Feature Model Extraction from Large Collections of Informal Product Descriptions

Jean-Marc Davril, Edouard Delfosse, Negar Hariri, Mathieu Acher, Jane Clelang-Huang, Patrick Heymans

Feature Model Extraction from Large Collections of Informal Product Descriptions

Jean-Marc Davril[1], Edouard Delfosse[1], Negar Hariri[2]
Mathieu Acher[3], Jane Cleland-Huang[2], Patrick Heymans[1]
[1]Faculty of Computer Science,University of Namur, Belgium
[2]School of Computing, DePaul University, Chicago, USA
[3]INRIA / Irisa, Unviersity of Rennes 1, France
{jmdavril, delfosse}@student.fundp.ac.be; nhariri@cs.depaul.edu;
jhuang@cs.depaul.edu; macher@irisa.fr; patrick.heymans@fundp.ac.be

ABSTRACT

Feature Models (FMs) are used extensively in software product line engineering to help generate and validate individual product configurations and to provide support for domain analysis. As FM construction can be tedious and time-consuming, researchers have previously developed techniques for extracting FMs from sets of formally specified individual configurations, or from software requirements specifications for families of existing products. However, such artifacts are often not available. In this paper we present a novel, automated approach for constructing FMs from publicly available product descriptions found in online product repositories and marketing websites such as SoftPedia and CNET. While each individual product description provides only a partial view of features in the domain, a large set of descriptions can provide fairly comprehensive coverage. Our approach utilizes hundreds of partial product descriptions to construct an FM and is described and evaluated against antivirus product descriptions mined from SoftPedia.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/ Specifications—*Methodologies*

General Terms

Algorithms, Management

Keywords

Product Lines, Feature Models, Domain Analysis

1. INTRODUCTION

The use of *Software Product Line (SPL)* engineering is becoming increasingly prevalent as a means to deliver high quality products with a shorter time-to-market at reduced costs [38]. According to the Software Engineering Institute, SPLs *"epitomize strategic, planned reuse, and represent a way of doing business that results in order-of-magnitude improvements in cost, time-to-market, and productivity''* [26, 32]. An SPL is a set of software-intensive systems that share a common, managed set of features developed from a common set of core assets in a prescribed way [16, 29]. SPL engineering aims to support the structured reuse of a wide range of software artifacts including requirements, design, code, and test cases [8, 12, 21, 29].

Feature Models (FMs) are one of the most popular formalisms for modeling and reasoning about commonality and variability of an SPL [17]. Depending on the level of abstraction and artifacts described, features may refer to a prominent or distinctive user-visible characteristic of a product or to an increment in a software code base [8, 7, 15]. A recent survey of variability modeling showed that FMs are by far the most frequently reported notation in industry [10]. Several academic or industrial tools have been developed to specify them graphically or textually and automate their analysis, configuration or transformation [30, 11, 9, 3, 37, 14]. FMs hierarchically organize a potentially large number of concepts (features) into multiple levels of increasing detail, typically using a tree. Variability is expressed in terms of mandatory, optional and exclusive features as well as logical constraints over the features. The conjunction of constraints expressed in an FM defines the set of all legal *configurations* of an SPL [19]. This is illustrated in Table 1 which lists all valid configurations for the FM shown in Figure 1.

SPL engineering includes two phases of *domain engineering* and *application engineering* [38]. Domain engineering involves analyzing a specific domain, discovering commonalities and variabilities, and then constructing core assets which will be used across the entire SPL [34]. In contrast, application engineering is concerned with building a specific product based upon the core assets of the product line. In this paper we focus on the process of constructing an FM as part of the domain engineering process. This process can be exceedingly time-consuming, yet can provide support during the domain engineering phase to help configure products, design a new family of products, expand an existing product line, or simply provide inputs into the requirements elicitation phase of a single application process.

