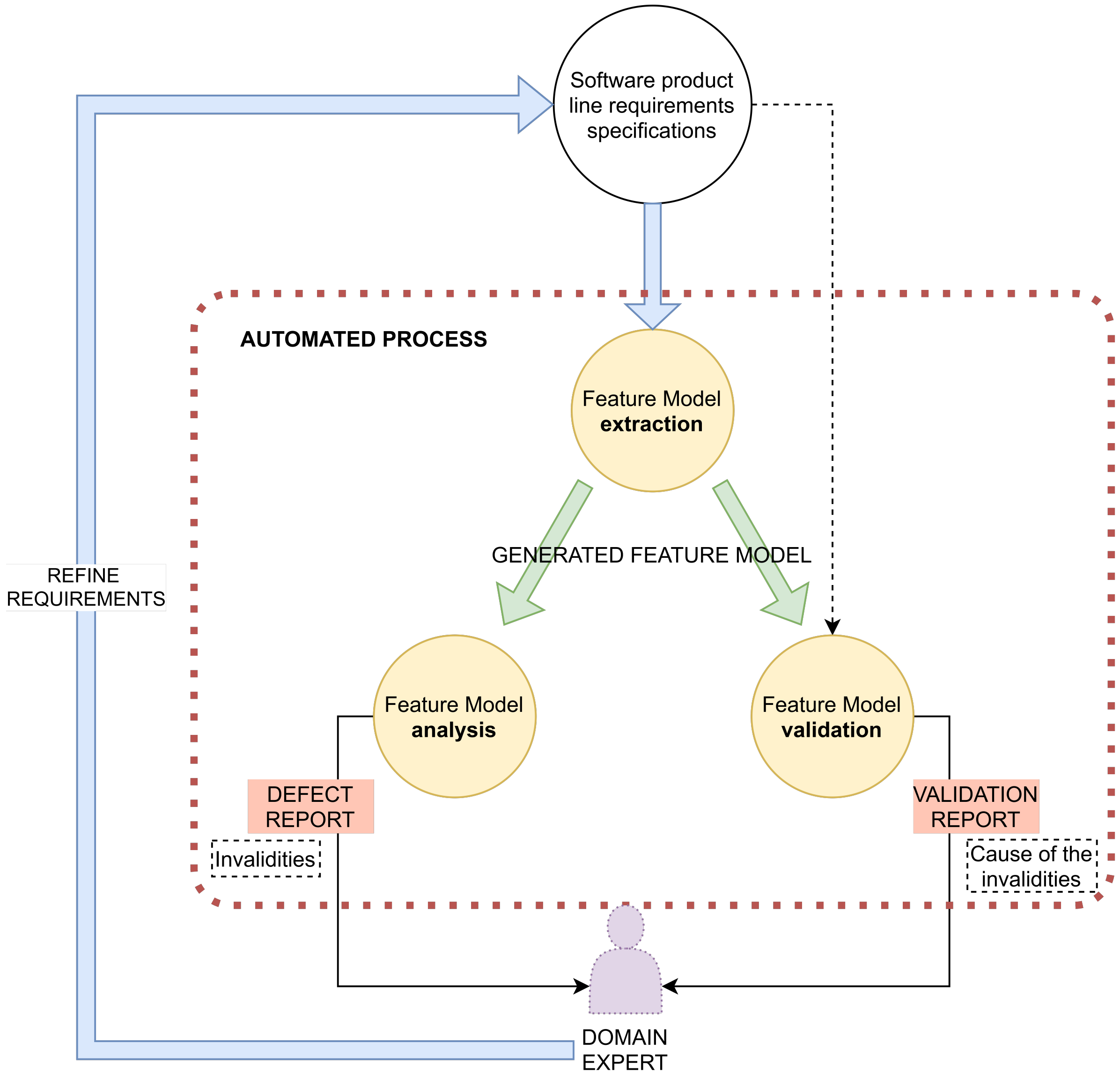

Software product line requirements specifications
AUTOMATED PROCESS
Feature Model **extraction**
GENERATED FEATURE MODEL
Feature Model **analysis**
Feature Model **validation**
DEFECT REPORT
Invalidities
VALIDATION REPORT
Cause of the invalidities
REFINE REQUIREMENTS
DOMAIN EXPERT

Figure 1.2: Guided requirements engineering.

the field of requirements engineering and Feature Models, identify specific research contributions which can be extended further and thereafter create a concrete implementation of an RE process utilizing FMs.

The main contributions of this thesis can be defined in terms of the following research goals:

Goal 1 : An elaborate and detailed literature review of all works related to the application of product line engineering tools and techniques on requirements engineering.

Goal 2 : An automated procedure for the extraction of a FM from a collection of unstructured informal text specifications.

Goal 3 : A semi-automatic approach for the validation of the extracted FM against the text specification using statistical machine learning techniques.

Goal 4 : A semi-automatic formal analysis of FM capable of identifying defects in the FM and provide feedback information that can guide Goal 2 to produce a better FM. This goal will take advantage of existing tools and solvers such as Alloy for evaluating the validation results obtained from Goal 3 on various case studies.

Figure 1.2 demonstrates the outcome of this research contribution. This thesis introduces a framework for guiding the requirements engineering process in the context of Software Product Lines. This framework includes an automated technique for extraction, validation and analysis of FMs so that it can be run in an iterative manner and reevaluated after every single change made to the specifications in order to check for the overall consistency of the feature requirements for the SPL. To this end, an automatic technique for feature extraction from informal requirements is implemented, so that the extracted FMs can be formally specified and automatically analyzed against the quality criteria such as inconsistencies, incompleteness etc. This together with the validation of FM with respect to the information present in the source documents can provide constructive feedback to the feature extraction mechanism [63]. This guided requirements engineering technique is focused towards gradually improving the quality of the extracted Feature Models and thus improve the quality of the software requirements specifications. The major challenge in this automated feedback mechanism was the non-trivial nature of refining software requirements based on the results of FM analysis and validation which involved the contributions of a domain expert.

This thesis is structured into five main chapters. We begin with a detailed account of all the literature surveyed as part of this research work in Chapter 2. In Chapters 3,4 and 5 we provide an explanation on the 3 processes involved in the new guided RE framework (refer Figure 1.2), which includes the discussions around existing research works, its limitations, identified solution approaches, comparison studies of these approaches and finally the implementation of the winning approach. We provide a summary of the main conclusions of the thesis and its impact in Chapter 6 and conclude by providing a list of future research directions in Chapter 7.

Chapter 2

State Of The Art

In the research findings [89] on deriving formal specification from informal requirements, it has been stated that, for ensuring reliability of software, which is an attribute of dependability of the software [204], the use of formal methods is inevitable. Formal methods are techniques based on mathematics and formal logic, used to model software systems as mathematical entities [94, 142]. From the survey results on the use of Formal methods in Requirements Engineering, by [204] it is seen that: *"Formal methods are used in specifying software, thereby developing a precise statement of what the software is to do, while avoiding constraints on how it is to be achieved. A complex specification may be decomposed into sub-specifications, each describing a sub-component of the software system"*. We have reviewed the current state-of-the-art on the use of formal methods in Feature Model based RE. Since the research goal is targeted towards creating an automated support for requirements engineering, it was essential to understand all the existing work related to the following components that supports this cause:

1. formalization in software requirements specifications
2. existing algorithms for translating requirements specifications to verifiable models
3. availability of tool support for model generation
4. validating requirements specification models
5. methods for analysis of specification models

Section 2.1 covers an introduction to the key concepts and vocabulary used in this thesis with an overview of the current research trends in the area of automated FM-based RE and the key developments and contributions which assisted this research effort. Section 2.2 elaborates on why feature-based models in product line engineering are effective for automated RE by providing references to existing work which have demonstrated promising results. Sections 2.2.1,2.2.2 and 2.2.3 covers all the findings from a systematic literature review and is categorized based on the extraction, validation and analysis of FMs, which are the identified components for supporting our vision of automated requirements engineering. Section 2.3 consolidates all the limitations identified for the different research results. And finally a summary of the major inferences from the literature review is provided in Section 2.4.

2.1 Literature review

According to a recent research publication which consolidates all the findings from a survey on the formalisation of system requirements and their validation [143] on real industrial projects, it is observed that the application of formal methods or notations for requirements specifications is still challenging. This is mainly because of the lack of technology readiness level of the tools that help support it. A similar study analyzing the application of SPL in industry in the last 20 years around the world [18], have found that the most reported problem area is in the adoption of SPL in practice. This means that, the current popular process of documenting requirements in the form of natural language text is still considered to be the most convenient for a large majority of the industry practitioners of RE. Therefore it is even more important to explore existing algorithms and tools that can support conversion of textual specifications to formal mathematical notations and models for ensuring unique, unambiguous and consistent requirements specifications. Among all the various research on formalization in RE, the application of feature oriented design and technique for generating verifiable models of the software product in terms of the product features had several contributions with promising results. This was a direct demonstration of converting structured or unstructured textual information about product features into Feature Models that can be analysed and verified. It was observed that such Feature Models were prominent in the field of product line engineering with software product lines at the core of this research topic.

The growing popularity of Software product lines (SPL) in the industrial space calls for more adaptable and convenient tools which could aid in the practice of product line engineering. One of the main reasons for the success of SPL, is re-usability. There are various research results which have explained the details of re-engineering software products from code. There are also a few other research works which have profound significance in reverse engineering of software products belonging to a product domain. But only a handful of research works have looked into extracting product features from the product descriptions of domain specific software. The core of SPL revolves around generating reference Feature Models (FM) which will assist in engineering newer products in future for any domain specific product line. Therefore, SPL corresponds to feature oriented software development where Feature Model extraction, Feature Model validation and Feature Model analysis would together guide the process of setting the references for newer products in the product line. This in turn assists the requirements engineering phase for new products coming from the same product line. The feature extraction, validation and analysis phases help in creating sound and complete FM.

Manually extracting implicit variability information is tedious and error-prone. Hence, several works [5, 20, 130, 137, 145, 154, 192, 200, 209] have proposed automatic or semi-automatic extraction considering different types of information sources, *e.g.*, design diagrams, source code, informal product descriptions, etc. In this thesis we are only focusing on work that relates to extracting FMs from documents written in *natural language*. Several approaches have been proposed to address this problem [10, 43, 44, 54, 73, 97, 105, 106, 128]. Bakar et al. [21] have systematically studied the various research works on Feature Model extraction approaches from natural language requirements. Arrabito et al. [15] have compared three NLP analysis tools used both in industry and research to compare their relative performance in detecting ambiguity and under-specification in software requirements documents. They have also performed a similar comparison study of NLP tools in the RE domain [14] which can extract product variation points. The effectiveness of each approach can be measured in terms of its ability to detect a high rate of meaningful features and relationships, while maintaining a low rate of false positives. However, it is hard to assess the relative merits of each approach for two reasons:

- There are only few papers that have provided a publicly available working implementation or prototype of the proposed method, or even sufficient details to reproduce it accurately.
- There is a lack of benchmarks that can be used to compare the quality of the outputs. As a result, papers are evaluated using different sources, and the inputs and/or the outputs are not publicly available.

Extracting variability data from product line specifications requires the use of text processing algorithms. Several open source Natural Language Processing (NLP) packages provide state-of-the-art methods for text analysis, but they are general-purpose tools that are not tailored to the requirements engineering domain. The problem of isolating key product features from requirements specifications and identifying their relationships is still a challenge in the FM-based requirements engineering (RE) research.

To start exploring automated model generation techniques from natural language text, a systematic review of the most relevant research work on the extraction of (a) only features, (b) only relationships and (c) both feature *and* feature relationships (the complete FM) from requirements documents was carried out. Before getting into the details of all the key findings, it is essential to understand about features and relationships. In a software system, a *feature* is a “user-visible aspect or characteristic of the domain” [101]. When considering a family of related software products, features may differ or be shared among the different products. A *Feature Model* (FM) is a visual hierarchical representation of these commonalities and variabilities.

Formally, a product line offers a set of features $\mathbb{F} = \{f_1, \ldots, f_m\}$. Products can be described as subsets of this set of features, although not all subsets are considered valid products. The set of valid products of a product line is described by its Feature Model. A Feature Model $\mathbb{FM}$ is a directed graph $\mathbb{FM} = (V, A)$ where the set of vertices is $V \in \mathbb{F}$, the set of features. Each arc $(x, y) \in A$ denotes a dependency between features x and y. Table 2.1 (left) lists the different types of one-to-one dependencies : mandatory, optional, requires or includes.

Moreover, sets of arcs starting from the same vertex, the *parent* feature, may describe more complex relationships with a set of *children* features. Table 2.1 (right) describes the potential one-to-many dependencies : conjunction, disjunction and alternative. These relationships define a tree with a *root* feature: the feature with no parents.

A subset $P \subseteq \mathbb{F}$ is a valid product according to a Feature Model $\mathbb{FM}$ if it includes the root feature and satisfies all dependencies among features in the FM:

One-to-one	One-to-many
[**M**] MANDATORY	[**A**] AND
[**O**] OPTIONAL	[**R**] OR
[**E**] EXCLUDES	[**X**] ALTERNATIVE/XOR
[**I**] REQUIRE/IMPLIES	

Table 2.1: Types of relationships among features.

1. mandatory(x, y): $x \in P$ implies that $y \in P$.
2. optional(x, y): $x \in P$ implies that either $y \in P$ or $y \notin P$.
3. requires(x, y): $x \in P$ implies that $y \in P$ and x != $parent(y)$.
4. excludes(x, y): $x \in P$ implies that $y \notin P$.
5. and(x, Y): $x \in P$ implies that $\forall y_i \in Y : y_i \in P$.
6. or(x, Y): $x \in P$ implies that $\exists y_i \in Y : y_i \in P$.
7. xor(x, Y): $x \in P$ implies that $(|Y \cap P| = 1)$.

Apart from these relationships, there are more complex feature relationships called cross-tree constraints. These indicate dependencies with multiple features described as a logical expression like propositional logic formula which are nothing but a notation with one or more conjunction, disjunction or negations. There are more recent research works [107] which demonstrate that any such cross-tree constraint in FM can be refactored to simple constraints that is covered by the named relationship types demonstrated in Table 2.1.

Keeping this knowledge about features and relationships in mind, we will proceed to go over the key findings consolidated from the literature review for feature extraction, feature relationships mining and automated FM model generation as described below.

Feature extraction Arora et al. [12] have developed an automated approach for extracting candidate glossary terms from requirements specifications documents. Even though glossary terms may include other concepts, they would also capture feature names. Similarly, Arora et al. [13] have evaluated the state-of-the-art in domain model extraction-based on ontological rules that have been proposed by various other research works. Two findings of their work indicate that: 90% of the extracted feature terms were either correct or partially correct; and from the set of identified relationships, only 36% were found relevant enough to appear in the Feature Model. Dumitru et al. [66] and Hamza et al. [82] have presented recommender systems that models and recommends product features for a given domain.

Relationship extraction Some works focus only on extracting feature relationships, considering the list of features as an input to their method. For instance, Yi et al. [209] proposed an approach for detecting *binary constraints* among features, which identifies only *requires* or *excludes* relationships among pairs of features. [161] is used for identifying cross-tree constraints among the extracted features. According to the authors, the tools are no longer maintained.

Feature Model extraction Itzik et al. [90] have considered two perspectives for the visualization of variability. These are *structural or objects-related* view and *functional or actions-related* view. This makes it clear that there can be different Feature Models created for a given SPL depending on what perspective a domain engineer is interested to see in order to make critical design decisions for the development of new products. They have studied the uses of Feature Models in software industry and depending on the various phases in the software development life cycle, the need for a different perspective of an FM is justified. Their method SOVA (*Semantic and Ontological Variability Analysis*) is supported with a tool [91] which is not publicly accessible. Hence, it was not possible to reproduce their results. For the evaluation of the framework created as a result of this thesis work, the FM studied in by Itzik et al. [90] has been used to compare our research results. Dumitru et al. [66] have used text mining and clustering algorithms for generating probabilistic Feature Models with feature relationships and cross tree constraints. Whereas Hamza et al. [82] have used text preprocessing and an NLP-based approach for identifying feature terms and constructing the FM. Weston et al. [201] have introduced a complete tool suite that can be used for extracting candidate FM from natural language text specifications. The tools ArborCraft (an Eclipse plugin) [5] is used for extracting Feature Models from text specifications and EA-Miner. Davril et al. [63] have proposed a clustering algorithm based approach for constructing FMs from publicly available product descriptions found in online product repositories and marketing websites. They have demonstrated the use of graph-based algorithms that can be applied on a directed weighted graphs of candidate feature terms which can then distinguish the feature relationship types based on extracted feature association rules. This approach also lacks publicly available tool support because of which the results in this paper could not be reproduced.

2.2 Feature-based requirements engineering

The concept of features and Feature Models (FMs) came along with the introduction of Feature Oriented Domain Analysis (FODA) [101]. According to Kang et al. [101] an application is a software system which provides a set of general services for solving some type of user problem. Using the FODA technique for a domain, the commonalities and variabilities (differences) of the problems that are addressed by the applications in the domain are analyzed in the domain-modeling phase and a number of models representing different aspects of the problem are produced.

From a requirements engineering perspective, a feature-based RE is an approach where FODA principles are used for the various RE processes. As we have mentioned previously, RE comprises mainly of the elicitation, analysis, specification and management processes. When applying feature-based RE the processes map into features and feature relationships extraction, FM analysis and FM validation which takes care of managing the specifications. Since FODA has proven to have produced promising research results when applied to software product line engineering, we have attempted to continue our research in the same direction. In the following subsections we will briefly describe the most relevant research results in the field of FM extraction, analysis and validation. The review findings are evaluated based on the following two criteria:

1. Known limitations and

2. Available tool support

2.2.1 Feature Model Extraction

The process of identifying features of a SPL from natural language specifications is called *feature extraction* [21, 120]. This computation requires considering the syntax (sentence structure) and/or the semantics (meaning) of the text. In other words, the process of generating a FM, is called FM extraction and can be divided into two activities: (1) feature identification/mining, i.e. identifying the list of relevant features; and (2) feature construction/modeling [1], i.e. providing a hierarchical classification of the features as a Feature Model where the dependencies and constraints among features are described in the form of a non-binary (multifurcating) tree. Such abstraction where informal requirements specifications documents needs to be read and processed into a concise diagrammatic representation can be done manually by a domain engineer who is well versed with multiple products belonging to a particular domain. But it is also well known that assimilating large volumes of text manually by a human could be error prone, ambiguous and inconsistent. This is one of the primary reasons why an automatic or a semiautomatic process could prove invaluable for continued developments in this area.

In order to find all relevant papers related to feature extraction from requirement specification documents using NLP techniques, the most widely used databases have been consider: IEEE Explore, SpringerLink, Association for Computing Machinery (ACM) Digital Library, Elsevier, Google Scholar and Digital Bibliography & Library Project (DBLP).

The initial search on all the databases was done based on the following search criteria:

Search terms:	(Product Features OR Feature Extraction) AND (Software Requirement OR Text Corpus OR NLP OR Natural Language)
Searched in:	Abstract OR Keyword OR Title

This resulted in thousands of papers from each database as shown in Table 2.2, including papers in different languages and from different research fields. A cause for the large number of matches was due to the term "feature", which is also used in the machine learning context with a different meaning.

Database	ACM	DBLP	Elsevier	Google Scholar	IEEE	SpringerLink
# of papers	4.787	4.052	5.362	12.700	5.689	10.660

Table 2.2: Search results

We further filtered the results based on the year of publication from the year 1990 until 2021 and excluded all the papers which were related to:

1. Machine learning and artificial intelligence
2. Natural language processing of languages other than English
3. Technical details of NLP algorithms
4. Application of NLP outside software engineering, e.g. in the medical field

We included only those papers which had contributed towards extracting features or Feature Models or feature relationships from software requirements specifications in the context of SPL which were documented in the English language. At the end of this selection process, there were a total of 27 papers, which were considered in this survey. A summary of this comparative evaluation is provided in Table 2.3. The table 2.3 illustrates the overall survey observations based on the following differentiating factors which are shown under each column:

1. *Information source*: Extraction of Feature Models can be done either from informal requirements, or business process models or even product line variability matrices. For this reason we have categorized and organized the different literature against their respective information source i.e. the source of software requirements specifications. In a few other literature we also saw that the Feature Extraction process starts from

the feature tree representations of requirements. Mostly they are abstracted from domain specific feature collections which are in line with SPL variabilities. These kind of diverse information sources that includes even use case diagrams or architectural diagrams have been classified under the *Others* category.

2. *Extraction strategy*: Based on the information source, the way by which the various features are extracted will also differ. In some cases the informal requirements are collected and represented in terms of graphs and constraints, from which the extraction of various features becomes more or less very intuitive. Whereas in other cases, a semi-formal algorithm is generated to extract the feature described by the requirements specifications. Frequency profiling is an Information Retrieval (IR)-based extraction technique based on the frequency of occurrence of words in the specification text. Finally, Variability modeling is an extraction technique where the features are identified and modeled as a variability of an existing Domain specific feature, thereby highlighting the measure of extensibility of the SPL features.

3. *Extraction results*: The result of the extraction is always a Feature Model. But the representation of these Feature Models varies among different proposals. Most commonly used visual notation of a Feature Model is called a Feature Diagram. This includes the *Hierarchical Feature Models*. Other representations include Feature Algebra and notations based on various types of grammars and propositional formulas [197]. For this reason we have classified the results under two categories: (1) Feature diagrams and (2) Feature Algebra which includes notations based on formal semantics.

4. *Tool support*: This is a classification based on the available tool support for the automatic or semi-automatic Feature Model extraction. Some tools are available in public domain and some are non-public, with a few that exists only as a prototype. For the rest either there are no mention of any automatic tool support or the reference explicitly mentions that there is no automatic support for it.

Before we get into the details of the literature review, it is essential to familiarize with the various terms and keywords used in the natural language processing context.

NLP Glossary

NLP considers a variety of problems regarding the understanding and manipulation of natural language text. Some of the goals of NLP research are the following:

- *parsing*, using parser programs to detect the structure and components of a sentence such as subject and object connecting a verb, [45],
- *keyword generation*, detecting high frequency words which could be candidates to describe the main topic discussed in the text [165]
- *keyword neighborhood*, detecting related keywords before and after the identified keyword [96]
- *named-entity recognition*, identifying structured information such as a date, a place or location [75],
- *co-reference resolution*, a process of detecting expressions referencing the same entity in a text [113],
- *dependency analysis*, identifying relationships between multiple contextual data in such a way that the sequence of information is significant in making the text meaningful,
- *information extraction*, a generalization of named-entity recognition [176],
- *sentiment analysis*, used to automatically understand the tone of the text to classify it as a positive sentiment or negative sentiment [99],
- *machine translation*, translating text from one language into another without changing its meaning [52], and
- *question answering*, replying to queries posed by humans using natural language.

Each of these goals are extremely complex due to the inherent characteristics of natural language, i.e. large vocabulary, flexible sentence structure, ambiguity, meaning depending on context, and others. For this reason, NLP methods are typically decomposed into a set of simpler tasks or resources that are required in order to solve the

Table 2.3: State of the art in Feature-Based Requirements Engineering.

Feature Model Extraction	Information Source			Extraction Strategy				Extraction Results		Tool Support
	Business Process Model	Informal Specification or text-based	Others	Graphs & constraints or algorithms	Requirements clustering or matrix-based	Frequency profiling	Variability modeling	Feature diagram	Feature algebra	Available
[147]			✓				✓	✓		Prototype
[41, 169]			✓	✓					✓	Prototype
[81, 135]		✓					✓		✓	Prototype
[83]		✓		✓				✓		Public
[125]			✓	✓					✓	No tool
[125, 140]		✓			✓				✓	Prototype
[122]	✓		✓		✓				✓	Prototype
[54]		✓			✓		✓		✓	Prototype
[44]		✓				✓			✓	Proprietary
[85]			✓	✓					✓	Prototype
[55]		✓					✓		✓	Prototype
[189]			✓				✓		✓	Public
[203]			✓	✓			✓	✓		No tool
[204]		✓					✓		✓	Proprietary
[7]			✓	✓			✓	✓		Public
[49, 118]		✓	✓	✓			✓		✓	No tool
[140]			✓				✓	✓		No tool
[214]		✓			✓			✓		Prototype
[63]		✓		✓				✓		Prototype

Table 2.4: State of the Art in Feature Model Analysis.

Automated Feature Analysis	Analysis properties			Type of input FM		Analysis Results			Tool Support
	Consistency	Variability analysis	Others	Feature diagrams	Feature algebra	Feedback	Counter example	Others	Availability and type
[80]	✓				✓	✓			Public, Alloy-based
[61]	✓		✓		✓	✓			Public, FAMA (FeAture Model Analyser)
[79, 201]	✓				✓	✓			Prototype, Alloy-based
[64]	✓		✓	✓		✓			Public, Alloy-based
[155]	✓		✓		✓		✓		Public, SAT solvers
[149]			✓	✓		✓			Public, Aspect oriented
[22]	✓			✓		✓			Public, SAT solvers
[23]	✓		✓		✓	✓			Prototype, Generative programming
[26]	✓				✓	✓			Public, Semantic markup language
[33, 185, 188, 190, 194, 197, 206, 210, 215]		✓	✓		✓	✓		✓	Prototype, Constraint Satisfaction Problems (CSPs)
[24]	✓		✓		✓		✓		Prototype, SAT solvers
[29, 30, 31, 203]		✓			✓			✓	Public, Propositional logic
[62, 133]		✓			✓			✓	Unknown
[84, 89]	✓			✓		✓			Prototype, Model checking
[101]	✓			✓				✓	Unknown
[81]		✓			✓	✓			Public, Common variability language
[1, 122, 134, 181]	✓		✓		✓			✓	Prototype, Theorem proving

complete problem. Table 2.5 provides a glossary of NLP terms, considering only those that are used in the context of the analysis of software product line specifications. This glossary will be used when comparing the different approaches. There are several NLP packages and tools providing the implementation of typical analysis tasks. Some examples are Apache OpenNLP[1], Cyc [131], Natural Language Toolkit (NLTK)[2] [40], The Stanford Natural Language Processing tools [3] and WordNet [4]. Many research works have been build upon these tools rather than developing their own infrastructure from scratch.

Most of the selected research papers have used various NLP techniques on different kinds of input data in order to extract useful information that contributes towards the respective RE goal. We will now present the details of the review by discussing the techniques being employed, the results, the shortcomings or limitations. In order to better understand the research trends, we have classified the findings based on five aspects of NLP in RE related research which are:

1. *Domain Agnostic?* Can the approach be applied to any SPL domain?
2. *Input:* What is the required format for the input? (e.g. structured text, tagged text, etc.)
3. *Methods:* Which standard NLP techniques have been used in the proposed method?
4. *Tool support:* Is there an automated tool implementing the method? Is it publicly available?
5. *Output:* What is the final output of the extraction process?

With reference to Table 2.6 the research works have been divided into five categories based on their input and output format. The first category of papers (T1) discusses the extraction of Feature Models from requirements documents. Category (T2) discusses extraction of list of *abstractions* from requirements specifications. Category (T3) is a collection of research works which are focused on extracting *keywords, feature names, list of commonalities* and *list of variability* from natural language requirements documents. The final categories (T4) and (T5) contains papers which discuss how to extract of abstracted features and business rules from glossary terms and other input file formats such as XML documents using NLP techniques.

FM extraction research has explored many different techniques from a diversity of fields: *Natural Language Processing* (NLP); *Information Retrieval* (IR) and *Machine Learning* (ML). The goal of previous works on feature extraction has been the derivation of a business vocabulary, domain model or FM from the textual requirements [12, 13, 20, 44, 63, 73, 82, 90, 105, 106, 128, 130, 137, 145, 154, 177, 192, 200].

To some extent, language lexical analyzers help a domain engineer make decisions and extract individual features and construct a FM [44]. Palmer and Liang [125] have presented an approach for automatic requirements analysis, which can be reused for feature extraction. The first part of this approach is based on indexing requirements. For this, they used the frequency method from the field of Information Retrieval (IR). This method was applied on the thesaurus of verbs, which could be used as classification criteria for requirements.

Regarding the application of ML for the detection of Feature Models, the use of supervised learning based on patterns of POS tags is significant. This strategy is related to [121] proposed by Yang et al. , where POS patterns are one of the attributes considered by supervised learning algorithms. However, this can be used with different supervised learning method (logistic regression) and could require less information from domain experts to build the dataset for training. There are also techniques driven by the computation of confidence scores to obtain efficiency when working with rather large textual specifications. For this reason, more computationally expensive techniques such as *named entity recognition* [40], *noun phrase chunking* [40] or methods such as *word embeddings* [98] or the use of domain vocabularies or ontologies have been considered.

The feature mining technique proposed by Davril et al. [63] includes several steps: mining raw feature descriptors, pre-processing, feature formation and finally naming the feature. This approach showcases a promising extrac-

[1] https://opennlp.apache.org/
[2] http://www.nltk.org/
[3] http://nlp.stanford.edu/software/
[4] https://wordnet.princeton.edu/

Id	Technique	Description of the technique or the terms
G1	Anaphora resolution	Pronoun resolution, which resolves references to earlier or later items in the discourse.
G2	Canonization, tokenization and sentence splitting	Tokens (which can be word or sentence), used to split the text into simpler manageable units. Breaking down sentences into lexically simplest paraphrase is also known as canonization.
G3	Corpora	A body of text like medical journals, speech, or such similar information that makes meaningful content in English language.
G4	Clustering	Clustering algorithms group a set of documents into subsets or clusters. The algorithms' goal is to create clusters that are coherent internally, but clearly different from each other. In other words, documents within a cluster should be as similar as possible; and documents in one cluster should be as dissimilar as possible from documents in other clusters.
G5	Duplicate removal	Removing duplicate text regions using a technique known as shingling. It detects the presence of duplicates by using a set of term frequency factor and checks if two sentences or documents are near duplicates of one another.
G6	Extract lexicon from glossary	A lexicon is a collection of information about the words of a language and the lexical categories to which they belong. Such information can be extracted from glossary terms where the words and phrases from the document are organized based on the domain.
G7	Inter-chunk relation recognition	Discovering the dependency graph between chunks found in the sentences.
G8	Inverse document frequency	Numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.
G9	Lemmatization	Stronger version of stemming using of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.
G10	Lexicon	A dictionary containing words and their meanings.
G11	Named Entity Recognition	Also known as entity identification, entity chunking and entity extraction. Process of identifying structured information within the text.
G12	Part of speech (POS)	Differentiating various components in every sentence and marking them as identifiable parts of English language text such as verbs, nouns, adjectives etc.
G13	Recognition of Semantic Relations between Named Entities	Method for detecting a semantic relation between a given pair of named entities, which may be located in different sentences.
G14	Semantic analysis and frequency profiling	Method used to discover keywords in the corpora which differentiate one corpus from another.
G15	Semantic tagging	Process of identifying meaningful terms in the text-based on the context and detecting these terms in the various regions of the corpus.
G16	Slicing	Process of extracting chunks of text-based on different criteria.
G17	Stemming	Technique to remove meaningless differences between words.
G18	Stop word removal	Removing extremely common words which do not contribute to the analysis.
G19	Term frequency and inverted term frequency	An area in the text that mentions a term more often has more to do with that term and therefore should receive a higher score or weight based on this term frequency. Inverted term frequency simply assigns a weight 0 when the searched term is not found in the corpus.
G20	Word-sense disambiguation	Method for identifying which of the several meanings of a word is used in a particular sentence.

Table 2.5: NLP Glossary

Table 2.6: Summary of the literature review.

Tools Category	Input	Output	NLP techniques used	Domain agnostic?	Citation	Tools
T1	Requirement documents and textual assets	Feature Model	duplicate removal (G5), inverse document frequency (G8), parts of speech tagging (G12), semantic tagging (G15), stemming (G17), term frequency (G19)	YES	[63] [91] [202] [82]	SOVA (Semantic and Ontological Variability Analysis) – ARBORCRAFT [202] and EA-Miner tools FFRE, a prototype recommendation tool
T2	Requirement documents and textual assets	List of abstractions	tokenization (G2), Custom algorithm, statistical analysis, use of WMATRIX, parts of speech tagging (G12), semantic relations (G13), frequency profiling (G14)	YES	[112] [166] [6]	2 programs (finder and strainer) The REVERE toolset CIRCE
T3	Natural language text documents	Keywords or Feature names or list of commonality and variability	tokenization (G2), clustering (G4), duplicate removal (G5), inverse document frequency (G8), lemmatization (G9), named entity recognition (G11), parts of speech tagging (G12), frequency profiling (G14), semantic tagging (G15), stemming (G17), stopword removal (G18), term frequency (G19)	YES	[97] [54] [9] [43] [44] [73] [105] [106] [128] [130] [20] [200]	Back propagation algorithm – Prototype software tool [TextReq] – Tool-based requirements management Prototype tool used for evaluation of the approach Prototype automated tool FDDetector (Feature duplication detector) NAPLES is an approach with tool support Un-nlse (natural language based tool) Approach that uses a combination of existing tools Backpropagation algorithm
T4	Morpho-syntactically annotated corpus coming from Poliqarp and other documents in XML format called CCL which is supported by Corpus2	Abstracted features	anaphora resolution (G1), tokenization (G2), syntactic relations recognition (G7), semantic relations (G13), slicing (G16), word sense disambiguation (G20)	YES	[47]	Feature extraction framework Fextor
T5	Glossary and Requirements documents (SRS)	Semantics of Business Vocabulary and Business Rules (SBVR) instance model	tokenization (G2), lexicon extraction (G6), syntactic relations recognition (G7), semantic relations (G13), stemming (G17)	NO (PROMISE requirements corpus)	[170]	Prototype tool

tion model that does not yet come with any automatic tool support for undertaking any of its processing stages. Moreover the mining is only done on publicly visible and known informal requirement statements available for a specific domain and not on a domain with a more constrained set of product descriptions. Most of the industrial custom software solutions deal with managing complex constrained requirements which are difficult to be validated against the complete quality requirements. For such situations, the feature extraction model identified by the feature mining method will not be useful: the requirements elicitation process is dynamic and changing with time, and because of this there is no consistent data available for applying the extraction algorithm on the mined features.

Boutkova et al. [44] have introduced a lexical analyzer for decomposing text documents into single words, which will be further processed until valid potential feature candidates will be identified. This approach is a semi-automatic one where the user selects the specification text from which the feature needs to be identified and an algorithm generates all the list of features from it automatically. They have described a method for easing the transition from product descriptions expressed in a tabular format to FMs that accurately represents them. This process is parameterized through a dedicated language and high-level directives (e.g., products/features scoping).

Haslinger et al. [84] have presented an algorithm to automatically extract a FM from a set of valid feature combinations. An approach proposed by Chen et al. [54] constructs FMs by requirements clustering. This approach has been successful to a very great extent in partially automating feature identification, feature organization and variability modeling of features based on the functional requirements specification of any software application. This is mainly because of its ability to construct multiple application feature trees, based on the various features explored for different sample applications and then creating a domain feature tree that will be a good representation of all the features pertaining to the studied application domain. The basis of this approach is the requirements relationship graph, from which the clustering is studied and features are identified. One of the limitations of this approach is that it is also being applied to data dominant domains and still expects functional requirements to be set in place before attempting to execute the clustering and extraction algorithms.

The papers [63, 90, 201] start from a requirements document specified in natural language text and produce a Feature Model as output. They have attempted to use basic POS tagging and semantic tagging applied together with term frequency technique to filter out probable feature names which thereafter aid in constructing the Feature Model. The tool ArborCraft claims to be capable of automatically extracting possible feature names, provided the complete text specifications is fed as the input to the tool.

The papers [6, 112, 166] have detailed about specific algorithms which internally use various NLP techniques such as canonization, tokenization, POS and semantic analysis for collecting different abstraction terms together with the detailed meanings of these terms. The research findings in these papers also present a sample of toolsets and programs that can be used to run the proposed algorithms on sample text input which would generate the lists of abstractions.

Arellano et al. [9] have published a work-in-progress paper that describes a new approach for the interpretation, organization and management of textual requirements through the use of application-specific ontologies and NLP. They have supported their findings with a prototype evaluation on a simplified model of an aircraft specification. Broda et al. [47] have discussed a complete feature extraction framework known as Fextor which uses very specialized form of text input and generates abstracted features. The specialized input is an annotated form of textual documents and other XML formatted content which is partially processed and refined by NLP tools.

Selway et at. [170] propose a deep natural language understanding approach to create complete and precise formal models from glossary terms and SRS (Software requirements specifications) documents. In their approach a feedback is provided to the user, which can be used to allow manual refinement of specifications into a precise and unambiguous form. The paper describes how natural language requirements are processed into models ingrained with semantic business rules and uses business vocabulary. The paper [6] describes a web-based environment which is used for aiding the requirements elicitation process. It is used for extracting abstract information from the specification text which is then used to create various models.

The papers [82, 104, 159] discuss feature extraction in a theoretical perspective and did not produce any automated or semi-automated tools with this purpose. For this reason we have classified them under research contributions targeting development of frameworks for feature extraction using NLP techniques.

The paper by Kevin et al. [159] is the oldest work among the collected papers. This work explicitly talks about the stages in RE where realistically NLP techniques can be used. The paper indicates that the technical specifications of the core product requirements can be formally structured which can be validated with the use of automated formal analysis on the processed output that results after applying NLP techniques. The paper has also emphasized on the importance of questions like *why*, *what-if* and *why-not* to be used for validating such formal requirements specifications extracted from the natural language text.

Kastrati et al. [104] demonstrate the use of structured queries to extract information from random text documents. They demonstrate the use of the NLP techniques *keyword neighborhood*, *named-entity recognition* and *dependency analysis* to query for specific kind of information. They had projected a future research direction where the query can be optimized and using multiple NLP techniques to arrive at a more precise information extraction. Hamza et al. [82] have demonstrated a prototype recommendation tool which will aid in recommending features and their relationships from the textual requirements specifications based on natural language processing techniques and heuristics.

Approaches such as the feature recommender system proposed by Dumitru et al. [66] require the input documents to follow a very constrained format, e.g. a list of bulleted product specifications, which makes it hard to be extended to arbitrary documents. Such recommender systems can become an integral part of feedback mechanism in FM extractor and visualization tools for the domain engineer who might be adjusting the generated FM to fit a particular perspective. The major limitation with this approach is its incapability to support larger requirements specifications document rather than just bullet points of product specifications. Even research works with tool support like Weston et al. [201] and Davril et al. [63] failed to provide sufficient details for future researchers to explore and extend their framework.

