Extracting Software Product Line Feature Models from Natural Language Specifications

Anjali Sree-Kumar
Universitat Oberta de Catalunya (UOC)
Barcelona, Spain
asree_kumar@uoc.edu

Elena Planas
Universitat Oberta de Catalunya (UOC)
Barcelona, Spain
eplanash@uoc.edu

Robert Clarisó
Universitat Oberta de Catalunya (UOC)
Barcelona, Spain
rclariso@uoc.edu

ABSTRACT

The specification of a family of software products may include documents written in natural language. Automatically extracting knowledge from these documents is a challenging problem that requires using Natural Language Processing (NLP) techniques. This knowledge can be formalized as a Feature Model (FM), a diagram capturing the key features and the relationships among them.

In this paper, we first review previous works that have presented tools for extracting FMs from textual specifications and compare their strengths and limitations. Then, we propose a framework for feature and relationship extraction, which overcomes the identified limitations and is built upon state-of-the-art open-source NLP tools. This framework is evaluated against previous works using several case studies, showing improved results.

CCS CONCEPTS

• **Software and its engineering → Requirements analysis**; **Software product lines**; *Software prototyping*;

KEYWORDS

Software Product Line, Natural Language Processing, Feature Model Extraction, Requirements Engineering, NLTK

ACM Reference Format:
Anjali Sree-Kumar, Elena Planas, and Robert Clarisó. 2018. Extracting Software Product Line Feature Models from Natural Language Specifications. In *SPLC '18: 22nd International Systems and Software Product Line Conference, September 10–14, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3233027.3233029

1 INTRODUCTION

A *Software Product Line* (SPL) is a family of systems that share a common set of core technical assets, with preplanned extensions and variations to address the needs of specific customers or market segments [12]. An advantage of product line engineering in software product development is the ease of reusing already tested and proven product features, which improves the time to market.

SPLC '18, September 10–14, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-6464-5/18/09…$15.00
https://doi.org/10.1145/3233027.3233029

Visualizing the variabilities and commonalities among the products in a product line is essential for making configuration decisions, *e.g.* including or excluding a product feature. To this end, we can use a graphical formalism like *feature model* (FM), which represents the standard features and the relationships between them for a family of products/systems in the domain [21]. Among other relationships, the FM identifies common (i.e., mandatory), alternative, and optional features in a given product line [21]. A domain engineer can then go through this model and configure new products based on the new requirements. Nevertheless, first it is necessary to extract the (formal) FM from the (informal) specification of the product line, which may include documents written in natural language.

Example 1.1. 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 1 depicts a potential feature model for this SPL, manually created by a domain engineer. It captures the features (bolded phrases) and relationships among those features (underlined words) described in the above excerpt which are relevant according to that engineer's perspective. Our goal is to automatically extract such 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.

The contributions of this paper 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 [7]; (3) an evaluation of its effectiveness with respect to the feedback from domain engineers; and (4) a comparison with other approaches from the literature.

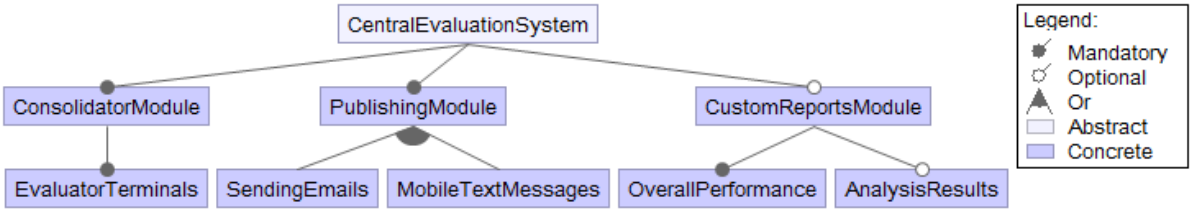

Figure 1: Manually created feature model in FeatureIDE [22] - Online Examination Coordinator.

The remainder of this paper is structured as follows. A review of existing research works on automatic or semi-automatic FM extraction is presented in Section 2. Section 3 describes the FeatureX framework for FM extraction, and Section 4 provides details on the implementation of this framework. Section 5 presents the preliminary 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. Finally, Section 6 concludes with a discussion on future work.