Given the arduous nature of manually constructing FMs, the SPL research community has shown significant interest in the ability to automatically generate FMs from exist-

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia
ACM 978-1-4503-2237-9/13/08
http://dx.doi.org/10.1145/2491411.2491455

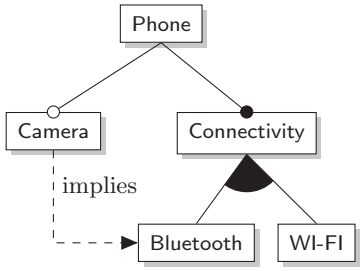


Figure 1: FM example

Camera	Connectivity	Bluetooth	WI-FI
X	X	X	
X	X	X	X
	X	X	
	X		X
	X	X	X

Table 1: Configurations for the FM above

ing data. There are several examples of prior work in the area. Czarnecki et al. [18] introduced probabilistic FMs and provided an extraction procedure that mined propositional formulas and soft constraints from a set of multiple configurations. However, their approach assumed that products were already formally described as sets of features. Similarly Acher et al. [2] described a way of extracting FMs from product descriptions but their approach assumed the availability of formal and complete descriptions of products as configurations of features. Chen et al. and Weston et al. described techniques for extracting an FM from informal specifications ([13], [39]). This approach is particularly useful in cases where an organization has an existing set of individual products and wishes to move towards an SPL approach. However, it also has certain limitations, because the constructed FM is constrained to the set of features described in the SRS for the existing set of products.

In this paper we focus on the scenario in which an organization has no existing product descriptions and must rely upon publicly available data from websites such as SoftPedia[1], CNET[2], and MajorGeeks[3], which provide feature lists for hundreds of thousands of products [22]. However, such product descriptions are generally incomplete, and features are described informally using natural language.

The task of extracting FMs from informal data sources involves mining feature descriptions from sets of informal product descriptions, naming the features in a way that is understandable to human users, and then discovering relationships between features in order to organize them hierarchically into a comprehensive model. In this paper, we base the feature extraction technique on our previously developed approach [22], and then introduce a novel technique for generating an FM from the set of extracted features. We describe and validate our approach using product descriptions for antivirus products mined from SoftPedia.

The remainder of the paper is structured as follows. Section 2 provides a general overview of our approach, while Section 3 describes feature modeling in general. Sections 4 and 5 describe the two main phases of our approach, namely feature mining, and subsequent construction of the FM. Section 6 describes results from the evaluation process, while Section 7 discusses possible threats to validity. Finally Section 8 describes related work, and Section 9 provides an analysis of our results and proposes ideas for future work.

[1]`http://www.softpedia.com/`
[2]`http://download.cnet.com/windows/`
[3]`http://majorgeeks.com/`

2. OVERVIEW

Our approach is summarized in Figure 2 and consists of two primary phases. In the first phase, software features are discovered from a set of informal product descriptions, while in the second phase the FM is constructed.

2.1 Mining Features

In step ❶, product specifications are mined from online software repositories. We used the *Screen-scraper* utility to scrape raw product descriptions for 165 antivirus products from *Softpedia*. In step ❷, these product specifications are processed in order to identify a set of features and to generate a product-by-feature matrix $P \times F$ in which the rows of the matrix correspond to products and the columns correspond to features. The $(i, j)^{th}$ entry of this matrix can take a value of 0 or 1, to represent whether the i^{th} product is known to include the j^{th} feature or not. Given the informal and incomplete nature of the product descriptions,we cannot differentiate between the case in which a feature is not included in the product, versus the case in which the feature is present in the product but not listed in the description. One of the challenges of our approach is therefore to construct an FM from relatively large quantities of incomplete information. In step ❸, meaningful names are selected for the mined features.

2.2 Building Feature Model

In the second phase, the product-by-feature matrix is used to infer feature associations and to create the FM. This process requires creating some intermediate structures. In step ❹ a set of association rules are mined for the features. In step ❺ these association rules are used to generate an *implication graph* (IG) which captures binary configuration constraints between features. In this context, the IG is a directed graph in which nodes represent features, and an edge exists between two features f_1 and f_2 if the presence of f_1 in a configuration implies the presence of f_2. In step ❻, the tree hierarchy and then the *Feature Diagram* (FD) are generated given the IG and the content of the features. Finally, step ❼ identifies *cross-tree constraints* (CTCs) and *OR-groups* of features. These two elements, in addition to the FD generated in step ❻, form the FM.