The method used by Itzik et al. [90] leads to creating larger Feature Models which would contain information similar to use case diagrams and class diagrams which are very granular in definition and would explode the FM solution space for larger software systems. Moreover in the engineering process of an SPL, a domain engineer would depend on other reusable software implementation artifacts like class diagrams and software architecture document to make critical decisions on the domain implementation details.

Apart from the above papers we also found the following papers valuable in terms of their contribution towards classifying the major research directions and approaches towards processing natural language text for generating meaningful software abstractions which can then be further analyzed. Acher et al. [1] takes software product descriptions as the input and converts them into Feature Models with a readable tree hierarchy that includes variability information. Their approach does not include the use of any NLP techniques. Stoiber et al. [180] discusses about transforming graphical software requirement models into Feature Models for a product line. ADORA (Analysis and Description of Requirements and Architecture) has its own language and tool that is used for this type of Feature Model generation and has been evaluated on specific graphical requirements models. Collobert et al. [58] have proposed a unified neural network architecture and learning algorithm that can be applied to various natural language processing tasks. This work claims to use very effective tagging algorithms for text corpora, such that it utilizes minimum computational resources. It also enables identification of valid feature keywords required for the purpose of feature extraction. Bogdanova demonstrates an algorithm which will automatically extract such features on any given textual content using various NLP techniques.Greenwood et al. [80] introduce a tool suite that is able to manage elements of variability among requirements specifications by automatic extraction of elements of variability (like features of an FM), enriching variability model, model adaption, configuration verification & validation and model synchronization to maintain consistency of the FM when changes are introduced.

One of the main steps involved in automated feature extraction is the identification of named-entities in any unstructured text. This is then processed using parsing and semantic analysis for more accurate feature extraction. Natural language toolkit (NLTK) is an open source library in Python, which is used both in industries and scientific research for the analysis and processing of text data. The papers [20, 43, 44, 54, 73, 97, 105, 106, 128, 130, 200] have resulted in several automated and semi-automated tools and approaches which will aid in the feature extraction goal. The list of existing tools offering feature extraction from natural language text is shown in Table 2.6. None of these tools offer a fully automated solution for feature extraction. All the tools have used different NLP techniques in various combinations significant to each of their extraction approaches. This survey has identified that there are commercially licensed software available for feature extraction like the one distributed by *Agilent Technologies*. The presence of such tools in the market highlights the use of complex tools in practice.

	Evaluation Criteria	Citations
METHOD	**Pattern matching**	[12, 73, 82, 90, 105, 137, 154, 192]
	Machine learning	[12, 43, 54, 63, 97, 105, 200, 209]
	NLP[†]	[5, 12, 13, 20, 44, 63, 73, 82, 90, 105, 106, 128, 130, 137, 145, 154, 192, 200, 201]
	Information retrieval	[5, 154]
GOAL	**Only features**[†]	[12, 20, 43, 44, 73, 97, 130, 145, 154, 192, 200]
	Only relationships	[209]
	Features and relationships	[5, 13, 54, 63, 82, 90, 105, 106, 128, 137, 201]
INPUT	**Text document**[†]	[12, 13, 43, 44, 82, 90, 105, 106, 130, 137, 192, 200, 201, 209]
	Words list	[13, 97]
	Document collection	[5, 54, 63, 73, 128, 145, 154]
RESULT	**Automated**	[12, 63, 73, 90, 105, 106, 128, 137, 145, 192, 201, 209]
	Semi-automated	[20, 44, 54, 130, 154]
	Prototype/Approach/Framework[†]	[5, 13, 20, 43, 63, 73, 82, 90, 97, 130, 154, 192, 200, 209]
	Tool support	[12, 73, 105, 106, 128, 145, 201]
LIMITATION	**Tool not available for evaluation**[†]	[5, 13, 20, 43, 44, 54, 63, 73, 82, 90, 97, 105, 106, 128, 130, 137, 154, 192, 200, 201, 209]
	Restricted inputs	[13, 20, 63, 97]
	Automated feature naming	[5, 44, 63, 90, 106, 137, 145, 192]
	Results not reproducible[†]	[5, 13, 20, 43, 44, 54, 63, 73, 82, 90, 97, 105, 106, 128, 130, 137, 154, 192, 200, 201, 209]
	Interaction with domain engineer	[82, 145, 201]

[†] The most prominent criteria in each category is highlighted in grey.

Table 2.7: Literature review summary.

Our literature review involved 27 different research works, which have been compared against each other and the summary is provided in Table 2.7. The review information is classified under 5 different categories, each having a set of evaluation criteria:

1. Technique used for FM extraction (METHOD) - Techniques like pattern recognition (including ontology-based), using machine learning algorithms, natural language processing techniques, information retrieval techniques, etc. Machine learning algorithms include clustering algorithms like *spherical k-means*, classifiers like *binary classifiers*, etc.

2. Goal of the proposed approach (GOAL) - Feature extraction, relationship mining or FM extraction.

3. Inputs to the approach or tool (INPUT) - Text documents, words list, unstructured collection of documents, etc.

4. Result or output of the research (RESULT) - Automated or semi-automated feature or FM extraction, availability of tool support, prototype of the proposed framework or approach, etc.

5. Presence of several limitations (LIMITATIONS) - Threats to validity, lack of real industrial case studies, unavailability of tool support, missing details of stated implementation approach etc.

2.2.2 Feature Model Validation

In this section, we briefly review the literature related to the key concepts and techniques used for the *validation of FMs* and the use of various *visualization* techniques. Once the FM has been extracted from the textual requirements, is it necessary to assess the level of concordance between this FM and the source documents. In this context, we are not considering the intrinsic quality of the FM, *e.g.*, measuring suitable metrics about its complexity or reusability [36]. Instead, we want to establish whether the FM is correct and complete with respect to the original requirements.

In the literature, this evaluation is typically performed manually, with the help of a domain expert who labels the extracted features as correct or incorrect and identifies missing features. Nevertheless, the evaluation efforts are considered insufficient or inadequate in most cases. According to a recent survey [120], out of 25 surveyed research works on FM extraction only 3 provided a reproducible evaluation using a sound methodology, with 13 providing a weak or non-existent evaluation. Moreover, in [21] a potential bias is identified which is that: in most works, the domain experts performing the evaluation are researchers or research students. Finally, due to non-available software tools and other reproducibility issues, there are no works attempting to compare different approaches in terms of relative accuracy and validity.

We aim to fill this gap by providing a visualization that can highlight the level of confidence offered by each element in the FM. Heatmaps are a popular visualization in the software domain [158]. Among other applications, they can be used to highlight coverage in software testing [72]; to measure code usage, performance or resource usage during code profiling; or to illustrate the impact of a software change [32]. In product line engineering, different types of visualizations have been used, *e.g.*, for the analysis and configuration of product lines [16, 152]. However, to the best of our knowledge, the proposed heatmap visualization is innovative both in its goal (validating FMs with respect to textual documents) and its use of the heatmap metaphor in the SPL context.

2.2.3 Feature Model Analysis

The automated analysis of FMs aims to extract consistency, correctness and completeness information from FMs using automated mechanisms [27]. In order to have automated mechanisms for FM analysis a semantics must be defined for FMs. These semantics are nothing but the use of mathematical logic to capture the Feature Model. This way the semantics is captured using Formal methods where the semantics is recorded in terms of propositional formulas. There are various criteria on which the FM analysis can be carried out and one of them is automatic consistency checking. Consistency is one aspect of the overall quality of a FM, which can be measured by (1) how adequately it captures a given domain and by (2) the integrity of the model itself with respect to the used modeling elements. While the adequate capture of the domain can only be analyzed and reviewed by domain experts, the integrity of the model can be determined by the occurrence of redundancies, anomalies and inconsistencies [135, 206, 213]. An adequate tool support is needed to manage features, their relationships and dependencies in order to guarantee the development of consistent models. Therefore the detection of redundancies and inconsistencies is a very important requirement on Feature Modeling tools. Benavides et al. [27] have surveyed the state-of-the-art on FM analysis and have consolidated the described set of analysis operations as shown in Table 2.8.

Void feature mode	Valid product
Valid partial configuration	All products
Number of products	Filter
Anomalies detection and Explanations	Corrective explanations
Feature Model relations	Optimization
Core features	Variant features
Atomic sets	Dependency analysis
Multi-step configuration	Other operations

Table 2.8: Analysis operations proposed by Benavides et al. [27].

Table 2.8 is an exhaustive list of all the analysis operations that have been covered for FM analysis by the various research works [27, 29, 30, 61, 64, 139, 146, 169, 188, 190, 213]. In the remainder of this section, we will discuss the automated support available for carrying out these analysis operations. Research work related to FM analysis is further classified based on the mathematical logic or language used for modeling and specifying the FM. The following classification has been extracted from the work of Benavides et al. [27].

i. **Description logic based analysis**
Wang et al. [198] proposed to use description logic reasoners for FM analysis. This proposal is based on the translation of FMs into an OWL DL (Domain Language) ontology. OWL DL is an expressive yet decidable sub language of OWL (Ontology Web Language) [26]. In that connection, it is possible to use automated tools such as RACER (Renamed ABox and Concept Expression Reasoner) for the automated analysis of FMs.

ii. **Constraint programming based analysis**
A Constraint Satisfaction Problem (CSP) is defined as a set of variables, each ranging on a finite domain, and a set of constraints restricting all the values that variables can take simultaneously. A solution to a CSP is an assignment of a value from its domain to every variable, in such a way that all constraints are satisfied simultaneously. The language used in CSP is more succinct than the one used in SAT or other propositional solvers because CSP solvers allow the use of numerical finite variables such as integers or sets, while for the use of those variables for representing the problem in SAT or BDD approaches, a translation is necessary [31]. Basic and extended FMs can be translated to CSPs [27].

iii. **Propositional logic and satisfiability (SAT) solvers**
Propositional-based Feature analysis [61] is the most widely used analysis method [22, 23, 24, 185]. It uses different solvers to represent and analyze the FMs. There are several research works [24, 27, 61, 64, 78, 79, 139, 190, 203] that propose the translation of basic FMs into propositional formulas. Batory [22] proposes the use of SAT solvers. Zhang et al. [214] propose the use of SVM system. The SVM [122] system is a system *"for checking finite systems against specification in temporal logic"*, however the proposal does not use temporal logic but propositional logic. Sun et al. [182] propose the use of Alloy Analyzer [64] that internally uses a SAT solver to check model satisfiability [61].

The summarized presentation of all the various research results on Feature Model Analysis has been illustrated in Table 2.4. The survey results in the Table 2.4 on Feature Model Analysis have been categorized based on the following criteria:

1. *Analysis properties*: Checking for consistency and measuring the variability is the most common analysis operations performed by a majority of the research. There were a few specific literature that explicitly talked about validity of feature configurations and optimization. They have been included under the *Others* category.

2. *Type of input Feature Model*: We have used the same category as from the Feature extraction survey table. Here the *Feature diagrams* also include state transition diagrams and other software design diagrams which demonstrate the dependencies and relationships between various features of the product. The *Feature algebra* includes Annotative feature representations [197] and the general formal propositional formulas and other semantic notations.

3. *Analysis results*: The analysis results are mostly used for evaluating one of the correctness or completeness properties of the Feature Models. A very few number of research works deal with providing an automatic feedback mechanism to improve the Feature Model so as to validate for good consistency and correctness measures. This has been recorded under *Feedback loop for refining requirements*. Most of the literature deals with measuring the performance of the analysis methodology, and do a comparison of analysis tools based on this performance results. These type of analysis results have been recorded under the category *Others*. A few other literature talks about analyzing the Feature Models and providing counter examples that invalidate the analysis operation. They do not give further explanations as to how the Feature Model can be correctly validated against the chosen analysis operation without generating a counter example. This has been recorded under *Generation of counter example scenarios*.

4. *Tool support*: This too is similar to the Feature extraction survey (see Table 2.3) table except that this also includes an additional information regarding the type of tool that has been used for analysis.

2.3 Limitations

The good news is that there has been several progressive effort towards studying and evaluating the feasibility of introducing automated mechanisms for the various RE processes. The bad news is that even after a decade of research on this field there are only a handful of results which are close to being adapted by the industrial practitioners. These are mainly attributed to the limitations identified in the research results. Below we summarize the key limitations and justify our motivation towards pursuing this research and taking a step further towards addressing at least some of them.

2.3.1 Lack of scalability

Since most of the research work explored the different possibilities for applying various information extraction methods on natural language text specifications, it was imperative to have used both simple and complex NLP techniques that are difficult to scale and perform well on larger text corpus. The enterprise software systems requirements from an industrial setup involved such large structured and unstructured data. It becomes important to have the research results applied to at least a sample of such larger requirements specifications and provide evaluation reports to inspire practitioners to take benefits of the available resources.

From among the research works that had attempted to do such industrial case studies the performance of the NLP technique used seemed to be deteriorating as the size of the text data grows. Part of this impact on the performance of the NLP techniques is attributed towards the complex parsing and computations that constitute these language processing methods. Further researching on these areas for improving the computational efficiency of natural language processing algorithms have become a significant contributor towards increasing the practicality of automated RE.

2.3.2 Missing implementation details

Most research publications include only a brief summary of the actual implementation approach and a detailed discussion of the research results. We recognized that this was due to space restrictions on many of these publications. In order to extend such research works it is essential to have at least a boiler plate implementation to get started and build further. This was a major missing factor due to which it was hard to reproduce the results. We did reach out to several of the authors connected to such works and were disappointed to hear that the source code or an implementation was not available for almost all of them.

2.3.3 Lack of standard validation approaches

Validating the research results are a major factor that contributes in building trust towards the approach or the method that was proposed for automated RE. In almost all of the cases, we found that the validation of such research results are not conducted using scientific and measurable parameters. This means that every work resulted in conclusion derived from manual evaluations done by either a domain engineer or the researchers themselves. There were no guidelines or a reference template which could be used for assigning a quality indicator for the results obtained from the case studies performed in these works. This makes it even more difficult to study and understand the relative benefits of a research work in comparison to the others.

2.3.4 Missing industrial case studies

In order to take the results from research to practice it is important to showcase the application and benefits of the approach or framework on real industrial projects. Most of the research works did not include any case studies involving large enterprise specification due to either of the following reasons:

- Business use case and specification documentations from most software companies are confidential documentations and will not be available for public research purposes.

- Researchers who were successful in securing such project documentations are unable to share the implementation details of their approach which could have helped us to build out a prototype demonstrating the method used by their paper.
- Community initiatives like http://www.splot-research.org/ are not current with the latest in software product line engineering thus lacking maintenance.
- Less partnerships between industry and research in this field of study.

Due to these practical limitations, we had challenges and difficulty in identifying the results that could be reused for bridging some of the gaps that prevents the use of automated RE in practice. Though our pursuit was to find works that can benefit from being extended further, we ended up identifying several good research works that can be combined in a smart way to get to a state where we can build out a limited version of a publicly available automated FM-based RE framework that is capable of being reused and improved. We believe that the work presented in this thesis will not only benefit the RE process followed in the SPL domain, but could potentially drive the larger software development ecosystem to adapt to a feature oriented design and development paradigm.

2.4 Summary

In this chapter we introduced the main subjects of this research effort which are Feature-based requirements engineering, Feature Models of software product line, Feature Model extraction, validation and analysis. We have categorized the findings of this systematic literature review in terms of the limitations and tool support availability that can enable the development of a unified automated FM-based RE framework for SPL. The review illustrated diverse directions of research like the use of natural language processing libraries, information retrieval techniques, machine learning approaches, model validators for consistency checking etc., which have been used to approach the problem of automating the processes in RE. Almost all the research work suffered from limitations related to reproducability and validation of the results.

In this thesis work we have attempted to reuse and extend some of the best techniques and heuristics which have been proposed and demonstrated in the surveyed research works. For example, the analysis of the Part-Of-Speech (POS) tag; the use of term frequency to estimate relevance within a set of documents; the use of synonyms to cluster related terms referring to the same features; or the distance within the document as a proxy of the degree of relation between two concepts etc. We had spent a good amount of time searching for such great research publications, understanding those contributions, its impact and limitations, and calibrating the future prospects of extending it to build adaptations of it etc. We realized that most of our literature review effort could be reused for evaluating any new developments in the field of engineering automated approaches for RE in practice. We now conclude this section with a discussion about a reference model which can be used for assisting future systematic literature reviews in this area of research.

2.4.1 Literature review evaluation template: A Checklist

As the research in this domain will keep progressing, it is important to have an extensible template that can be used for evaluating upcoming research contributions and place them in the appropriate category. This will aid in detecting significant contributions in the future which can be integrated into existing tools and frameworks. We now present a template based on a checklist for evaluating new research contributions dedicated to feature-based requirements engineering approaches, frameworks and tools:

- ☐ It support one or more input formats (like text, list of words, diagrams, etc.).
- ☐ Support for automated mode of input? (or manual, upload to tool, presence of converters, ...).
- ☐ NLP techniques are used.
- ☐ Is it introducing a new NLP technique?
- ☐ Can the feature extraction algorithm scale for large requirements specifications?

- ☐ Is it capable of measuring the quality of extracted feature names ?
- ☐ What are the heuristics for selecting feature names and can they be quantitatively defined (like in terms of confidence weights, term ranking, term significance, frequency of occurrence or correlation with other significant terms, . . .)?
- ☐ Can it discover relationships between candidate feature terms?
- ☐ Are the relationships marked with attributes (like mandatory, optional, or, alternative)?
- ☐ Are the results of the FM extraction displayed as a diagram or text? For example in the form of a Feature diagram, Tree view using text indentation, etc.
- ☐ Are the feature extraction algorithm configurable?
- ☐ Can the user update the extraction rules in order to extract more accurate features and relationships (like a feedback mechanism which can assist the algorithm to learn and adapt based on manual refinements)?
- ☐ Can the output be translated into formal notations (like Alloy [92])?
- ☐ Is there a publicly available tool support? For evaluation of work related to tool, we have formulated a checklist based on the following definitions of a fully automated tool.
    - ☐ Be able to take one or more file formats as input for the text specifications (like PDF, TXT, DOC, DOCX, . . .)
    - ☐ Be able to extract the complete text excluding the graphical/diagrammatic information with a measured level of accuracy and data loss expectations.
    - ☐ Be able to process the text data to generate a list of words or phrases which are potential feature names tallied against the percentage of confidence depending on the rule set used to extract this information.
    - ☐ Be able to derive the relationships between the features.
    - ☐ Be able to present the extracted features and relationships in the form of some visualization. Such a visualization could accept user edits to update the model and/or the extraction rule for allowing a feedback loop.
    - ☐ Be capable of transforming the generated model to a format that can be consumed by a Feature Model analysis tools which can perform automated analysis operations and provide valuable feedback for validating the model and thereby improving the requirements.
- ☐ Has it been tested for real world software requirements specifications?

Chapter 3

Automated Feature Model extraction

Organizations dealing with software assets often have to manage different variants, configurations or releases of applications or services. This family of interrelated *software products*, with some common traits and variation points, is called a *software product line* (SPL). Formalizing a software product line can help managing the software portfolio of an organization, *e.g.*, identifying all the potential products in the SPL or semi-automating the development of new products.

In order to manage a software product line, different sources of information can be considered. For example, the relationships between products can be defined in design models [126] or be implicit in the source code of the different products [65], *i.e.*, in the sharing of libraries or code fragments. This information can also be made explicit in the form of *textual documents* [121, 208] that either detail the characteristics of specific products (a set of *product specifications* [25, 50, 145]); or in requirements documents that describe two or more products.

A more formal description of a software product line is a type of graphical diagram called *Feature Model* (FM) [101]. A Feature Model identifies a collection of salient characteristics of the products which are called *features*. Then, it describes the *relationships* among those features, *e.g.*, whether the use of a feature in a given product requires (or excludes) the use of another feature. This information is depicted graphically as a rooted tree, where features are vertices and relationships[1] are parent-child arcs.

Feature Models (FM) are formal notations capable of describing the features of the products in a SPL. This field of research gained much popularity since the inception of the *Feature Oriented Domain Analysis* (FODA) paradigm [68, 123]. An FM consists of a finite catalog of features and the relationships among them, e.g. "every product with feature *A* must also include feature *B*". However, the requirements specifications of a SPL and even the specific product specifications are all commonly captured in the form of natural language text. Therefore, generating a FM from the original specification is a very complex task. To this end, it is essential to use automated extraction tools based on NLP for identifying both the candidate product feature names and their inter-dependencies.

Natural language processing (NLP) refers to all types programmatic manipulation on any language used by humans to communicate in their daily lives [40]. NLP techniques range from simple mechanisms for counting the word frequency in a given text to algorithms for automatically extracting the facts and concepts described in a complex paragraph. Although NLP emerged in the 1950s, it is still a very active research field and it is employed in a wide variety of application domains, from spam filters to intelligent personal assistants. In the software engineering domain, NLP has been used to gather information from requirements documents, product specifications and stakeholder feedback in a (semi)automatic way [11, 60, 136, 174, 205]. One particular area of interest has been the description of *Software Product Lines* (SPL), i.e. families of related software products sharing a set of features.

[1] There may also be more complex dependencies, called *cross-tree constraints*, that must be described textually.

3.1 Introduction

As was explained in section 2.2.1, there were several research contributions related to automated FM extraction, most of which had limited information on how to systematically use the approach or the tool. There were several interesting aspects about the different NLP techniques and machine learning algorithms used for detecting the presence of potential candidate features in very large text corpus. It was important to notice that a Feature Model is generated during the domain analysis process of domain engineering phase. This corresponds to the requirement analysis phase of the application engineering process [8]. During the domain analysis phase the domain expert together with the software product stakeholders identify the key features of the product. These product features can have one or more dependencies. It is important to be able to detect the impact of the changing feature requirements on these feature dependencies as the domain analysis phase evolves. Feature Models play a significant role in recognizing such dependencies in a graphical way and communicate about such impact effectively. Hence FM becomes a convenient mechanism for the domain engineer to better understand and manage the product variability configuration which are nothing but the features and their dependencies. Thus the automatic or semi-automatic extraction of FM is an important step towards automating the variability modeling of a software product line. For the purpose of this thesis we consider automated FM extraction as the combination of the results obtained from automated feature extraction and automated relationship extraction. The majority of the reviewed literature work attempted the automated feature extraction process. It was evident from the lesser number of research works on automated relationship extraction that it was a more complex process and required detailed study of the language semantics rather than its syntax.

To recap, the major limitations identified in the related work are:

1. **Tool availability**: Lack of publicly available tool support for FM extraction. When it is available, then the tools are either unsupported or not compatible with current software versions.

2. **Restricted inputs**: Input documents must follow a specific format [66], which makes it hard to be extended to arbitrary documents. Such restriction implies having strict document formatting in terms of the document and sentences structure, e.g. a list of bulleted product specifications. This kind of an input expects bulleted sentences with appropriate keywords that can be easily extracted by locating those phrases.

3. **Automatic feature naming**: No support for assigning names for the discovered features, limited support with manual intervention required or poor conventions for automated naming [5].

4. **Reproducibility**: Lack of sufficient details to reproduce the experiments. Input documents and/or outputs not available.

5. **Interaction with the domain engineer**: Only some proposals [66, 82, 145] consider an iterative process allowing an interface for the domain engineer to provide feedback in order to refine the results of the extraction process.

Apart from these limitations, some methods have specific shortcomings. For instance, the method used by Itzik et al. [90] aims to capture the domain model and variability information in the same formalism. As a result, the amount of information would explode for complex families of software products, making their method less practical from a variability modeling perspective.

Considering these limitations as challenges in the practical application of automated FM extraction, this thesis provides an NLP based working solution which addresses most of the identified deficiencies. The goal is to automatically extract potential candidate features and relationships from the natural language text. Given the complexity of this task and the large amount of information in product line specifications, this computation will provide a list of candidate features and relationships, which should be then reviewed by a domain engineer to discard false positives. To ease this task, the list of candidate features should be as complete and succinct as possible. This thesis describes a novel domain independent automated Feature Model extraction framework that has been applied on various case studies and generated promising results. The coming sections cover detailed discussions around the heuristics and architecture of this framework.

The remainder of this chapter is structured as follows. Section 3.2 describes the FeatureX framework for FM extraction, and Section 3.3 provides details on the implementation of this framework. Section 3.4 presents the results of the framework evaluation in comparison with previous works and provides a list of key indicators that illustrate the major advantages of the proposed framework.

3.2 The FeatureX framework overview

There were several research contributions [5, 82] in the Feature Model extraction research. Though most of these contributions lacked implementation details, we were still able to derive the needful inspiration and apply it to fit our requirements. This effort involved unifying the various strategies applied by different researchers and progressing towards a centralized framework for enabling easy practice of Software Product Line Engineering. Identifying extraction rules for detecting meaningful relationships in the context of Feature Models that can connect two or more features using the relationship types **Mandatory**, **Optional**, **Alternative**, **And**, **Or**, **Exclude** and **Require** is one of the main thesis contributions. To demonstrate the advantages of the proposed framework over existing rules in research contributions, a comparison of the results obtained by the framework is done against the results of other research works and corresponding benchmark FMs created by domain engineers. In addition to this, the quality of term/concept extraction rules to identify the features is measured in terms of the *Precision* and *Recall*, the details of which is explained in Section 3.4. This will help in identifying the confidence level at which the extracted features can be placed.

With this goal in mind, the next section 3.2.1 provides an architectural overview of the proposed framework. The feature relationships and constraints are identified based on heuristics that are explained in the section 3.2.2. These heuristics distinguish two types of relationship categories: one-to-one relationships and one-to-many relationships as was described in Table 2.1. Section 3.3 provides all the details of how the framework is implemented and section 3.4 discusses the case study setup and results. This chapter concludes with a detailed account of all the known threats to the validity of this framework in section 3.5 and a chapter summary in section 3.6.

3.2.1 Architecture

FeatureX, is an integrated approach and a framework for automated FM extraction from textual requirements specification documents. To achieve this goal, FeatureX needs to perform three subtasks. First, it needs to *identify candidate features* within the text. Second, it needs to *identify candidate relationships* among those features and *assign them a suitable type*. And finally, it needs to decide the *direction* of those relationships, distinguishing between parent and child features and highlighting the root node of the Feature Model.

The architecture of FeatureX aims to solve these three challenges sequentially. The main components of FeatureX are presented in Figure 3.1:

1. **Lexical analysis module** which includes text preprocessing, entity extraction and connected noun phrases extraction,

2. **Machine learning module** used for classifying a candidate feature term as a root feature and then determining the term frequency of that candidate in text for assigning weights that is used for final root feature identification, and

3. **Feature relationships mining module** which entails the implementation of the heuristics for identifying and categorizing feature relationship types. The relationship text files shown as output in Figure 3.1 contains the results of the FeatureX relationships mining heuristics.

The lexical analysis phase resulted in a large exhaustive set of potential candidate feature terms that can be further clustered and reduced to detect well formed product features. This processing phase was also responsible for identifying the list of candidate root feature for the software product line to which the input requirement specifications belonged. Once the potential features were identified it was necessary to label them with appropriate names

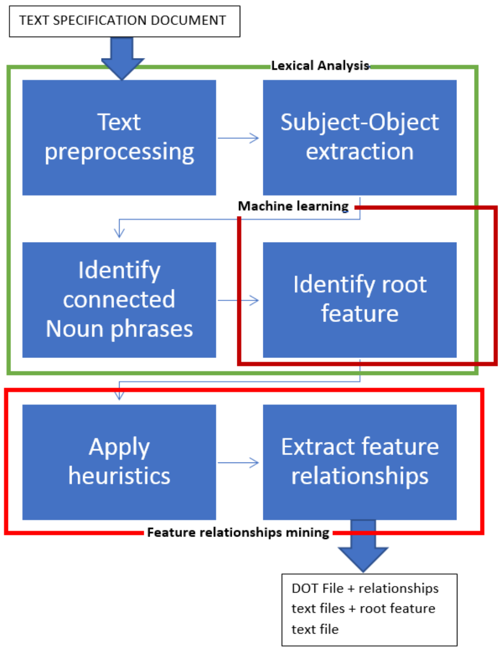

TEXT SPECIFICATION DOCUMENT
Lexical Analysis
Text preprocessing
Subject-Object extraction
Machine learning
Identify connected Noun phrases
Identify root feature
Apply heuristics
Extract feature relationships
Feature relationships mining
DOT File + relationships text files + root feature text file

Figure 3.1: FeatureX overview for processing a single product specification document.

that evidently reflected the intentions of the feature thereby being meaningful for the domain engineer who would review it.

The relationship mining phase required a deep dive into the semantic analysis and correlations that could exist between the candidate feature terms. This was the most complex contribution of this thesis as it involved exploration of various techniques that could assist in mining correlation between concepts identified from the text information. Understanding the various parts of speech for the English language and being able to comprehend the different possibilities of representing an information with the same meaning, was essential for devising a strategy that could work for both structured and unstructured text.

We have used NLTK (Natural Language ToolKit) for all of the lexical analysis and have applied machine learning technique (Naive-Bayes algorithm) for identifying and weighing the candidate feature terms. NLTK [40] is an open source library in Python, which is used both in industry and scientific research for the analysis and processing of text data. The most significant part of our work is the contribution towards *feature relationships mining* which will be discussed to a very great detail in the next subsection 3.2.2. We have used Pattern library [173] ($pattern.text$) for text parsing in order to filter irrelevant terms by explicitly creating a catalog of nonsense terms that do not make sense for a given domain and then apply the text parsers from this library. We have used $pattern.graph$ from the same library for visualizing connected candidate feature terms. The synonym detection provided by WordNet [141] for clustering similar candidate features helped with identifying related feature terms. Open source PDF parsers like PDFMiner[2] was used for generating the raw text corpus from inputs that were provided as PDF requirements specifications files to FeatureX.

To clearly understand the process of solutioning the Feature Model extraction problem from text, it is important to run through a couple of examples manually and recognize the patterns that could exist and could be taken advantage of while coming up with the heuristics.

Example 1

[2]https://github.com/euske/pdfminer

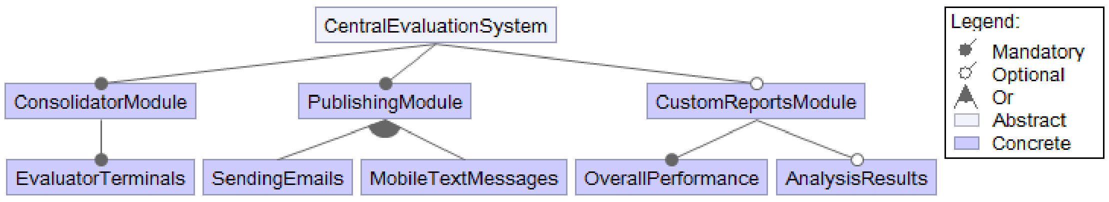


Figure 3.2: Manually created Feature Model in FeatureIDE [103] - Online Examination Coordinator.

Let us consider an excerpt of a real requirements specification for an online examination coordinator system product.

> *"The **central evaluation system** collects all the evaluation results from the various **evaluator terminals** using the **consolidator module** and enters them into the results database where the scores for various courses are segregated or grouped based on candidates. These results will be used by the **publishing module** which is responsible for **sending emails** or **mobile text messages** to the candidates with their complete score cards. The publishing module can be configured using a **custom reports module** for generating customized views of the reports which can be used for understanding the **overall performance** of the class and provide informational **analysis results** on the exam to the teachers and other administrators."*

Figure 3.2 depicts a potential Feature Model for this SPL, manually created by a domain engineer. It captures the features (bold phrases) and relationships among those features (underlined words) described in the above excerpt which are relevant according to that engineer's perspective.

To start the explanation of the framework, we will first provide a glimpse of the comparison study carried out between feature extraction techniques defined in the literature. For this we have selected and implemented the rules identified from [12, 13] using **NLTK** and generated the candidate features and their relationships. Since we cannot provide all the details of our findings we have illustrated the comparison of feature terms extracted from a single requirements statement using the rules from each of the papers in the below example:

Example 2

We highlight the features extracted using [13] in **bold**, [12] in square brackets '[]' and the actual expected features as suggested by a domain engineer in underline. We are using a sentence from the specification excerpt presented in *Example* 1.

> *"**These [results]** will be used by **the [publishing module]** which is responsible for sending **[emails] or mobile [text messages]** to **the [candidates]** with **their complete [score cards]**."*

The relationships or associations are defined by the noun phrases connected using any one of the phrases shown in Table 3.1, which have been extracted using the association rules taken from the respective approaches.

As can be seen from Table 3.1 and Example 2, there are 2 association phrases found in the sample text which are *'be used'* and *'is'*. This establishes a relationship between **publishing module** and **emails**. But when we look back at Figure 3.2, there exists a relationship between the feature terms **publishing module**, **sending emails** and **mobile text messages**. The existing association rules fail to identify the **Or** relationship, while the extracted feature terms are mostly correct.

Similarly, let us consider sample texts from semi-structured and unstructured text specifications. A semi-structured data in our context is defined as that which contains relevant pieces of information that are obtained by sources like

Association phrases	
is	based
be used for	segregated
be used	customized
be configured	collects
grouped based on	

Table 3.1: Relationship phrases detected by the association rules from [12, 13].

voice converted into text or consolidated points from a discussion etc. which does not follow a consistent template or language style [193] . Whereas an unstructured text is one that is obtained from sources like product specifications from online forums, e-commerce websites and web marketplaces etc. which includes data from random sources with very diverse language usage and styles [117]. Below is an example of a semi-structured text.

> "Antivirus Scanner:Protection Cloud Technology:PUA Shield:Award-winning protection from malware (viruses, Trojans, worms, etc.) Scans unknown files in real time for malware and exploits. Identifies potentially unwanted applications, hidden programs bundled with other software. They display ads, slow down your PC and redirect you to other websites.hidden within legitimate software."

An example of an unstructured text would be a set of product reviews for a specific product, written by customers in social media. For example the below text shows the reviews written for a camera on Amazon website.

> "Already obsolete as Nikon readies to launch its mirrorless Z-mount marvel. High shutter speeds require a grip. Top LCD stopped working within days. Some low ISOs actually seem *noisier* at 1:1 than the d800, including ISO 400. The top LCD and view-finder content disappeared. Touch screen is nice. Video still 8-bit and thus has very limited utility in the end."

For the above examples, extracting meaningful feature terms is rather very challenging as the context of the text in most of these cases cannot be distinctly identified. Under such circumstances the simplest approach is to extract noun phrases that repeat itself and mark them as candidate feature terms. Whereas to detect relationships we must attempt to apply more human intuitive ways of processing text like meaning analysis, topic analysis etc. that are capable of highlighting the key connections between topics in such descriptions of the product. Therefore, irrespective of the format in which the textual data is presented, our goal is to demonstrate the feasibility of an automated process that is capable of mining meaningful feature descriptions and indicate their dependencies. This thesis work is the first step towards tailoring the various existing approaches to derive a complete solution with an accepted range of false positive errors.

There can be extraction rules that can identify all or most of the possible relevant feature terms, but the relationships are dependent on the perspective of the domain engineer who is evaluating the FM and the style of the language used in the text. This means that we cannot have a single set of feature relationships that works for any domain engineer as the interpretations are very subjective. Keeping this in mind, we have identified a set of natural language processing (NLP) techniques that can bring out meaningful features. Mining relevant relationships was challenging when targeting to consider all kinds of text (structured, semi-structured or unstructured) inputs. The NLP techniques that have been used in the FeatureX framework for extracting candidate feature terms are defined as follows:

- Stop words removal [40]: remove commonly used words such as *'a'*, *'an'*, *'the'*, etc.;
- Tokenization [40]: breaking down sentences into words and punctuations (a.k.a. tokens);
- Part of Speech (POS) tagging [40]: labeling words with known lexical categories;
- Subject-Object phrases (entity) extraction using consecutive Noun phrase chunking [40]: select subset of

tokens corresponding to individual noun phrases; and

- Term Frequency and Inverse Document Frequency (TF-IDF) [150] for identifying the root feature: assigning weights for defining the importance of the words in a collection of documents. This was also used for selecting the best fit feature name among the group of candidate terms.

In addition to the above, we have maintained a list of words identified as *nonsense words*, that did not make sense to be considered when parsing sentences for domain specific terms. Such words included terms like *day, alone, same, part, feeling, hence*, etc.

In the next section we study in detail, the heuristics used in FeatureX for relationships mining together with its formal definitions.

3.2.2 Heuristics

As the candidate feature terms extraction is independent of the context and style of the text, direct application of NLP techniques are utilized for that purpose. On the other hand feature relationships detection relies on the meaning and style of the language that is used in the text. Such relationships can be classified into two categories based on the number of features they connect. These are one-to-one relationship, where one feature term connects to exactly one other feature term and one-to-many relationship, where a feature can have two or more features connected to it. In order to identify these relationship types, it is essential to have a generic approach for detecting the baseline parent-child hierarchy between features which works for any kind of text input. For this, rather than treating sentences or parts of sentences as distinct pieces of information, we consider it as a linked chain of correlated information which would make sense if read in the order in which they appear in the text. This is the way a human brain processes text. The brain understands the context by reading the text line-by-line and then based on the understanding of the read lines the brain proceeds to the next line in the text and connects the context. We have used this analogy to identify hierarchy and relationships between the subjects and objects in the parsed sentences with that of the subjects and objects in the succeeding sentences.

The subject/object noun phrases become the candidate feature terms which will be set as a parent or a child feature based on the position of its first occurrence in the text. For any two related candidate feature terms f_1, f_2 if f_1 occurs first in the text then $f_1 = parent(f_2)$ and f_2 becomes the child feature of f_1. This way we extract the parent-child relationships between these candidate feature terms which forms the baseline set of relationships. This hierarchy will be refined or modified only when the specific type of the relationship is identified. More on this will be explained in the next sections where each of the two relationships categories and their characteristics in terms of natural language patterns used for extracting them are explained. This is similar to what has been studied by several other researchers in related works [37, 90, 212].

CC	Coordinating conjunction	**RB**	Adverb
CD	Cardinal number	**NNS**	Noun, plural
DT	Determiner	**PDT**	Predeterminer
IN	Preposition or conjunction	**NNP**	Proper noun, singular
JJ	Adjective	**VB**	Verb, base form
MD	Modal verb	**VBG**	Verb, gerund, participle
NN	Noun, singular or mass	**VBN**	Verb, past participle

Table 3.2: Natural language POS Tag abbreviations.

One-to-one mapping relationships

Mandatory and Optional features. These features can be identified by the presence of modal verbs in a given sentence. Model verbs can express necessity or possibility, such as *'can'*, *'could'*, *'may'*, *'might'*, *'must'*, *'shall'*, *'should'*, *'will'* or *'would'*.