2 RELATED WORK

Extracting implicit variability information manually is tedious and error-prone. Hence, several works 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 Section, we focus only on works that extract FMs from documents written in *natural language*.

Several approaches have been proposed to address this problem [1, 4, 8, 9, 11, 16, 20, 23–27, 29, 31, 34, 35, 37]. Bakar et al. [5] have systematically studied the different feature extraction approaches from natural language requirements proposed in the literature. 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 proposal for two reasons:

- There are few papers that provide a publicly available working implementation or prototype of the proposed method, or even sufficient details to reproduce it accurately.
- There is a lack of public 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.

A review of the most relevant existing proposals to extract (a) only features, (b) only relationships and (c) both feature *and* feature relationships (the complete FM) from requirements documents has been carried out. We now discuss each of these categories and finally, in Subsection 2.1, provide a succinct comparison among these approaches and report their limitations in Subsection 2.2.

Feature extraction. Arora et al. [2] 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. [3] 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 domain model.

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. [37] proposed an approach for detecting *binary constraints* among features, which identifies only *requires* or *excludes* relationships among pairs of features.

FM extraction. Itzik et al. [18] have considered two perspectives for the visualization of variability. These are *structural or objects-related* view and *functional or actions-related* view. 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 [19] which is not publicly accessible. Hence, it was not possible to reproduce their results. For the evaluation of our framework, the FM studied in paper [18] has been used to compare the results.

Dumitru et al. [15] and Hamza et al. [17] have presented recommender systems that models and recommends product features for a given domain. Dumitru et al. [15] have used text mining and clustering algorithms for generating probabilistic feature models with feature relationships and cross tree constraints. Whereas Hamza et al. [17] have used text preprocessing and an NLP-based approach for identifying feature terms and constructing the FM.

Weston et al. [36] 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) [1] is used for extracting feature models from text specifications and EA-Miner [33] is used for identifying cross-tree constraints among the extracted features. According to the authors, the tools are no longer maintained.

Davril et al. [13] 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

Table 1: Literature review summary.

	Evaluation Criteria	Citations
TECHNIQUE	**Pattern matching**	[2, 16–18, 23, 27, 31, 34]
	Machine learning	[2, 8, 11, 13, 20, 23, 35, 37]
	NLP†	[1–4, 9, 13, 16–18, 23–27, 29, 31, 34–36]
	Information retrieval	[1, 31]
GOAL	**Only features**†	[2, 4, 8, 9, 16, 20, 26, 29, 31, 34, 35]
	Only relationships	[37]
	Features and relationships	[1, 3, 11, 13, 17, 18, 23–25, 27, 36]
INPUT	**Text document**†	[2, 3, 8, 9, 17, 18, 23, 24, 26, 27, 34–37]
	Words list	[3, 20]
	Document collection	[1, 11, 13, 16, 25, 29, 31]
RESULT	**Automated**	[2, 13, 16, 18, 23–25, 27, 29, 34, 36, 37]
	Semi-automated	[4, 9, 11, 26, 31]
	Prototype/Approach/Framework†	[1, 3, 4, 8, 13, 16–18, 20, 26, 31, 34, 35, 37]
	Tool support	[2, 16, 23–25, 29, 36]
LIMITATION	**Tool not available for evaluation**†	[1, 3, 4, 8, 9, 11, 13, 16–18, 20, 23–27, 31, 34–37]
	Restricted inputs	[3, 4, 13, 20]
	Automated feature naming	[1, 9, 13, 18, 24, 27, 29, 34]
	Results not reproducible†	[1, 3, 4, 8, 9, 11, 13, 16–18, 20, 23–27, 31, 34–37]
	Interaction with domain engineer	[17, 29, 36]

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

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.

2.1 Comparing existing methods

Our review involved 23 different research works, which have been compared against each other and the summary is provided in Table 1. The review information is classified under 5 different categories, each having a set of evaluation criteria:

(1) Technique used for FM extraction (TECHNIQUE) - 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) Limitations (LIMITATION) - As described in section 2.2.