3. BACKGROUND

Before describing our approach in greater detail, we illustrate different types of syntactical constructions used in FMs and we present the underlying formalism. An FM is a hierarchical organization of features that aims to represent the constraints under which features occur together in products configurations. Returning to the example of Figure 1, the FM is depicted using the standard visual notation and aims to represent a family of mobile phones. Each combination of features (a.k.a. configuration) that does not violate the constraints of the FM corresponds to a specific product, for instance, a mobile phone that exhibits the features

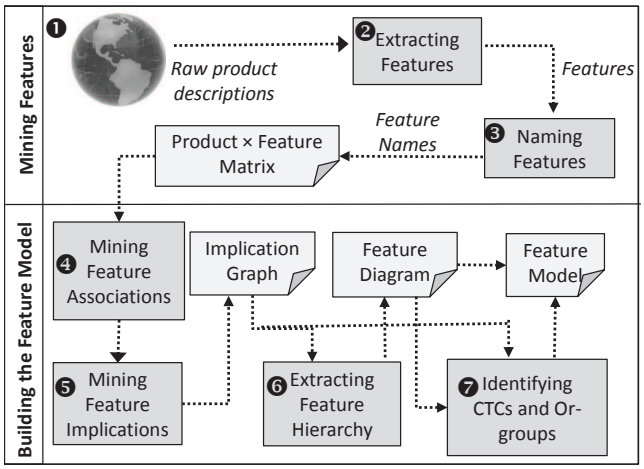
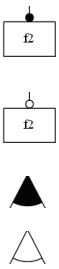


Figure 2: Fully automated, two-step process: mining features and building FM

Connectivity, Bluetooth and Camera (see also Table 1) is a valid product. Figure 1 shows some of the constructions: every mobile phone has a Connectivity feature, the presence of a Camera is optional, a mobile phone supports at least Bluetooth or WI-FI (and possibly both of them). Finally, the presence of a Camera implies the presence of Bluetooth. Importantly, the hierarchy imposes some constraints: the presence of a *child* feature in a configuration logically implies the presence of its *parent* (e.g., the selection of Bluetooth implies the selection of Connectivity). The hierarchy also helps to conceptually organize the features into different levels of increasing detail. Specifically, relationships between a parent and its child features can be as follows:

1. **Mandatory** The presence of the parent feature in a configuration implies the presence of the child feature.
2. **Optional** The child feature may or may not be present in a configuration which contains its parent feature.
3. **OR** If the parent is selected, then at least one of its child features must be selected.
4. **XOR** If the parent is selected, exactly one of the child features should be selected.



Formally, an FM is composed of an FD and a set of *Cross-Tree Constraints (CTCs)* (see definition hereafter). The FD of an FM is a tree representing the hierarchical structure among features as well as variability information exposed above. CTCs are additional configuration constraints that cut through the tree structure of the FD. In line with [1, 6], we define these concepts as follows:

Definition 1. An FD is a tuple FD (F, E, E_m, G_o, G_x) where (i) F is a finite set of features and $E \subseteq F \times F$ represents the set of edges from child features to their parents so that (F, E) forms a tree (ii) $E_m \subseteq E$ is a set of mandatory edges; (iii) G_o and G_x represent the sets of OR-groups and XOR-groups respectively. These are sets of non-overlapping subsets of E so that all the edges from the same subset of E share a common parent feature.

There exists two kinds of CTCs. (i) *Implies* constraints are binary implications between two features ($A \Rightarrow B$). Figure 1 gives an example of an implies CTC between the features Camera and Bluetooth. (ii) *Excludes* constraints express that the presence of a feature in a configuration implies the absence of another feature ($A \Rightarrow \neg B$).