Example 3

Let us look at an example taken from the excerpt presented in *Example 1.1*:

> *"The **publishing module** <u>can</u> be configured using a **custom reports module**"*

Here the modal verb **'can'** is connecting the feature terms **publishing module** and **custom reports module**. Based on our application of this rule on the various case studies we have performed it was observed that this type of a connection is valid for most of such cases occurring in all types of the textual data. With this we suggest a rule that there exists a one-to-one mapping relationship between the two feature terms and the mapping type could be **Optional** as the modal verb does not express obligation like *'must'* or *'should'*.

Nevertheless, the above rule is still unable to identify that the features **publishing module** and **custom reports module** are not directly related. Thus, the presence of modal verb alone cannot ensure the validity of the relationship. In order to improve the quality of our proposed feature relationships, we consider two additional elements: the presence of *adverbs of time* and the use of *past/present participles*.

Adverbs of time are special adverbs giving an indication as to when in time an action occurred, for what duration and how often. Some examples of such adverbs are *'often'*, *'never'*, *'always'*, *'frequently'*, *'normally'*, *'usually'*, *'generally'*, *'regularly'*, *'occasionally'*, *'sometimes'*, *'hardly'* or *'rarely'*. The main advantage of considering adverbs of time is that they always have a very consistent position in a sentence, bound by the semantic rules of the English language. In this way, it is easy to propose a generic rule that works for any kind of sentences. We label this set of words as ***adv*** for any further reference in this thesis.

This rule, in conjunction with the detection of contiguous sequences of past and present participle, can help identify the presence or absence of direct relationship. Such participle combinations are identified by checking for a verb ending with *-ing* which is immediately preceded by a verb ending with *-ed*. Example; *'..configured using..,'*. This indicates that the features are related and neither one is a sub-feature of the other.

<u>*Excludes and Require/Implies features.*</u> Now we can extend the one-to-one mapping rules by adding new rules based on determiners /predeterminers and conditional verbs. Determiners and predeterminers like *'all'*, *'every'*, *'each'*, *'any'* and *'some'* can provide information on whether the relationship is mandatory or optional and give an indication on the cardinality of the mapping function similar to the presence of conditional verbs in a sentence. We label this set of words as ***det*** for further reference. This facilitates the identification of dependencies between disconnected features such as **require/implies** relationships. These cross-tree relationships between 2 feature terms that will belong to 2 different sub-trees of the Feature diagram representation, are known as Cross-tree constraints (CTC). For a very simplistic rule related to **implies**, a conditional sentence with the '*if-then-else*' construct can be used to identify conditional dependencies amongst the features. Such a rule is also included in the rules set.

Example 4

Since we do not have an *excludes* and *require* relationship in the excerpt from *Example 1.1*, we use a different example for demonstrating the semantic significance of *determiners* and *predeterminers* in mining these cross-tree constraints (CTC):

> *"**Headphones** could be <u>generally</u> provided for <u>all</u> devices with **FM radio** capability"*

Here, the adverb *'generally'*, the modal verb *'could'* indicates the optional nature of the relationship and the predeterminer *'all'* signifies an implication. This establishes a *requires* CTC between **headphones** and **FM radio**. If it happens that based on the other feature terms extracted from the remaining part of the specification, **Headphones** is the direct parent feature of the **FM radio** then the relationship might be labeled as **Optional**.

A well-defined set of rules can be defined which can be applied on a list of words $\Im$ for any given tokenized sentence from a specification document. $\mathbb{F}$ is a set of candidate feature terms for which a baseline set of pairwise hierarchy is defined as $\mathbb{H}$. Then, for any given pair of *feature terms* $f_1, f_2 \in \mathbb{F}$ such that $(f_1, f_2) \in \mathbb{H} \cup (f_2, f_1) \in \mathbb{H}$, the following rule is satisfied if the corresponding pattern applies:

$$\mathbf{R1}: Loc(f_1) < Loc(MD) < Loc(f_2)$$
$$\mathbf{R2}: Loc(f_1) < Loc(adv) < Loc(f_2)\,\mathbf{or}\,Loc(f_1) < Loc(det) < Loc(f_2)$$
$$\mathbf{R3}: Loc(f_1) < Loc(VBN - VBG) < Loc(f_2)$$
$$\mathbf{R4}: Loc("if") < Loc(f_1) < Loc("then") < Loc(f_2)$$

where, MD, VBN, VBG are parts-of-speech (POS) tags as defined in Table 3.2 which occur exactly one time, and for any word *x*, *Loc(x)* is a word location/position function defined as:

$$Loc(x) = f(\Im) = x' \left| \begin{Bmatrix} x' = \Im.index(x) \\ 0 \leqslant x' < \Im.length \end{Bmatrix} \right.$$

We now consider the above 4 rules as categorical variables for classifying a detected relationship. This means that the 4 classifiers can be used to make a decision on the classification of the relationship. The *any-of* classification χ_r for $\Re$ different classifiers applied on any feature terms pair (t_1, t_2) will return either 1 or 0, and can be defined as:

$$\chi_r(t_1, t_2) \in \{1, 0\}\ for\ any\,\mathbf{r} \in \{R_1, R_2, R_3, R_4\}$$

Once the classifier identifies the categories, a decision matrix as shown in Table 3.3 is used to decide the final relationship type of the selected pair of features. This can be defined as:

$$\chi(t_1, t_2) = \sum_{r=1}^{|\Re|} \chi_r = \begin{Bmatrix} max(|\mathbf{M}|, |\mathbf{O}|, |\mathbf{I}|, |\mathbf{E}|) \\ \emptyset \end{Bmatrix}$$
$$where\,\Re = \{R_1, R_2, R_3, R_4\}$$

	1	0
R_1	**M** , **O**	**M** , **O**, **I** , **E**
R_2	**M** , **O**, **I** , **E**	**O**
R_3	$\emptyset$	**M** , **O**
R_4	**I** , **E**	**O**

Table 3.3: Decision matrix.

One-to-many mapping relationships

Or, And and Alternative features. These are feature relationship mappings that extend from one feature to multiple features. Such feature relationships may or may not be mined from a single product specification. A more accurate assessment can happen when there are several product specifications involved.

Example 5

Consider the following examples from two different products (SelfKey[3] and AuthenticID[4]) of blockchain-based user identity software applications:

SelfKey

[3] https://selfkey.org/wp-content/uploads/2017/11/selfkey-whitepaper-en.pdf

[4] https://tokens.authenticid.co/images/2017/08/AuthenticID-Smart-Identity-Token-Sale_Whitepaper_2.0_13-Sept-2017_FINAL.pdf

*"We believe that **proof of individuality** (POI) can be better solved with **biometrics** than what is currently being proposed in the Ethereum community – **a series of video calls**."*

AuthenticID

*"Use **biometrics** and device-related factors to determine if the person whose picture is on the ID document is actually present. **Facial recognition**, **liveness detection** and **other methods** are **key components**."*

Here we detect the presence of coordinating conjunctions **CC** (see Table 3.2) like **and** and **or** and prepositions and conjunctions **IN** (see Table 3.2) like **if**. This will detect whether two features:

- must coexist when a parent feature is present (**AND relation**) or
- they may or may not coexist when a parent feature is present (**OR relation**)
- they coexist in a mutually exclusive way when a parent feature is present (**XOR relation**)

The one-to-many relationships can be visualized as a set of feature pairs which have same or semantically similar parent features. In the mentioned example it can be seen that both products have attempted to use **biometrics** for validating the proof of existence of a person in their respective systems. In addition to that, there is another *alternative* suggested in one of the product specifications as *series of video calls* which can be considered either as an **OR** relation or **XOR** relation, but never an **AND** relation. Finally the features **liveness detection**, **facial recognition** and **other methods** have an **AND** relationship with the feature **biometrics**. A part of the identity software FM is shown with the mentioned features and relationships in Figure 3.3. Note that by default an optional or mandatory relationship with a set of features is an **AND** connection in FeatureIDE tool [103].

The example illustrates the role of conjunctions and propositions in the sentence for identifying such feature pairs. To automate this process in FeatureX, there are 2 kinds of extraction rules applied. One that pertains only to a single product specification and the other that pertains to a situation where there are more than one product specifications. Given a specification $\mathbb{S}$ and any 2 feature pairs $(f_1, f_2), (f_3, f_4)$ from $\mathbb{S}$, if $f_1 = synonym(f_3)$ then there exists a relationship $\mathbb{R}$ where

$$\mathbb{R} \in \{AND, OR, XOR\}$$
$$(f_1/f_3) \xrightarrow{\mathbb{R}} f_2$$
$$(f_1/f_3) \xrightarrow{\mathbb{R}} f_4$$

The specific relationship is identified based on the presence of conjunctions and prepositions as explained in the example. Similarly, for more than one product specifications $\mathbb{S}_1, \mathbb{S}_2$ and feature pair (f_1, f_2) from $\mathbb{S}_1$ and (f_1', f_2') from $\mathbb{S}_2$ if $f_1' = synonym(f_1')$ then the feature pairs have one-to-many relationship similar to the previous definition. For this, the evaluation of similarity between feature terms in done using the synonym function from the WordNet [141] library and choosing an appropriate feature term among the candidate parent features follow the same rules for feature name selection (Section 3.2.1).

To summarize, the feature relationships extraction heuristics in **FeatureX** comprises of the steps shown in Algorithm 1.

It can be very optimistic to assume that applying such simple rigid rules can indeed work for any kind of text specifications irrespective of the inherent ambiguity and grammatical styles possible in English language. For this reason we would like to emphasize that the rules presented in this work must be considered as an extensible set which can be improved as and when newer specifications are being tested with FeatureX. The feedback from the domain engineer will be used as the driver for updating these rule sets in future.

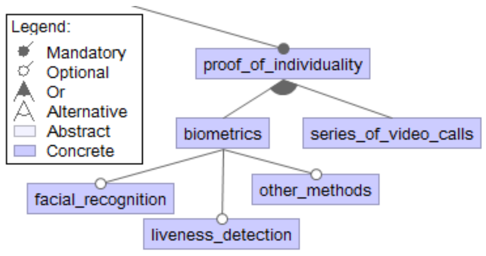


Figure 3.3: Partial view of Identity software FM (manually created in FeatureIDE [103]).

```
Input  : A set of candidate feature terms $FT$, selected root feature $F_{root}$ and the raw text content $T$
Output: Set of 4 text files with heuristics results
Function RelationshipMiner($T, FT, F_{root}$):
    foreach $(f,g) \in FT$ do
        ▷ Detect one-to-one relations
        pair ← SetDirection(f,g,T) ▷ e.g.  f → g
        foreach $rule \in [R_1, R_2, R_3, R_4]$ do
            ▷ apply pattern matching rule
            ▷ append to R1.txt/R2.txt/R3.txt/R4.txt depending on the current rule
            results ← ClassifyFeaturePair(T, pair, rule, $F_{root}$);
        end
    end
    WriteToResultFile(results);
    return ▷ R1.txt,R2.txt,R3.txt,R4.txt
```

Algorithm 1: FeatureX relationships extraction algorithm.

3.3 Implementation

The FeatureX framework has been implemented in Python and its source code is available in GitHub as the **FeatureX** library[5]. The implementation relies on the Natural Language Toolkit (NLTK) [40] for using natural language processing techniques like Part Of Speech (POS) tagging, lemmatization, named entity recognition, stemming, term frequency and inverse document frequency, etc. The implementation is complete with the programming of the feature relationships mining heuristics as described in Section 3.2.2. Algorithm 3 describes the logical flow of the steps used by FeatureX for processing the text document into a candidate FM. This implementation incorporates the best from the results shared by various previous works.

Algorithm 1 shows the implementation of relationships extraction heuristics for mining one-to-one relationships. Algorithm 3 processes the raw varied format text data, to a DOT file (DOT is a graph description language, also referred to in this thesis as dot file) with structured feature terms and their relationships. This output becomes a candidate Feature Model for one product specification document. Therefore, if we have n different product specifications, the algorithm will result in n sets of output dot files. The next step is to correlate these dot files and merge to a single list of *CTC*, *Mandatory*, *Optional*, *Alternative*, *And* and *Or* features. This is demonstrated in the Algorithms 2 and 4. These are saved in separate text files with content similar to that shown in Figure 3.4. These files can then be processed further for analysis, validation and finally visualization of the complete Feature Model. This step of generating the graphical model from the FeatureX's output is described in Chapter 4.

For devising a feasible implementation plan for the Feature Model extraction problem, we have broken down the

[5] https://github.com/5Quintessential/FeatureX

[6] https://vergecurrency.com/assets/Verge-Anonymity-Centric-CryptoCurrency.pdf, http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf, https://cryptonote.org/whitepaper.pdf

```
Input  : An array of processed DOT file $D_{dot}$ and root feature $F_{root}$ ($O_{arr} = [D_{dot}, F_{root}]$)
Output: 6 lists with relationships results
Function RefineOneToOneRelations($O_{arr}$):
    foreach ($D_{dot}, F_{root}$) ∈ $O_{arr}$ do
        ▷ Refine one-to-one relations
        foreach ($record$) ∈ $D_{dot}$ do
            if $Relationship(record) == 'M'$ then
                ▷ appending to list of mandatory relations
                $L_{mandatory}.append(record)$ ;
            else if $Relationship(record) == 'O'$ then
                ▷ appending to list of optional relations
                $L_{optional}.append(record)$ ;
            else
                ▷ appending to list of cross-tree constraints relations
                $L_{CTC}.append(record)$ ;
            end
        end
        ▷ consolidate all relations across multiple specifications documents
        ▷ Optimize the lists
        $L_{mandatory} \leftarrow$ RemoveDuplicates($L_{mandatory}, F_{root}$);
        $L_{optional} \leftarrow$ RemoveDuplicates($L_{optional}, F_{root}$);
        $L_{CTC} \leftarrow$ RemoveDuplicates($L_{CTC}, F_{root}$);
    end
    WriteResultsToList($L_{mandatory}, L_{optional}, L_{CTC}$);
    return ▷ $L_{mandatory}, L_{optional}, L_{CTC}$
Function MatchPatternsForOneToMany($O_{arr}$):
    foreach ($D_{dot}, F_{root}$) ∈ $O_{arr}$ do
        ▷ Refine one-to-one relations
        foreach ($record$) ∈ $D_{dot}$ do
            ▷ Detect if there exist a one-to-many relationship and identify the
              relationship type name
            $record_{rel} \leftarrow$ ComputeRelationship($record, O_{arr}$) ▷ apply pattern matching rule
              across all the DOT files
            if $Relationship(record) == 'A'$ then
                $L_{And}.append(record)$ ▷ appending to list of AND relations
            else if $Relationship(record) == 'R'$ then
                $L_{Or}.append(record)$ ▷ appending to list of OR relations
            else
                ▷ it should be an XOR relation
                $L_{alternative}.append(record)$ ▷ appending to list of XOR relations
            end
        end
        ▷ consolidate all relations across multiple specifications documents
        ▷ Optimize the lists
        $L_{And} \leftarrow$ RemoveDuplicates($L_{And}, F_{root}$);
        $L_{Or} \leftarrow$ RemoveDuplicates($L_{Or}, F_{root}$);
        $L_{alternative} \leftarrow$ RemoveDuplicates($L_{alternative}, F_{root}$);
    end
    WriteResultsToList($L_{And}, L_{Or}, L_{alternative}$);
    return ▷ $L_{And}, L_{Or}, L_{alternative}$
```

Algorithm 2: FeatureX advanced relationships algorithm.

```
Input  : A product specification document D
Output: A dot file with all feature terms pairs
▷ extract plain text from the document (e.g.  PDF)
T ← ExtractText(D, PDFMinerLibrary);
▷ apply NLP techniques for lexical/syntactical analysis, e.g.  POS tagging
P ← PreprocessText(T);
P ← CleanWords(P, NonsenseWordsList);
P ← OptimizeForSynonyms(P, WordNetLibrary);
▷ find features by collecting Subject-Object phrases and apply chunking rules for
  multi-term features
FT ← IdentifyCandidateFeatureTerms(P);
▷ use frequency distribution to find the most relevant features
$F_{root}$ ← IdentifyCandidateRootFeatures(P,FT);
▷ visualize the connected candidate features
VisualizeFeatureTermsCluster(FT, PatterLibrary);
▷ Calling the relationships heuristics algorithm
REL ← RelationshipMiner(P,FT, $F_{root}$);
▷ compute final classification using decision matrix
DR ← ApplyDecisionMatrix(REL);
▷ Constructing dot file representing the FM
result ← (FT, $F_{root}$, DR);
WriteToResultFile(result);
▷ FM.dot file, RootFeature.txt
```
Algorithm 3: FeatureX algorithm.

```
Input  : A list of dot files ($DF$) corresponding to different product specifications document
Output: Set of 6 text files with feature relationships results
Function DotFileComparer($DF$):
    ▷ DOT is a graph description language
    foreach (file) ∈ $DF$ do
        ▷ Identify one-to-one relations
        Relationships1 ← RefineOneToOneRelations(file) ;
        ▷ Append processing results to file
        results ← UpdateProcessingResults(Relationships1);
        ▷ Detect one-to-many relations
        Relationships2 ← MatchPatternsForOneToMany(file) ;
        ▷ Append processing results to file
        results ← UpdateProcessingResults(Relationships2);
    end
    ▷ All feature term relationships except complex cross tree constraints
      (represented in the form of propositional formulas with more than 3
      propositional variables) are outputted to TXT files corresponding to each
      feature relationship name
    WriteToResultFile(results);
    return ▷ Mandatory.txt,Optional.txt,
      And.txt,Or.txt,Alternative.txt, CTC.txt
```
Algorithm 4: FeatureX Dot file comparer algorithm.

```
transaction_ledge->mint_transaction
transaction_ledge->backing_escrow_pool
transaction_output->first_coin_commitment

transaction->routing
```

Figure 3.4: Automatically generated feature terms and relationships - Case study : Cryptocurrency[6].

FM extraction process to logical sub-processes. The results from each sub-process will feed into the next process in the form of process inputs. To help understand the chronological order of the sub-processes in play the flowchart in figure 3.5 illustrates the flow on how to read the algorithms associated with the FeatureX tool. The approach that is used to guide this implementation for automatic feature extraction uses the understanding gained from the way human brain make sense of the content being read. An overly simplified interpretation of this process can be explained as follows:

- Line by line interpretation of the meaning
- Associating the new understanding with that of the previously read line or lines
- Creating a set of key concepts being explained in the text
- Highlighting the main subject of the text

We propose a novel approach which makes use of techniques available in natural language processing, statistical analysis of words and phrases frequencies, measuring text similarity and relations and with the use of clustering and classification algorithms.

3.4 Extraction results

In order to evaluate the FeatureX framework, we compare the FM extracted using our tool with (a) the FM proposed by domain engineers and (b) the FM proposed by other methods for FM extraction from textual specifications. The contributions of domain engineers has been key for evaluation of the results, as the benchmark FM is always a model manually generated by a domain engineer. The results for six different case studies are presented in these results. These case studies are categorized into two groups:

CS (1) Product lines not studied in other previous works:

(a) Blockchain based identity software: 2 specification documents (whitepapers for SelfKey and AuthenticID), part of which are depicted in Figure 3.3.

(b) Cryptocurrency: 3 specification documents (whitepapers for zerocash, verge & cryptonote) part of the result was demonstrated in Figure 3.4.

CS (2) Product lines studied in previous works:

(a) Car crash management system: 1 specification document [51] and reference FM used from [160].

(b) Anti-virus software: 1 specification document with multiple product descriptions [63].

(c) E-shop software: 1 specification document [82, 90].

(d) Smart home software: 1 specification document [201].

The evaluation results show several improvements in the quality and presentation of results which includes: appropriate feature names; clarity in the relationship hierarchy; elimination of irrelevant feature terms; and produces a manageable set of features and relationships even from very large specification documents. This makes the validation of the output FM easier for a domain engineer.

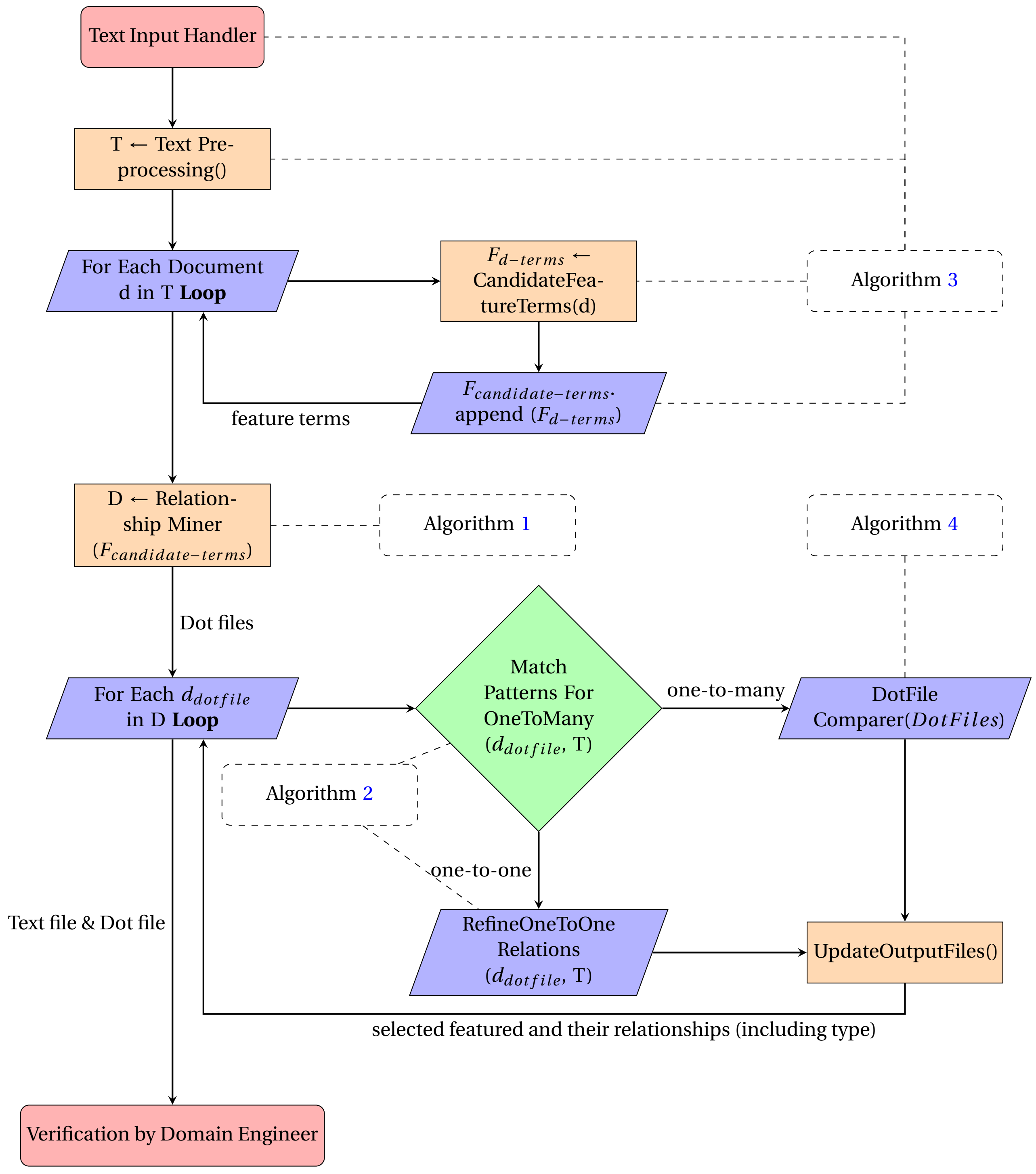

Text Input Handler
T ← Text Pre-processing()
For Each Document d in T **Loop**
$F_{d-terms}$ ← CandidateFeatureTerms(d)
Algorithm 3
$F_{candidate-terms}$. append ($F_{d-terms}$)
feature terms
D ← Relationship Miner ($F_{candidate-terms}$)
Algorithm 1
Algorithm 4
Dot files
For Each $d_{dotfile}$ in D **Loop**
Match Patterns For OneToMany ($d_{dotfile}$, T)
one-to-many
DotFile Comparer($DotFiles$)
Algorithm 2
one-to-one
Text file & Dot file
RefineOneToOne Relations ($d_{dotfile}$, T)
UpdateOutputFiles()
selected featured and their relationships (including type)
Verification by Domain Engineer

Figure 3.5: FM extraction : Sub-processes flowchart

In order to compare the results obtained by FeatureX to those from existing works [63, 82, 90, 201], we have used the expertise of the two senior engineers who have manually created the FMs for CS(1a), CS(1b) and CS(2a). The same engineers were presented with actual text specifications used by FeatureX, and were asked to generate the benchmark FMs for CS(2b), CS(2c) & CS(2d). Thus, manually generated benchmark FMs for all the case studies were available to be used for a collaborative analysis and comparison of models. This can help evaluating the improvements in the FeatureX generated models when compared to the models from the literature. To this end, the domain engineers were provided with the FMs used the literature [63, 82, 90, 201] and the corresponding FeatureX generated FMs. Then, they were asked to analyze each FM using an evaluation template. This template assesses the suitability of each relationship in the FM, seen as a pair of related features (f_1, f_2), according to four qualitative criteria:

1. *Feature naming appropriateness*: Are the name of the features f_1 and f_2 suitable? If it is not suitable, propose the correct name.
2. *Correctness of relationship*: Should the relation among features (f_1, f_2) exist in the FM?
3. *Feature hierarchy appropriateness*: Is the direction of the relation (f_1, f_2) correct (f_1 is the parent) or should it be the reverse (f_2 is the parent)?
4. *Confidence in the relationship classification*: Which is the type of relationship (e.g. mandatory (M), optional (O) or alternative (A))? Which is the degree of confidence regarding this type of relationship?

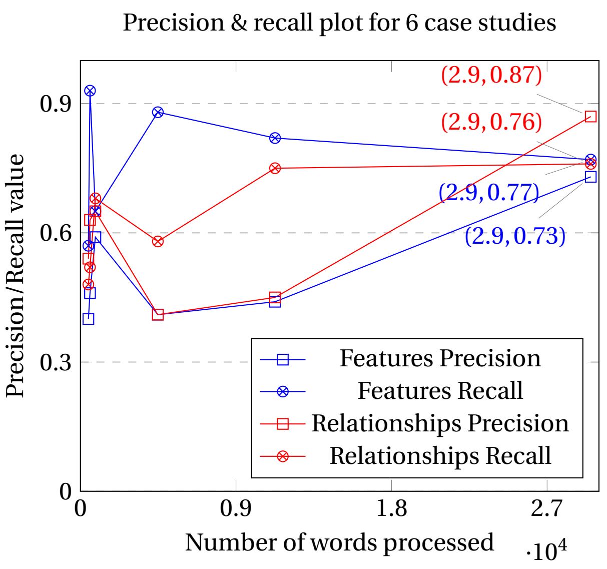


Figure 3.6: FeatureX evaluation with trend lines.

A sample output of the above analysis study[7] is shown in Table 3.4. This information is consolidated into a list of valid relevant features and relationships for all the case studies, which plays an important role in evaluating the quality of the FMs generated from FeatureX and comparing it with previous methods. The following sections describe this comparison in detail.

Candidate feature pair	Corrected name (if needed)	Relationship exists?	Correct direction?	Type* and confidence
(anti_virus, scans)	(anti_virus, scans)	✓	✓	**M** 100%
(anti_virus, email_spam)	(anti_virus, scans)	✓	✓	**M** 100%
(scans, malware)	(scans, malware)	✓	✓	**A** 60%
(spyware_protection, anti_spyware)	(scan, spyware_protection)	×	×	**A** 60%
(anti_virus, product_management)	(anti_virus, product_management)	✓	✓	**M** 100%
(anti_spyware, remove trojan)	(spyware_protection, remove_trojan)	✓	✓	**O** 60%
⋮	⋮	⋮	⋮	⋮

*As defined in Table 2.1.

Table 3.4: Domain engineer analysis of FMs for Anti-virus product line (CS(2b)).

3.4.1 Evaluation metrics

Two significant metrics are used in the comparison: *precision* (measuring how many of the extracted features are relevant) and *recall* (measuring how many of the relevant features were successfully identified). Ideally, an effective extraction framework should exhibit both high precision, with few false positives among the candidate features and high recall, with few missing features among the candidates. In the context of feature terms extraction,

[7]The complete details of this evaluation can be found here: https://github.com/5Quintessential/FeatureX/blob/master/results/EvaluationReport.pdf

precision and **recall** are defined as follows:

$$Precision = \frac{|\{\text{relevant features}\} \cap \{\text{extracted features}\}|}{|\{\text{extracted features}\}|}$$

$$Recall = \frac{|\{\text{relevant features}\} \cap \{\text{extracted features}\}|}{|\{\text{relevant features}\}|}$$

Apart from precision and recall, the following metrics are also presented for each example:

1. Number of words in the specification used, i.e. the size of the documents used for processing and deriving the FM.
2. Number of candidate features extracted *before* i.e. the total number of keywords or feature terms or feature clusters derived during the extraction process.
3. Percentage of correct feature names identified.
4. Percentage of valid cross tree constraints mined.

For all metrics, the values compare the output produced by FeatureX with the results produced by the manual study performed by the domain engineers as described previously.

3.4.2 Observations

The papers [63, 82, 90, 201] presented the FM generated for 3 different case studies (CS(2b), CS(2c) and CS(2d)) using their respective approaches and tools. Overall, the FMs generated by FeatureX for the same case studies were comparable with the results presented in the mentioned papers: there was a **90%** match between features and **45%** match between relationships. In the following, we compare the outputs in more detail to assess the benefits of using FeatureX with respect to state-of-the-art methods.

A comparison of the visualized FMs generated by FeatureX in relation to the resulting FMs from the respective research works are presented in Table 3.5. Figure 3.6 is a scatter plot of the precision and recall values obtained for all the 6 case studies against the number of words used to derive each of those results. We compare the outputs of FeatureX with the tools used for each case study in Tables 3.6 and 3.7. The observations made on the results are:

- The correctness of feature relationships is improving when the size of the processed text data is increasing. This could mean that there is more information in larger text which is useful to narrow down feature relationships to a more succinct set of values, thus increasing the precision.
- The automated names assigned for the features were accurate between 41%-60% of the features in all the case studies. For the rest, the domain experts selected a name which does not appear in the text.
- The FeatureX processing time for each of the case studies was similar (around 3 minutes) irrespective of the size of the input text used.

A threat to validity of this evaluation is the limited number of domain engineers who were involved in creating the manual FMs used as the benchmark and the limited number of case studies used for evaluation. The correctness and validity of the benchmark FM could be argued, but the similarity (≈90%) between the FeatureX FMs with the results obtained by other research supports the validity of the extraction methods used in the framework. Using simple POS-based rules for extracting valid feature terms might seem trivial, but these rules have proved to be capable of detecting a significant number of potential features and relationships. These can be extended to include more rules to allow significant improvement in the completeness and correctness of features and relationships.

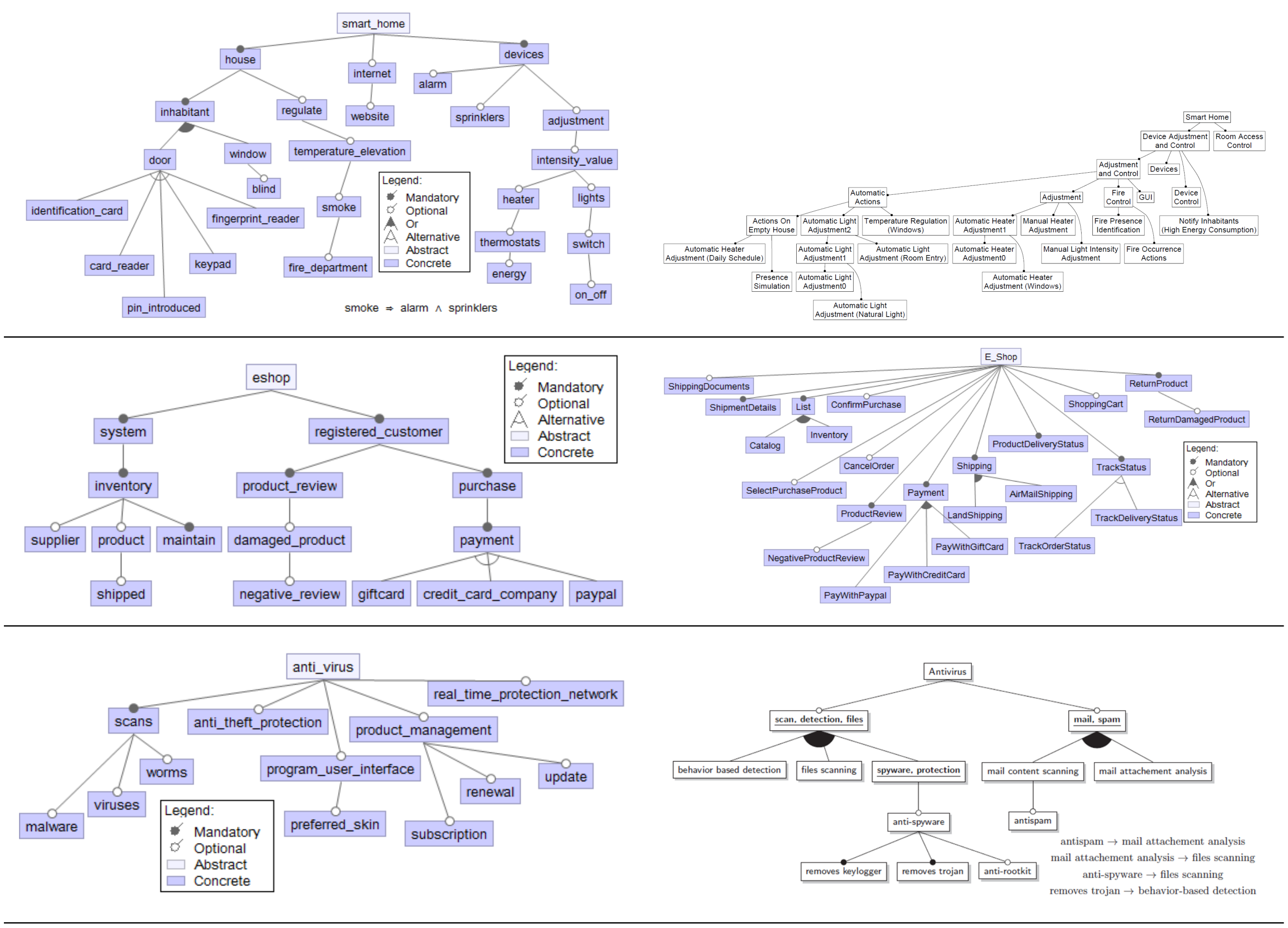


Table 3.5: FM comparison with FM in Literature (SmartHome[201], Eshop [90] & Anti-virus [63])

Case study	# words	# extracted features	% correct feature names	% valid cross-tree constraints
CS(1a)	11258	72	44%	60%
CS(1b)	29528	237	52%	75%
CS(2a)	4479	68	41%	88%
CS(2b)	864	220	60%	50%
CS(2c)	459	40	60%	68%
CS(2d)	552	63	48%	80%

Table 3.6: FeatureX performance.

3.5 Threats to Validity

Though the best efforts have been kept for ensuring maximum coverage of relevant papers to be included in the literature survey, it is important to mention the possible weaknesses of this work in terms of coverage and review method.

We have searched through the significant databases and have selected the most relevant and significant papers

	FeatureX								Other tools							
Case study	Features				Relationships				Features				Relationships			
	Rel	Extr	**Precision**	**Recall**	Rel	Extr	**Precision**	**Recall**	Rel	Extr	**Precision**	**Recall**	Rel	Extr	**Precision**	**Recall**
CS(1a)	15	32	0.44	0.82	14	31	0.45	0.75	-	-	-	-	-	-	-	-
CS(1b)	20	27	0.73	0.77	23	26	0.87	0.76	-	-	-	-	-	-	-	-
CS(2a)	13	32	0.41	0.88	12	31	0.41	0.58	-	-	-	-	-	-	-	-
CS(2b)	15	26	0.59	0.65	16	25	0.65	0.68	7	13	0.53	0.55	6	12	0.50	0.37
CS(2c)	6	16	0.40	0.57	8	15	0.54	0.48	13	24	0.54	0.45	12	23	0.57	0.43
CS(2d)	13	29	0.46	0.93	17	28	0.63	0.52	16	27	0.59	0.57	15	26	0.66	0.55

Rel: Number of extracted features considered relevant by a domain engineer. **Extr**: Total number of extracted features (both relevant and irrelevant).

Table 3.7: Comparing feature and relationship extraction between FeatureX and state-of-the-art approaches.

for this survey subject using a very generic set of keywords. Though the search resulted in thousands of results, the detailed filtering of papers was done manually. It could be possible that some research work could have been missed due to human error. We have not searched in a few other databases such as *Scopus* and *ScienceDirect*. But we believe that this will not pose a serious threat as we have started our work by reviewing a recent survey work by Bakar et al. [20], who have included these missing databases in their survey.

Regarding the evaluation of tool support, we have attempted to contact the authors and get access to the tools used in that paper. When such contacting efforts failed, we have excluded the practical evaluation of the research results for those papers and have cited the unavailability of tool support as a limitation.

3.6 Summary

In this chapter we discussed about the main limitations in the tools and techniques available for extracting *product features* and *product feature relationships* from Software Product Line specifications. Keeping these limitations in mind we explained the details about a novel approach proposed by this thesis for FM extraction from SPL specifications. This was followed by a discussion around the way we devised the mentioned approach together with an elaborate study on evaluating the approach against existing research works.

The contributions of this chapter are: (1) the description of a novel FM extraction framework from natural language specifications (in English), which we call *FeatureX*; (2) an implementation of the proposed framework as an open source tool using the state-of-the-art NLP package NLTK [40]; (3) an evaluation of its effectiveness with respect to the feedback from domain engineers; and (4) a comparison with other approaches from the literature.

Chapter 4

FM Validation

While FMs are very useful to describe and manage an SPL, producing them is a complex task as they require profound domain knowledge. Thus, even though they can be produced by human experts, there has been a significant amount of research in automating the generation of a FM from textual documentation [21, 120]. As a result, FMs can be generated using a variety of techniques, either manual or (semi-)automatic, and produce different FMs depending on the approach used.