2.2 Limitations

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 [15], 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 [1].
(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 [15, 17, 29] 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. [18] 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 perspective.

3 THE FeatureX FRAMEWORK

The identified limitations are the motivation to unify the various strategies applied by different researchers and move towards a centralized framework for enabling easy practice of Software Product Line Engineering. Identifying extraction rules for detecting

Table 2: Types of relationships among features.

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

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 the main contribution of this paper. 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 5. This will help in identifying the confidence level at which the extracted features can be placed.

With this goal in mind, in the next section we first provide some background on FMs (Subsection 3.1). Then, an overview of the proposed framework is described (Subsection 3.2). The feature relationships and constraints are identified based on heuristics that are explained in Subsection 3.3. These heuristics distinguish two types of relationship categories: one-to-one relationships (Subsection 3.3.1) and one-to-many relationships (Subsection 3.3.2).

3.1 Feature Models

In a software system, a *feature* is a "user-visible aspect or characteristic of the domain" [21]. 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 (left) lists the different types of one-to-one dependencies we consider: 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 (right) describes the potential one-to-many dependencies we support: 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*:

(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 $(|Y \cap P| = 1)$.

3.2 Framework overview

Our framework, **FeatureX**, is an integrated approach 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 children 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 2:

(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 our proposed heuristics for identifying and categorizing feature relationship types. The relationship text files shown as output in Figure 2 contains the results of the FeatureX relationships mining heuristics.

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 [7] 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 3.3. We have used Pattern library [14] (*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. We have used the synonym detection provided by WordNet [28] for clustering similar candidate features. We have also used open source PDF parsers like PDFMiner[1] for generating the raw text corpus from inputs that were provided as PDF requirements specifications files to FeatureX.

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 [2, 3] 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:

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

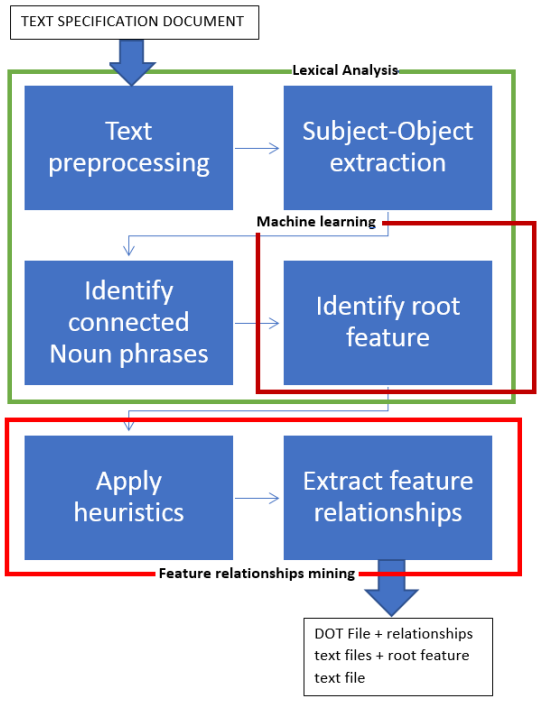

Figure 2: FeatureX overview for processing a single product specification document.

Example 3.1. We highlight the features extracted using [3] in **bold**, [2] 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 1.1.*

> *"**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, which have been extracted using the association rules taken from the respective approaches.

Table 3: Relationship phrases detected by the association rules from [2, 3].

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

As can be seen from Table 3 and Example 3.1, 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 1, 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.

This indicates that we can have extraction rules that can identify 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 format of the text. This means, 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 more challenging especially when targeting to consider all kinds of text (structured, semi-structured or unstructured) inputs. For extracting all candidate feature terms, the following NLP techniques are used:

- Stop words removal [7]: remove commonly used words such as *'a'*, *'an'*, *'the'*, etc.;
- Tokenization [7]: breaking down sentences into words and punctuations (a.k.a. tokens);
- Part of Speech (POS) tagging [7]: labeling words with known lexical categories;
- Subject-Object phrases (entity) extraction using consecutive Noun phrase chunking [7]: select subset of tokens corresponding to individual noun phrases; and
- Term Frequency and Inverse Document Frequency (TF-IDF) [30] 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, having, hence*, etc. In the next section we study in detail, the heuristics used in FeatureX for relationships mining together with its formal definitions.