Table 2: Features from an Antivirus Product

Safepay	Keeps hackers at bay by automatically opening online banking pages in a separate, secure browser.
Dashboard	See all the status and licensing information about your software and services in your own, MyBitdefender dashboard. Now accessible from anywhere, anytime, from any Internet-connected device.
Security Widget	Enables you to keep track of all of your security related tasks, plus lets you quickly and easily drag-and-drop files for quick scanning for viruses right from your desktop!
Parental Control	Blocks inappropriate content, restricts Web access between certain hours, and helps you remotely monitor your children's online activity even on Facebook!
USB Immunizer	Immunizes any Flash Drive from viruses, when they are connected to your computer, so that you never worry again about USBs infecting you.
Active Virus Control	Proactive, dynamic detection technology which monitors processes behavior in real-time, as they are running, and tags suspicious activities.

4. FEATURE MINING

To mine features from a database of software specifications we adopted an approach used in our previous work [22] in which we constructed a feature recommender system based on features described using natural language and mined from SoftPedia. However, our prior work made no attempt to construct a more formal FM, which is the focus of this current paper. Furthermore, we replace the previous feature naming algorithm to generate names more suited to appearing in an FM. In this section we summarize the process used to mine features from product descriptions.

4.1 Mining Raw Feature Descriptors

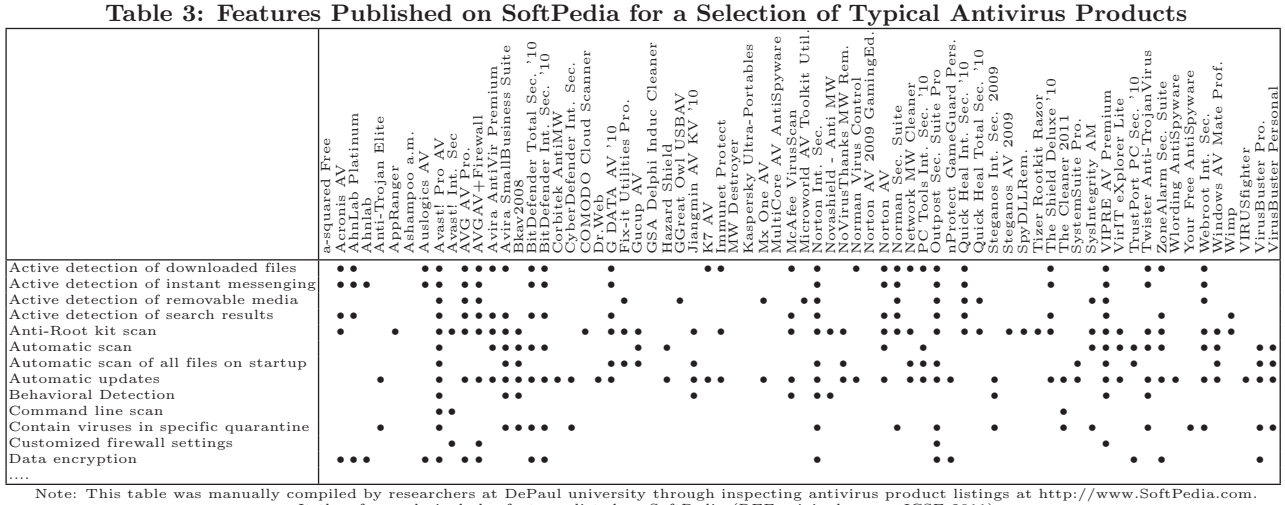
Raw feature descriptors are retrieved from a repository of software products. For purposes of this paper the *Screen-scraper* utility was used to scrape the raw software descriptions of 165 antivirus products from *softpedia.com*. Softpedia products tend to include a general product description followed by a list of bulleted items describing specific features. The set of feature descriptors for each product was extracted by parsing the software description into sentences and adding the items in the bulleted list of features. There were a total of 4,396 descriptors mined from the antivirus category. Table 2 provides examples from one particular product. It is important to note that similar features are described using very different terms, and furthermore that the list of descriptors for each product is incomplete.