Nevertheless, a key weakness of current research is the lack of a mechanism to correlate the information of the FM with the source documents. In particular, we would be interested in assessing the correctness and the completeness of the different elements of a FM, that is, measuring if the source documents justify the choice of a certain feature or relationship and if there are any missing elements. Typically, current works, including the FeatureX framework, measure this accuracy by producing another FM manually with the help of a domain expert, and comparing it with the original FM. Nevertheless, this comparison has several shortcomings. First, it is "expensive" in the sense that it requires the involvement of a human expert. Second, it may be ambiguous, as different experts may produce different reference FMs, leading to different results in the comparison. And third, it does not take advantage of the information available in the source documents, which may include data that has been neglected by the expert.

In this chapter, we propose a method to assess the quality of a FM with respect to one or more textual specification documents which was used as the source documents for generating the FM. Quality is defined as having both *correct* and *complete* sets of features and relationships, that is, all elements should be supported by the information in the text and there should not be missing elements. The specific contributions of our work on FM validation are the following:

1. A method that assigns a *confidence score* in the [0..1] range to each feature and relationship in the FM. This score measures the likelihood/probability that the textual documents identify this element as a relevant feature or relationship in the product line.
2. A visual representation of the confidence scores in the form of a heatmap of the FM, that is both intuitive and informative for product designers.
3. Heuristics to highlight potentially relevant missing features or relationships to designers.

This information has a wide variety of applications: validating the method used to generate the FM; identifying potential fixes or improvements to the FM; comparing and choosing between alternative FMs produced from different sources; improving the understanding of the FM and the SPL; or detecting errors or omissions in the source documents.

The remainder of this chapter is structured as follows. Section 4.1 provides an overview of the proposed FM validation approach. Section 4.2 describes how the correctness and completeness of features and relationships is measured by means of Machine Learning techniques. Section 4.3 describes the visualization of this correctness and completeness using a heatmap. In Section 4.4, we apply our evaluation method to several case studies and discuss our findings. And finally in Section 4.5, we summarize the chapter, highlighting the threats to validity and conclusion.

4.1 Overview

The idea behind validation of Feature Models originates from the fact that there is no single FM that can be identified as the only correct representation of all the features and relationships of a given software product line specification. This is mainly attributed to the inherent nature of textual information which can be interpreted differently by different domain engineers. This leads to the generation of multiple FMs from the same specification sources. To demonstrate this, we have selected the below excerpt from the specifications of an e-commerce product line and apply parts of speech (POS) tagging of the highlighted sentences.

> *"**The system supports paying with credit cards.** If a customer pays with a credit card, the system approves first the payment by contacting the credit card company. When the system completes recording an order, the supplier can ship the ordered products to the customer. The system sends the shipping documents via email. When the system finalizes a software order details, the supplier ships the ordered product via email. The system sends the shipping documents. The system supports different shipping options. However, if a customer buys a very small product, the system supports only air mail shipping. The system displays the available products. When a registered customer buys a product, the system updates the inventory. **The system supports paying with gift cards.** If a customer pays with a gift card, the system supports only land shipping. A customer can track the purchase status. The system provides details on the product delivery status. **The system supports paying with PayPal. If a customer pays with PayPal, the System verifies the PayPal payment information.**"*

POS tagged sentences:

[('The', 'DT'), ('system', 'NN'), ('supports', 'VBZ'), ('paying', 'VBG'), ('with', 'IN'), ('credit', 'NN'), ('cards', 'NNS'), ('.', '.')] [('The', 'DT'), ('system', 'NN'), ('supports', 'VBZ'), ('paying', 'VBG'), ('with', 'IN'), ('gift', 'NN'), ('cards', 'NNS'), ('.', '.')] [('If', 'IN'), ('a', 'DT'), ('customer', 'NN'), ('pays', 'VBZ'), ('with', 'IN'), ('PayPal', 'NNP'), (',', ','), ('the', 'DT'), ('System', 'NNP'), ('verifies', 'VBZ'), ('the', 'DT'), ('PayPal', 'NNP'), ('payment', 'NN'), ('information', 'NN'), ('.', '.')]

Manually, looking at the sentence, it can be inferred that the following could be the probable features of the product and are the alternative payment methods supported by the system:

- **'the system supports paying'** is the root of this sub-tree of the FM.
- **'credit cards'** is one of the features of **'the system supports paying'**.
- **'gift cards'** is one of the features of **'the system supports paying'**.
- **'PayPal'** is yet another feature of **'the system supports paying'**.

In order to automate this kind of information detection from any such specification text, it is important to get some level of understanding of the language used in the text in terms of the grammatical structure, tense used and the writing style. While parsing the tokens (words) from left to right and picking up the relevant tokens to be considered as a feature, it is possible to determine the criteria that decides when to pick the token and when to stop picking it, as it is an n-gram selection. With that in mind, we target to identify the sequence of the various distinct POS tag sequences in the text and manually check the significance of these tag sequences in contributing towards labeling a phrase as a feature. Therefore the n-gram selection can reduce into a POS tag sequence selection problem. For the current example under review we will label the features as F_1, F_2, F_3 and F_4.

- F_1 = 'the system supports paying' = DT NN VBZ VBG
- F_2 = 'credit cards' = NN NNS

- F_3 = 'gift cards' = NN NNS
- F_4 = 'paypal' = NNP

For each token, starting from the first token, we can compute the distinct transitions to other tags in the text as is shown in Table 4.1.

Tag sequence	count	probability in text
DT → NN	14	0.0006
NN → VBZ	105	0.0003
VBZ → VBG	6	0.0000345
VBG → IN	3	0.00001
IN → NN	7	0.00004
NN → NNS	13	0.00007
IN → DT	14	0.00008
VBZ → IN	4	0.00002
IN → NNP	2	0.00001
NNP → DT	1	0.000005
DT → NNP	2	0.00001
NNP → VBZ	1	0.000005
VBZ → DT	38	0.0002
NNP → NN	1	0.000005
NN → NN	24	0.0769

Table 4.1: POS tag sequence probabilities

Such tag sequence probability information from the text, highlights the language style used and the characteristics of the English grammar used in the text. Based on several manual evaluation of product specification documents it is observed that the feature terms would mostly contain one or more type of nouns and verbs. This understanding can be used to formulate a strategy for identifying the POS tag sequences that can be applied on the text corpus and potential candidate feature terms could be identified. Therefore the probable feature terms that could be identified from the text are: F_1' = *the system supports*, F_2' = *with credit cards*. F_3' = *with gift cards*. F_4' = *with paypal*.

We consider this to be a valid identification of feature terms based on the characteristics of the given corpus, independent of any domain knowledge. The FM for this is illustrated in Figure 4.1. The FM demonstrates a product variation that is extracted from the reviewed textual specification. If this information can be combined with a heuristics for detecting feature relationships, we could potentially generate a probabilistic Feature Model, which, for this example, is manually generated using FeatureIDE [103]. Figure 4.1 displays the 3 payment alternatives of the system.

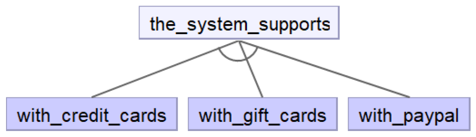


Figure 4.1: Part of the FM for E-Shop generated using FeatureIDE [103].

The figure 4.2 shows the features and relationships identified from literature and the FM created by a domain engineer for the same specification. By comparing the two FMs it is evident that the text has been interpreted differently as is shown in Table 4.2. It is obvious that such differences are always going to be present if different domain experts generate the FM. Given such a situation, it is most ideal to be able to validate FM against the information from the source text that was used for the FM generation, rather than gauging it against the model generated by one or more domain engineers. This way there is an opportunity to compare the reasoning behind

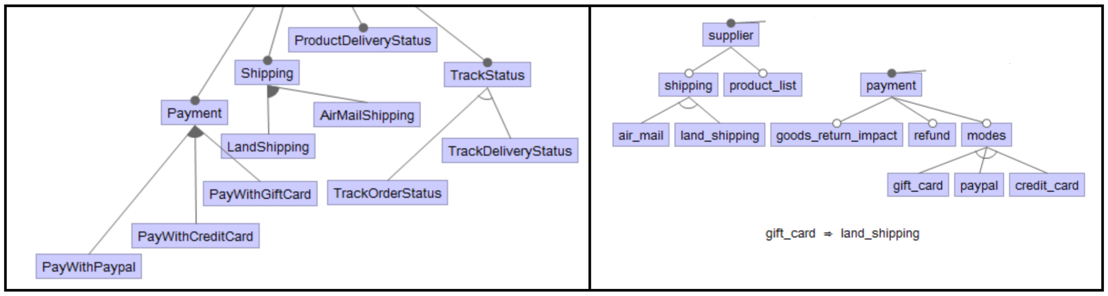


Figure 4.2: FM from literature (left) vs manually created FM in FeatureIDE [103] (right)

the specific FM configuration with that of the text information present in the source documents. This type of validation can also help identify inconsistent information in the text document and assist the RE practitioner to correct them based on the FM validation results. Moreover, the current inability to evaluate FM in a deterministic way further threatens the validity of the FM extraction approaches identified by various research works. Therefore, this way of cross checking the information demonstrated by the FM against the information in the source text can alleviate the FM verification and validation problem to some extend due to its generic nature which is not biased by any preconceived knowledge of the domain.

Features	**Literature FM**	**Manual FM**
- Gift Card - Credit card - PayPal	Optional payment methods without mutual exclusion	Mutually exclusive alternatives
Shipping options	Or relationship	Mutually exclusive alternatives
Cross-tree constraints	No constraint for the relationship between land shipping and gift card payment option	Includes the constraint between land shipping and gift card payment
Missing features	- Recording Order - Display Products - Track Purchase Status - Update Inventory	- Recording Order - Product delivery status - Track Purchase Status - Update Inventory

Table 4.2: FM comparison

With this intend in mind, we now explain the inputs and the output of the proposed FM verification and validation method. The inputs to this method are the two descriptions of a software product line:

1. A text document or a set of documents describing the SPL in natural language. These documents can be user manuals, product specifications, product comparisons or requirements documents for the complete SPL.

2. A Feature Model that captures the features and relationships in the SPL. It can either be produced manually by a domain expert or generated by a (semi-)automatic extraction method.

Our goal is to quantify the level of concordance between these two descriptions using Machine Learning and Natural-Language Processing. There are two directions in which this concordance should be checked:

1. *Correctness*: The features in the FM should refer to concepts that are mentioned in the source documents (either literally, using synonyms or related terms) and that are relevant within the text. Moreover, relationships in the FM should connect features that are linked within the text.

2. *Completeness*: All relevant concepts in the textual documents should appear as features in the FM.

Given a textual description of a product line and/or the products it includes, there may be several alternative formalizations of the SPL that can be considered correct. For instance, there could be different understandings of what concepts should be considered a feature, how to group them and choose the name for a particular feature or what are the relations between features. Thus, in general it will not be possible to establish correctness and completeness with a ”yes or no” answer. Instead, we will compute a [0..1] *confidence score* for each concept and relationship in the textual requirements. A higher confidence score means that the concept is more relevant within the text, *i.e.*, it is used more often or has been indirectly referred to several times. Then, we will match these scores with the features and relationships in the FM to identify which elements are supported by the information in the text and which relevant elements are potentially missing.

Finally, we will produce a graphical visualization of this concordance, to make this information more usable and informative to designers and domain experts. This visualization will use the *heatmap* metaphor, where elements are painted with a color according to their confidence: colder colors (blue) for low confidence and hotter colors (red) for higher confidence.

4.1.1 Case Studies

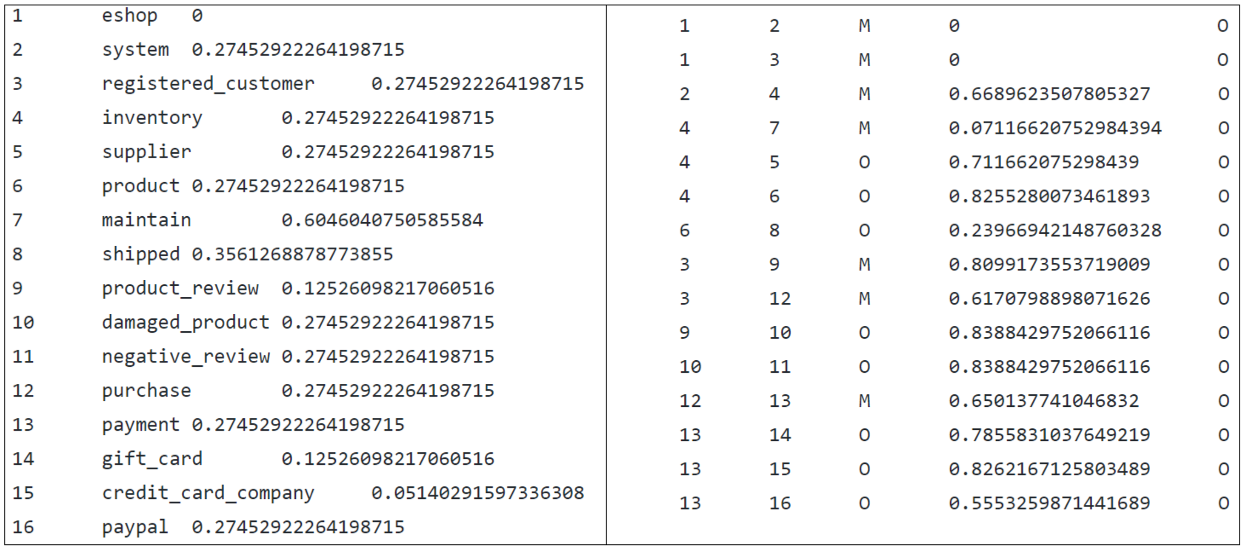
In order to evaluate and illustrate our approach, we will use four different case studies of product lines described both as a textual specification and a Feature Model. Most of these case studies have already been introduced in section 3.4 except for the SUV product line. We have also not considered to use the previous case studies of home automation tool and anti-virus specifications for FM validation. While working with the mentioned 2 specifications as part of the FM extraction, we detected a large number of inconsistencies and incompleteness in the specifications itself which were misleading the FM extraction results. Therefore these 2 case studies have not been considered here. For the rest, as we will be using parts of the case studies as examples to explain different components of our method, we briefly reintroduce them before describing the approach in detail. The case study specifications have been carefully selected from 4 different product lines or domains with varying size or length of the product descriptions. This ranges from a small toy example with 44 sentences that describe the product feature specifications to a very large specification with 55000 sentences that is spread across 6 different documents. This way we aim to compare the performance of the FM validation approach and identify areas of improvement which will be scoped for future work in this research subject.

The case studies have been used for generating two or more Feature Models using FeatureX framework, methods proposed in other research literature and manually by domain experts. At that point the 2 inputs required for this FM validation method is ready, which are, an FM and the text data that was used for generating the FM. The FM validation approach processes this information and generates confidence scores against each feature and its relationship. This means that for an input text $\mathbb{T}$ if there are 2 Feature Models $\mathbb{FM}_1$, $\mathbb{FM}_2$, then for each feature and its relationships $f, f_r \in \mathbb{FM}_1 \cup \mathbb{FM}_2$, confidence scores $\mathbb{C}_f, \mathbb{C}_{f_r}$ is computed for every feature and its relationships.

		Textual requirements		**FM**
ID	**Product line**	**Product**	**Sentences**	**Source**
S1	SUV	Audi	8.328	FeatureX [177]
		Mazda	7.055	
		Toyota	4.819	
		Infinity	9.366	
		Honda	8.162	
		Mercedes	8.871	
S2	Cryptocurrency	Verge	246	Feature X [177] +
		ZeroCash	1.461	Domain expert [177]
		CryptoNote	411	
S3	bCMS	Complete SPL	338	Feature X [177] + Domain expert [160]
S4	E-shop	Complete SPL	44	Extracted in [82, 90, 177]

Table 4.3: Summary of the case studies.

These computed confidence scores are initially saved into text files similar to the one shown in figure 4.3 for the E-Shop FM produced by literature. It is then converted into the dot file format by the heatmap generator module that aids in graphically visualizing this result. The FM input provided to the validation method is in the form of a text file that will list out the features and relationships in the order in which it appears in the tree structure using the breadth first or level-order traversal. The features are listed out with their corresponding incremental ID number and the relationships are documented in the form of 2 feature id numbers and the abbreviated relationship type table 2.1. Therefore the first 2 columns from the left hand side of the table and the first 3 columns in the right hand side of the table in figure 4.3 is provided as input to the validation logic. The FM validation logic then fills-in the confidence score values for the features and the relationships together with the expected relationship type based on the information in the textual specification.



```
1       eshop   0
2       system  0.27452922264198715
3       registered_customer     0.27452922264198715
4       inventory       0.27452922264198715
5       supplier        0.27452922264198715
6       product 0.27452922264198715
7       maintain        0.6046040750585584
8       shipped 0.3561268878773855
9       product_review  0.12526098217060516
10      damaged_product 0.27452922264198715
11      negative_review 0.27452922264198715
12      purchase        0.27452922264198715
13      payment 0.27452922264198715
14      gift_card       0.12526098217060516
15      credit_card_company     0.05140291597336308
16      paypal  0.27452922264198715
```

```
1       2       M       0                       O
1       3       M       0                       O
2       4       M       0.6689623507805327      O
4       7       M       0.07116620752984394     O
4       5       O       0.71166207529843 9      O
4       6       O       0.8255280073461893      O
6       8       O       0.23966942148760328     O
3       9       M       0.8099173553719009      O
3       12      M       0.6170798898071626      O
9       10      O       0.8388429752066116      O
10      11      O       0.8388429752066116      O
12      13      M       0.650137741046832       O
13      14      O       0.7855831037649219      O
13      15      O       0.8262167125803489      O
13      16      O       0.5553259871441689      O
```

Left hand side - Features in chronological order in which it appears in the FM and Confidence score.
Right hand side - feature #1, feature #2, Found relationship type in FM, Score, Expected relationship type (refer table 2.1).

Figure 4.3: FM validation results for E-Shop FM generated from literature.

Here are the specifics of the textual data that feeds into the FM validation method for the different case studies that have been used in this evaluation.

- Case study **S1** describes the product line of *mid-size SUVs* (sport utility vehicles) from six different car manufacturers. The textual specification is the owner manual (a PDF document) for each brand, with around 8.000 to 9.000 sentences per document. The FM to be evaluated is produced by the FM extractor FeatureX [177].
- Case study **S2** specifies a product line of cryptocurrencies, studying the specification documents of three different currencies. In addition to automatically extracted FMs for this product line, there is a FM defined by a domain expert.
- Case study **S3** is a specification of a product line for a crash management system (bCMS) [51] specified in a single document. It has been previously used by several researchers as a case study for feature extraction [57, 87, 177]. Moreover, a reference FM [160] proposed by a domain expert is also available.
- Case study **S4** describes the order processing module of electronic commerce applications (E-shop). This product line has also been used as a benchmark by previous works on feature extraction [82, 90, 177], so several FM descriptions are available.

Table 4.3 summarizes the details of the four case studies, including the length of the specification (number of sentences) and the source of the Feature Model (either defined by a domain experted or extracted using an automatic tool). All the inputs (source documents and FMs) and outputs of the validation are available in a public repository in GitHub[1].

Features			Relationships			
Id	**Name**	**Score**	**Parent**	**Child**	**Type**	**Score**
1	mid_size_suv	0.69	1	2	Mandatory	0.67
2	driving_assist_system	0.95	1	3	Mandatory	0.59
3	driving_safety_system	0.97	1	4	Mandatory	0.99
4	emission_control_system	0.92	1	5	Mandatory	0.42
5	climate_control_system	0.95				

Table 4.4: Sample evaluation of a FM in the SUV case study.

4.2 Feature Model evaluation

In this section, we describe the process that we use to identify features and relationships in a set of textual documents and assign them a confidence score. Table 4.4 shows an example of the expected output of this process, which indicates a high level of confidence in the proposed features but a low level of confidence in some of the relationships. We aim to provide this confidence score (a) for each feature and relationship in a FM and (b) for salient features detected in the analysis of the textual documents.

First, we present the strategy for predicting features in subsection 4.2.1. Then, subsection 4.2.2 explains the prediction of relationships. A simplified version of the complete procedure, combining feature prediction, relationship prediction and additional heuristics, is listed in Algorithm 5. Later, Section 4.3 will discuss how these confidence scores can be shown to designers in an intuitive and informative way.

4.2.1 Validating features

Overall strategy

Feature extraction is a process where a word or a phrase (sequence of words) is identified as a representation of a distinct ability, a user visible characteristic or a supporting prominent and distinctive functionality of the software product [101].

In order to increase the applicability of our method to as many product lines as possible, we have made two important design decisions. First, textual requirement specifications can be very long documents, for instance, thousands of sentences in our case studies. As some NLP tasks can be computationally very expensive, we have opted for techniques that are efficient, discarding methods that may yield a better accuracy but incur in a significant overhead (such as methods based on semantics). And, second, we will not require a domain vocabulary or ontology as an additional input, as it may be unavailable for some SPLs.

Following these decisions, we have used several IR/NLP techniques for the identification of features:

- *Part-of-speech(POS) tagging* [40]: This analysis assigns categories (such as noun, verb or adjective) to a term, considering its definition and contextual information, *e.g.*, its position in the sentence and adjacent words.
- *Term frequency-Inverse document frequency (TF-IDF)* [132]: This metric measures the frequency and the relative frequency of a term in a document or corpus.
- *Detection of synonyms*: A concept can be described using different terms in different parts of a document. We use the WordNet lexical database [40] together with the Python Enchant[2] library to detect such synonyms and take them into account in the TF-IDF computation.

[1]https://github.com/5Quintessential/FMEvaluator-ML
[2]https://www.abisource.com/projects/enchant/

Phrase Id	**Response Variable** (Y)	**NNP** (x_1)	**VB** (x_2)	**NN** (x_3)	**VBN** ... (x_4)	**VBP** (x_m)
1	Feature	2	0	0	...	0
2	Not-Feature	0	1	1	...	0
3	Not-Feature	1	0	0	...	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮

POS tags used in this dataset

Noun: NN = singular noun, NNP = singular proper noun
Verb: VB = base, VBN = past participle, VBP = present tense

Table 4.5: Sample labeled dataset used for training.

- *Pronoun resolution*: Pronouns can refer to some feature that is mentioned previously, either in the same or sentence or in previous ones. Similarly to the case of synonyms, we track the use of pronouns to improve the accuracy of our TDF-IDF metric.
- *Removal of stop words and meaningless words:* Text typically includes common words that do not affect the meaning of the text (*e.g.,* the, this, ...). Furthermore, if the requirement documents use the PDF format, there may be errors in the extraction of text, such as embedded symbols or graphics in the text, problems with hyphened words, ... All these words should be removed from the text prior to our analysis.

These techniques will be complemented with supervised learning to detect *patterns* in the text that indicate potential references to a feature. These patterns will be defined as *sequences of POS tags*, where we only consider the count of each type of POS tag in the sequence (but not its position). In order to search for these patterns, the specification will be split into groups of n consecutive words (n-grams) where n will be given by the length of the longest positive (feature) sequence we have identified.

The rationale for this analysis is that features in a given domain may follow or use certain common structures, so studying sample features can help us identify additional ones. As we are using supervised learning, we will require a labeled dataset for training with both *positive* and *negative* examples, that is, sequences of POS tags that refer to a correct feature and sequences that do not. Positive examples can be obtained, for instance, from the Feature Model (allowing a domain expert to screen representative feature names) or from the textual requirements (the table of contents or glossary). For instance, Table 4.5 shows a potential dataset that could be used for training. After training, the classifier can inspect the remaining n-grams in the document to identify feature candidates.

Classification using Machine Learning

In ML, a *response* or *dependent variable* is the output of the model that is being studied. *Explanatory* or *independent variables*[3] are measured or set by the researcher and become the input to the model. These variables are called explanatory because they explain how the response variable is affected by their changes.

The explanatory variables selected to model this ML problem are the counts for the POS tags in a given n-gram, while the response variable Y has two potential values: 0 (the n-gram is not a feature) or 1 (it is a feature). As the response variable can have only two values, this problem definition fits the framework of *binomial logistic regression* [129, 153]. We chose to use logistic regression [156] only because we are interested in the probability of the response variable being a feature. We chose this supervised machine learning algorithm because (i) it provides a measure of probability, rather than simply answering "yes" or "no"; and (ii) it can take advantage of domain expert knowledge for selecting explanatory variables and generating the input dataset for the ML algorithm.

[3]In supervised learning, a more popular term to refer to explanatory variables is *feature*. To avoid ambiguity with the features of a SPL, we use the term "explanatory variable", which is more typically used in statistics but describes the same concept.

Id	Product line	Significant POS tag set
S1	Mid-size SUV	NN, NNP, NNPS, NNS VBD, VBZ, VBP, VBG
S2	Cryptocurrency	NN, NNP, NNPS VB, VBZ, VBP, VBG
S3	bCMS	NN, NNP VB
S4	E-shop	NN, NNP VB

Table 4.6: Relevant POS tag sets in the four case studies.

Our case studies include on an average 32 distinct POS tags. Considering the problem of overfitting, we used Lasso regression [138] for selecting the most significant explanatory variables. This process is also known as *dimensionality reduction* which is used as a shrinkage method and for variable selection. Table 4.6 shows the relevant explanatory variables identified for the four case studies. It shows that the classifier will mostly consider noun phrases (POS tags beginning with N) and verb phrases (POS tags beginning with V).

The linear model used by the definition of the logistic regression paradigm is a linear expression as shown below:

$$\beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \beta_3 \cdot x_3 + \ldots + \beta_m \cdot x_m$$

where $x_1, x_2, .., x_m$ are the discrete variables that holds the count of each different POS tags and the β values are the coefficients computed for the respective POS tag.

We will not detail how the values of the β coefficients are established during the training phase, as this is out of the scope of this paper (see [156]). Nevertheless, we will show how they can be used to classify a new n-gram after training.

For instance consider an example from our case study S1, mid-size SUVs. One of the features of an SUV, *traffic sign recognition* is tagged as *(traffic, NNP), (sign, NN), (recognition, NN)*, where the tags correspond to two singular nouns (NN) and a singular proper noun (NNP). β_1 and β_2 are the coefficients for the tags NN and NNP, respectively. The corresponding x values are the count of each of type of tags in our n-gram. Thus, in this example, $x_1 = 2$ and $x_2 = 1$. Hence the probability of this phrase *traffic sign recognition* being a feature is computed as $P(Y = 1) = p$. By definition of logistic regression:

$$log\left(\frac{p}{1-p}\right) = \beta_1 \cdot x_1 + \beta_2 \cdot x_2$$

where the *LHS* function (the *logistic function*, that gives its name to this type of regression) is defined as the logarithm of odds. The respective β values from this case study are $\beta_1 = 1.378$ and $\beta_2 = 1.224$. Then,

$$log\left(\frac{p}{1-p}\right) = (1.378 * 2 + 1.224 * 1)$$

$$p = \frac{e^{(1.378*2+1.224*1)}}{1 + e^{(1.378*2+1.224*1)}}$$

Hence $p = 0.982$, which is very close to 1. This means that this phrase has a very high probability of being a feature.

With regards to the implementation, we have used R [69] to generate a summary of the logistic regression model fitted to the training dataset. We used the R library `glmnet` for applying the Lasso regression shrinkage method and saved the resulting reduced set of explanatory variables and the corresponding coefficients. These explanatory variables (POS tags) are then used by the feature classifier/predictor implemented in Python, which uses NLTK (Natural Language Toolkit) to perform NLP tasks such as computing POS tags and synonyms lookup.

Summary

We define a labeled dataset with positive and negative examples of features, describing the sequence of POS tags in each of them. This dataset will be used to train a classifier using supervised learning. This classifier will analyze the input text and identify a set of candidate features. In order to optimize the total number of candidate features and to ensure an efficient code execution time even for larger text corpus, a combination of TF-IDF metric and synonym term replacement has been used.

4.2.2 Validating relationships

Overall strategy.

The features prediction phase of our analysis was able to detect significant candidate feature terms, which we also refer to as concepts. To complete the validation of the FM, we need to establish two traits. First, we need to check if the features that are related in the FM also exhibit that connection in the text. And second, if the type of connection between them is consistent. For the sake of conciseness, we will focus our discussion on the first dimension (quantifying the degree of relation between two concepts) and provide a couple of useful heuristics for validating the type of relationship.

Similar to that of feature prediction, some other approaches to establish the distance between concepts (such as *word embeddings*) may be computationally complex. Instead, we will trade off potential gains in accuracy to ensure an efficient computation. In this case, we will define a *distance metric* based on the relative position of the two concepts within the source documents.

Estimating the closeness between two features.

Intuitively, if two concepts are related, this relation will manifest in their position in the source documents. That is, related concepts tend to be introduced at the same point of the discussion or, at least, to be presented close to each other. Therefore, we define a *distance metric* $d_{a,b}$ between two features a and b that measures the least number of sentences that separate their mentions within the document. For instance, features introduced in the same sentence would be at a distance 0. Whereas the distance will be 1 if they were mentioned in consecutive sentences, and so on. In this computation, it is very important to consider synonyms in order to properly identify the various occurrences of each feature.

We have modified the distance calculation approach for computational efficiency based on a heuristic. *The distance between 2 features is same as the difference between the distances between each feature and the root feature.* This means that instead of computing distances between each pair of features, we compute the distance of each feature to the root feature of the FM (the most relevant concept in the document, *e.g.*, determined using the TF-IDF metric). Then, the distance between two concepts is established as the difference between their distances to the root feature.

$$d_{a,b} = \left| d_{a,root} - d_{b,root} \right|$$

When a specification consists of several documents (each can have very different lengths), it is necessary to *normalize* these distances in order to allow them to be comparable. We use the *min-max standardization* to convert an arbitrary distance into a value in the [0..1] interval:.

$$d'_{a,b} = \frac{d_{a,b} - min(d)}{max(d) - min(d)}$$

Using this normalized distance, the computation of a confidence score for relationships is straightforward:

$$\text{confidence}_{x,y} = 1 - d'_{x,y}$$

The closer the distance, higher the confidence.

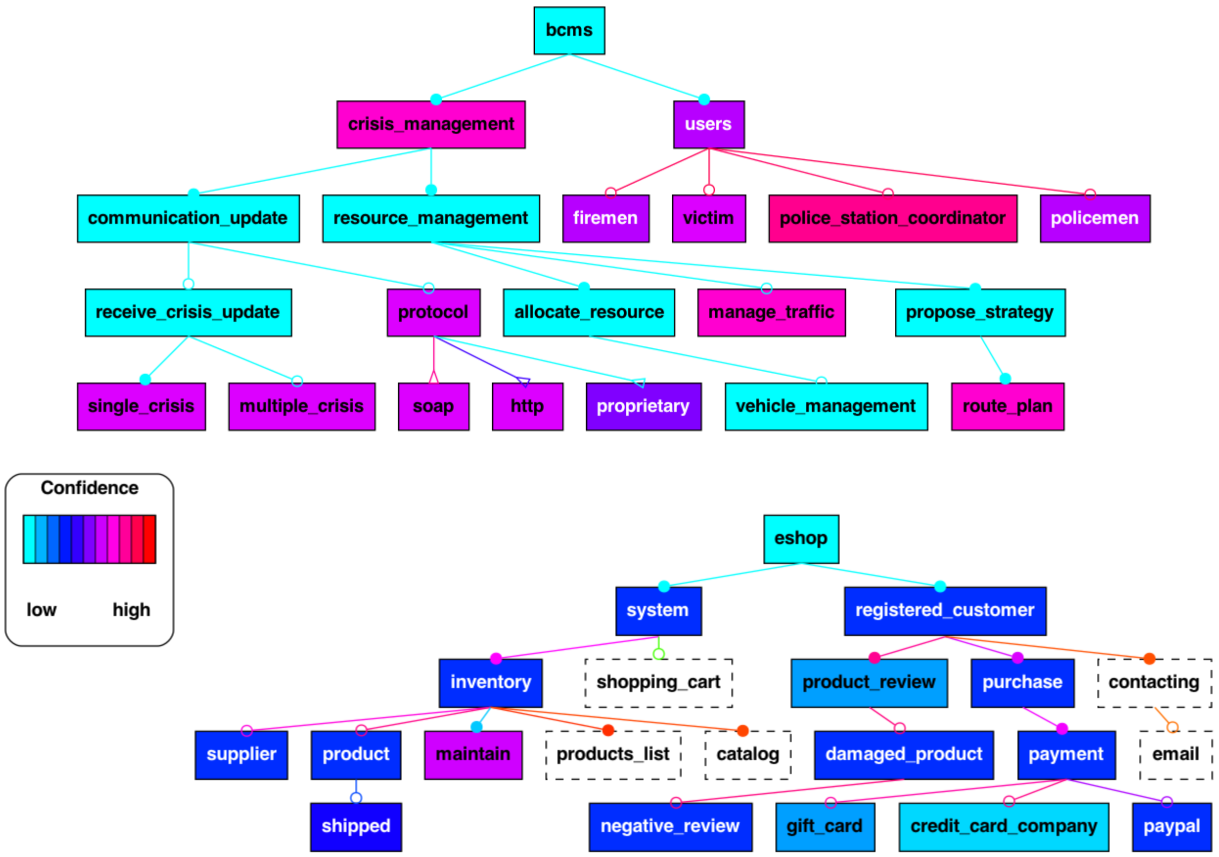


Figure 4.4: Examples of the heatmap visualization in two case studies: (top) Fragment of bCMS (S3); (bottom) E-Shop (S4).

Evaluating the relationship type.

The identification of the relationship type may require a more complex analysis, considering its semantics, the use of determiners (all/some), conjunctions (and/or) or the use of modal verbs (may/might/must), *e.g.* [177].

Given that our visualization focuses on describing the strength of the relationship between two features, we have not aimed to explore new techniques for evaluating the relationship type. Instead we use a heuristic for identifying mandatory versus optional features. *The text documents of a product line consists of a set of product specifications. A feature is mandatory if the feature (or a synonym) is referenced in all documents. If not, then we assume it is optional.*

4.3 Heatmap visualization

In order to graphically visualize and evaluate the validity of the information about the correctness of any given FM, we will paint the FM as a heatmap. Each feature and relationship in the FM will be given a color according to the probability score computed by our predictor models, from blue (low, *i.e.*, less probable) to red (high, *i.e.*, more probable). In this way, designers can quickly assess which parts of the FM are backed by the information in the textual specification and which are not.

```
Input  : A set of product specification documents $\mathbb{D}$
Output: A Feature predictor model ($\mathbb{M}$) and a Relations predictor table ($\mathbb{R}_T$)
Function FeaturesPredictor($\mathbb{D}$):
    foreach $d \in \mathbb{D}$ do
        ▷ Extract text from file
        T ← ExtractText(d)
        ▷ remove symbols and special characters
        ▷ remove unnecessary spaces
        ▷ apply sentence tokenize
        $\mathbb{S}$[] ← PreProcessText(T)
        $corpus$ ← $corpus$.append(GenerateLabeledText($\mathbb{S}$))
    end
    ▷ Preparing training data
    F[] ← ReadManualFeatures()
    ▷ logistic regression training data
    LR ← GenerateTrainingData($corpus$)
    ▷ compute coefficients
    Coeff ← ComputeLogisticRegressionCoefficients(LR)
    ▷ Prepare test data
    TD ← GenerateTestData($\mathbb{S}$)
    ▷ Repeat train and test until an acceptable misclassification rate
    FeaturePredictorModel ← Model(FinalLogisticRegression)
    $\mathbb{M}$ ← FeaturePredictorModel
Function RelationsPredictor($corpus$):
    ▷ Compute tf-idf scores for all distinct words
    $\mathbb{S}_{tfidf}$ ← GetTfidfAndSort($corpus$, 'desc')
    ▷ Select Root feature word
    $\mathbb{F}_{root}$ ← GetRootFeatureWord($\mathbb{S}_{tfidf}$)
    ▷ Get all the noun and verb based phrases with an arbitrary n-gram sequences
    $CandidateFeatures$ ← GenerateNounVerbPhrases($corpus$)
    foreach cf ∈ $CandidateFeatures$ do
        ▷ Count the number of phrases separating the phrase and the nearest phrase
          with the root feature word
        $dist_{cf}$ ← ComputeDistance(cf, $\mathbb{F}_{root}$)
        ▷ Store all distance values for all phrases
        $all\_dist$.append($dist_{cf}$)
    end
    ▷ Normalize all distance values and generate the scores table
    $\mathbb{R}_T$ ← GenerateDistanceScoresTable($all\_dist$, $CandidateFeatures$)
```

Algorithm 5: FMEvaluator model generator.

To create this heatmap of the FM, our approach uses the Graphviz graph visualization package[4]. To this end, our approach generates a description of the FM in the `dot` textual notation used by Graphviz. The features and relationships become vertices and edges in the graph, and the probability score from our analysis of the textual specification is translated into a color in the blue-red scale. Then, Graphviz prepares the layout of the features and creates the colored heatmap that is presented to designers.

Figure 4.4(top) shows an example of this heatmap visualization. It depicts a fragment of the FM defined by the domain expert for the bCMS case study (**S3**), a crisis management system. We notice that most leaf features from the FM have a high confidence, such as *victim*, *route_plan* or *soap*. Meanwhile, most internal features have a low confidence and possibly arise from the expert's knowledge of the domain, rather than the contents of the document. Nevertheless, internal features such as *protocol*, *users* or *crisis management* play a relevant role in the textual specification.

In addition to visualizing the correctness of the FM, it is also possible to visualize its completeness in a similar manner. The main idea is to show, along with the features in the FM, those concepts that play a relevant role in the textual specification but are not included in the FM. These "candidate" features are shown in the position where they best fit in the current FM, as a suggestion about potentially missing features. Visual clues identify these as "candidates" in contrast with real features. This information can be shown in the heatmap alongside the correctness information. An example of this visualization is shown in Figure 4.4(bottom) for the FM extracted in the E-Shop case study by the tool FeatureX [177]. This heatmap includes the suggested missing concepts, depicted as dashed rectangles with a white background. Notice that some of the identified features (*shopping cart*, *e-mail* and *catalog*) are reasonable suggestions in an e-commerce domain.

We consider that this heatmap visualization is very informative to designers and can provide useful feedback about the alignment of the FM and the textual specification it is based on. Another advantage of this type of depictions is that it is not tied to a specific method for computing a confidence score: if we use a different approach for measuring confidence, we can show its results using this visualization. A known shortcoming of this visualization is the limitation with respect to depicting confidence for relationships. Remember that there are two dimensions of confidence regarding relationships: (a) whether two features should be connected in the FM and (b) whether the type of relationship (*e.g.*, mandatory or optional) in the FM is correct. Given that we are only using color to classify relationships (to keep the visualization easy to understand and avoid information overload), it can only show one dimension of confidence, or a weighted combination of both dimensions.