3.3 Heuristics

In this work, feature relationships have been classified into two categories based on the number of features they connect. 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. A human 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. 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 [6, 18, 38].

Table 4: Natural language POS Tag abbreviations.

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

3.3.1 *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 3.2. Let us look at an example taken from the excerpt presented in *Example 1.1*:

> *"The **publishing module** can 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 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 features, 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. 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 paper.

This rule, in conjunction with the detection of contiguous sequences of past and present participle, can help us 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.

Excludes and Require/Implies features. We further extend the one-to-one mapping rules by adding 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. For **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 our rules set.

Example 3.3. 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** are generally provided for all devices with **FM radio** capability"*

Here, the adverb *'generally'* indicates the optional nature of the relationship and the predeterminer *'all'* signifies an implication. This establishes a *requires* CTC between **headphones** and **FM radio**.

A well-defined set of rules can be defined which can be applied on a list of words $\mathfrak{I}$ 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:
R1 : $Loc(f_1) < Loc(MD) < Loc(f_2)$
R2 : $Loc(f_1) < Loc(adv) < Loc(f_2)$ **or** $Loc(f_1) < Loc(det) < Loc(f_2)$
R3 : $Loc(f_1) < Loc(VBN - VBG) < Loc(f_2)$
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 4 which occur exactly one time, and for any word x, *Loc(x)* is a word location/position function defined as:

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

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 $\mathfrak{R}$ 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 5 is used to decide the final relationship type of the selected pair of features. This can be defined as:

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

Table 5: Decision matrix.

	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**

3.3.2 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 3.4. Consider the following examples from two different products (SelfKey[2] and AuthenticID[3]) of blockchain-based user identity software applications:

SelfKey

> *"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 4) like **and** and **or** and prepositions and conjunctions **IN** (see Table 4) 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**)

[2]https://selfkey.org/wp-content/uploads/2017/11/selfkey-whitepaper-en.pdf
[3]https://tokens.authenticid.co/images/2017/08/AuthenticID-Smart-Identity-Token-Sale_Whitepaper_2.0_13-Sept-2017_FINAL.pdf

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**. Due to space limitations, only a part of the identity software FM is shown with the mentioned features and relationships in Figure 3. Note that by default an optional or mandatory relationship with a set of features is an **AND** connection in FeatureIDE tool [22].

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 $\mathbb{R} \in \{AND, OR, XOR\}$ and $(f_1/f_3) \xrightarrow{\mathbb{R}} f_2$ and $(f_1/f_3) \xrightarrow{\mathbb{R}} f_4$. 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 [28] library and choosing an appropriate feature term among the candidate parent features follow the same rules for feature name selection (Section 3.2).

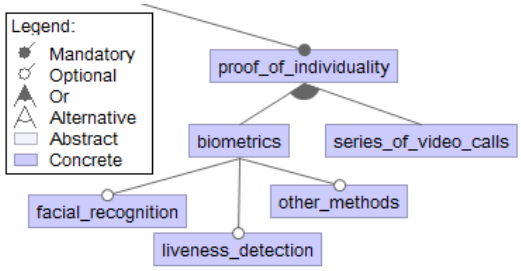

Figure 3: Partial view of Identity software FM (manually created in FeatureIDE [22]).

To summarize, the feature relationships extraction heuristics in **FeatureX** comprises of the steps shown in Algorithm 2.

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.