4.2 Preprocessing

In preparation for feature extraction, all of the feature descriptors were preprocessed by stemming each word to its morphological root, removing stop words (i.e., extremely common words such as *this* and *provides*). The remaining descriptor was then modeled as a vector of terms, in which each dimension of the vector corresponds to one of the terms in the vocabulary. We use the *tf-idf* (term frequency - inverse document frequency) scheme to assign a weighting to term t in descriptor d as follows:

$$w_{t,d} = tf_{t,d} * log(\frac{N}{df_t}) \quad (1)$$

Table 3: Features Published on SoftPedia for a Selection of Typical Antivirus Products



Note: This table was manually compiled by researchers at DePaul university through inspecting antivirus product listings at http://www.SoftPedia.com. It therefore only includes features listed on SoftPedia (REF original source ICSE 2011)

where, $tf_{t,d}$ represents the number of times that term t occurs in d, df_t is the number of descriptors that contain term t, and N is the total number of descriptors.

4.3 Feature Formation

To identify coherent features, we first determined the similarity of each pair of descriptors through computing the inner product of their corresponding vectors. The resulting similarity scores were then used to cluster descriptors into similar groups in which each group represents a feature. We adopted a two-stage *Spherical k-Means (SPK-Means)* clustering algorithm [20] which has previously been shown to perform well for clustering features [22]. The first stage is similar to the K-Means clustering algorithm which starts by randomly selecting k centroids, where k is the desired number of clusters. In each iteration of K-Means clustering, each of the instances is assigned to the nearest cluster and at the end of the iteration each centroid is recomputed based upon its assigned set of descriptors. This iterative process continues until the centroids stabilize. The second stage of the *SPK-Means* clustering involves the incremental optimization of the objective function through an iterative process. At each iteration, one of the instances is randomly selected and moved to another cluster to maximize the gain of the objective function. The centroids are then updated and this process continues until convergence.

4.4 Naming the Feature

For purposes of constructing an FM, the clusters represent features which will be presented to human users and must therefore be assigned meaningful names. Based on informal experimentation we developed a cluster-naming process that involved selecting the most frequently occurring phrase from among all of the feature descriptors in the cluster. This approach is similar to the method presented in [25] for summarizing customer reviews. To identify the most frequently occurring phrase we use the Stanford Part-of-Speech (POS) tagger[4] to tag each term in the descriptors with its POS. The descriptors are then pruned to retain only nouns, adjectives, and verbs as the other terms were found not to add useful information for describing a feature.

[4]http://nlp.stanford.edu/software/tagger.shtml

Frequent itemsets are then discovered for each of the clusters. In this context, frequent itemsets are sets of terms that frequently co-occur together in the descriptors assigned to the same cluster. More formally, the support of an itemset I is the number of descriptors in the cluster that contain all the terms in I. Given a pre-determined *itemset support threshold*, s, I is considered frequent if its support is equal or larger than s. We set this threshold to be 25% of the total number of descriptors in each cluster.

Various algorithms exist for mining frequent itemsets including the Apriori [4] and FPGrowth [23] algorithms. We chose to use FPGrowth as it is shown to be memory-efficient and hence suitable for the size of our data set.

To select the name for a cluster, all of its frequent itemsets of maximum size, FIS_{max} are selected. Next, all feature descriptors in the cluster are examined. In each feature descriptor, the shortest sequence of terms which contains all the words in FIS_{max} is selected as a candidate name. For example, let $FIS_{max} = \{prevents, intrusion, hacker\}$. For a given feature descriptor such as: *prevents possible intrusions or attacks by hackers trying to enter your computer*, the selected candidate name is *prevents possible intrusions or attacks by hackers*. Finally, the shortest candidate name is selected, as this reduces the verbosity of the feature name.

5. BUILDING THE FEATURE MODEL

This section describes the algorithm used to automatically construct the FM (see Algorithm 1). It takes as input the product-by-feature matrix created during the feature mining phase and the set of feature descriptors assigned to each feature cluster. The complete process was derived incrementally through a series of informal design and evaluation iterations. We present only our final algorithm here, although two variants are compared in Section 6 of this paper.

The logical constraints have largely influenced our adopted process. We first utilize binary implication rules between features to construct a so-called *implication graph* (more details are given hereafter), then add disjunctive rules to complete the implication graph, and finally perform additional processing to transform it into a complete FM (including the synthesis of OR-groups and CTCs). The main steps of the algorithm are now briefly explained.

Mining feature associations The first step of the pro-

cess involves mining association rules between features (lines 2-5 in Algorithm 1). Two types of association rules are mined: *binary implications* and *disjunctive rules*.