Finally, the designer might want to understand the reasons that motivate the confidence level of a particular feature. For instance, she may be interested in knowing why a concept has been proposed as a relevant missing feature in the FM or why two features of the FM have a different level of confidence. In order to explain these decisions, we propose to augment the heatmap visualization with links to the relevant parts of the textual requirements. Thus, the rectangle for a feature *F* would include a link to a separate document containing a list of sentences of the textual specification that refer to *F*, either using the same terms literally or using synonyms. This specific functionality is not implemented in the current version of the tool but is considered as a future extension.

4.4 Results and discussion

In this section we start by validating the accuracy of our evaluation method (Subsection 4.4.1). Then, we describe the application of our method to the four case studies (Subsection 4.4.2). Finally, we identify the major threats to validity (Subsection 4.4.3).

4.4.1 Validating the evaluation approach

In order to determine whether our FM evaluation method accurately classifies concepts as relevant features and relationships, we need to validate the confidence scores it computes.

[4]https://graphviz.org

Total number of observations: **n**	**Predicted** *Feature*	**Predicted** *Non-Feature*
True *Feature*	True Positive (tp)	False Negative (fn)
True *Non-Feature*	False Positive (fp)	True Negative (tn)

$$n = tp + fp + tn + fn$$

Table 4.7: Confusion matrix.

One way to control the classification process is by setting a suitable value for the *cut-off probability*, the threshold value beyond which we consider the phrase to be classified as a feature. By decreasing or increasing this value, we can control the number of candidates, and thus, the rate of misclassified phrases.

To study the performance of our binomial classification model we have used a *confusion matrix*. A confusion matrix is a simple table that shows the total number of *True Positive* (tp), *True Negative* (tn), *False Positive* (fp) and *False Negative* (fn) as shown in Table 4.7. Using the confusion matrix, we have generated the classification report with the standard *precision*, *recall* and F_1*-score* measures defined as usual:

$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}},\ \text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}},\ F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

We consolidated the findings on the changes of the misclassification rate based on the changing cut-off probability values and plotted the *empirical receiver operating characteristics (ROC) curve* [46] which displays the true positive rates against the false positive rates for all tested cut-off probabilities. The higher the area-under-the-curve (AUC), the better test accuracy which in turn indicates better model performance.

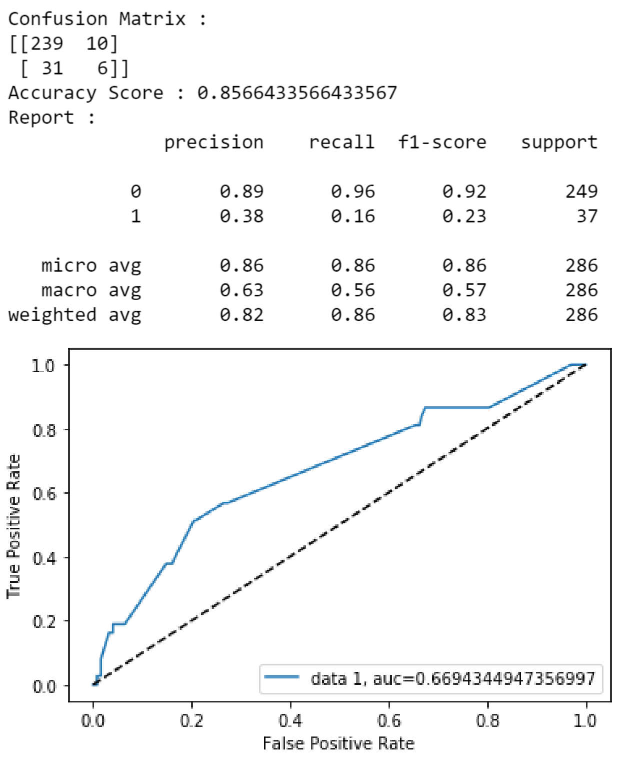
When comparing the plots for the different case studies we observed that the smaller datasets had a better model performance. The models for studies **S1** and **S2** have delivered a performance in the range of poor-fair, whereas the smaller studies **S3** and **S4** have fair-good performance range. We have set the performance ranges based on the regular ranges used by other similar studies, which assign a rating of “poor” for any $auc < 0.7$, “fair” for $0.7 < auc < 0.8$ and “good” for $auc > 0.8$. The confusion matrix and the plot for the case study **S3** is shown in Figure 4.5.

For the larger datasets, the generation of the predictor model required a lot of time and resulted in a variety of data related issues. When looking into the characteristics of these documents that differentiates them during the classification, it was observed that:

- Text with embedded picture representation of symbols and images is not processed correctly and gets classified as features. An example of such problems is shown in Figure 4.6.
- A multi-column specification text with more than 2 columns is resulting in a large number of features due to improper text extraction from the PDF files.

Table 4.8 shows the structural differences in the content. We have observed that documents which had repeated words and phrases over multiple consecutive sentences, showed natural coherence between these sentences and guided the predictor for narrowing down on the right set of feature terms based on this simple POS tag related probability scores.

In our initial attempts, it was also observed that the feature predictor performance for larger specifications based on POS tags is contributing only for a 30% of features prediction. There were a large number of phrases under false negative for all the different cut-off values. This indicated that the model needed to be improved by adding additional dependent variables that could enhance the predictability. The text characteristics that were not being accounted for are:

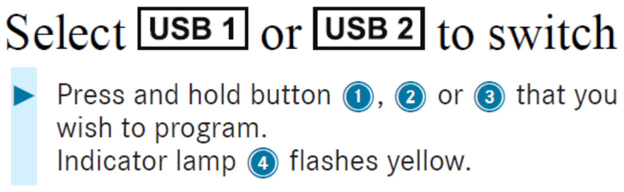


```
Confusion Matrix :
[[239  10]
 [ 31   6]]
Accuracy Score : 0.8566433566433567
Report :
              precision    recall  f1-score   support

           0       0.89      0.96      0.92       249
           1       0.38      0.16      0.23        37

   micro avg       0.86      0.86      0.86       286
   macro avg       0.63      0.56      0.57       286
weighted avg       0.82      0.86      0.83       286
```

Figure 4.5: Confusion matrix and ROC curve for the case study S3 (bCMS).

Spec	Structure
Honda	3 column page, with sec, subsec & bullet points
Toyota	Full page, with sec, paragraphs & bullet points
Mercedes	3 column page, with paragraphs & bullet points
Audi	2 column page, with sec, paragraphs & bullet points
Infinity	3 column page, with sec, subsec & bullet points
Mazda	Full page, with sec & paragraphs

Table 4.8: Comparing sections on 'Child safety' for the case study S1.



Select USB 1 or USB 2 to switch

▶ Press and hold button 1, 2 or 3 that you wish to program.
Indicator lamp 4 flashes yellow.

Figure 4.6: Sample image and symbols embedded in the textual documents from case study S1 (mid-size SUV).

1. Subject and object in a sentence
2. Cross references to a previously mentioned subject/object using pronouns
3. Modal verbs POS tags

Thus, we included the above characteristics by extending the POS tags to include *pronouns* and *modal verbs* and

Tachometer	Braking system
TPMS Indicator Tachometer	Mitigation Braking System
Indicator Tachometer Speedometer	Collision Mitigation Braking
Tachometer Speedometer U.S.	Braking Brake System
speedometer tachometer odometer	Steering Wheel Braking
Interface Show Tachometer	Apply engine braking

Table 4.9: Feature terms representing the same product feature.

improved the TF-IDF score by introducing a subject object reference counter that keeps track of the total number of times the same subject and object have been talked about in the documents at multiple locations using pronouns. We regenerated the predictor model of the very large case study **S1** which has more than 5.000 sentences in each specification document. After that, the predictor performance increased by 20% with $auc > 0.7$. This clearly indicates that the adjustments in the ML encoding made in accordance to the style of the language used in the text have a positive impact on the predictive power of the ML model.

At this stage we have both the feature and relations predictors. Since the feature terms will not be identical in all the documents due to differences in the language and the vernacular specific to each product, we group them together and mark them with labels **F**<id>, representing a feature term and id referring to a unique identifier, thus creating a dictionary of distinct feature terms. This way it is easy to detect similar feature terms with same meaning present in the different product specification documents and get the appropriate POS tags information to compute the feature probability and its relations probability with another feature term.

	FM1 generated by FeatureX [177]						FM2 generated manually/from literature					
	Features			Relationships			Features			Relationships		
Case study	**Precision**	**Recall**	F_1	**Precision**	**Recall**	F_1	**Precision**	**Recall**	F_1	**Precision**	**Recall**	F_1
S1	0.62	0.55	0.58	0.47	0.55	**0.51**	0.72	0.58	**0.64**	0.81	0.37	0.51
S2	0.88	0.75	**0.81**	0.58	0.45	**0.51**	0.54	0.45	0.49	0.57	0.43	0.49
S3	0.91	0.62	0.74	0.67	0.89	**0.76**	0.76	0.74	**0.75**	0.64	0.53	0.58
S4	0.82	0.77	**0.79**	0.71	0.48	0.57	0.75	0.68	0.71	0.94	0.55	**0.69**

The best F_1-score for features and relations in each case study is highlighted in bold.

Table 4.10: Comparison of two FMS using the feature and relationship predictors.

4.4.2 Results of the case studies

In order to evaluate the FMs against its reference specification texts, we have listed its features and relationships, together with the confidence scores produced by our method. Table 4.4 shows an example of this information. Then, a Java program transforms this tables into a text file in `dot` format depicting the heatmap which is used by the Graphviz tool to produce the output image[5].

Case study **S1**, being the largest of the four, had relatively poor FM validation results overall when compared to the other smaller studies **S3, S4**. For instance, it identified close to 1.000 distinct candidate features. Among those, only 50% were being used in one or more FMs that were being evaluated. This is mainly attributed to the low accuracy of the predictor models, which suffers from low performance due to several missing considerations during the ML model building as explained in subsection 4.2.1.

One of the applications of our method is the comparison among several candidate FMs for the same textual requirements. In Table 4.10, we have used our validation approach to perform this comparison in our four case

[5]The Java source file, the output tables and the heatmaps are available in the `src` and `results` folders of the Github repository (see Section 4.1.1).

studies. On one hand, we have used the FMs generated by FeatureX [177], and on the other, FMs extracted using other approaches (see Table 4.3).

This comparison is not intended to show whether FeatureX is better than other methods: as FeatureX includes similar heuristics to the ones used in this paper, this evaluation would be biased in its favor. Nevertheless, we think that this comparison can still reveal useful insights. For instance, in some case studies (such as **S1**) both FMs exhibit a comparable accuracy. However, in several examples FeatureX achieves a lower precision but slightly better recall in predicting relationships. This probably hints that other methods are more conservative when predicting relationships, preferring less false positives at the expense of potentially more false negatives. Knowing this information, designers can select the most suitable FM extraction method according to their priorities, either minimizing false positives (proposed features/relationships are more likely to be correct) or minimizing false negatives (it is less likely to miss a relevant feature/relationship).

4.4.3 Threats to validity

Internal validity.

The proposed approach for computing the confidence score for features and relationships uses mainly syntactic information (POS tags) and information about the frequency of terms. Even though some semantic information is considered (synonyms), the accuracy of the approach could be improved by combining this information with more sophisticated approaches (*word embeddings*, domain ontologies). Obviously, the potential gains in accuracy will require devoting more computational resources.

Another threat for this study is related to the limited training data available for the supervised learning used in the prediction of features. This, in turn, affects the accuracy of the prediction model. To alleviate this problem, we used Lasso [138] in order to accommodate the smaller set of data points available for training in comparison to the number of available predictor variables. Nevertheless, the impact of limited data on the predictive power of the model is not negligible. In fact, it could possibly lead to a number of omissions of significant feature terms. This can be minimised by including an unsupervised learning approach like k-means clustering and comparing the generated clusters with that of the identified features and feature groups. This approach is one of the considered approaches for future work.

External validity.

Our approach supports the analysis of textual information. Nevertheless, it cannot handle other types of information that may appear in requirements documents such as tables, graphs or images and may have problems with formatted text (*e.g.*, bullets or tabulated text). This was evidently experienced with the case study **S1**. Also for instance, it cannot take advantage of *product comparison matrices* [163], if available.

Regarding its output, the proposed method can evaluate the correctness and completeness of features and validate the correctness of the proposed relationships. Nevertheless, it has several shortcomings when it comes to evaluating relationships.

- It does not attempt to establish the level of confidence on the *type* or relationship, only on the strength of the connection between two features. We have only described heuristics for the identification of *mandatory* and *optional* relationships between features. Hence, this analysis will not validate other relationship types in more complex FMs, such as *exclusive-or (xor)* or *cross-tree constraints*. Other research works (*e.g.*, [119]) have also faced challenges in the accurate detection of these types of constraints. Even though mandatory and optional are the fundamental types of relationships to build a simple FM, we aim to extend our work for the detection of other relationships in the future.

- So far our approach does not propose candidate missing relationships like it does for relevant missing features. Given that our analysis receives a FM as input (which is already shaped as a tree), these missing relationships would have to be cross-tree constraints, *e.g.*, *requires* or *excludes*.

4.5 Summary

In this chapter we introduced the necessity of having a generic FM validation and verification capability in order to be able to evaluate the relative benefits of an FM over another FM, produced for the same textual specification. In the course of explaining this motivation, we identified that such an unbiased process can be even used for providing feedback to the RE practitioner regarding the correctness and completeness of the specified product line requirements. We then go on to explain what are the inputs to this validation method and the expected outputs. The subsections of this chapter were divided in terms of the 2 different elements that constitutes an FM which are the features and the feature relationships. In each of the sections 4.2.1, 4.2.2 we provided a detailed account of the heuristics involved in the formulation of the validation approach for the 2 constituting elements of an FM and the algorithms used for the same. A discussion on the heatmap visualization and the guidelines for interpreting the results was elaborated in section 4.3. We then concluded the chapter with a detailed analysis of the evaluation results and the threats to its validity in section 4.4.

Chapter 5

Automated FM Analysis

Feature Models (FMs) are a mechanism to model variability among a family of closely related software products, i.e. a software product line (SPL). Analysis of FMs using formal methods can reveal defects in the specification such as inconsistencies that cause the product line to have no valid products.

A popular framework used in research for FM analysis is Alloy, a light-weight formal modeling notation equipped with an efficient model finder. Several works in the literature have proposed different strategies to encode and analyze FMs using Alloy. However, there is little discussion on the relative merits of each proposal, making it difficult to select the most suitable encoding for a specific analysis need. In this paper, we describe and compare those strategies according to various criteria such as the expressivity of the FM notation or the efficiency of the analysis. This survey is the first comparative study of research targeted towards using Alloy for FM analysis.

This review aims to identify all the best practices on the use of Alloy, as a part of a framework for the automated extraction and analysis of rich FMs from natural language requirement specifications.

5.1 Introduction

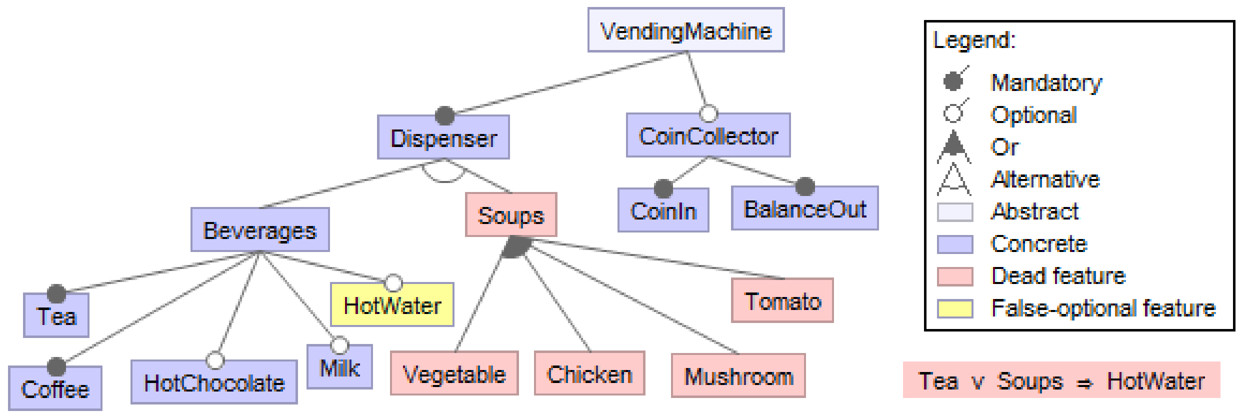
Many software systems are not one-of-a-kind, but a family of related products in a given application domain called *software product line* (SPL). Products in a SPL can be differentiated by their *features*, defined as "increments in program functionality" [24] or "user-visible aspects or characteristics of the domain" [101]. Hence, modeling a SPL requires describing the features and the *relationships* among them such as dependencies or incompatibilities. That is, unlike traditional models of information systems (considering a single product), models of SPLs capture the variability among a family of products.

Feature Models (FMs) [27] are a popular family of notations, capable of describing complex SPLs. FMs can be constructed for a given application domain, using a methodology known as *Feature-Oriented Domain Analysis* (FODA) [101]. The output of this process is a FM, a diagram describing a complete SPL as a set of features and relationships. The diagram looks like a connected graph with the boxes/nodes in the diagram representing *Features*, edges representing relationships, and cross-tree constraints expressed as Propositional logic formulas. A product becomes a *configuration* of this diagram, i.e. a subset of features that satisfies all the relationships and constraints. Throughout this paper we will use *product configuration*, *product* and *configuration* with interchangeable meanings.

Given an FM, it is important to detect potential *defects*, e.g. no valid product, *dead features* (that cannot appear in any product), *false optional features* (that are required in every product), etc. For instance, the FM illustrated in Fig. 5.1 is the model of a vending machine product line. A cross–tree constraint makes *HotWater* a required feature when the features *Tea* or *Soups* are selected in the FM. This makes *Soups* unselectable in any configuration as the

Dispenser can either have *Beverages* or *Soups* but never both. Meanwhile, optional feature *HotWater* is required by mandatory feature *Tea* (making it mandatory).

Figure 5.1: The vending machine FM created in FeatureIDE [103].



There are several techniques and tools that have been considered for this purpose, as surveyed in [27]. Among these tools, a popular choice is Alloy [92], a light-weight formal modeling language with an efficient model finder. The generic semantics and versatility of the relational logic used in Alloy together with a fully automated analysis [187] makes Alloy an ideal choice for building complex automated tools based on it. Other options include *Description Logic based* and *Constraint programming based* approaches as illustrated in the survey by Benavides et al. [27].

The differences among the existing research works using Alloy lie in the formalization of the FM. Different formalizations provide unique benefits, e.g. readability or efficiency of the analysis. However, there is a lack of information regarding the relative merits of each proposal, making it difficult to select the most suitable one for a specific type of analysis. In this paper, we review the state of the art in the analysis of FMs using Alloy. Our goal is to characterize the strengths and weaknesses of various Alloy encoding approaches from several perspectives such as expressivity of the FM, efficiency of the analysis, existence of tool support, etc.

We present our findings over the following sections. Section 5.2 briefly discusses the related surveys and technical details about the Alloy analyzer and the FM analysis operations. Section 5.3 presents the selection criteria used to locate the relevant works and the assessment criteria used to evaluate them. Then, Sections 5.4 and 5.5 discuss the results of the evaluation. Section 5.6 describes the major challenges identified in our survey with an indication of future research directions and Section 5.7 summarizes the conclusions of this survey.

5.2 Background

In this section we briefly discuss the existing surveys on this research subject. Then, we present a short summary of the required technical know–how about Alloy and FM in general with an introduction to the benchmark FM. The section concludes with a short overview of the specific FM analysis operations that are considered by various research papers.

5.2.1 Related surveys

There are 3 surveys [27, 115, 184] which have systematically studied the state-of-the-art on FM analysis. Most of the high level findings related to tools support and capability of executing analysis operations have been documented in those surveys, but with no specific focus towards using Alloy as the analysis tool. Benavides et al. [27] have consolidated a complete list of analysis operations which are fundamentally required for a fully operational end-to-end FM analysis automated tool. We have used this list as our reference to evaluate the support levels extended by the different papers that are reviewed during this survey.

5.2.2 Benchmark Feature Model

For comparing the selected research works, a suitable FM has to be set as the benchmark. We have used the bCMS software product line [51, 160] (see Fig. 5.2) for this purpose. The bCMS-SPL is a case study defined for the Workshop on Comparing Modeling Approaches (CMA'2011). It describes a family of car crash emergency systems in a FM with 28 features and 30 relationships among features. Two of these relationships are simple (at most 2 features involved) cross–tree constraints.

5.2.3 Alloy semantics

In Alloy, a model is described as a collection of *signatures* (`sig`), which identify the potential types for objects. Signatures can have *fields* with values like references to other objects, sets of objects or mappings among objects. *Facts* (`fact`) are constraints that capture the well-formedness rules of the model. Then, it is possible to check two types of properties: *assertions* (`assert`), i.e. searching an instance that violates a condition, or *predicates* (`pred`), i.e. searching an instance that fulfills it. When defining complex facts, predicates or assertions, it is possible to use *functions* (`fun`) to reuse and encapsulate large subexpressions.

Properties can be expressed using a relational logic that combines features from first-order logic (quantifiers and Boolean operators), set operations (navigations through references and mappings) and the operators of relational calculus (e.g. computing the differences among two relations).

5.2.4 Analysis operations

In order to evaluate each encoding and the analysis operations supported by them, we have listed all the operations as described by Benavides et al. [27] across 2 tables: Table 5.1 (which lists out only those operations which are included in one or more research works considered in the survey) and Table 5.2 in Appendix 7.5 (which enlists all the remaining analysis operations which are significant as mentioned by Benavides et al. [27], but not covered by any research work, both among the ones considered in our review and in the larger space of research works

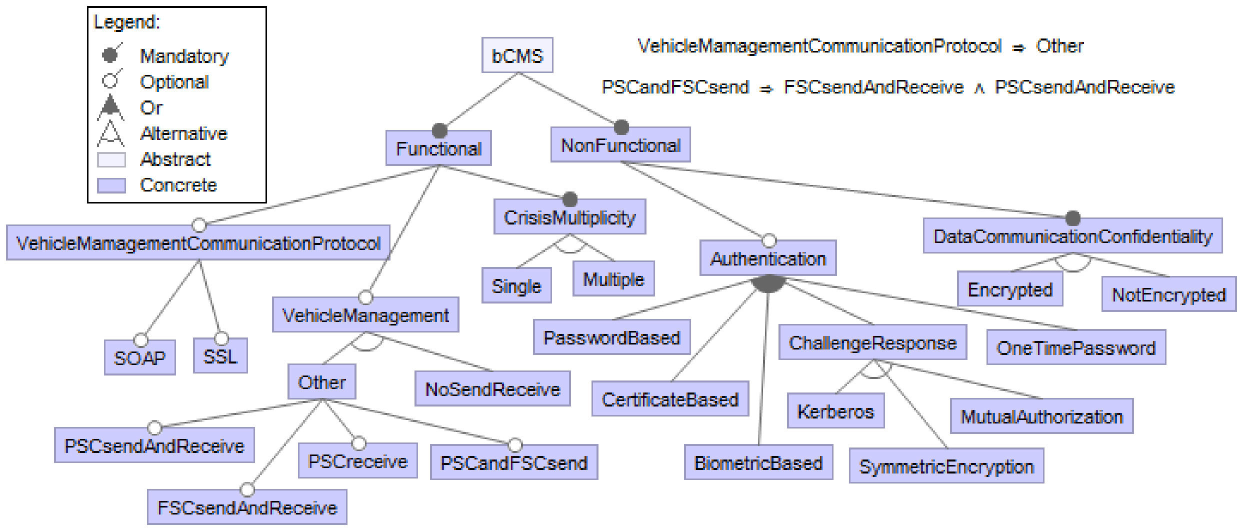


Figure 5.2: The bCMS FM [160] created using FeatureIDE [103].

related to FM analysis using Alloy as the model finder). More explanations on the importance and relevance of each analysis operation can be found in [27].

Id	Operation	Analysis operation description	Supported by
A1	Void FM	A Feature Model is void if it represents no products	[88, 144, 157, 200]
A2	Valid product	A product is valid if it belongs to the set of products defined by the FM	[2, 53, 76, 78, 88, 144, 157, 183, 200]
A3	All products	All valid products possible from the FM	[78]
A4	Filter	For a given partial product configuration this analysis will result in a list of valid products that can be generated from this partial configuration	[2]
A5	Dead features	A feature is dead if it cannot appear in any of the products of the SPL	[76, 78]
A6	Wrong cardinalities	These appear in cardinality-based Feature Models where cross-tree constraints are involved. A group cardinality is wrong if it cannot be instantiated	[2]
A7	Refactoring	An FM is a refactoring of another one if both represent the same set of products while having a different structure	[200]
A8	Commonality	This operation takes an FM and a configuration as input and returns the percentage of products represented by the model including the input configuration	[2]
A9	Variability factor	This operation takes an FM as input and returns the ratio between the number of products and 2^n where n is the number of features considered.	[101]
A10	Degree of orthogonality	This operation takes an FM and a subtree (represented by its root feature) as input and returns the ratio of the total number of products of the FM and the number of products of the subtree	[2]

Table 5.1: FM analysis operations commonly found across multiple research works.

Table 5.2 shows the remaining analysis operations that are important to be performed on FMs as indicated by Benavides et al. [27]. Table 5.3 shows the sources of the search results of all the papers that were found during the start of this survey on which we had further applied the inclusion and exclusion criteria to filter out the reviewed 9 papers.

5.3 Review method

The review begins by considering existing surveys (e.g. [27, 115, 184]) and searching the following terms using a logical *and* operator for the combination, in the abstract and keyword lists of the following scientific databases:

Search terms:	Feature Model, Alloy, analysis, software product line, verification
Data sources:	IEEE Xplore, ACM Digital Library, ISI Web of Knowledge

The search process is iterated on the list of bibliographic references in each of the identified papers, in order to identify any potentially missing references. Overall, this initial search produced 47 research papers. Fig. 5.3 presents a timeline of these references while Table 5.3 in Appendix 7.5 identifies their venue of publication.

In order to refine this collection of references, Table 5.4 describes the inclusion and exclusion criteria used to select only the specific works related to our survey interest. On one hand, we focus on works dealing with the analysis of FMs following the *Feature-Oriented Domain Analysis* (FODA) conventions [101]. On the other hand, methods that include information beyond Feature Models, e.g. behavioral models [64] or use-case models [4], are omitted. As it is shown in Table 5.4, we only consider papers using Alloy as the verification engine. Thus, we omit any paper that uses alternative formalisms (e.g. constraint programming or description logic) or any other propositional logic provers [27, 115].

After applying the inclusion and exclusion criteria from Table 5.4, the final set of selected papers consists of 9 research papers which qualified to be included for our detailed review process. These papers from 2005 until 2015 include: 6 papers from conference proceedings [53, 88, 144, 157, 183, 200], 2 from journal publications [2, 76] and 1 technical report [78].

Id	**Operation**	**Analysis operation description**
A11	Valid partial configuration	An incomplete product configuration which in this partial state does not introduce any contradictions and can be still extended into a complete product
A12	Count	The number of valid products possible from the FM
A13	Conditionally dead features	A feature is conditionally dead if it becomes dead under certain circumstances (e.g. when selecting another feature)
A14	False optional features	A feature is false optional if it is included in all the products of the product line despite not being modeled as mandatory
A15	Redundancies	An FM contains redundancies when some semantic information is modeled in multiple ways
A16	Explanations	This operation takes an FM and an analysis operation as input and returns information about the reasons of why or why not the corresponding response of the operation?
A17	Corrective explanations	A corrective explanation provides suggestions to solve a problem, usually once this has been detected and explained
A18	Generalization	An FM, **F**, is a generalization of another one, **G**, if the set of products of **F** maintains and extends the set of products of **G**
A19	Specialization	An FM, **F**, is a specialization of another one, **G**, if the set of products of **F** is a subset of the set of products of **G**
A20	Arbitrary edit	There is no explicit relationship between two FMs
A21	Optimization	This operation is chiefly useful when dealing with extended FMs where attributes are added to features. In this context, optimization operations may be used to select a set of features maximizing or minimizing the value of a given feature attribute
A22	Core features	This operation takes an FM as input and returns the set of features that are part of all the products in the SPL
A23	Variants	Variant features are those that do not appear in all the products of the software product line
A24	Atomic sets	An atomic set is a group of features (at least one) that can be treated as a unit when performing certain analyses. The intuitive idea behind atomic sets is that mandatory features and their parent features always appear together in products and therefore can be grouped without altering the result of certain operations
A25	Dependency analysis	This operation takes an FM and a partial configuration as input and returns a new configuration with the features that should be selected and/or removed as a result of the propagation of constraints in the model
A26	Multi–step configuration	A multi–step configuration problem is the process of producing a series of intermediate configurations, i.e. a configuration path, going from an FM configuration to another
A27	Homogeneity	A more homogeneous FM would be one with few unique features in one product (i.e. a unique feature appears only in one product) while a less homogeneous one would be one with a lot of unique features.
A28	ECR	Extra Constraint Representativeness - This operation takes an FM as input and returns the degree of representativeness of the cross-tree constraints in the tree
A29	LCA	Lowest Common Ancestor - This operation takes a Feature Model and a set of features as input and returns a feature that is the lowest common ancestor of the input features
A30	Root features	This operation takes an FM and a set of features as input and returns a set of features that are the root features in the model

Table 5.2: Analysis operations not discussed by any Alloy related research work till date.

5.3.1 Threats to validity

Before we go on further with the details of this literature survey, we would like to identify a set of potential weaknesses in our review process:

- We have not included all possible list of literature databases, e.g. DBLP or Google scholar. On the other hand, the considered surveys [27, 115, 184] are very recent (2010, 2014, 2015), and so they offer a good coverage of recent works.
- Other combinations of equivalent terms, such as *"variability modeling"*, *"automated FM analysis"*, etc. have not been considered. As a result of our search, and to the best of our knowledge, these nine selected papers are the most relevant for the topic addressed in this review.
- Marcilio et al. [139] and Liang et al. [124] have been successful in evaluating SAT-based FM analysis on very large FMs such as the FM of Linux kernel (5814 features), and found that the results were promising in terms of scalability of the FM and solver execution times. All such evaluations were done on FMs represented in Conjunctive Normal Form (CNF) which can be fed as input to SAT solvers such as the SAT4j standalone SAT

Source	**No:**	**Details**
Conferences	**36**	Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ) Asia-Pacific Software Engineering Conference (APSEC) Int. Conf. on Advanced Information Systems Engineering (CAiSE) Int. Conf. on Coordination models and languages (COORDINATION) European Conf. on Modelling Foundations and Applications (ECMFA) 15th Int. Conf. on Conceptual Structures (ICCS) Int. Conf. on Engineering of Complex Computer Systems (ICECCS) Int. Conf. on Formal Engineering Methods (ICFEM) ACM Symposium on Applied Computing (SAC) Brazilian Symposium on Software Engineering (SBES) Int. Conf. on Software Product Lines (SPLC)
Journals	**15**	ACM Computing Surveys ACM Transactions on Software Engineering and Methodology Artificial Intelligence for Engineering Design, Analysis and Manufacturing Communications of the ACM IET Software Information and Software Technology Journal of Computational Science Journal of Information Systems Journal of Knowledge Engineering and Soft Data Paradigms Journal of Logical and Algebraic Methods in Programming Journal of Software Journal of Systems and Software Journal of Universal Computer Science
Workshops	**6**	Annual IEEE Software Engineering Workshop Int. Workshop on Analysis of Software Product Lines Int. Workshop on Variability Modelling of Software-Intensive Systems First Alloy Workshop
Others	**15**	PhD Thesis Technical reports Newsletters

Table 5.3: Source of the papers obtained in the initial search.

Inclusion criteria	**Exclusion criteria**
1. Discusses FM properties that can be analyzed using Alloy	1. Does not consider FODA-style FMs describing a SPL
2. Focuses on Alloy-based Feature Model analysis of SPLs	2. Does not consider the analysis of the FM
3. Introduces a new FM encoding or improve/extend existing encoding to include a new analysis	3. Does not use Alloy as the underlying verification engine

Table 5.4: Selection criteria used in this literature review.

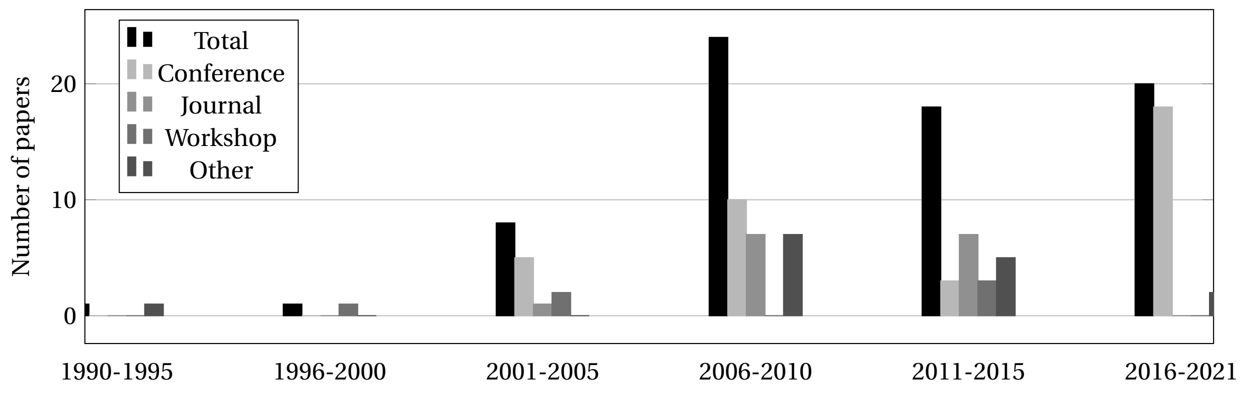


Figure 5.3: Collected papers statistics.

solver. In this paper we could not use such very large FMs in CNF format mainly because we lacked the tool support for converting such CNF notations into Alloy specifications. Moreover it would be difficult to validate the results of analysis of various FM analysis operations on such large FMs. Therefore the scalability of Alloy encoding for larger FM specifications and its performance with respect to analysis execution time for such large FMs has not been covered in this paper.

- We did not quantify the encoding performance in terms of the time taken to manually convert any FM to Alloy specification (ease of use).
- As there is only one case study the respective merits of each encoding may vary on different FMs.

5.4 Evaluation

Our review was intended towards answering the following research questions for each selected work:

1. What is the goal of FM analysis, i.e. is there a motivating scenario?
2. What degree of expressivity is supported in the FM, i.e. what kind of relationships are allowed?
3. What kind of analysis can be performed on the FM? What is the efficiency of those analysis?
4. What kind of tool support is available for generating the Alloy specification automatically?

5.4.1 Formalization strategies

This subsection provides an overview of the general formalization of FMs in the Alloy notation, in order to facilitate the comparison among the different formalization strategies. All selected works use common conventions in their Alloy encodings. For instance, all works share common concepts among the signatures they declare: `Feature`, `Relationship` and `Configuration`. A `Feature` represents a major functionality of a software product. `Relationships` are connections between features. Most connections are between a complex feature (*parent*) and its set of sub-features (*children*), defining a tree-like structure. However, it is possible to define *cross-tree* relationships connecting arbitrary features, for example, stating that a feature requires another feature. Finally, a `Configuration` (also called `Concept` in [200]) is a set of features that define a product of an SPL.

5.4.2 Goal of analysis

The goal of FM analysis in general should be to identify all sorts of problems (like dead features, presence of false optional features etc.) in the FM and suggest corrective measures to make the FM valid. For example, if we consider the *Vending machine* FM shown in Fig. 5.1 it has both *dead features* and *false optional features.* Both these problems can be resolved if a *'Refactoring' (A7)* FM analysis operation is applied and the structure of the FM is changed to Fig. 5.4. This FM is found to be completely valid with 162 possible product configurations (as calculated by FeatureIDE [103]). In our survey, every selected research work was targeting to emphasize on some or the other analysis operation on the Feature Model. The most common analysis operations are: check if an FM is void (A1) [88, 144, 157, 200], finding invalid/valid configurations (A2) [2, 53, 76, 78, 88, 144, 157, 183, 200] and finding dead features (A5) [76, 78]. The support extended for each analysis operation by the different selected papers in our review, has been marked under the **Supported by** column of Table 5.1.

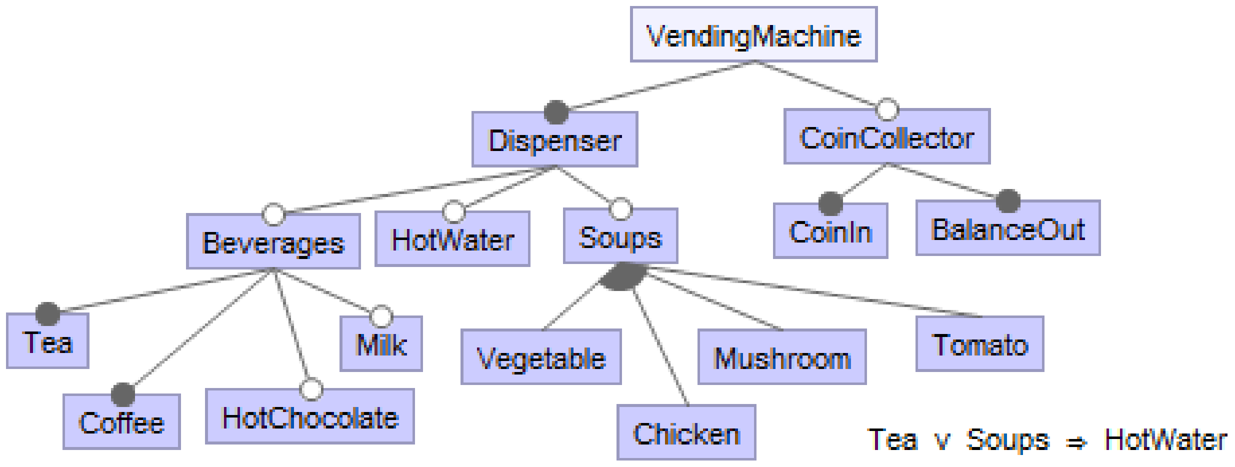


Figure 5.4: The corrected vending machine FM created in FeatureIDE [103].