Input : A set of candidate feature terms *FT*, selected root feature *R* and the raw text content *T*
Output: Set of 4 text files with heuristics results
Function RelationshipMiner(*T*, *FT*, *R*):
  foreach (f,g) ∈ *FT* **do**
    ▷ Detect one-to-one relations
    pair ← SetDirection(f,g,T) ▷ e.g. f → g
    foreach rule ∈ [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, R);
    end
  end
  WriteToResultFile(results);
  return ▷ R1.txt,R2.txt,R3.txt,R4.txt

Algorithm 1: FeatureX relationships extraction 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);
▷ apply NLP techniques for lexical/syntactical analysis, e.g. POS tagging
P ← PreprocessText(T);
▷ find features by collecting Subject-Object phrases and apply chunk rules for multi-term features
FT ← IdentifyCandidateFeatureTerms(P);
▷ use frequency distribution to find the most relevant features
R ← IdentifyCandidateRootFeatures(P,FT);
▷ Calling the relationships heuristics algorithm
REL ← RelationshipMiner(P,FT, R);
▷ compute final classification using decision matrix
DR ← ApplyDecisionMatrix(REL);
▷ Constructing dot file representing the FM
result ← (FT, R, DR);
WriteToResultFile(result);
▷ FM.dot file, RootFeature.txt

Algorithm 2: FeatureX algorithm.

4 IMPLEMENTATION

The FeatureX framework has been implemented in Python and its source code is available in GitHub as the **FeatureX** library[4]. The implementation relies on the Natural Language Toolkit (NLTK) [7] 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 3.3. Algorithm 2 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 2 processes the raw varied format text data, to a 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 Algorithm 3. These are saved in separate text files with content similar to that shown in Figure 4. These files can then be processed automatically to generate the complete feature model using an existing FM modeling tool like FeatureIDE [22]. In this work, this last step (the generation of the FeatureIDE graphical model from FeatureX's output) is performed manually.

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*):
  foreach (file) ∈ *DF* **do**
    ▷ Detect one-to-many relations
    Relationships ← MatchPatternsForOneToMany(file) ;
    ▷ Append to file processing results
    results ← UpdateFileProcessingResults(Relationships);
  end
  WriteToResultFile(results);
  return ▷ Mandatory.txt,Optional.txt, And.txt,Or.txt,Alternative.txt,CTC.txt

Algorithm 3: FeatureX Dot file comparer algorithm.

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

Figure 4: Automatically generated feature terms and relationships - Case study : Cryptocurrency[5].

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

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

5 PRELIMINARY RESULTS

In order to evaluate our 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. Six different case studies have been performed for this evaluation. 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.
- (b) Cryptocurrency: 3 specification documents (whitepapers for zerocash, verge & cryptonote) part of the result was demonstrated in Figure 4.

CS (2) Product lines studied in previous works:
- (a) Car crash management system: 1 specification document [10] and reference FM used from [32].
- (b) Anti-virus software: 1 specification document with multiple product descriptions [13].
- (c) E-shop software: 1 specification document [17, 18].
- (d) Smart home software: 1 specification document [36].

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 produce 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.

In order to compare the results obtained by FeatureX to those from existing works [13, 17, 18, 36], 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 as compared to the models from the literature.

To this end, the domain engineers were provided with the FMs used the literature [13, 17, 18, 36] 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?

A sample output of the above analysis study[6] is shown in Table 6. 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.

5.1 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, **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.

5.2 Results of the evaluation

The papers [13, 17, 18, 36] 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.

We compare the outputs of FeatureX with the tools used for each case study in Tables 7 and 8. 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.

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

Table 6: Domain engineer analysis of FMs for Anti-virus product line (CS(2b)).

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.

Table 7: FeatureX performance.

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

	FeatureX								Other tools							
	Features				Relationships				Features				Relationships			
Case study	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).

- 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.

6 CONCLUSIONS AND FUTURE WORK

The proposed framework (FeatureX) for extracting FMs from natural language specifications of software product lines 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). Moreover, FeatureX is programmed using Python and the source is available publicly for future researchers to evaluate and extend it further.

As future work, we intend to formally analyze the correctness and completeness of the FM and automatically detect inconsistencies, thereby providing feedback to the domain engineer notifying of potential defects (such as dead features, duplicate features, constraint violations, etc.) in the specification.

ACKNOWLEDGMENTS

The authors would like to thank Joost Noppen for his assistance regarding the use of the ArborCraft tool.

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).