Implication graph The binary implications are used to build an implication graph (line 7), defined as a directed graph in which nodes are features and edges represent logical implications between features. Because the implication graph is a directed graph, it does not present the look-and-feel of an FM in which features are grouped and arranged in a hierarchical fashion. In fact every possible hierarchy represents a spanning tree of the implication graph.

Extraction of feature hierarchy The next step is to find subgroups of features. To accomplish this task we utilized the *SPK-Means* clustering algorithm to group features into N clusters such that features in the same cluster are similar in textual content (line 9). The implication graph is then split into N subgraphs, representing the N identified clusters (line 12). This is accomplished through removing the edges that connect features across different clusters and adding them to a set of *CTCs*. We then utilized the *Agape*[5] library of graph algorithms to look for the strongly connected components (SCCs) in each of the subgraphs using Tarjan's algorithm [35] (line 13). Next, FDs are built for each of the subgraphs (line 14). The final FD is then built by merging these individual FDs. Associations between the different FDs are identified through mining an additional set of association rules at the cluster level (line 16). We substitute each feature by the cluster it belongs to in the product-by-feature-matrix before re-executing the association rule mining algorithm. These associations are then used to aggregate the FMs (lines 17-20).

Recovery of CTCs and OR-groups. In the final step, OR-groups and CTCs are recovered (line 22).

These steps are explained in greater detail below.

Algorithm 1 Feature Model Extraction

```
1: ▶ Association rules mining
2: function AssocRules(P, MIS)
3:     F ← CFP − Growth(P, MIS)   ▷ Frequent Itemsets
4:     A ← Agrawal(F, MinSup)
5:     return A

6: ▶ Implication graph
7: G(V, E) ← IG(AssocRules(Configurations, MIS))

8: ▶ Clustering the features
9: C ← SPK − Means(Features)

10: ▶ Building FDs for the clusters
11: for i ← 1 to n do                ▷ Number of clusters
12:     G_i ← SubGraph(G, C_i)       ▷ Cluster i
13:     SCC_i ← StronglyConnectedComponents(G_i)
14:     FD_i ← FeatureDiagram(G_i, SCC_i)

15: ▶ Aggregating the FDs
16: A ← AssocRules(ConfigsClusters, MISClusters)
17: G ← merge({FD_1, ..., FD_n}, A)
18: SCC ← StronglyConnectedComponents(G)
19: FD ← FeatureDiagram(G, SCC)
20: FD ← PrimeImplicates(FD)

21: ▶ Recovery of CTCs and OR-groups
22: CTC ← G − MG                     ▷ Cross-Tree Constraints
```

5.1 Mining Feature Associations

There are various kinds of associations between software features in a domain. Having a dataset of valid configurations, association rule mining methods can be exploited to identify these associations. These algorithms were originally proposed for market basket analysis in order to identify the relations between shopping items. For example, a rule in the form of $flour \Rightarrow egg$ indicates that customers who buy flour are likely to buy eggs as well. The rules are discovered based on a database of customers' shopping transactions.

To discover the association rules for features, the product-by-feature matrix, $P \times F$, is used as the training data to discover the relationships between the features. Each row of this matrix corresponds to a product, and therefore represents a potentially incomplete, yet otherwise valid configuration of features. Mining association rules is a two-step process. The first step is to discover the *frequent itemsets* (as explained in section 4.4), while in the second step, association rules are generated for the set of frequent itemsets.

Given an association rule in the form of $A \Rightarrow B$, A and B are both frequent itemsets and $A \cap B = \emptyset$. The *support* of this rule indicates the proportion of transactions that contain both itemsets A and B:

$$support(A \Rightarrow B) = \frac{|S(A \cup B)|}{|S|} \quad (2)$$

where S is the multiset of all transactions and $S(A \cup B)$ is the multiset of transactions that contain both A and B.

Given a predefined threshold value, σ, rules which have support $\geq \sigma$ are accepted.

The *confidence* of this rule indicates the proportion of transactions that contain itemset B among the transactions that contain the itemset A.

$$confidence(A \Rightarrow B) = \frac{support(A \cup B)}{support(A)} \quad (3)$$

The FM is constructed based on binary implications and disjunctive rules.