5.4.3 Degree of expressiveness

Expressivity has been evaluated in terms of the ability to encode all possible relationships and constraints between features in the Alloy notation. We have consolidated all such relationships and constraints and tabulated them under the **Relationships** sub–table of Table 5.5. This table shows that some papers lack support for some types of relationships in the FM. This was purely a choice of the respective authors, considering only the properties of interest and a trade-off between expressivity and efficiency.

5.4.4 Efficiency of the analysis

To measure the efficiency, we have compared the execution time for various analysis operations using version 4.2 of the Alloy Analyzer (using the default SAT4j solver [34]). We have used the Alloy encoding proposed by each paper to model the bCMS FM and then performed different analysis operations. Also note that we have not included any evaluation of the analysis performance compared to other verification engines beyond the Alloy Analyzer.

The details of the execution time in milliseconds can be found in Table 5.5.

5.4.5 Available tool support

Regarding the source of the FMs, there is no standard textual format to represent FODA style Feature Models. Hence, all works start from a manually created FM. In some cases [2, 53, 76, 78, 88, 183], the FM is a formal specification created specifically for analysis purposes. In others [144, 157, 200], the FM is the manual translation of the requirements document of a SPL.

Regarding the generation of the Alloy specification from the FM, the inference was again that there was a lack of tool support: only Nakajima et al. [144] have a prototype tool called FD-Checker that can generate the Alloy specification from a propositional logic formula. All the other papers do not have any kind of tool support, so the

Alloy formalization needs to be generated manually. In our opinion, this is a significant finding in this literature survey. This has also made it very difficult for us to evaluate the research results of the bCMS case study, as it involved manual translation of FM specifications to the respective Alloy encoding.

For each reference (1$^{\text{st}}$ row) in Table 5.5 the set of supported relationships in the FM, i.e. their expressivity (2$^{\text{nd}}$ row), the availability of tool support for automatically generating the formal Alloy specification (3$^{\text{rd}}$ row) and the set of analysis operations offered by each method and its efficiency (4$^{\text{th}}$ row) in terms of the execution time for each analysis, has been summarized.

5.5 Discussion

In the following subsections we discuss the details of each encoding approach followed by the different research papers considered in this review in chronological order. Each research paper corresponds to one approach, also referred to as one strategy.

5.5.1 Alloy encoding and analysis of bCMS FM

In this subsection we include all major findings related to each selected paper and would provide the information based on the following criteria:

- Overview of the encoding strategy used - Specialty of the encoding and identifying the distinguishing aspect of the strategy.
- Strengths and limitations of the encoding - Comparison with the encoding described by other research papers.
- Ease of reproducing the research results - Discusses the effort to encode the bCMS FM and perform the analysis operations.
- Major technical roadblocks - Discusses the technical limitations and roadblocks that were encountered while reproducing the analysis results on the encoded bCMS FM.

Wang et al. [200]: The encoding is thoroughly described in the paper. This is the only encoding which is able to perform a check for semantic equivalence between an FM and its refactored model. Unlike other papers this approach did not discuss about finding dead features, finding all possible product configurations or provide explanations for analysis results. The encoding was easy to reuse for the bCMS FM and the results stated in the paper were completely reproduced without any technical issues.

Gheyi et al. [78]: Gheyi proposed two theories for Feature Model analysis, the R-Theory and the G-Theory. The R-Theory defined a FM as a set of features. The G-Theory was a concise approach towards creating reusable constructs that are generically implemented in order to allow easy FM specification in Alloy. The encoding was more mature than the specifications of Wang et al. [200] in terms of expressivity as it was the only encoding among the surveyed papers that supported finding all valid products from any FM. It was easy to generate Alloy specifications for bCMS FM. FM analysis such as checking for valid/invalid product configurations, identifying dead features and collecting all valid product configurations was performed on bCMS FM and the results are presented in Table 5.5. A major roadblock in the implementation of these theories in Alloy for the bCMS FM was the use of logical constructs (recursion) that are unsupported by the current version of the Alloy notation. This was resolved when we directly contacted the authors and got a solution from them.

Tanizaki et al. [183]: The encoding described in this paper is the most difficult to understand as the semantics used is very different from what we have seen in all the other approaches. Here the concept of a Feature-Model-Connection has been introduced in order to support traceability of configuration changes in a software system. The time taken to specify the bCMS FM was thrice the time what we took for the other encodings. The main roadblock was in understanding the encoding in terms of FM analysis as they were mainly targeted towards backtracking FM changes to identify any new invalid FM state. This is more or less a combination of analysis properties

A12 & A13. But we were unable to reproduce the research results. This was mainly because of a lack of clarity in the encoding (too many signatures with no reusable functions or predicates which can aid in specifying bCMS FM in Alloy) encoding and incomplete encoding information in the paper (finding a product instance using the address book example from the paper was not directly reproduced because of the missing encodings which the authors have explicitly mentioned of having excluded it from the paper).

Nakajima et al. [144]: The most remarkable aspect of this encoding is that the features and its relationships are all encoded in terms of propositional logic formulas (using many `sig` and `fact`), rather than using the `fun` or `pred` constructs of Alloy. This makes the encoding effort for very large FMs a time consuming and error-prone process. The major strength of this encoding lies in its direct and simple to understand encoding semantics. But unfortunately this approach has a very long running time for the *A2* analysis operation (see Table 5.5) even for such a small sized FM. Unlike Gheyi et al. [78], this encoding is not very compact while it is more flexible and potentially easier to be generated **automatically** for any given FM. For this reason they had introduced a tool called FD-Checker which could have been useful to convert propositional logic formulas to Alloy specifications of the FM. Nevertheless, the tool is currently not available in the public domain and we did not contact the authors as we were working on a smaller FM which we managed to encode manually.

Finkel et al. [76]: The goal of this paper was to detect all possible configurations and find if the model has any optional-feature flaw and missing-feature flaw. If a feature is marked as optional but exists in all valid products from the FM, then the FM is said to have an optional-feature flaw. Whereas, in the same way if a mandatory feature is absent in all products then the FM has a missing-feature flaw. For specifying the bCMS FM we had to manually encode the features as different signatures and specify relationships as propositional logic formulas in Alloy, similarly to Nakajima et al. [144]. Therefore the encoding was simple and easy to reproduce but it was time consuming and error-prone. This calls for automated tools to support the translation of FM to Alloy specifications.

Ripon et al. [157]: The main objective of this paper was to present an approach for formalizing and verifying SPL FMs with support for automatically generating customized products based on user requirements. They have used the same Alloy encoding as described by Gheyi et al. [78] in their G-Theory and Wang et al. [200], and hence have similar findings. This can be seen from the results of analysis in Table 5.5.

Huang et at. [88]: The encoding is very similar to Nakajima et al. [144] as they have extended the same encoding to include signatures (`sig`) that will enable analysis operations to check for valid/invalid sub–FMs in an FM. This approach is very verbose and error-prone due to relying completely on propositional logic formulas. Two analysis operations *A1 & A2* are successfully performed using this encoding on bCMS. Though the authors have even mentioned about detecting valid/invalid sub–models that aid in refining the overall FM using Refactoring (*A7*), this was not explicitly demonstrated in the paper and hence it is not included in the bCMS analysis.

Jaime et al. [53]: The objective of this paper was to detect conflicts in the product configuration process which occur when intended features cannot be selected because of other choices of features included into the product. For this they have identified and included one more type of relationship which is *'Non-selectable'* in addition to the relationship encodings provided by Gheyi [78]. An FM is considered as invalid if full mandatory features that must be included in all the configurations are also non-selectable. The encoding was easy to understand and apply for specifying the bCMS FM. The analysis operation *A2* took slightly more time compared to other approaches though the encoding was better encapsulated with appropriate predicates and functions for easy FM specification. There were no technical roadblocks.

Ajoudanian et al. [2]: Constraint-based FMs with cardinalities are known as extended Feature Models. Hence they are Feature Models with attributes. This paper describes a promotion technique in Alloy which claims to significantly improve the efficiency of analysis operations performed on such extended Feature Models. We were unable to reproduce the results of the paper using the example and Alloy specifications provided in it. When the Alloy encoding details were directly used, it showed several syntax errors in the Alloy editor such as missing **'{'**, **'.'** between `univ` and **'('**, undefined variables, invalid Alloy symbols for *less than or equal* and *greater than or equal*. Therefore it was difficult to reuse the encoding and reproduce their results with bCMS FM.

Reference	Wang [200]	Gheyi [78]	Tanizaki [183]	Nakajima [144]	Finkel [76]	Ripon [157]	Huang [88]	Jaime [53]	Ajoudanian [2]
Expressivity	R1-7	R1-3, R5	R1-3, R5-7	R1-7	R1-3, R5	R1-5	R1-3	R1-3, R5-7	R1-3, R5
Tool support	None	None	None	FD-Checker	None	None	None	None	None
Analysis* (Identifier + Execution time)	A1 5 ms A2 7 ms A7 1 ms	A5 25 ms A2 20 ms A3 30 ms	A2 10 ms	A1 8 ms A2 610 ms	A2 7ms A5 4 ms	A1 10 ms A2 8 ms	A1 5 ms A2 5 ms	A2 22 ms	A2 5 ms

* Execution time of each analysis reported on the bCMS case study of Fig. 5.2
Settings: PC with 16Gb RAM and a 3.4GHz processor. Alloy Analyzer version 4.2 (SAT4j SAT solver).
The Alloy source files for the bCMS product line, encoded using each approach are available at :
https://github.com/5Quintessential/FMAlloyAnalysis

Relationships

Id	**Definition**
R1	*Mandatory(x):* Feature x must appear if parent(x) is included
R2	*Optional(x)* ***or*** *Optional Or(x):* Feature x may or may not appear if parent(x) is included
R3	*Alternative*(X) ***or*** *XOR*(X): Exactly one feature $x_i \in X$ must appear if parent(X) is selected
R4	*Optional Alternative(X):* At most one feature $x_i \in X$ must appear if parent(X) is selected
R5	*Or(X):* At least one feature $x_i \in X$ must appear if parent(X) is selected
R6	*Requires(x, y):* Feature x must appear when feature y is included
R7	*Excludes(x, y):* Feature x and y cannot appear both

Table 5.5: Review summary on the analysis of FM using Alloy.

5.6 Challenges

The major challenge lies in having future research works targeting to address the missing analysis operations (Table 5.2) using Alloy analyzer. The analysis operations such as *Valid partial configuration*, *Dependency analysis*, *Redundancies* and *Commonality analysis* have great significance when it comes to the industrial application of FM analysis during real product development. With the available research results we cannot implement a fully operational FM analysis automated tool based on Alloy without these missing operations that are crucial for generating Feature Models with lesser defects. Furthermore, it would be challenging to study in detail, all the formal proposals for semantics of Feature Models and how Alloy could be used to support each one of those semantics. As a starting point we would refer to the works of Schobbens et al. [168] and Amador et al. [67]. From the summary of review it can be inferred that some of the existing research results can be reused for specific analysis operations. Future research must focus to resolve all the limitations identified in our review.

5.7 Summary

The evaluation summary is illustrated in Table 5.5. After analyzing and comparing the execution time for different operations, we have identified the list of various encodings that can be used for the respective analysis operations based on the encoding efficiency and execution time. This is summarized in Table 5.6.

We have identified two types of shortcomings in this literature survey that dealt with papers dealing with the analysis of FMs using Alloy: *theoretical limitations* and *practical limitations*.

Analysis operation	Encoding strategy	Analysis operation	Encoding strategy
A1 (Void FM)	Wang et al. [200]	A6 (Cardinalities)	Supported in [2]*
A2 (Valid product)	Wang et al. [200]	A7 (Refactoring)	Wang et al. [200]
A3 (All products)	Gheyi et al. [78]	A8 (Commonality)	Supported in [2]*
A4 (Filter)	Supported in [2]*	A9 (Variability)	Supported in [2]*
A5 (Dead features)	Gheyi et al. [78]	A10 (Orthogonality)	Supported in [2]*

* These are discussed by Ajoudanian et al. [2] but we were unable to reproduce the results due to various reasons as explained in Sec. 5.5

Table 5.6: Analysis operation and best encoding strategy based on lowest execution time and highest expressiveness.

Theoretical limitations. Most encodings only support parent-child relationships and trivial requires/excludes cross-tree constraints. Furthermore, only one of the papers demonstrated the encoding to compute the entire set of products in the product line by extending support for checks such as counting the number of valid configurations. There is a wide scope for improving the encodings to allow more expressivity so that the other missing analysis operations (among the list of 30 as shown in Table 5.2) can also be automated.

Practical limitations. There were several challenges in our attempts to replicate each approach in the bCMS case study.

Firstly, only Nakajima et al. [144] describe a tool to generate the Alloy specification automatically. Even though, the tool is still available, it is not available for download and use. Therefore, almost at all times during the review process, the Alloy encodings had to be generated manually.

Furthermore, the encoding proposed by Ripon et al. [157] is not fully described. This means that some significant encoding is not provided in the paper and is neither available online. Attempts to fill the missing gaps yielded results which did not conform to the expected output obtained for the same analysis in the rest of encodings.

Finally, none of the works provided a large-scale FM for experimentation. Even though some approaches [2] claimed being validated on FMs with hundreds of features, the examples are not available and authors did not respond to inquiries.

Chapter 6

Conclusions

The contributions of this thesis are threefold. First, we have proposed an open source Feature Model extraction framework called FeatureX. Second, we introduce a Feature Model validation approach which checks the correspondence between an FM and the set of input textual documents, and provides an intuitive visualization. Third, we review and compare different encoding strategies for analyzing Feature Models using Alloy.

6.1 Summary of contributions

The proposed framework (FeatureX) for extracting FMs from natural language specifications of software product lines (see Chapter 3) aims to address the limitations of previous works in this area of research. Experimental results show improvements in terms of recall (lack of false positives) with respect to previously published methods, while keeping a comparable degree of precision (detecting most relevant features). FeatureX is developed in Python and the source is available publicly for future researchers to evaluate and extend it further.

The FM validation approach (see Chapter 4) provides a method for the validation of FMs with respect to the textual specifications from which they have been generated. Given a FM and a set of textual documents, we assess if the FM is consistent and complete with respect to the source documents. This information is useful for a variety of tasks, like improving the FM or detecting omissions in the source documents. The method starts by analyzing the source documents using NLP techniques. First, the designer selects a few representative features from the FM. Using supervised learning, the program recognises the syntactic structure and identify the remaining feature-like concepts in the text. This information is then used to compute a confidence score for each element (features and relationships) in the FM. Finally, this confidence score is represented using a heatmap visualization of the input FM. As an advantage, this visualization is decoupled from the previous steps, thereby allowing it to be used in conjunction with any alternative method that can compute confidence scores for validating features and relationships in a FM.

Moreover, this thesis also highlights the importance of future research in the direction of supporting automated FM analysis using Alloy, by identifying the limitations in the current state-of-the-art approaches (see Chapter 5). We have reviewed prior contributions related to automated analysis of FM using Alloy. The major shortcoming identified was linked to the lack of tool support for converting informal FM specifications to the respective Alloy encodings. There are also several other practical issues that hinder the industrial application of these research results such as: out-of-date tools and Alloy formats, several syntactic errors in the Alloy specifications provided in the papers, lack of complete descriptions for the Alloy theories, lack of large-scale examples and a large number of analysis operations which are still not supported using Alloy. Several previous works have proposed the analysis of FODA-style FMs through Alloy. All formalizations but [78] are unable to reason about more than one product configuration.

6.2 Discussion

The research results have demonstrated evidence of sustainable solution that can be developed for realizing a fully automated FM based requirements engineering tool that can be easily adapted within the industrial setting with minimal or no learning curve. One of the key areas for future improvements is the performance of natural language processing algorithms for complex meaning analysis and semantic attribution of parts in the text. The application of more involved machine learning algorithms must be studied in detail to facilitate scalable FM extraction, analysis and validation solutions.

Moreover, the results from this thesis highlight the significance of NLP driven approaches for requirements engineering in the context of software product lines. It not only proves effective in terms of the time and effort spent on understanding and meaningfully linking the software product requirements, but also showcases the impact of such automation capabilities in determining the quality of the specifications. As a result, it has been observed that over the past decade the contributions in this area of research has been extensive with a consistent increase in the number of research publications.

Another strength of the contributions of this thesis is the availability of the implementation source code for the FM extraction, analysis, validation and visualization modules and the textual extracts of all the case studies used in this research work. These artefacts can be used to reproduce the results and customize it further to suite new research requirements. There needs to be more detailed study on the applications of deep learning algorithms for identifying and classifying features of the software product/ product line specifications. It can even be possible that such complex machine learning algorithms can contribute towards identifying semantic connections between phrases and sentences thereby supporting identification and extraction of feature relationships.

One of the first challenges was converting unstructured textual data into labelled datasets with appropriate attributions that can assist the application of machine learning tools. This was accomplished during the development of FeatureX and it delivered results with reasonable accuracy. Nevertheless, it is necessary to fine-tune the strategies and heuristics used for such data transformations in order to identify grounded semantic rules that can function for any textual corpus in English language irrespective of the domain. To this end, future research should include specification documentations from a wider variety of domains. Exploring more recent NLP libraries can also prove beneficial in terms of the performance of the NLP techniques.

Regarding FM analysis, one of the main challenges is the *state explosion problem*: adding features and relations to a FM greatly increases the number of the states that need to be explored. Thus, it is necessary to ensure that this analysis is scalable and applicable in practice.

6.3 Revisiting the original research questions

After presenting the conclusions of the thesis, it is interesting to reflect back on the contributions of this work towards finding answers to the research questions we started off with. We will now summarize the answers to each research question in terms of the conclusive inferences obtained from this research work.

RQ 1 : *What tools are available for automatic analysis of feature models generated from requirements specifications and how can it be used for encoding and analysing various specification data?*

This research work used **Alloy analyzer** tool for automatic analysis of FM. Alloy was selected mainly because it was the only open source tool that used first-order logic and has been already used in a wide range of software modeling applications. There were also other tools like **Z notation**, Object Constraint Language in UML (**OCL**), Vienna Development Method (**VDM**) etc. which mainly supported higher order logic thus curtailing the automated analysis capability beyond a certain point. The largest product line case study investigated using Alloy was the product specifications of the car crash management system as discussed in the section 5.5.

RQ 2 : *What Natural language processing (NLP) based heuristics can be used for extracting features and relationships from textual specifications?*

NLP techniques are at the core of automatic FM extraction. Most of the time, the input data source will constitute textual specifications, NLP must be used for automatically refining, enhancing and processing information into data points on which complex computation can be performed. The experiments conducted in this research has shown that the computations cannot be effective if the data points resulting from NLP techniques lack quality in terms of correctness of language interpretation. For this reason there is a huge potential for exploring the affects of applying various NLP techniques (see Table 2.5) on framing better heuristics that can lead to improved accuracy in FM extraction.

RQ 3 : *How can the reliability of the automated FM extraction results be validated, analyzed and quantified?*

The research results presented in section 4.2 are able to demonstrate the feasibility of having a fully automated FM validation approach that is independent of the domain knowledge. This approach was only relying on the language properties of the textual data that was used for creating the FM irrespective of the way it was used to create the FM. The key findings have highlighted the significance of the application of machine learning techniques on performing such unbiased FM validation. This not only has improved the reliability of such validation results but can also evolve into reference standard for validating FM's. This in itself is one of the main contributions of this work which, the progressive future research can reuse and build on.

RQ 4 : *What are the evaluation results from the application of automated FM extraction, validation and analysis on various case studies?*

This thesis work, as shown in Figure 1.2, has successfully been able to prototype a framework for automated FM extraction, validation and analysis, wherein the interventions of a domain expert for considering domain knowledge has been drastically minimized. The results have demonstrated that high quality FM generation can be achieved in a domain-independent way. The automation of this process has facilitated the extraction of FMs, without impacting the performance across any size of input documents. This capability of enabling guided requirements engineering process is a unique and novel contribution of this work.

Chapter 7

Future Work

This chapter provides the details pertaining to the continued research in the field of automated tools for software requirements engineering using Feature Models. Several opportunities for applying the key research findings from this work is presented in the coming sections. In addition to that, the sections in this chapter are organised based on the improvements that can be applied to the methods used in this thesis, extending the evaluation of the thesis results based on latest related research results, and thus proposing research direction for addressing the known limitations in the effectiveness of this thesis work towards the software requirements engineering processes. The chapter concludes with a summary of this discussion with an overview of the most recent relevant research publications and its contribution towards the future extensions of this thesis work.

7.1 Introduction

The previous chapters explain the progression of transforming a textual documentation of system needs, functionalities and constraints (software product line requirements specification) into a configurable software product line model known as the Feature Model. Such Feature Models are used for both validating the correctness of the specified requirements and its completeness based on the available product line domain knowledge. Due to the intrinsic ambiguous characteristics of the English language, developing heuristics that can be used for automatic extraction and evaluation of FM is already an open and challenging research problem. Being a complex problem, this was broken down into simple sub problems for which independent solutions were developed as part of this thesis. The results thus obtained from evaluating the thesis outcomes have proven that an integrated software requirements engineering tool is feasible and can allow better predictability of software project success. An important inference from the thesis results is regarding the level of abstraction that can be accommodated by a Feature Model. A product feature could potentially be a combined effect of several product sub-features. Ability to determine and apply such level of abstraction could help control the complexity of the FM. These are some of the potential avenues to explore as part of the continued research in this field. In order to explore such new dimensions of this thesis research problems, it is important to address some of the necessary fine-tuning of the methods used and extend the case studies to include much complex software products. With this in mind, we will now proceed to explain the details of such improvements in section 7.2, with identifying opportunities for researching on complex software product requirements in section 7.3 and hence defining a path to achieve an integrated solution for enabling the practice of guided requirements engineering for software products in section 7.4. The chapter concludes with a summary in section 7.5 which includes the quick highlights of the future endeavors and provides a brief account of the latest research publications that can facilitate the proposed extended research work.

7.2 Improvements

The focus of this research was to prove the feasibility of having a semi-automated guided software requirements engineering process for large scale industrial software development. As with any similar feasibility study and evaluation of methods, this research used heuristics and omitted one of the complex components of the research subject, which is the cross tree constraints in Feature Models. The coming sections explain how to systematically address such shortcomings related to limitations imposed by both the methods used and out-of-scope components of the research objectives.

7.2.1 FM cross-tree constraints

Cross-tree constraints in Feature Models range from the one-to-one relationship of **requires** and **excludes** to complex constraints represented as a propositional logic formula. Such propositional expressions can involve more that two propositional variables, which in this case are the product features. This makes it a complex constraint that needs to be met in order to fulfil the requirements of a valid product configuration of the software product line under study. The presence of such constraints is crucial and significant in FM research [108] and cannot be avoided when developing a solution for industrial applications.

In the literature, cross-tree constraints (CTC) have been studied in 2 different ways. One is the exploration of whether such constraints are replaceable with simple one-to-one or one-to-many feature relationships. And the other is to explore more structured representations for these constraints so that their analysis and validation can be accomplished with ease.

This thesis does not propose any mechanisms to extract CTC information from text specifications and thus does not include any approaches for validating them. Extracting such constraints is one of the directions of future work after this thesis.

7.2.2 Machine learning algorithms

The results generated by the methods proposed in this thesis does not ensure complete accuracy. The approach and heuristics used in this thesis take advantage of syntactic structures within the text and only partially exploit semantic information. Accuracy can be improved by using other machine learning algorithms that are better suited for the analysis of semantic properties in English text.

Some of the other machine learning algorithms which had been used during this research work includes, Hidden Markov models, word embedding and artificial neural networks, all of which had yielded promising results but were computationally intensive and required larger datasets. As the availability of data was a major limitation for this research, these algorithms did not prove very effective.

An important line of future research will be improving the data repository of natural language specifications for enabling continued research in this field. Without this in place, application of complex machine learning algorithms like recursive or recurrent neural networks is not possible. The results of this thesis have confirmed that the use of unsupervised machine learning technique like clustering may result in a large number of false positives. It was also observed that the general accuracy of the supervised machine learning algorithms like regression and classification resulted in better predictions. Thus, it would be necessary to devote effort to creating, maintaining and extending useful datasets which are either directly obtained from industrial software projects or are at least close to such real world specifications.

7.2.3 Domain-specific vocabularies

Since text documents come with inherent ambiguity, it is also essential to evolve domain specific vocabulary in order to improve predictability of the feature and relationship extraction algorithms. This thesis has attempted to create FM extraction and validation methods irrespective of any domain knowledge. From the results produced, it was observed that certain heuristics worked for some text data with great accuracy whereas it did not work that well for the others. It can thus be inferred that though the heuristics are looking only at the syntactic attributes of the language used, the vernacular and vocabulary used within the specific domain plays an important and significant role in the precision of the decision making process of the predictor algorithms. This calls for extending this research to study each application domain independently and understand such dependencies on vocabulary for achieving better accuracy.

7.3 Evaluations

Similar to several other research works in this area, this thesis also suffers from the lack of availability of large datasets which can help assess the quality of the FM extraction and validation results. For the most part, these evaluations had been heavily dependent on the expertise of an RE practitioner or a domain engineer. The contributions of this thesis has taken a step forward by introducing partially automated FM validation approaches that rely only on the information present in the source documents. This is very significant as it has been now demonstrated that the presence or absence of any feature and its relationships can be argued purely based on the evidence that can be found from the source text it used to extract that information. This removes the dependency on a domain engineer or an RE practitioner to help validate an FM. But the heuristics used for such automated validation approaches needs much refinement in order to account for the actual meaning of the text in the context of the specific product line. This can be achieved only by introducing better AI based machine learning algorithms, which in turn depends heavily on the availability of large datasets. Therefore, as the next steps towards refining the heuristics it is necessary to expand the case study repository to include data from various diverse product lines and create new avenues for stronger collaborations with the software industry to be able to generate datasets that replicate realistic software product line scenarios.

7.3.1 Advanced case studies

The evaluations performed in this thesis were obtained from publicly available sources. The specific case study related to the sports utility vehicles (SUV), utilized the various SUV owners manual as the product specification documents. This was the largest dataset studied in this thesis work. Though this was an attempt towards evaluating the thesis approach on a very large product line specification, the text in these documents did not replicate a realistic software product requirements specification. The owners manual is written and organized in a way that it can be easily understood by the user of the vehicle and uses a writing style where the features of the SUV has already been clearly identified and will not involve any inconsistencies as the product itself has evolved over several years and is now producing a much stable version with well-defined product configurations. As part of future research we would like to identify product line requirements specifications for domains like microprocessor chips, search engines, email web clients, operating systems, banking software, flight reservation software etc. which could demonstrate different language usage styles and vocabulary. Datasets generated from such diverse domains will be used for more thorough training and testing of the ML algorithms involved in the methods introduced in this thesis. It is essential to encourage involvement of industrial practitioners into the development of automated tools for SPL. For this reason it is important to target some effort in this direction.

7.3.2 Industrial collaboration

As part of extensive training of ML models, we would like to collaborate with interested industrial partners for gaining access to appropriate set of specification documentations which are used for software product development in the respective domains. Based on the results of the continued refinement of the methods, we would like to access the benefits of such automated tools and techniques on the quality and success of the products delivered. This should be an ongoing, continuous improvement cycle where the artefacts from industry propels improvements in the method which in turn provides feedback to the practitioners for modifying and correcting the identified defects in the used SPL specifications. This would eventually lead to more seamless integration of academic research results into industrial practice at a steady and consistent pace.

7.4 Guided RE tool

An integrated automated solution for requirements engineering would be the future goal of this research. We believe that this would lead us to find the path towards such an integrated, assisted, recommending-system based guided requirements engineering process for software product lines. This system should be able to transform RE from an intuition and domain-expertise based process to a template based guided approach where a new product will simply be an amended configuration of an existing similar product or product line specification. It will allow the product stakeholders to visualize the finished product with all the prime features and dependencies much ahead of its actual development.

Any changes to the product configuration would automatically highlight the impact on the other product features and their dependencies. It would also suggest possible ways of fixing the breaking changes. Such feedback is undoubtedly the most useful guidelines for RE practitioners who handle requirements change management. Processing large volumes of text and automatically providing a visual representation in terms of the Feature Model which gets updated as and when the changes are made to the text, would be the work for the future that would make use of the methods proposed in this thesis. Such a user interface powered by the natural language processing and machine learning based computing engine in the back-end, would disrupt the way software companies handle requirements engineering today.

7.5 Summary

This chapter elaborated the future directions for continued research in the field of automated tools for requirements engineering based on Feature Models. The immediate focus is given to exploring various natural language processing techniques together with the application of complex machine learning algorithms in Feature Model extraction and validation approaches. This is mainly attributed to improve the language interpretation capabilities of the automated techniques used in this research from being syntactic to becoming more semantic. More precision, in the way the language communicates the intentions of a product or its features, means less ambiguity in the creation of a Feature Model. This will not only lead to more consistent use of language for documenting requirements, but would also lead to ease in its computational abilities like validation and defect detection and correction.

Though this thesis has included most of the feature relationships, it has not accounted for complex propositional logic expression based cross-tree constraints. The need for such constraints when modeling a software system needs to be studied in depth in order to propose effective ways of extracting this information from textual descriptions. This is one of the priority tasks lined up for the future work as the FM is deemed incomplete without all documented constraints and dependencies.

The goal of continued research in this field is to introduce an automated integrated solution for requirements engineering process in industrial practice with effective user interface that promotes its adoption in real world requirements engineering processes. This can become a reality only through extensive collaboration between the greater research community and the software industries. As part of future work, we dedicate a significant part of the overall effort towards this direction and intend to have a positive impact on the pace at which this research progresses.

There are several new contributions in the areas of evaluating significance of Feature Modeling in software product lines [171], reverse engineering SPL using FM [167], techniques for resolving conflicting system goals using FM [164], classification and systematic review of FM defects [38], support for industrial practitioners to migrate from traditional single product development lifecycle to software product line development lifecycle [110], developing textual language for Feature Modeling [3], Feature Model evolution [71] and tools for extracting variability from natural language text [70], etc. details of which can be used or inspire the integrated solution. There is still a very long way to go, but the path is more clear as we keep taking a step forward.

Bibliography

[1] Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Charles Vanbeneden, Philippe Collet, and Philippe Lahire. On Extracting Feature Models from Product Descriptions. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, page 45–54, New York, NY, USA, 2012. Association for Computing Machinery.

[2] Shohreh Ajoudanian and Seyed-Hassan Mirian Hosseinabadi. Automatic promotional specialization, generalization and analysis of extended feature models with cardinalities in alloy. *Journal of Logical and Algebraic Methods in Programming*, 84(5):640–667, 2015.

[3] Ali Fouad Al-Azzawi. PyFml - a Textual Language For Feature Modeling. *International Journal of Software Engineering & Applications (IJSEA)*, abs/1802.05022, 2018.

[4] Mauricio Alférez, Roberto E. Lopez-Herrejón, Ana Moreira, Vasco Amaral, and Alexander Egyed. Consistency checking in early software product line specifications - the VCC approach. *Journal of Universal Computer Science*, 20(5):640–665, May 2014.

[5] Vander Alves, Christa Schwanninger, Luciano Barbosa, Awais Rashid, Peter Sawyer, Paul Rayson, Christoph Pohl, and Andreas Rummler. An exploratory study of information retrieval techniques in domain analysis. In *12th International Software Product Line Conference SPLC'08*, pages 67–76, September 2008.

[6] Vincenzo Ambriola and Vincenzo Gervasi. Processing natural language requirements. *Proceedings of the IEEE International Automated Software Engineering Conference, ASE*, pages 36–45, 1997.

[7] Michal Antkiewicz and Krzysztof Czarnecki. FeaturePlugin: Feature Modeling Plug-in for Eclipse. In *Proceedings of the 2004 Object-Oriented Programming, Systems, Languages & Applications (OOPSLA) Workshop on Eclipse Technology eXchange*, eclipse '04, pages 67–72, New York, NY, USA, 2004. ACM.

[8] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Basic Concepts, Classification, and Quality Criteria*, pages 47–63. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[9] Andres Arellano, Edward Carney, Lockheed Martin, College Park, and Mark A Austin. Natural Language Processing of Textual Requirements. *ICONS 2015 : The Tenth International Conference on Systems Natural*, pages 93–97, 2015.

[10] Eleonora Arganese, Alessandro Fantechi, Stefania Gnesi, and Laura Semini. Nuts and bolts of extracting variability models from natural language requirements documents. In *Integrating Research and Practice in Software Engineering*, volume 851 of *Studies in Computational Intelligence*, pages 125–143. Springer, 2020.

[11] Venera Arnaoudova, Sonia Haiduc, Andrian Marcus, and Giuliano Antoniol. The use of text retrieval and natural language processing in software engineering. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, volume 2, pages 949–950, May 2015.

[12] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Automated extraction and clustering of requirements glossary terms. *IEEE Transactions on Software Engineering*, 43(10):918–945, 2016.

[13] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Extracting domain models from natural-language requirements: Approach and industrial evaluation. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS'16)*, pages 250–260. ACM, 2016.

[14] Monica Arrabito, Alessandro Fantechi, Stefania Gnesi, and Laura Semini. A comparison of NLP tools for RE to extract variation points. In *Joint Proceedings of REFSQ-2020 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track co-located with the 26th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2020), Pisa, Italy, March 24, 2020*, volume 2584 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

[15] Monica Arrabito, Alessandro Fantechi, Stefania Gnesi, and Laura Semini. An experience with the application of three NLP tools for the analysis of natural language requirements. In *Quality of Information and Communications Technology - 13th International Conference, QUATIC 2020, Faro, Portugal, September 9-11, 2020, Proceedings*, volume 1266 of *Communications in Computer and Information Science*, pages 488–498. Springer, 2020.

[16] Mohsen Asadi, Samaneh Soltani, Dragan Gasevic, and Marek Hatala. The effects of visualization and interaction techniques on feature model configuration. *Empirical Software Engineering*, 21(4):1706–1743, 2016.

[17] Aybüke Aurum and Claes Wohlin. Requirements engineering: setting the context. In *Engineering and managing software requirements*, pages 1–15. Springer, 2005.

[18] Maider Azanza, Leticia Montalvillo, and Oscar Díaz. 20 years of industrial experience at SPLC: a systematic mapping study. In *SPLC '21: 25th ACM International Systems and Software Product Line Conference, Leicester, United Kingdom, September 6-11, 2021, Volume A*, pages 172–183. ACM, 2021.

[19] Jongsu Bae and Sungwon Kang. A method to generate a feature model from a business process model for business applications. In *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, pages 879–884, 2007.

[20] Noor Hasrina Bakar. Latent Semantic Analysis and Particle Swarm Optimization for Requirements Reuse in Software Product Line : A Research Plan. In *SPLC'2013 Doctoral Symposium*, pages 0–9, November 2015.

[21] Noor Hasrina Bakar, Zarinah M. Kasirun, and Norsaremah Salleh. Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. *Journal of Systems and Software*, 106:132–149, 2015.

[22] Don Batory. Feature models, grammars, and propositional formulas. In *Proceedings of the 9th International Conference on Software Product Lines*, SPLC'05, pages 7–20, Berlin, Heidelberg, 2005. Springer-Verlag.

[23] Don Batory. A tutorial on feature oriented programming and the ahead tool suite. In *Proceedings of the 2005 International Conference on Generative and Transformational Techniques in Software Engineering*, GTTSE'05, pages 3–35, Berlin, Heidelberg, 2006. Springer-Verlag.

[24] Don Batory, David Benavides, and Antonio Ruiz-Cortés. Automated analysis of feature models: Challenges ahead. *Communications of the ACM*, 49(12):45–47, dec 2006.

[25] Guillaume Bécan, Razieh Behjati, Arnaud Gotlieb, and Mathieu Acher. Synthesis of attributed feature models from product descriptions: Foundations. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, page 1–10, New York, NY, USA, 2015. Association for Computing Machinery.

[26] Sean Bechhofer. *OWL: Web Ontology Language*, pages 2008–2009. Springer US, Boston, MA, 2009.

[27] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010.

[28] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. Using Java CSP Solvers in the Automated Analyses of Feature Models. In *Post-Proceedings of the Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE). LNCS 4143*, page 2006, 2005.

[29] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. A first step towards a framework for the automated analysis of feature models. *Managing Variability for Software Product Lines: Working With Variability Mechanisms*, 85:1–5, 2006.

[30] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. FAMA: Tooling a Framework for the Automated Analysis of Feature Models. In *In Proceeding of the First International Workshop on Variability Modelling of Software intensive Systems (VAMOS)*, pages 129–134, 2007.

[31] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Automated reasoning on feature models. In *Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, pages 491–503. Springer Berlin Heidelberg, 2005.

[32] Omar Benomar, Houari Sahraoui, and Pierre Poulin. Visualizing software dynamicities with heat maps. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*, pages 1–10. IEEE, 2013.

[33] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. A survey of variability modeling in industrial practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-Intensive Systems*, VaMoS '13. Association for Computing Machinery, 2013.

[34] Daniel Le Berre. SAT4J solver. Technical report, Université d'Artois, 2010.

[35] Daniel M Berry and Brian Lawrence. Requirements engineering. *IEEE software*, 15(2):26–29, 1998.

[36] Carla I. M. Bezerra, Rossana M. C. Andrade, and José Maria S. Monteiro. Measures for quality evaluation of feature models. In *Software Reuse for Dynamic Systems in the Cloud and Beyond*, pages 282–297. Springer International Publishing, 2014.

[37] M P S Bhatia, Akshi Kumar, and Rohit Beniwal. Ontology based framework for detecting ambiguities in software requirements specification. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 3572–3575, March 2016.

[38] Megha Bhushan, Arun Negi, Piyush Samant, Shivani Goel, and Ajay Kumar. A classification and systematic review of product line feature model defects. *Software Quality Journal*, 28(4):1507–1550, Dec 2020.

[39] Think Big and Act Small. CHAOS MANIFESTO 2013. http://versionone.com/assets/img/files/ChaosManifesto2013.pdf, 2013.

[40] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

[41] Margot Bittner, A. Botorabi, Alexander Poth, Mark-Oliver Reiser, and Matthias Weber. Managing variability and reuse of features and requirements for large and complex organizational structures. *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 469–470, 2005.

[42] Günter Böckle. *Introduction to Software Product Line Engineering*, pages 3–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[43] Dasha Bogdanova. Extraction of High-Level Semantically Rich Features from Natural Language Text. In *ADBIS 2011, Research Communications, Proceedings of the 15th East-European Conference on Advances in Databases and Information Systems*, pages 262–271, 2011.

[44] Ekaterina Boutkova and Frank Houdek. Semi-automatic identification of features in requirement specifications. In *2011 IEEE 19th International Requirements Engineering Conference*, pages 313–318. IEEE, 2011.