REFERENCES

[1] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. 2008. An Exploratory Study of Information Retrieval Techniques in Domain Analysis. In SPLC'08. 67–76. https://doi.org/10.1109/SPLC.2008.18

[2] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer. 2016. Automated Extraction and Clustering of Requirements Glossary Terms. IEEE Transactions on Software Engineering 99 (2016), 918–945. https://doi.org/10.1109/TSE.2016.2635134

[3] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer. 2016. Extracting Domain Models from Natural-language Requirements: Approach and Industrial Evaluation. In MODELS'16. ACM, 250–260. https://doi.org/10.1145/2976767.2976769

[4] N. Bakar. 2015. Latent Semantic Analysis and Particle Swarm Optimization for Requirements Reuse in Software Product Line : A Research Plan.. In SPLC'2013 Doctoral Symposium. 0–9.

[5] N. H. Bakar, Z. M. Kasirun, and N. Salleh. 2015. Feature Extraction Approaches from Natural Language Requirements for Reuse in Software Product Lines. J. Syst. Softw. 106, C (Aug. 2015), 132–149. https://doi.org/10.1016/j.jss.2015.05.006

[6] M. P. S. Bhatia, A. Kumar, and R. Beniwal. 2016. Ontology based framework for detecting ambiguities in software requirements specification. In INDIACom'16. 3572–3575.

[7] S. Bird, E. Klein, and E. Loper. 2009. Natural Language Processing with Python (1st ed.). O'Reilly Media, Inc.

[8] D. Bogdanova. 2011. Extraction of High-Level Semantically Rich Features from Natural Language Text. In ADBIS'11. 262–271.

[9] E. Boutkova and F. Houdek. 2011. Semi-automatic identification of features in requirement specifications. In RE'11. IEEE, 313–318. https://doi.org/10.1109/RE.2011.6051627

[10] A. Capozucca, B. H. C. Cheng, G. Georg, N. Guelfi, P. Istoan, G. Mussbacher, S. P. Mandalaparty, and A. Moreira. 2012. Requirements Definition Document for a Software Product Line of Car Crash Management Systems. Technical Report. U. of Nice Sophia Antipolis, I3S CNRS. https://doi.org/10.1007/978-3-642-16086-8

[11] K. Chen, W. Zhang, H. Zhao, and H. Mei. 2005. An approach to constructing feature models based on requirements clustering. In RE'05. IEEE, 31–40. https://doi.org/10.1109/RE.2005.9

[12] P. Clements and L. Northrop. 2001. Software Product Lines: Practices and Patterns. Addison-Wesley Professional.

[13] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans. 2013. Feature Model Extraction from Large Collections of Informal Product Descriptions. In ESEC/FSE'2013. ACM, 290–300. https://doi.org/10.1145/2491411.2491455

[14] T. De Smedt and W. Daelemans. 2012. Pattern for Python. J. Mach. Learn. Res. 13 (June 2012), 2063–2067. http://dl.acm.org/citation.cfm?id=2188385.2343710

[15] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli. 2011. On-demand Feature Recommendations Derived from Mining Public Product Descriptions. In ICSE'11. ACM, 181–190. https://doi.org/10.1145/1985793.1985819

[16] A. Ferrari, G. O. Spagnolo, and F. Dell'Orletta. 2013. Mining commonalities and variabilities from natural language documents. In SPLC '13. 116. https://doi.org/10.1145/2491627.2491634

[17] M. Hamza and R. J. Walker. 2015. Recommending Features and Feature Relationships from Requirements Documents for Software Product Lines. In RAISE'15. IEEE Press, 25–31.

[18] N. Itzik and I. Reinhartz-Berger. 2014. Generating Feature Models from Requirements: Structural vs. Functional Perspectives. In SPLC'14 Workshops. ACM, 44–51. https://doi.org/10.1145/2647908.2655966

[19] N. Itzik and I. Reinhartz-berger. 2014. SOVA-A tool for Semantic and Ontological Variability Analysis. In CAISE-Forum'14 (CEUR), Vol. 1164. 177–184.

[20] T. Jo. 2003. Neural Based Approach to Keyword Extraction from Documents. In ICCSA'03. 456—-461.

[21] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021. Software Engineering Institute, Carnegie Mellon University.