5.1.1 Binary Implications

A **binary implication** $f_1 \rightarrow f_2$, expresses an implication between two single literals, and is discovered through mining frequent itemsets. Standard approaches for mining frequent itemsets utilize a single minimum support threshold; however this can be problematic. Setting the threshold too low produces *false positives*, i.e. frequent itemsets that are not representative of actual co-occurrence patterns of features. On the other hand, setting the threshold too high, results in *false negatives*, i.e. patterns of feature co-occurrence which are not captured as frequent itemsets.

This situation is particularly common when the distribution of features over the dataset is not normal and some of the features occur frequently while others appear rarely in transactions. To deal with this problem, we use the CFP-growth algorithm ([27]) which allows multiple minimum support values to be used to mine frequent items sets. We tweaked the different minimum support values of the features until they all appeared in the association rules.

[5] `https://traclifo.univ-orleans.fr/Agape/`

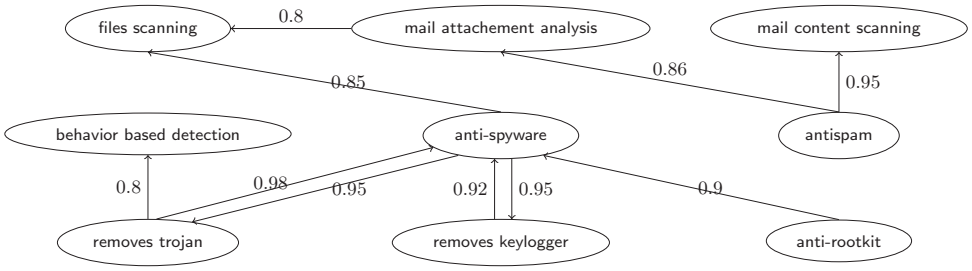


Figure 3: Implication Graph

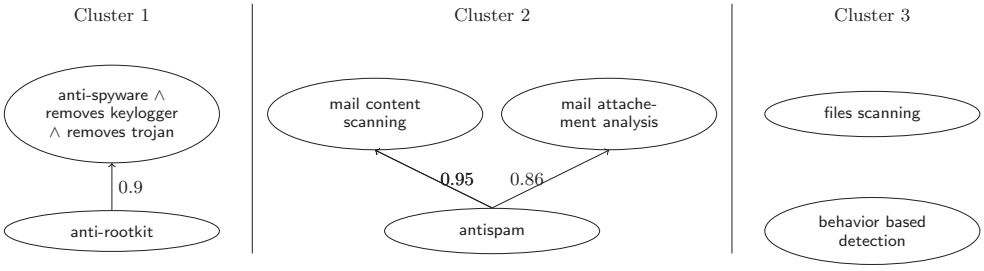


Figure 4: Implication Graph divided in subgraphs after clustering

Agrawal's Apriori algorithm [4] was then applied to discover the association rules in order to discover binary implications.

5.1.2 Disjunctive Rules

A **disjunctive rule** $f_1 \rightarrow f_2 \vee f_3 \vee ... \vee f_n$, represents an implication from a single literal to a *clause* (i.e., a disjunction of literals). The disjunctive rules are found by identifying *prime implicates* among features. A clause C is an *implicate* of the logical expression e, iff C is not a tautology and $e \Rightarrow C$ is a tautology. An implicate C of expression e is said to be *prime* iff there is no clause C' obtained by removing any literal from C such that C' is an implicate of e. By observing whether disjunctions of features are prime implicates, we identify OR-groups of minimal size among features. Disjunctive rules for which the features in the prime implicate share a common parent in the hierarchy are represented as OR-groups. In case of an incomplete dataset, a threshold can be tuned so that the disjunctive rules involved in the computation of the prime implicates are the rules with a support higher than this given threshold.

5.1.3 Exclusion Clauses

While FMs generally include exclusion clauses in the form $f_1 \rightarrow \neg f_2$, the nature of the informal product descriptions used in our approach makes this infeasible. This is because, as previously explained, the absence of a feature in the product-by-feature matrix does not differentiate between the case that the feature is not present versus the case in which the feature is not described in the product description. Therefore, our procedure does not include the mining of exclusion clauses.