[45] Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. In *Association for Computational Linguistics (ACL)*, 2016.

[46] Andrew P. Bradley. The use of the area under the ROC Curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition*, 30(7):1145–1159, July 1997.

[47] Bartosz Broda, Pawel Kedzia, Michal Marcinczuk, Adam Radziszewski, Radoslaw Ramocki, and Adam Wardynski. Fextor: A feature extraction framework for natural language processing: A case study in word sense disambiguation, relation recognition and anaphora Resolution. *Studies in Computational Intelligence*, 458(Ml):41–62, 2013.

[48] Jean-Michel Bruel, Sophie Ebersold, Florian Galinier, Manuel Mazzara, Alexandr Naumchev, and Bertrand Meyer. The role of formalism in system requirements. *ACM Computing Surveys (CSUR)*, 54(5):1–36, 2021.

[49] Gino Brunetti and Borut Golob. A feature-based approach towards an integrated product model including conceptual design information. *Computer-Aided Design*, 32(14):877–887, 2000.

[50] Yong Cai and Chun Liu. Extracting phrases as software features from overlapping sentence clusters in product descriptions. *IEEE Access*, 8:11174–11185, 2019.

[51] Alfredo Capozucca, Betty Cheng, Geri Georg, Nicolas Guelfi, Paul Istoan, and Gunter Mussbacher. Requirements definition document for a Software Product Line of car crash management systems. Technical report, Computer Science Department, Colorado State University, 2011.

[52] Daniel Cer, Michel Galley, Daniel Jurafsky, and Christopher D. Manning. Phrasal: A Toolkit for Statistical Machine Translation with Facilities for Extraction and Incorporation of Arbitrary Model Features. In *North American Association for Computational Linguistics - Human Language Technologies (NAACL-HLT) Demonstration Session*, HLT-DEMO '10, pages 9–12, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[53] Jaime Alberto Chavarriaga Lozano, Carlos Francisco Noguera Garcia, Viviane Jonckers, and Rubby Casallas. *Supporting Multi-level Configuration with Feature-Solution Graphs – Formal Semantics and Alloy Implementation.* Technical Report, Software Language Lab, Berlin, 2013.

[54] Kun Chen, Wei Zhang, Haiyan Zhao, and Hong Mei. An approach to constructing feature models based on requirements clustering. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 31–40. IEEE, 2005.

[55] Andreas Classen, Quentin Boucher, and Patrick Heymans. A text-based approach to feature modelling: Syntax and Semantics of TVL. *Science of Computer Programming*, 76(12):1130–1143, December 2011.

[56] Andreas Classen, Arnaud Hubaux, and Patrick Heymans. Analysis of Feature Configuration Workflows. *17th IEEE International Requirements Engineering Conference*, pages 381–382, August 2009.

[57] Philippe Collet and Philippe Lahire. Feature modeling and separation of concerns with FAMILIAR. In *2013 3rd International Workshop on Comparing Requirements Modeling Approaches (CMA@RE)*, pages 13–18, July 2013.

[58] Ronan Collobert. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

[59] Colleen Cool. The concept of situation in information science. *Annual Review of Information Science and Technology (ARIST)*, 35:5–42, 2001.

[60] Kenneth Cosh. Current research themes in software engineering: An application of text mining. In *2016 8th International Conference on Knowledge and Smart Technology (KST)*, pages 125–129, 2016.

[61] David Benavides Cuevas. *On The Automated Analysis of Software Product Lines using Feature Models. A framework for developing automated tool support.* PhD thesis, Universidad de Sevilla, 2007.

[62] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. In *Software Process: Improvement and Practice*, page 2005, 2005.

[63] Jean-Marc Davril, Edouard Delfosse, Negar Hariri, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans. Feature model extraction from large collections of informal product descriptions. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 290–300, New York, New York, USA, 2013. Association for Computing Machinery.

[64] David Dietrich, Pourya Shaker, Joanne M. Atlee, Derek Rayside, and Jan Gorzny. Feature interaction analysis of the feature-oriented requirements-modelling language using Alloy. In *Proceedings of the Workshop on Model-Driven Engineering, Verification and Validation*, MoDeVVa '12, pages 17–22, New York, NY, USA, 2012. ACM.

[65] Nicolas Dintzner, Arie van Deursen, and Martin Pinzger. FEVER: Extracting Feature-Oriented Changes from Commits. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, page 85–96, New York, NY, USA, 2016. Association for Computing Machinery.

[66] Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 181–190, 2011.

[67] Amador Durán, David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. FLAME: a formal framework for the automated analysis of software product lines validated by automated specification testing. *Journal Software and Systems Modeling*, pages 1–34, 2015.

[68] Alireza Ensan, Ebrahim Bagheri, Mohsen Asadi, Dragan Gasevic, and Yevgen Biletskiy. Goal-oriented test case selection and prioritization for product line feature models. In *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, pages 291–298, 2011.

[69] Brian S. Everitt and Torsten Hothorn. *A Handbook of Statistical Analyses Using R, Second Edition.* Chapman & Hall/CRC, 2nd edition, 2009.

[70] Alessandro Fantechi, Stefania Gnesi, Samuele Livi, and Laura Semini. A SpaCy-Based Tool for Extracting Variability from NL Requirements. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume B*, SPLC '21, page 32–35, New York, NY, USA, 2021. Association for Computing Machinery.

[71] Kevin Feichtinger, Daniel Hinterreiter, Lukas Linsbauer, Herbert Prähofer, and Paul Grünbacher. Supporting feature model evolution by suggesting constraints from code-level dependency analyses. In *Proceedings of the 18th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, GPCE 2019, page 129–142, New York, NY, USA, 2019. Association for Computing Machinery.

[72] Robert Feldt, Miroslaw Staron, Erika Hult, and Thomas Liljegren. Supporting software decision meetings: Heatmaps for visualising test and code measurements. In *39th Euromicro Conference on Software Engineering and Advanced Applications*, pages 62–69. IEEE Computer Society, sep 2013.

[73] Alessio Ferrari, Giorgio O. Spagnolo, and Felice Dell'Orletta. Mining commonalities and variabilities from natural language documents. In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, page 116, New York, NY, USA, 2013. Association for Computing Machinery.

[74] Giacomo Ferrari. State of the art in computational linguistics. *Linguistics today: Facing a greater challenge*, pages 163–186, 2004.

[75] Jenny Rose Finkel and Christopher D. Manning. Joint parsing and named entity recognition. In *North American Association of Computational Linguistics (NAACL)*, 2009.

[76] Raphael Finkel and Barry O'Sullivan. Reasoning about conditional constraint specification problems and feature models. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 25:163–174, 5 2011.

[77] Stuart Geman and Mark Johnson. Probability and statistics in computational linguistics, a brief review. *Mathematical foundations of speech and language processing*, pages 1–26, 2004.

[78] Rohit Gheyi, Tiago Massoni, and Paulo Borba. A theory for feature models in Alloy. In *Proceedings of the ACM SIGSOFY First Alloy Workshop*, page 71–80, Portland, United States, November 2006. Citeseer.

[79] Rohit Gheyi, Tiago Massoni, and Paulo Borba. Automatically checking the correctness of feature model refactorings. *Journal of Universal Computer Science*, 17(5):684–711, 2011.

[80] Phil Greenwood, Ruzanna Chitchyan, Awais Rashid, Joost Noppen, Franck Fleurey, and Arnor Solberg. Modelling adaptability and variability in requirements. *2011 IEEE 19th International Requirements Engineering Conference*, pages 343–344, August 2011.

[81] Joel Greenyer, Amir Molzam Sharifloo, and Politecnico Milano. Efficient Consistency Checking of Scenario-Based Product-Line Specifications. *2012 IEEE 20th International Requirements Engineering Conference*, pages 161–170, 2012.

[82] Mostafa Hamza and Robert J. Walker. Recommending features and feature relationships from requirements documents for software product lines. In *2015 IEEE/ACM 4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, pages 25–31. IEEE Press, 2015.

[83] Maarit Harsu. A survey on domain engineering. Technical report, Tampere University of Technology, 2002.

[84] Evelyn Nicole Haslinger, Roberto Erick Lopez-Herrejon, and Alexander Egyed. On extracting feature models from sets of valid feature combinations. In *Fundamental Approaches to Software Engineering*, pages 53–67, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[85] Bin He, Wei Song, and Yangang Wang. A feature-based approach towards an integrated product model in intelligent design. *The International Journal of Advanced Manufacturing Technology*, 69(1-4):15–30, 2013.

[86] Adithya Hemakumar. Finding Contradictions in Feature Models. In *Workshop on Analysis of Software Product Lines (ASPL)*, pages 183–190, 2008.

[87] Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes. Software product line engineering: A practical experience. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, SPLC '19, page 164–176, New York, NY, USA, 2019. Association for Computing Machinery.

[88] Changyun Huang, Yasutaka Kamei, Kazuhiro Yamashita, and Naoyasu Ubayashi. Using Alloy to support feature-based DSL construction for mining software repositories. In *Proceedings of the 17th International Software Product Line Conference Co-Located Workshops*, SPLC '13 Workshops, page 86, New York, New York, USA, 2013. Association for Computing Machinery.

[89] Dubravka Ili. Deriving Formal Specifications from Informal Requirements. In *31st Annual International Computer Software and Applications Conference*. IEEE Comput. Soc. Press, 2007.

[90] Nili Itzik and Iris Reinhartz-Berger. Generating Feature Models from Requirements: Structural vs. Functional Perspectives. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, SPLC '14, pages 44–51, New York, NY, USA, 2014. Association for Computing Machinery.

[91] Nili Itzik and Iris Reinhartz-Berger. SOVA-A tool for Semantic and Ontological Variability Analysis. In *The 26th International Conference on Advanced Information Systems Engineering (CAISE'2014)*, volume 1164 of *Central Europe (CEUR) Workshop Proceedings*, pages 177–184, 01 2014.

[92] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis.* The MIT Press, 2012.

[93] Daniel Jackson and Emina Torlak. Alloy : a language and tool for relational models. Technical report, MIT, 2010.

[94] Daniel Jackson and Jeannette Wing. Formal Methods Light. *Computer, IEEE Computer Society*, 29(4):20–22, 1996.

[95] Ashok Jashapara. The emerging discourse of knowledge management: a new dawn for information science research? *Journal of information science*, 31(2):136–148, 2005.

[96] Shengyi Jiang, Guansong Pang, Meiling Wu, and Limin Kuang. An improved K-nearest-neighbor algorithm for text categorization. *Expert Systems with Applications*, 39(1):1503–1509, 2012.

[97] Taeho Jo. Neural Based Approach to Keyword Extraction from Documents. In *International Conference on Computational Science and Its Applications (ICCSA'03)*, pages 456–461, 2003.

[98] Sangkeun Jung. Semantic vector learning for natural language understanding. *Computer Speech & Language*, 56:130–145, 2019.

[99] Dan Jurafsky, Victor Chahuneau, Bryan R. Routledge, and Noah A. Smith. Narrative framing of consumer sentiment in online restaurant reviews. *First Monday*, 2014.

[100] Uday Kamath, John Liu, and James Whitaker. *Deep learning for NLP and speech recognition*, volume 84. Springer, 2019.

[101] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[102] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. *Annals of Software Engineering*, 5(1):143–168, 1998.

[103] Christian Kästner, Thomas Thüm, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. FeatureIDE: A Tool Framework for Feature-oriented Software Development. In *2009 IEEE 31st International Conference on Software Engineering*, ICSE '09, pages 611–614, Washington, DC, USA, 2009. IEEE CS.

[104] Fisnik Kastrati, Xiang Li, Christoph Quix, and Mohammadreza Khelghati. Enabling structured queries over unstructured documents. *Proceedings - IEEE International Conference on Mobile Data Management*, 2:80–85, 2011.

[105] Colin Kelly, Barry Devereux, and Anna Korhonen. Automatic extraction of property norm-like data from large text corpora. *Cognitive Science*, 38:638–682, 2014.

[106] Amal Khtira, Anissa Benlarabi, and Bouchra El Asri. A Tool Support for Automatic Detection of Duplicate Features during Software Product Lines Evolution. *IJCSI International Journal of Computer Science*, 12(4):1–10, July 2015.

[107] Alexander Knüppel. *The role of complex constraints in feature modeling.* PhD thesis, Institut für Softwaretechnik und Fahrzeuginformatik, 2017.

[108] Alexander Knüppel, Thomas Thüm, Stephan Mennicke, Jens Meinicke, and Ina Schaefer. Is there a mismatch between real-world feature models and product-line research? In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, page 291–302, New York, NY, USA, 2017. Association for Computing Machinery.

[109] Thorsten Koch, Jörg Holtmann, David Schubert, and Timo Lindemann. Towards feature-based product line engineering of technical systems. *Procedia Technology*, 26:447–454, 2016. 3rd International Conference on System-Integrated Intelligence: New Challenges for Product and Production Engineering.

[110] Oman Komarudin, Daya Adianto, and Ade Azurat. Modeling requirements of multiple single products to feature model. *Procedia Computer Science*, 161:107–114, 2019. The Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia.

[111] Ivano Lauriola, Alberto Lavelli, and Fabio Aiolli. An introduction to deep learning in natural language processing: models, techniques, and tools. *Neurocomputing*, 470:443–456, 2022.

[112] Daniel M. Berry Leah Goldin. AbstFinder, A Prototype Natural Language Text Abstraction Finder for Use in Requirements Elicitation. *Proceedings of the First International conference on Requirement Engineering*, 4:375–412, 1997.

[113] Heeyoung Lee, Marta Recasens, Angel Chang, Mihai Surdeanu, and Dan Jurafsky. Joint entity and event coreference resolution across documents. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2012.

[114] Kwanwoo Lee, Kyo Kang, Wonsuk Chae, and Byoungwook Choi. Feature-based approach to object-oriented engineering of applications for reuse. *Software Practice And Experience*, 30:1025–1046, July 2000.

[115] Uwe Lesta, Ina Schaefer, and Tim Winkelmann. Detecting and Explaining Conflicts in Attributed Feature Models. *Electronic Proceedings in Theoretical Computer Science*, 182:31–43, 2015.

[116] Feng-Lin Li, Jennifer Horkoff, Alexander Borgida, Giancarlo Guizzardi, Lin Liu, and John Mylopoulos. From stakeholder requirements to formal specifications through refinement. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 164–180. Springer, 2015.

[117] Lin Li, William M. Campbell, Cagri Dagli, and Joseph P. Campbell. Making sense of unstructured text data. *Computing Research Repository (CoRR)*, abs/1704.05505, 2017.

[118] Long Li, Haiyan Zhao, and Wei Zhang. MbFM: A matrix-based tool for modeling and configuring feature models. *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 325–326, September 2012.

[119] Yang Li. Feature and variability extraction from natural language software requirements specifications. In *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 2*, SPLC '18, page 72–78, New York, NY, USA, 2018. Association for Computing Machinery.

[120] Yang Li, Sandro Schulze, and Gunter Saake. Reverse engineering variability from natural language documents: A systematic literature review. In *21st International Systems and Software Product Line Conference, SPLC 2017*, pages 133–142. ACM, 2017.

[121] Yang Li, Sandro Schulze, and Jiahua Xu. Feature terms prediction: A feasible way to indicate the notion of features in software product line. In *Proceedings of the Evaluation and Assessment in Software Engineering*, EASE '20, page 90–99, New York, NY, USA, 2020. Association for Computing Machinery.

[122] Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham. SVM Based Learning System For Information Extraction. In *International Workshop on Deterministic and Statistical Methods in Machine Learning*. Springer, 2005.

[123] Xiaoli Lian and Li Zhang. Optimized feature selection towards functional and non-functional requirements in software product lines. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 191–200, 2015.

[124] Jia Hui Liang, Vijay Ganesh, Krzysztof Czarnecki, and Venkatesh Raman. SAT-Based Analysis of Large Real-World Feature Models is Easy. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, pages 91–100, New York, NY, USA, 2015. Association for Computing Machinery.

[125] YiQing Liang and James D. Palmer. Indexing and clustering of Software Requirements Specifications. *Information and Decision Technology*, 18 (4):283–299, 1992.

[126] Sascha Lity, Sophia Nahrendorf, Thomas Thüm, Christoph Seidl, and Ina Schaefer. 175 In *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems*, VAMOS 2018, page 27–34, New York, NY, USA, 2018. Association for Computing Machinery.

[127] Bang Liu and Lingfei Wu. Graph neural networks in natural language processing. In *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 463–481. Springer, 2022.

[128] Neil Loughran, Americo Sampaio, and Awais Rashid. From requirements documents to feature models for aspect oriented product line implementation. In *Model Driven Development for Product Lines @ 9th international conference on Model Driven Engineering Languages and Systems*, volume 3844 of *MODELS'06*, pages 262–271, 2006.

[129] Anastassia Loukina, Klaus Zechner, Lei Chen, and Michael Heilman. Feature selection for automated speech scoring. In *Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 12–19, Denver, Colorado, June 2015. Association for Computational Linguistics.

[130] Stephen G. MacDonell, Kyongho Min, and Andrew M. Connor. Autonomous requirements specification processing using Natural Language Processing. In *14th International Conference on Intelligent and Adaptive Systems and Software Engineering*, pages 266–270. Computing Research Repository (CoRR), July 2005.

[131] Kavi Mahesh, Sergei Nirenburg, Jim Cowie, and David Farwell. An Assessment of CYC for Natural Language Processing. *Memoranda in Computer and Cognitive Sciences MCCS-96-296, Computing Research Laboratory, New Mexico State University, Las Cruces, NM*, 1996.

[132] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, USA, 2008.

[133] Mike Mannion. Using first-order logic for product line model validation. In *Software Product Lines*, volume 2379 of *Lecture Notes in Computer Science*, pages 176–187. Springer Berlin Heidelberg, 2002.

[134] Mike Mannion and Javier Camara. Theorem proving for product line model verification. In *Software Product-Family Engineering*, volume 3014 of *Lecture Notes in Computer Science*, pages 211–224. Springer Berlin Heidelberg, 2004.

[135] Thomas von der Maßen and Horst Lichter. Determining the variation degree of feature models. In *Software Product Lines*, volume 3714 of *Lecture Notes in Computer Science*, pages 82–88. Springer Berlin Heidelberg, 2005.

[136] Satoshi Masuda, Tohru Matsuodani, and Kazuhiko Tsuda. Automatic generation of utp models from requirements in natural language. In *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–6, April 2016.

[137] Mariem Mefteh, Nadia Bouassida, and Hanêne Ben-Abdallah. Mining feature models from functional requirements. *The Computer Journal*, 59(12):1784–1804, 2016.

[138] Scott Menard. *Logistic Regression: From Introductory to Advanced Concepts and Applications.* SAGE, 2010.

[139] Marcilio Mendonca, Andrzej Wąsowski, and Krzysztof Czarnecki. SAT-Based Analysis of Feature Models is Easy. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 231–240, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[140] Andreas Metzger, Klaus Pohl, Patrick Heymans, Pierre-Yves Schobbens, and Germain Saval. 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)*, pages 243–253, October 2007.

[141] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, November 1995.

[142] MIT. MIT Lecture notes. http://web.mit.edu/16.35/www/lecturenotes/FormalMethods.pdf.

[143] Konstantinos Mokos and Panagiotis Katsaros. A survey on the formalisation of system requirements and their validation. *Array*, 7:100030, 2020.

[144] Shin Nakajima. Semi-automated Diagnosis of FODA Feature Diagram. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2191–2197. ACM, 2010.

[145] Sana Ben Nasr, Guillaume Bécan, Mathieu Acher, João Bosco Ferreira Filho, Nicolas Sannier, Benoit Baudry, and Jean-Marc Davril. Automated extraction of product comparison matrices from informal product descriptions. *Journal of Systems and Software*, 124:82–103, 2017.

[146] Nan Niu and Steve Easterbrook. Concept analysis for product line requirements. In *Proceedings of the 8th ACM International Conference on Aspect-oriented Software Development*, AOSD '09, pages 137–148, New York, NY, USA, 2009. ACM.

[147] Briony J. Oates. *Researching information systems and computing.* SAGE, 2006.

[148] Ana Carolina Oran, Gleison Santos, Bruno Gadelha, and Tayana Conte. A framework for evaluating and improving requirements specifications based on the developers and testers perspective. *Requirements Engineering*, 26(4):481–508, 2021.

[149] Zachary J. Oster, Ganesh Ram Santhanam, and Samik Basu. Automating analysis of qualitative preferences in goal-oriented requirements engineering. *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 448–451, November 2011.

[150] Jiaul H. Paik. A Novel TF-IDF Weighting Scheme for Effective Ranking. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 343–352. Association for Computing Machinery, 2013.

[151] Lev Pevzner and Marti A Hearst. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1):19–36, 2002.

[152] Andreas Pleuss, Rick Rabiser, and Goetz Botterweck. Visualization techniques for application in interactive product configuration. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, New York, NY, USA, 2011. Association for Computing Machinery.

[153] Reid Pryzant, Youngjoo Chung, and Dan Jurafsky. Predicting sales from the language of product descriptions. In *Association for Computing Machinery's Special Interest Group on Information Retrieval (ACM SIGIR) Workshop on eCommerce*, SIGIR 2017, 2017.

[154] Thomas Quirchmayr, Barbara Paech, Roland Kohl, and Hannes Karey. Semi-automatic Software Feature-Relevant Information Extraction from Natural Language User Manuals. In *Requirements Engineering: Foundation for Software Quality*, pages 255–272. Springer International Publishing, 2017.

[155] Mark-Oliver Reiser and Matthias Weber. Managing Highly Complex Product Families with Multi-Level Feature Trees. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, RE '06, pages 146–155. IEEE Computer Society, 2006.

[156] John A. Rice. *Mathematical Statistics and Data Analysis.* Belmont, CA: Duxbury Press., third edition, 2006.

[157] Shamim Ripon, Keya Azad, Sk Jahir Hossain, and Mehidee Hassan. Modeling and Analysis of Product-line Variants. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*, SPLC '12, pages 26–31, New York, NY, USA, 2012. Association for Computing Machinery.

[158] David Rothlisberger, Oscar Nierstrasz, Stéphane Ducasse, Damien Pollet, and Romain Robbes. Supporting task-oriented navigation in IDEs with configurable heatmaps. In *2009 IEEE 17th International Conference on Program Comprehension*, pages 253–257. IEEE, 2009.

[159] Kevin Ryan. The role of natural language in requirements engineering. *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 240–242, 1993.

[160] Rick Salay, Michalis Famelis, Julia Rubin, Alessio Di Sandro, and Marsha Chechik. Lifting Model Transformations to Product Lines. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 117–128, New York, New York, USA, 2014. Association for Computing Machinery.

[161] Américo Sampaio, Awais Rashid, Ruzanna Chitchyan, and Paul Rayson. *EA-Miner: Towards Automation in Aspect-Oriented Requirements Engineering*, volume 4620 of *LNCS*, pages 4–39. Springer Berlin Heidelberg, 2007.

[162] Prahladavaradan Sampath, Silky Arora, and S Ramesh. Evolving specifications formally. In *2011 IEEE 19th International Requirements Engineering Conference*, pages 5–14, August 2011.

[163] Nicolas Sannier, Mathieu Acher, and Benoit Baudry. From comparison matrix to variability model: The wikipedia case study. In *28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 580–585, 2013.

[164] Theresia Ratih Dewi Saputri and Seok-Won Lee. Addressing sustainability in the requirements engineering process: From elicitation to functional decomposition. *J. Softw. Evol. Process*, 32(8), August 2020.

[165] Manolis Savva, Angel X. Chang, Christopher D. Manning, and Pat Hanrahan. TransPhoner: Automated Mnemonic Keyword Generation. In *Proceedings of the SIGCHI Special Interest Group Conference on Human Factors in Computing Systems*, CHI '14, page 3725–3734. Association for Computing Machinery, 2014.

[166] Pete Sawyer, Paul Rayson, and Roger Garside. REVERE: Support for Requirements Synthesis from Documents. *Information Systems Frontiers*, 4(3):343–353, 2002.

[167] Alexander Schlie, Alexander Knüppel, Christoph Seidl, and Ina Schaefer. Incremental Feature Model Synthesis for Clone-and-Own Software Systems in MATLAB/Simulink. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*, SPLC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[168] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. Feature Diagrams: A Survey and a Formal Semantics. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 139–148, September 2006.

[169] Sergio Segura. Automated Analysis of Feature Models Using Atomic Sets. In *12th International Conference on Software Product Lines*, SPLC 2008, pages 201–207. IEEE Computer Society, 2008.

[170] Matt Selway, Wolfgang Mayer, and Markus Stumptner. Semantic Interpretation of Requirements through Cognitive Grammar and Configuration. *Springer International*, 8862:496–510, 2014.

[171] Samuel Sepúlveda and Marcelo Esperguel. Systematic Mapping Protocol: Reasoning Algorithms on Feature Model. *Computing Research Repository (CoRR)*, abs/2103.16325, 2021.

[172] Edgar Serna and Alexei Serna. Power and limitations of formal methods for software fabrication: Thirty years later. *Informatica*, 41(3), 2017.

[173] Tom De Smedt and Walter Daelemans. Pattern for python. *Journal of Machine Learning Research*, 13:2063–2067, June 2012.

[174] Hécio A. Soares and Raimundo S. Moura. A methodology to guide writing Software Requirements Specification document. In *2015 Latin American Computing Conference (CLEI)*, pages 1–11, 2015.

[175] Ian Sommerville. *Software Engineering*. Addison-Wesley, Harlow, England, 9 edition, 2010.

[176] Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. Punctuation: Making a point in unsupervised dependency parsing. In *Computational Natural Language Learning (CoNLL)*, 2011.

[177] Anjali Sree-Kumar, Elena Planas, and Robert Clarisó. Extracting software product line feature models from natural language specifications. In *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1*, SPLC '18, pages 43–53, New York, NY, USA, 2018. ACM.

[178] Anjali Sree-Kumar, Elena Planas, and Robert Clarisó. Validating feature models with respect to textual product line specifications. In *15th International Working Conference on Variability Modelling of Software-Intensive Systems*, VaMoS'21, New York, NY, USA, 2021. Association for Computing Machinery.

[179] Anjali Sree-Kumar, Elena Planas, and Robert Clarisó. Analysis of feature models using alloy: A survey. *Electronic Proceedings in Theoretical Computer Science*, 206:46–60, Mar 2016.

[180] Reinhard Stoiber, Samuel Fricker, Michael Jehle, and Martin Glinz. Feature unweaving: Refactoring software requirements specifications into software product lines. *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference, RE2010*, pages 403–404, 2010.

[181] Detlef Streitferdt, Matthias Riebisch, and K. Philippow. Details of formalized relations in feature models using OCL. In *Proceedings of 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2003.*, pages 297–304. IEEE Computer Society, 2003.

[182] Jing Sun, Hongyu Zhang, Yuan Fang, and Hai Wang. Formal semantics and verification for feature modeling. In *Engineering of Complex Computer Systems, 2005. ICECCS 2005. Proceedings. 10th IEEE International Conference on*, pages 303–312, June 2005.

[183] Hiroaki Tanizaki and Takuya Katayama. Formalization and Consistency Checking of Changes of Software System Configurations Using Alloy. In *2008 15th Asia-Pacific Software Engineering Conference*, pages 343–350. IEEE, December 2008.

[184] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. A classification and survey of analysis strategies for software product lines. *ACM Comput. Surv.*, 47(1):6:1–6:45, June 2014.

[185] Thomas Thüm, Don Batory, and Christian Kästner. Reasoning about edits to feature models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 254–264, Washington, DC, USA, 2009. IEEE Computer Society.

[186] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming*, 79(0):70–85, 2014.

[187] Emina Torlak and Greg Dennis. Kodkod for Alloy Users. *Alloy Workshop*, 2006.

[188] Pablo Trinidad, David Benavides, Amador Durán, Antonio Ruiz-Cortés, and Miguel Toro. Automated Error Analysis for the Agilization of Feature Modeling. *Journal of Systems and Software*, 81(6):883–896, Jun 2008.

[189] Pablo Trinidad and Antonio Ruiz-Cortés. Abductive Reasoning and Automated Analysis of Feature Models: How are they connected? In *3rd. International Workshop on Variability Modelling of Software-intensive Systems (2009)*, VaMoS 2009, pages 145–153, 2009.

[190] Pim van den Broek and Ismênia Galvão. Analysis of feature models using generalised feature trees. In *Third International Workshop on Variability Modelling of Software-intensive Systems, VaMoS 2009*, ICB-Research Report, pages 29–35, Essen, Germany, January 2009. Universität Duisburg-Essen.

[191] Axel Van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proceedings of the 22nd international conference on Software engineering*, pages 5–19, 2000.

[192] Carlos Vicient, David Sánchez, and Antonio Moreno. An automatic approach for ontology-based feature extraction from heterogeneous textual resources. *Engineering Applications of Artificial Intelligence*, 26(3):1092–1106, March 2013.

[193] Paul Viola and Mukund Narasimhan. Learning to extract information from semi-structured text using a discriminative context free grammar. In *SIGIR'05*, pages 330–337. ACM, 2005.

[194] Ben L Di Vito and Larry W Roberts. Using Formal Methods to Assist in the Requirements Analysis of the Space Shuttle GPS Change Request. Technical Report August, NASA Langley Technical Report Server, 1996.

[195] Jan vom Brocke, Alan Hevner, and Alexander Maedche. *Introduction to Design Science Research*, pages 1–13. Springer International Publishing, Cham, 2020.

[196] Thomas von der Maßen and Horst Lichter. Requiline: A requirements engineering tool for software product lines. In *Software Product-Family Engineering*, volume 3014 of *Lecture Notes in Computer Science*, pages 168–180. Springer Berlin Heidelberg, 2004.

[197] Eric Walkingshaw. Features and Feature Models : A Survey of Variation Representations. Technical report, School of EECS, Oregon State University, 2010.

[198] Hai Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, and Jeff Pan. A semantic web approach to feature modeling and verification. In *In Workshop on Semantic Web Enabled Software Engineering (SWESE '05)*, 2005.

[199] Hai H. Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, and Jeff Pan. Verifying feature models using OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):117–129, June 2007.

[200] Jia-Bing Wang, Hong Peng, and Jing-Song Hu. Automatic keyphrases extraction from document using backpropagation. In *2005 International Conference on Machine Learning and Cybernetics*, volume 6, pages 633–641. IEEE, 2005.

[201] Nathan Weston, Ruzanna Chitchyan, and Awais Rashid. A Framework for Constructing Semantically Composable Feature Models from Natural Language Requirements. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 211–220, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[202] Nathan Weston and Awais Rashid. ArborCraft: Automatic Feature Models from Textual Requirements Documents. *Proc. of the 15th Workshop on Early aspects (EA), 8th Int. Conference on Aspect-Oriented Software Development (AOSD)*, pages 45–46, 2009.

[203] Jules White, Brian Dougherty, Doulas C. Schmidt, and David Benavides. Automated reasoning for multi-step feature model configuration problems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 11–20, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[204] Jim Woodcock and Juan Bicarregui. Formal Methods : Practice and Experience. *ACM Computing Surveys*, 41(4):1–40, 2009.

[205] Prasanth Yalla and Nakul Sharma. Parsing natural language text of use case description. In *2014 Conference on IT in Business, Industry and Government (CSIBIG)*, pages 1–3, 2014.

[206] Hua Yan, Wei Zhang, Haiyan Zhao, and Hong Mei. An optimization strategy to feature models' verification by eliminating verification-irrelevant features and constraints. In *Formal Foundations of Reuse and Domain Engineering*, volume 5791 of *Lecture Notes in Computer Science*, pages 65–75. Springer Berlin Heidelberg, 2009.

[207] Rongjie Yan, Chih-Hong Cheng, and Yesheng Chai. Formal consistency checking over specifications in natural languages. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1677–1682. IEEE, 2015.

[208] Badamasi Imam Ya'u, Azlin Nordin, and Norsaremah Salleh. Software requirements patterns and meta model: A strategy for enhancing requirements reuse (rr). In *2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*, pages 188–193, November 2016.

[209] Li Yi, Wei Zhang, Haiyan Zhao, Zhi Jin, and Hong Mei. Mining binary constraints in the construction of feature models. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 141–150. IEEE, September 2012.

[210] Li Yi, Haiyan Zhao, Wei Zhang, and Zhi Jin. CoFM: An environment for collaborative feature modeling. *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 317–318, September 2012.

[211] Pamela Zave. Classification of research efforts in requirements engineering. *ACM Computing Surveys (CSUR)*, 29(4):315–321, 1997.

[212] Jiayi Zhang, Bilal Ahmad, Daniel Vera, and Robert Harrison. Ontology based semantic-predictive model for reconfigurable automation systems. In *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, pages 1094–1099, July 2016.

[213] Wei Zhang, Hong Mei, and Haiyan Zhao. Feature-driven requirement dependency analysis and high-level software design. *Requirements Engineering*, 11(3):205–220, 2006.

[214] Wei Zhang, Haiyan Zhao, and Hong Mei. A propositional logic-based method for verification of feature models. In *Formal Methods and Software Engineering*, volume 3308 of *Lecture Notes in Computer Science*, pages 115–130. Springer Berlin Heidelberg, 2004.

[215] Hong Zhu and Lingzi Jin. Scenario Analysis in an Automated Tool for Requirements Engineering. *Requirements Engineering*, 5(1):2–22, July 2000.

Appendix A

Comparison between FM - FeatureX vs manual

bCMS

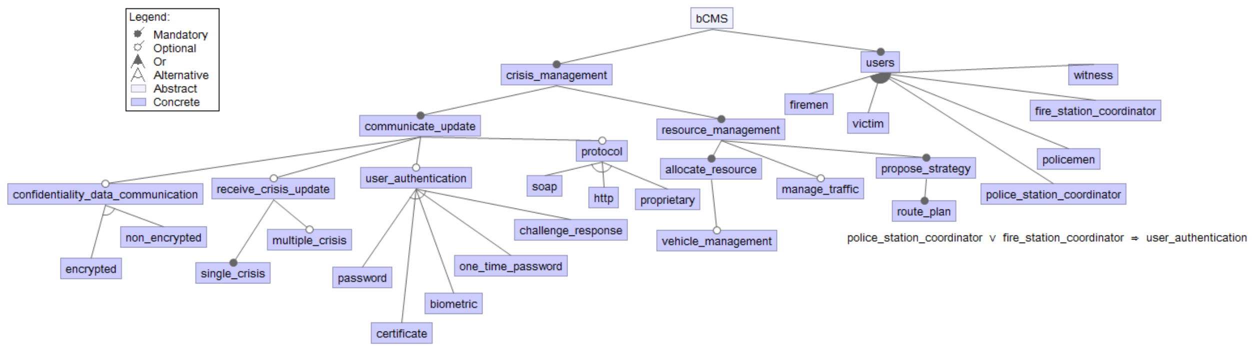


Figure A.1: bCMS - benchmark FM (manually created in FeatureIDE).

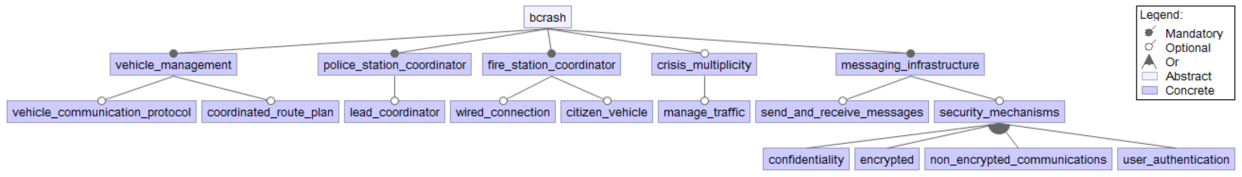


Figure A.2: bCMS - FeatureX FM (manually created in FeatureIDE).

Cryptocurrency

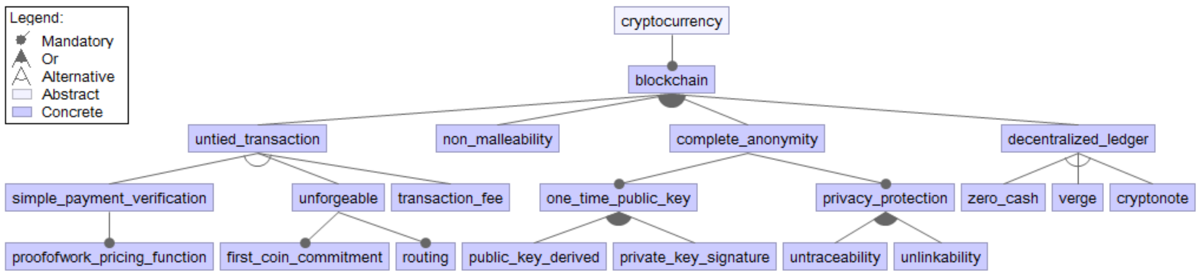


Figure A.3: Cryptocurrency - benchmark FM (manually created in FeatureIDE).

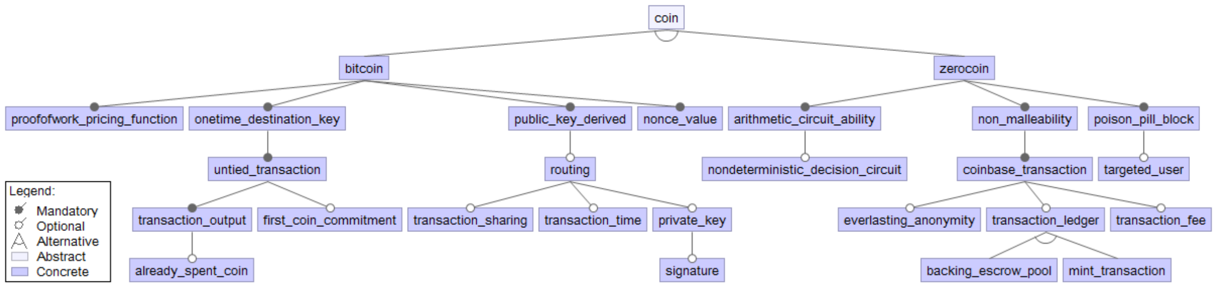


Figure A.4: Cryptocurrency - FeatureX FM (manually created in FeatureIDE).