[22] C. Kästner, T. Thüm, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, and S. Apel. 2009. FeatureIDE: A Tool Framework for Feature-oriented Software Development. In ICSE'09. IEEE CS, 611–614. https://doi.org/10.1109/ICSE.2009.5070568

[23] C. Kelly, B. Devereux, and A. Korhonen. 2014. Automatic extraction of property norm-like data from large text corpora. Cognitive Science 38 (2014), 638–682. https://doi.org/10.1111/cogs.12091

[24] A. Khtira, A. Benlarabi, and B. E. Asri. 2015. A Tool Support for Automatic Detection of Duplicate Features during Software Product Lines Evolution. IJCSI International Journal of Computer Science 12, 4 (2015), 1–10.

[25] N. Loughran, A. Sampaio, and A. Rashid. 2006. From requirements documents to feature models for aspect oriented product line implementation. In MDD for Product Lines @ MODELS'06 (LNCS), Vol. 3844. 262–271. https://doi.org/10.1007/11663430_27

[26] S. G. MacDonell, K. Min, and A. M. Connor. 2005. Autonomous requirements specification processing using Natural Language Processing. In IASSE'05. ISCA, 266–270.

[27] M. Mefteh, N. Bouassida, and H. Ben-Abdallah. 2016. Mining Feature Models from Functional Requirements. Comput. J. 59, 12 (2016), 1784–1804. https://doi.org/10.1093/comjnl/bxw027

[28] George A. Miller. 1995. WordNet: A Lexical Database for English. Commun. ACM 38, 11 (Nov. 1995), 39–41. https://doi.org/10.1145/219717.219748

[29] S. Ben Nasr, G. Bécan, M. Acher, J. Bosco Ferreira Filho, N. Sannier, B. Baudry, and J.-M. Davril. 2017. Automated extraction of product comparison matrices from informal product descriptions. J. Syst. Softw. 124 (2017), 82 – 103. https://doi.org/10.1016/j.jss.2016.11.018

[30] J. H. Paik. 2013. A Novel TF-IDF Weighting Scheme for Effective Ranking. In SIGIR'13. ACM, 343–352. https://doi.org/10.1145/2484028.2484070

[31] T. Quirchmayr, B. Paech, R. Kohl, and H. Karey. 2017. Semi-automatic Software Feature-Relevant Information Extraction from Natural Language User Manuals. In Requirements Engineering: Foundation for Software Quality. Springer, 255–272.

[32] R. Salay, M. Famelis, J. Rubin, A. Di Sandro, and M. Chechik. 2014. Lifting Model Transformations to Product Lines. In ICSE'14. ACM, 117–128. https://doi.org/10.1145/2568225.2568267

[33] A. Sampaio, A. Rashid, R. Chitchyan, and P. Rayson. 2007. EA-Miner: Towards Automation in Aspect-Oriented Requirements Engineering. LNCS, Vol. 4620. Springer, 4–39. https://doi.org/10.1007/978-3-540-75162-5_2

[34] C. Vicient, D. Sánchez, and A. Moreno. 2013. An Automatic Approach for Ontology-based Feature Extraction from Heterogeneous Textual Resources. Eng. Appl. Artif. Intell. 26, 3 (March 2013), 1092–1106. https://doi.org/10.1016/j.engappai.2012.08.002

[35] J. Wang, H. Peng, and J.-S. Hu. 2005. Automatic keyphrases extraction from document using backpropagation. In ICMLC'05, Vol. 6. 633–641. https://doi.org/10.1109/ICMLC.2005.1527596

[36] N. Weston, R. Chitchyan, and A. Rashid. 2009. A Framework for Constructing Semantically Composable Feature Models from Natural Language Requirements. In SPLC'09. Carnegie Mellon University, 211–220.

[37] L. Yi, W. Zhang, H. Zhao, Z. Jin, and H. Mei. 2012. Mining binary constraints in the construction of feature models. In RE'12. 141–150. https://doi.org/10.1109/RE.2012.6345798

[38] J. Zhang, B. Ahmad, D. Vera, and R. Harrison. 2016. Ontology based semantic-predictive model for reconfigurable automation systems. In INDIN'16. 1094–1099. https://doi.org/10.1109/INDIN.2016.7819328