Identity

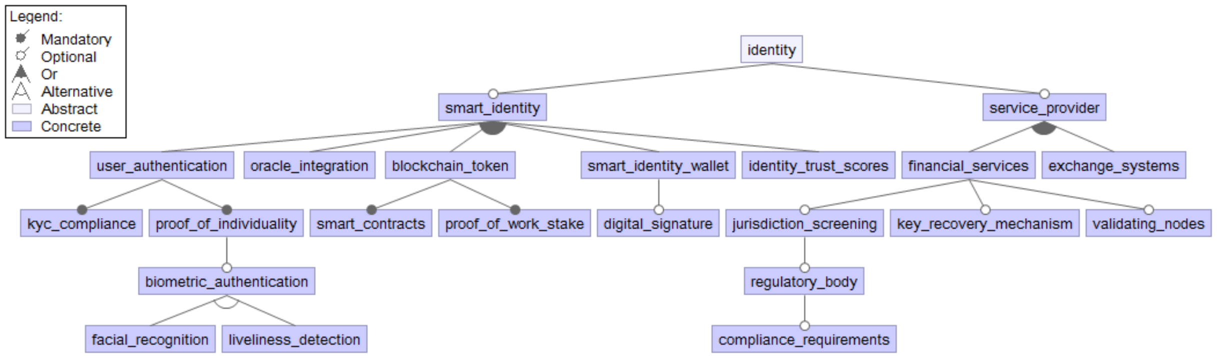


Figure A.5: Identity - benchmark FM (manually created in FeatureIDE).

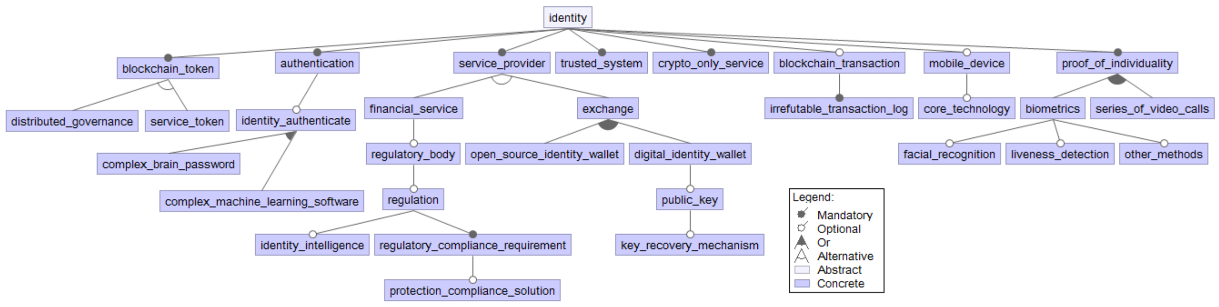


Figure A.6: Identity - FeatureX FM (manually created in FeatureIDE).

Antivirus

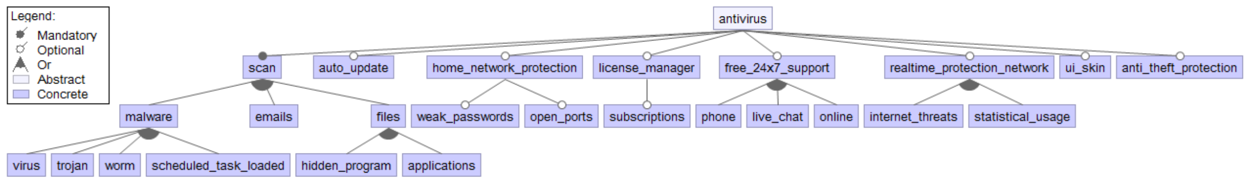


Figure A.7: Antivirus - benchmark FM (manually created in FeatureIDE).

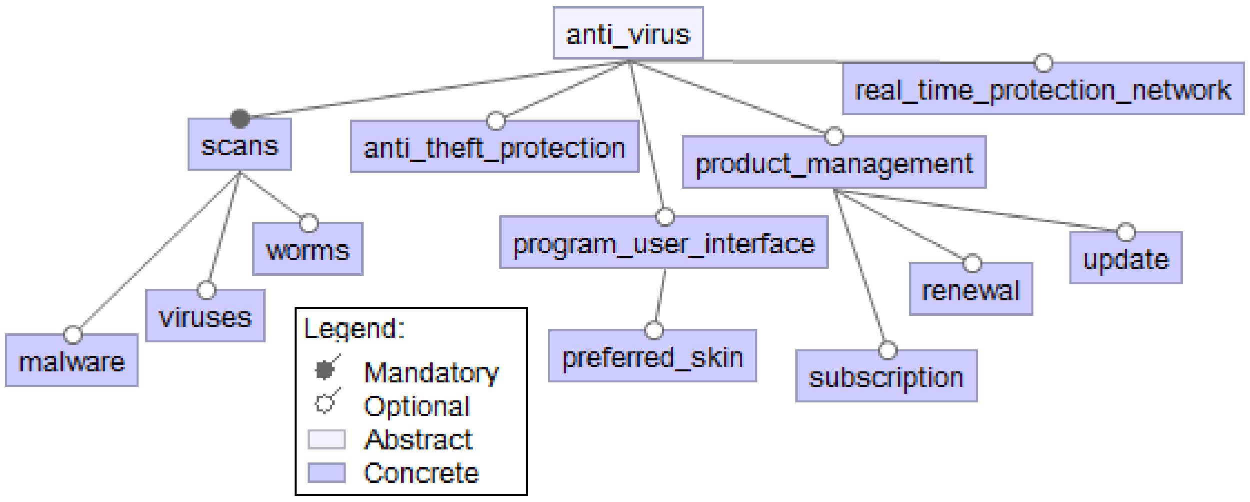


Figure A.8: Antivirus - FeatureX FM (manually created in FeatureIDE).

E-shop

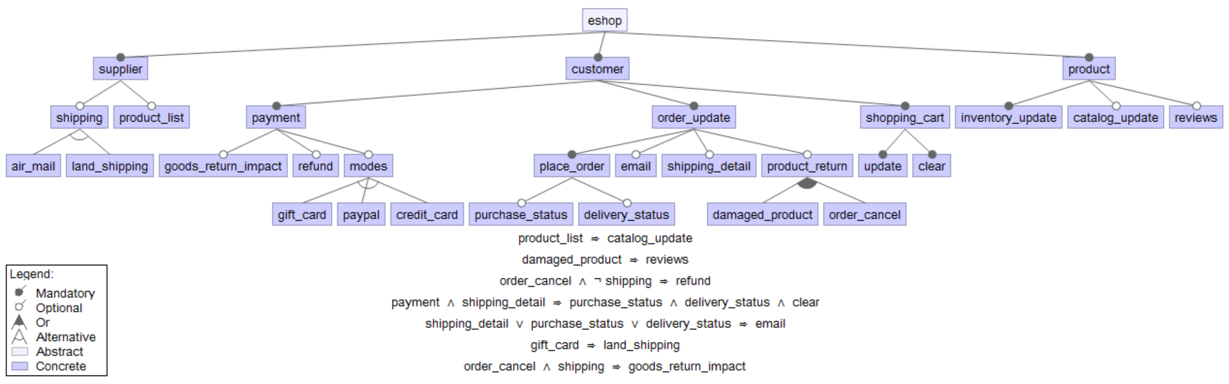


Figure A.9: E-shop - benchmark FM (manually created in FeatureIDE).

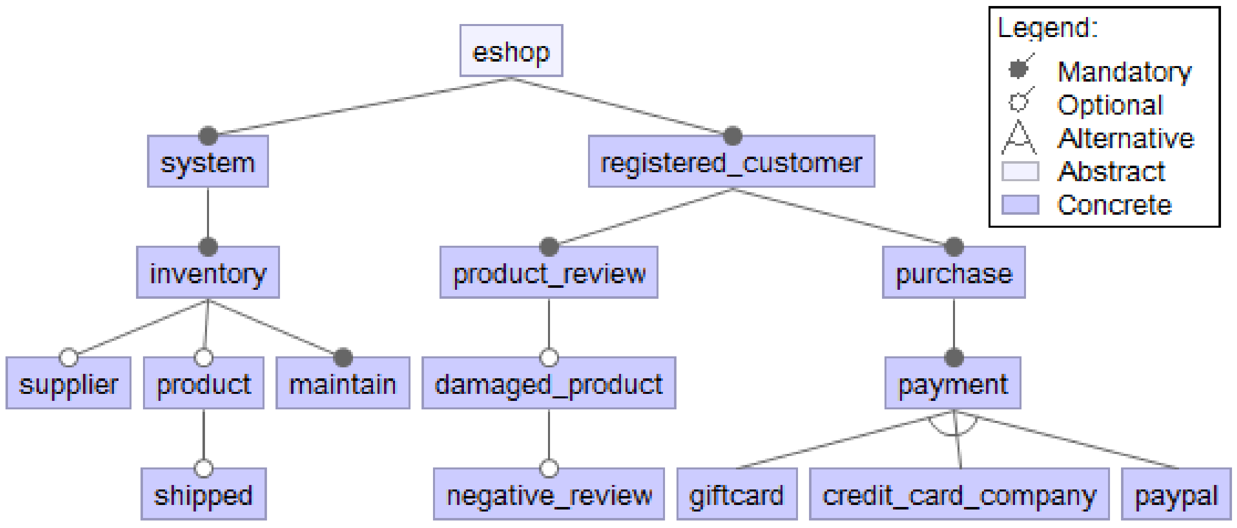


Figure A.10: E-shop - FeatureX FM (manually created in FeatureIDE).

Smart Home

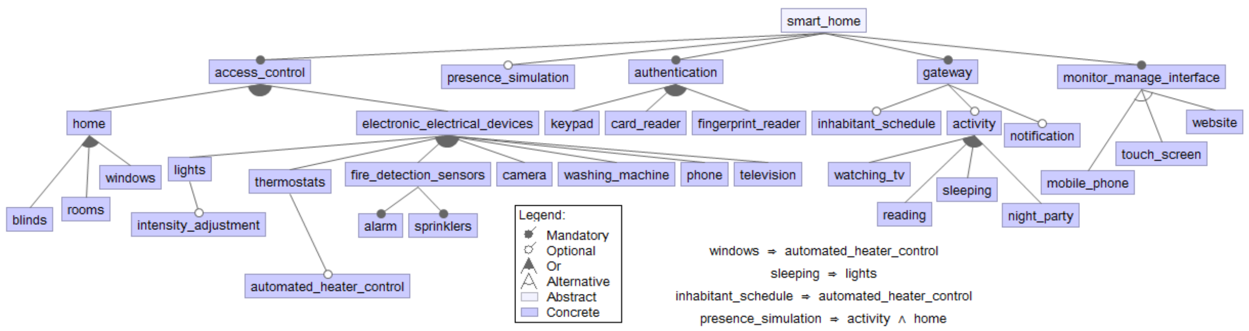


Figure A.11: Smart Home - benchmark FM (manually created in FeatureIDE).

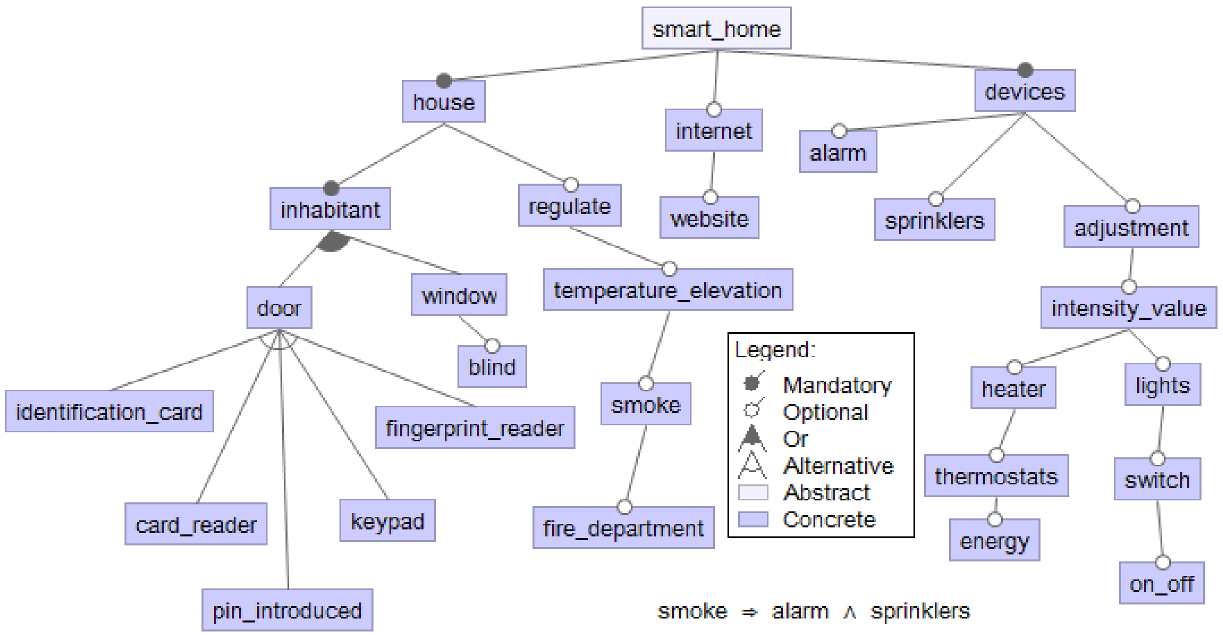


Figure A.12: Smart Home - FeatureX FM (manually created in FeatureIDE).

A.1 FMs from literature

A.1.1 E-shop

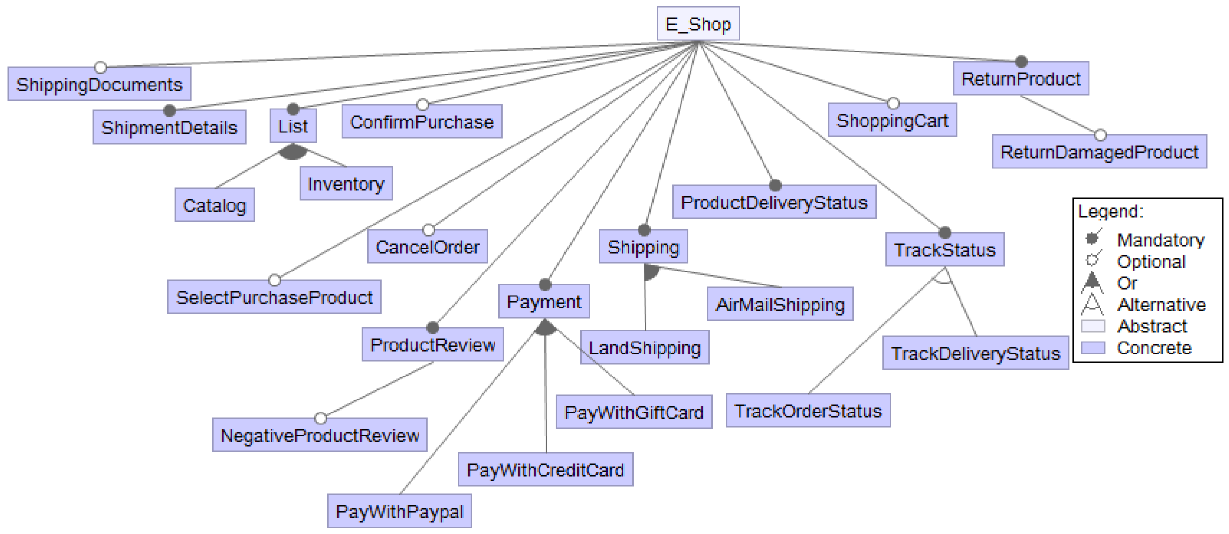
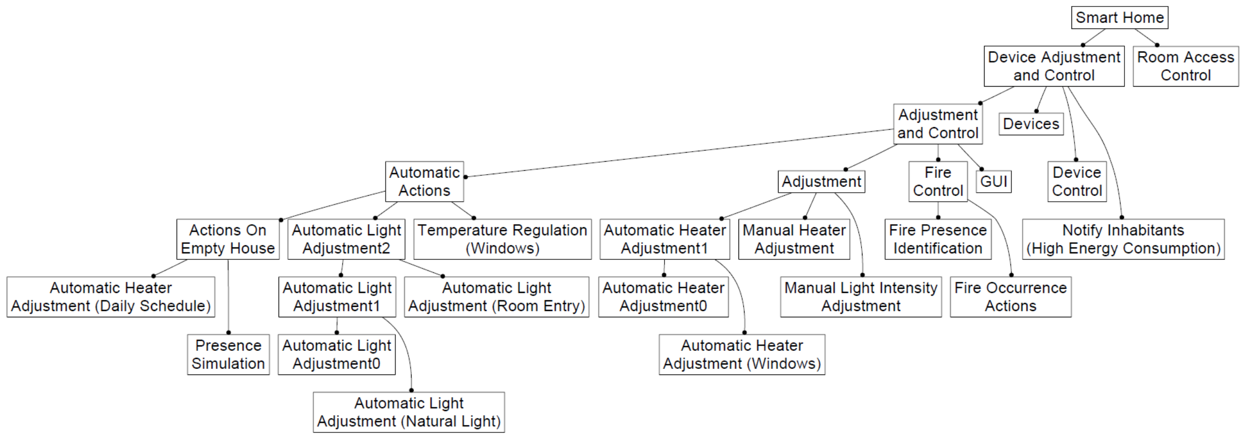


Figure A.13: E-shop - FM from literature (manually created in FeatureIDE).

A.1.2 Smart home



Smart Home
Device Adjustment and Control
Room Access Control
Adjustment and Control
Devices
Automatic Actions
Adjustment
Fire Control
GUI
Device Control
Actions On Empty House
Automatic Light Adjustment2
Temperature Regulation (Windows)
Automatic Heater Adjustment1
Manual Heater Adjustment
Fire Presence Identification
Notify Inhabitants (High Energy Consumption)
Automatic Heater Adjustment (Daily Schedule)
Automatic Light Adjustment1
Automatic Light Adjustment (Room Entry)
Automatic Heater Adjustment0
Manual Light Intensity Adjustment
Fire Occurrence Actions
Presence Simulation
Automatic Light Adjustment0
Automatic Heater Adjustment (Windows)
Automatic Light Adjustment (Natural Light)

Figure A.14: Smart Home - FM from literature.

A.1.3 Anti-virus

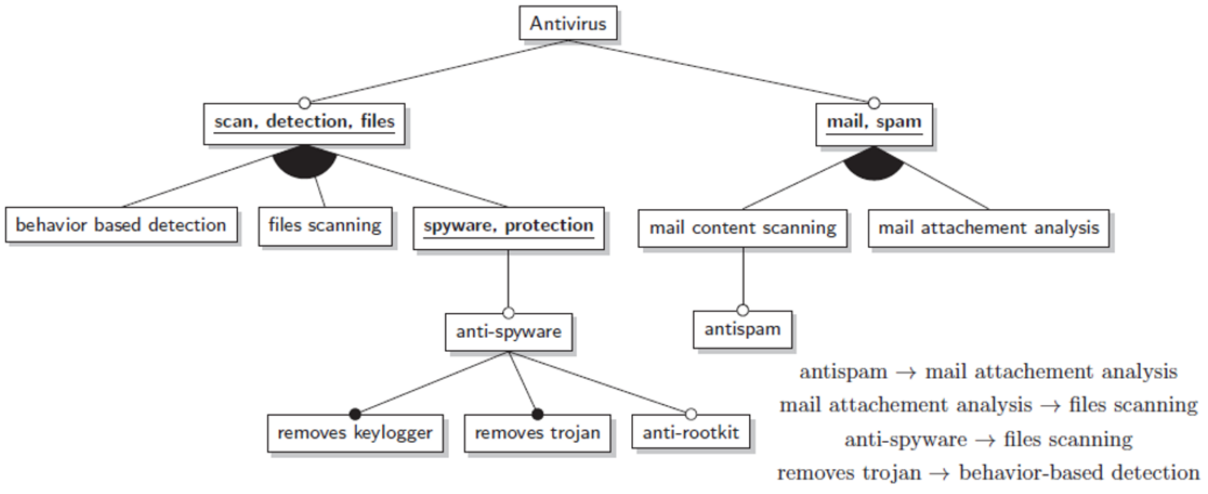


Figure A.15: Anti-virus - FM from literature.

A.2 Manual evaluation of FeatureX models with benchmark models by domain engineers

A.2.1 Legend used for relationship type

One-to-one mapping	One-to-many mapping
[**M**] MANDATORY	[**A**] AND
[**O**] OPTIONAL	[**R**] OR
[**E**] EXCLUDES	[**X**] ALTERNATIVE/XOR
[**I**] REQUIRE/IMPLIES	

Table A.1: Types of relationships among features.

A.2.2 Crash management product line

Candidate feature pair	Corrected name (if needed)	Relationship exists?	Correct direction?	Confidence and type
(bcrash, vehicle_management)	(bcrash, vehicle_management)	✓	✓	100% **M**
(bcrash, fire_station_coordinator)	(bcrash, fire_station_coordinator)	✓	✓	100% **M**
(bcrash, police_station_coordinator)	(bcrash, police_station_coordinator)	✓	✓	100% **M**
(bcrash, crisis_multiplicity)	(bcrash, crisis_multiplicity)	✓	✓	100% **O**
(bcrash, messaging_infrastructure)	(bcrash, messaging_infrastructure)	✓	✓	100% **M**
(vehicle_management, vehicle_communication_protocol)	(vehicle_management, communication_protocol)	✓	✓	60% **O**
(vehicle_management, coordinated_route_plan)	(vehicle_management, coordinated_route_plan)	✓	✓	60% **O**
(police_station_coordinator, lead_coordinator)	-	×	×	-
(fire_station_coordinator, wired_connection)	-	×	×	-
(fire_station_coordinator, citizen_vehicle)	(fire_station_coordinator, citizen_vehicle)	✓	✓	60% **O**
(crisis_multiplicity, manage_traffice)	(vehicle_management, manage_traffice)	✓	×	80% **O**
(messaging_infrastructure, send_and_receive_messages)	(messaging_infrastructure, send_and_receive_messages)	×	✓	90% **M**
(messaging_infrastructure, security_mechanisms)	(messaging_infrastructure, security_mechanisms)	✓	×	90% **M**
(security_mechanisms, confidentiality)	(security_mechanisms, confidentiality)	✓	✓	90% **R**
(security_mechanisms, encrypted)	(security_mechanisms, encrypted)	✓	✓	90% **R**
(security_mechanisms, non_encrypted_communications)	(security_mechanisms, non_encrypted)	✓	✓	90% **R**
(security_mechanisms, user_authentication)	(security_mechanisms, user_authentication)	✓	✓	90% **R**

Table A.2: FeatureX - bCMS

A.2.3 Cryptocurrency product line

Candidate feature pair	Corrected name (if needed)	Relationship exists?	Correct direction?	Confidence and type
(coin, bitcoin)	(coin, bitcoin)	✓	✓	90% **X**
(coin, zerocoin)	(coin, zerocoin)	✓	✓	90% **X**
(bitcoin, proofofwork_pricing_function)	(bitcoin, proofofwork_pricing_function)	✓	✓	90% **M**
(bitcoin, onetime_destination_key)	(bitcoin, onetime_destination_key)	✓	✓	90% **M**
(bitcoin, pulic_key_derived)	-	×	×	-
(bitcoin, nonce_value)	(bitcoin, nonce_value)	✓	✓	80% **M**
(zerocoin, arithmetic_circuit_ability)	(zerocoin, arithmetic_circuit_ability)	✓	✓	80% **M**
(zerocoin, non_malleability)	(zerocoin, non_malleability)	✓	✓	80% **M**
(zerocoin, poison_pill_block)	(zerocoin, poison_pill_block)	✓	✓	70% **M**
(onetime_destination_key, united_transaction)	-	×	×	-
(united_transaction, transaction_output)	-	×	×	-
(united_transaction, first_coin_commitment)	-	×	×	-
(transaction_output, already_spent_coin)	-	×	×	-
(public_key_derived, routing)	(public_key, routing)	×	✓	70% **M**
(routing, transaction_sharing)	(routing, transaction_sharing)	✓	✓	90% **O**
(routing, transaction_time)	-	×	×	-
(routing, private_key)	(routing, private_key)	×	✓	80% **M**
(private_key, signature)	(private_key, signature)	×	✓	90% **M**
(arithmetic_circuit_ability, non_deterministic_decision_circuit)	(arithmetic_circuit_ability, non_deterministic_decision_circuit)	✓	✓	100% **O**
(non_malleability, coinbase_transaction)	(transaction, non_malleability)	✓	×	100% **M**
(coinbase_transaction, everlasting_anonymity)	(coinbase_transaction, anonymity)	×	✓	90% **M**
(coinbase_transaction, transaction_ledger)	(coinbase_transaction, transaction_ledger)	✓	✓	80% **O**
(coinbase_transaction, transaction_fee)	(coinbase_transaction, transaction_fee)	×	✓	100% **M**
(transaction_ledger, backing_escrow_pool)	-	×	×	-
(transaction_ledger, mint_transaction)	-	×	×	-

Table A.3: FeatureX - Cryptocurrency

A.2.4 Identity software product line

Candidate feature pair	Corrected name (if needed)	Relationship exists?	Correct direction?	Confidence and type
(identity, blockchain)	(identity, blockchain)	✓	✓	100% **M**
(identity, authentication)	(identity, authentication)	✓	✓	100% **M**
(identity, service_provider)	-	×	×	-
(identity, trusted_system)	(identity, trusted_system)	✓	✓	100% **M**
(identity, crypto_only_service)	-	×	×	-
(identity, blockchain_transaction)	(identity, blockchain_transaction)	✓	✓	80% **O**
(identity, mobile_device)	(identity, mobile_device)	✓	✓	100% **O**
(identity, proof_of_individuality)	(identity, proof_of_individuality)	✓	✓	100% **M**
(blockchain_token, blockchain_governance)	(blockchain_token, blockchain_governance)	✓	✓	70% **X**
(blockchain_token, service_token)	(blockchain_token, service_token)	✓	✓	70% **X**
(authentication, identity_authenticate)	(authentication, identity_authenticate)	×	✓	100% **M**
(service_provider, financial_service)	(service_provider, financial_service)	✓	✓	50% **X**
(service_provider, exchange)	(service_provider, exchange)	✓	✓	70% **X**
(blockchain_transaction, irrefutable_transaction_log)	(blockchain_transaction, irrefutable_transaction_log)	✓	✓	100% **M**
(mobile_device, core_technology)	-	×	×	-
(proof_of_individuality, biometrics)	(proof_of_individuality, biometrics)	✓	✓	100% **R**
(proof_of_individuality, series_of_video_calls)	(proof_of_individuality, series_of_video_calls)	✓	✓	50% **R**
(biometrics, facial_recognition)	(biometrics, facial_recognition)	✓	✓	100% **O**
(biometrics, liveness_detection)	(biometrics, liveness_detection)	✓	✓	100% **O**
(biometrics, other_methods)	-	×	×	-
(identity_authenticate, complex_brain_password)	(identity_authenticate, complex_brain_password)	✓	✓	80% **R**
(identity_authenticate, complex_machine_learning_software)	(identity_authenticate, machine_learning_software)	✓	✓	60% **R**
(financial_service, regulatory_body)	-	×	×	-
(exchange, open_source_identity_wallet)	(exchange, open_source_identity_wallet)	✓	✓	80% **R**
(exchange, digital_identity_wallet)	(exchange, digital_identity_wallet)	✓	✓	90% **R**
(digital_identity_wallet, public_key)	(digital_identity_wallet, public_key)	×	✓	90% **M**
(public_key, key_recovery_mechanism)	(public_key, key_recovery_mechanism)	✓	✓	100% **O**
(regulatory_body, regulation)	-	×	×	-
(regulation, identity_intelligence)	(regulation, identity_intelligence)	×	✓	80% **O**
(regulation, regulatory_compliance_requirement)	-	×	×	-
(regulatory_compliance_requirement, protection_compliance_solution)	-	×	×	-

Table A.4: FeatureX - Identity SPL

A.2.5 Anti-virus product line

Candidate feature pair	Corrected name (if needed)	Relationship exists?	Correct direction?	Confidence and type
(anti_virus, scans)	(anti_virus, scans)	✓	✓	100% **M**
(anti_virus, anti_theft_protection)	(anti_virus, anti_theft_protection)	×	✓	100% **M**
(anti_virus, product_management)	(anti_virus, product_management)	✓	✓	100% **O**
(anti_virus, real_time_protection_network)	(anti_virus, real_time_protection_network)	✓	✓	100% **O**
(anti_virus, program_user_interface)	(anti_virus, program_user_interface)	✓	✓	80% **O**
(scans, malware)	(scans, malware)	×	✓	60% **R**
(scans, viruses)	(scans, viruses)	×	✓	60% **R**
(scans, worms)	(scans, worms)	×	✓	60% **R**
(program_user_interface, preferred_skin)	(program_user_interface, preferred_skin)	✓	✓	70% **O**
(product_management, subscription)	(product_management, subscription)	✓	✓	100% **O**
(product_management, renewal)	(product_management, renewal)	✓	✓	100% **O**
(product_management, update)	(product_management, update)	✓	✓	100% **O**

Table A.5: FeatureX - Anti-virus

A.2.6 E-shop product line

Candidate feature pair	Corrected name (if needed)	Relationship exists?	Correct direction?	Confidence and type
(eshop, system)	-	×	×	-
(eshop, registered_customer)	(eshop, customer)	✓	✓	100% **M**
(system, inventory)	(eshop, inventory)	✓	✓	100% **M**
(registered_customer, product_review)	(registered_customer, product_review)	×	✓	90% **O**
(registered_customer, purchase)	(customer, purchase)	✓	✓	90% **M**
(inventory, supplier)	(inventory, supplier)	×	✓	70% **M**
(inventory, product)	(inventory, product)	×	✓	70% **M**
(inventory, maintain)	(inventory, maintain)	✓	✓	70% **M**
(product_review, damaged_product)	-	×	×	-
(damaged_product, negative_review)	-	×	×	-
(purchase, payment)	(purchase, payment)	✓	✓	90% **M**
(payment, gift_card)	(payment, gift_card)	✓	✓	100% **A**
(payment, credit_card_company)	(payment, credit_card_company)	✓	✓	100% **A**
(payment, paypal)	(payment, paypal)	✓	✓	100% **A**

Table A.6: FeatureX - E-shop

A.2.7 Smart home product line

Candidate feature pair	Corrected name (if needed)	Relationship exists?	Correct direction?	Confidence and type
(smart_home, house)	-	×	×	-
(smart_home, internet)	-	×	×	-
(smart_home, devices)	(smart_home, devices)	✓	✓	100% **M**
(devices, alarm)	(devices, alarm)	×	✓	90% **A**
(devices, sprinklers)	(devices, sprinklers)	×	✓	90% **A**
(devices, adjustment)	-	×	×	-
(adjustment, intensity_value)	(devices, adjust_intensity_value)	✓	✓	100% **O**
(intensity_value, heater)	(devices, heater)	×	✓	60% **R**
(intensity_value, lights)	(devices, lights)	×	✓	60% **R**
(heater, thermostats)	-	×	×	-
(thermostats, energy)	-	×	×	-
(lights, switch)	-	×	×	-
(switch, on_off)	-	×	×	-
(internet, website)	-	×	×	-
(house, inhabitant)	-	×	×	-
(house, regulate)	(house, regulate)	×	✓	60% **M**
(inhabitant, door)	-	×	×	-
(inhabitant, window)	-	×	×	-
(window, blind)	-	×	×	-
(door, identification_card)	(door, identification_card)	✓	✓	70% **X**
(door, card_reader)	(door, card_reader)	✓	✓	80% **X**
(door, pin_introduced)	(door, pin_introduced)	✓	✓	80% **X**
(door, keypad)	(door, keypad)	✓	✓	80% **X**
(door, fingerprint_reader)	(door, fingerprint_reader)	✓	✓	80% **X**
(regulate, temperature_elevation)	(regulate, temperature_elevation)	✓	✓	100% **O**
(temperature_elevation, smoke)	(temperature_elevation, smoke)	✓	✓	100% **O**
(smoke, fire_department)	(smoke, fire_department)	✓	✓	100% **O**

Table A.7: FeatureX - Home automation

A.3 Manual evaluation of FM generated by methods used in literature

Candidate feature pair	Corrected name (if needed)	Relationship exists?	Correct direction?	Confidence and type
(E_shop, ShipmentDetails)	(E_shop, ShipmentDetails)	✓	✓	90% **M**
(E_shop, ShippingDocuments)	-	×	×	-
(E_shop, List)	-	×	×	-
(E_shop, ConfirmPurchase)	(E_shop, ConfirmPurchase)	×	✓	90% **M**
(E_shop, SelectPurchaseProduct)	(E_shop, PurchaseProduct)	×	✓	90% **M**
(E_shop, CancelOrder)	(E_shop, CancelOrder)	✓	✓	90% **O**
(E_shop, ProductReview)	(E_shop, ProductReview)	×	✓	90% **O**
(E_shop, Payment)	(E_shop, Payment)	✓	✓	80% **M**
(E_shop, Shipping)	(E_shop, Shipping)	✓	✓	80% **M**
(E_shop, ProductDeliveryStatus)	-	×	×	-
(E_shop, ShoppingCart)	(E_shop, ShoppingCart)	×	✓	60% **R**
(E_shop, TrackStatus)	(E_shop, TrackStatus)	×	✓	60% **R**
(E_shop, ReturnProduct)	(E_shop, ReturnProduct)	×	✓	60% **R**
(List, Catalog)	-	×	×	-
(List, Inventory)	-	×	×	-
(ProductReview, NegativeProductReview)	-	×	×	-
(Payment, PayWithCreditCard)	(Payment, CreditCard)	×	✓	90% **X**
(Payment, PayWithGiftCard)	(Payment, GiftCard)	×	✓	90% **X**
(Payment, PayWithPaypal)	(Payment, Paypal)	×	✓	90% **X**
(Shipping, LandShipping)	(Shipping, Land)	✓	✓	90% **X**
(Shipping, AirMailShipping)	(Shipping, AirMail)	✓	✓	90% **X**
(TrackStatus, TrackOrderStatus)	(TrackStatus, OrderStatus)	×	✓	90% **O**
(TrackStatus, TrackDeliveryStatus)	(TrackStatus, DeliveryStatus)	×	✓	90% **O**
(ReturnProduct, ReturnDamagedProduct)	(ReturnProduct, DamagedProduct)	✓	✓	90% **O**

Table A.8: E-shop

Candidate feature pair	Corrected name (if needed)	Relationship exists?	Correct direction?	Confidence and type
(Smart Home, Device Adjustment and Control)	(Smart Home, Device Adjustment and Control)	✓	✓	100% **M**
(Smart Home, Room Access Control)	(Smart Home, Room Access Control)	×	✓	100% **O**
(Device Adjustment and Control, Adjustment and Control)	-	×	×	-
(Device Adjustment and Control, Devices)	-	×	×	-
(Device Adjustment and Control, Device Control)	-	×	×	-
(Device Adjustment and Control, Notify Inhabitants (High Energy Consumption))	-	×	×	-
(Adjustment and Control, Automatic Actions)	(Adjustment and Control, Automatic Actions)	×	✓	60% **O**
(Adjustment and Control, Fire Control)	(Adjustment and Control, Fire Control)	×	✓	60% **O**
(Adjustment and Control, Adjustment)	-	×	×	-
(Adjustment and Control, GUI)	(Adjustment and Control, GUI)	×	✓	60% **O**
(Automatic Actions, Actions On Empty House)	(Automatic Actions, On Empty House)	×	✓	80% **O**
(Automatic Actions, Automatic Light Adjustment2)	(Automatic Actions, Light Adjustment)	×	✓	80% **O**
(Automatic Actions, Temperature Regulation (Windows))	(Automatic Actions, Temperature Regulation)	×	✓	90% **O**
(Adjustment, Automatic Heater Adjustment)	(Automatic Actions, Heater Adjustment)	×	✓	60% **O**
(Adjustment, Manual Light Intensity Adjustment)	(Device Adjustment and Control, Manual Light Intensity Adjustment)	✓	✓	90% **O**
(Adjustment, Manual Heater Adjustment)	(Device Adjustment and Control, Manual Heater Adjustment)	×	✓	90% **O**
(Fire Control, Fire Presence Identification)	(Fire Control, Fire Presence Identification)	✓	✓	80% **M**
(Fire Control, Fire Occurrence Actions)	(Fire Control, Fire Occurrence Actions)	✓	✓	80% **M**
(Actions On Empty House, Automatic Heater Adjustment (Daily Schedule))	-	×	×	-
(Actions On Empty House, Presence Simulation)	(On Empty House, Presence Simulation)	×	✓	90% **O**
(Automatic Light Adjustment2, Automatic Light Adjustment1)	-	×	×	-
(Automatic Light Adjustment2, Automatic Light Adjustment(Room Entry)	-	×	×	-
(Automatic Light Adjustment1, Automatic Light Adjustment0)	-	×	×	-
(Automatic Light Adjustment1, Automatic Light Adjustment (Windows)	-	×	×	-

Table A.9: Smart home

Candidate feature pair	Corrected name (if needed)	Relationship exists?	Correct direction?	Confidence and type
(Antivirus, scan/detection/files)	(Antivirus, scan/detection/files)	×	✓	80% **M**
(Antivirus, mail/spam)	(Antivirus, mail/spam)	✓	✓	80% **O**
(scan/detection/files, behavior based detection)	-	×	×	-
(scan/detection/files, files scanning)	(scan, files)	✓	✓	80% **R**
(scan/detection/files, anti-spyware/protection)	(detection, spyware_protection)	✓	✓	80% **R**
(anti-spyware/protection , removes keylogger)	(spyware_protection , removes keylogger)	✓	✓	90% **M**
(anti-spyware/protection , removes trojan)	(spyware_protection , removes trojan)	×	✓	90% **O**
(anti-spyware/protection , anti-rootkit)	(spyware_protection , anti-rootkit)	✓	✓	90% **O**
(mail/spam, mail content scanning)	(mail_spam, content scanning)	✓	✓	90% **R**
(mail/spam, mail attachement analysis)	(mail_spam, attachement analysis)	✓	✓	90% **R**
(mail content scanning, antispam)	(content scanning, antispam)	✓	✓	90% **O**
(spyware/protection,anti-spyware)	(spyware/protection,anti-spyware)	×	✓	90% **M**
(antispam, mail attachment analysis)	(antispam, mail attachment analysis)	✓	✓	60% **I**
(mail attachment analysis, files scanning)	(mail attachment analysis, files scanning)	✓	✓	60% **I**
(anti-spyware, file scanning)	(anti-spyware, file scanning)	✓	✓	60% **I**
(removes trojan, behavior-based detection)	(removes trojan, behavior-based detection)	✓	✓	60% **I**

Table A.10: Anti-virus