5.2 Mining the Implication Graph

The set of binary implications can be represented as an *Implication Graph* in which nodes represent features and a directed edge connects f_1 to f_2 for each implication $f_1 \rightarrow f_2$. Figure 3 shows an example of an implication graph which was automatically generated for a subset of features for the antivirus category of products in our dataset. However, in this example, feature names were assigned manually. Later, we provide examples of automated feature naming.

Association rules are a form of partial implication between features. Deterministic association rules are implications with 100% confidence; while non-deterministic rules have confidence of less than 100%. This is reflected in the weights assigned to the directed edges connecting any two given features f_1 and f_2 in the implication graph. Each weight represents the confidence of the rule $f_1 \rightarrow f_2$.

Given the incomplete product data mined from Softpedia, and the sparsity of the resulting product-by-feature matrix, it is necessary to consider association rules with confidence lower than 100%. Our product-by-feature matrix contains 165 products for 80 features with an average of 6.5 features per product. Certain features appear in as few as five products, and no pair of features occur together with a confidence of 100%. Therefore in order to build a dependency structure, it was necessary to reduce the confidence threshold to a value below 100%.

AND-groups. In the implication graph, there exists sets of features that are mutually implied and have a common parent. They form a so-called AND-group. All these groups can be determined by identifying *Strongly Connected Components* (SCC) in the implication graph. A SCC in a graph G is a subgraph of G in which a path exists in both directions between each pair of nodes in the subgraph. AND-groups can be synthesized as *mandatory* features in an FD whenever the parent-child relationships between them have been determined (see next section).

We reduce the implication graph by merging all the nodes in the same SCC. These merged nodes represent conjunctions of features and form AND-groups. For example in Figure 3, *anti-spyware*, *removes keylogger* and *removes trojan* form a SCC and can be merged together. The AND-groups are then included in the new resulting implication graph (see Figure 4).

5.3 Extraction of Feature Hierarchy

Given an implication graph, there are many potential feature hierarchies (and thus FDs) which could be extracted. The challenge is to identify the diagram that will ultimately produce the "best" hierarchy. It is well known that different groups of people will organize features in different ways and therefore produce a variety of hierarchies from the same set of features. The problem is not only identifying a single correct hierarchy (i.e., a spanning tree), but also finding a good one which organizes features in a meaningful way. One viable approach is to treat the extraction process as a spanning tree optimization problem.

Our approach selects the final feature hierarchy using a combination of text-mining and co-occurrences between features. First, the SPK-Means clustering algorithm clusters features into N clusters according to terms used in their associated feature descriptors. We used the same approach as in our prior work [22] to set the number of clusters (N). This clustering step is conducted because in many cases, human-created FMs include subtrees of related features. For each cluster, we then extract the subgraph of the implication graph that contains all the features belonging to this cluster, and only those. Therefore we can reduce the scope of possible FDs to a selection of N FDs from N subgraphs of the implication graph. For example, the implication graph shown in Figure 4 shows the division of the implication graph in Figure 3 into three subgraphs. Each one of the three FDs selections is realized by applying Edmonds' optimum branching algorithm ([36]) to the subgraph, which can be seen as a *Minimum Spanning Tree* algorithm for a directed graph. In our case the weights of the edges are conditional probabilities between feature occurrences and the algorithm is used to compute maximal trees.

For each subgraph, an artificial root node is added and elements under the root form an OR-group. Each node in the subgraphs represent features, except for these artificial roots which are abstract nodes that are similar to those created by experts when they write FMs manually. Graphically, we represent these artificial roots as boxes containing the themes of the clusters (i.e., the words that were attributed the greater TF-IDF weights in the vector space representation of the clusters). Figure 5 shows three examples of these automatically generated abstract nodes : $scan, detection, files$, $spyware, protection$ and $mail, spam$. Finally, all the resulting FDs need to be connected together into a single system-wide FD. This is accomplished by mining association rules between clusters from the product-by-feature matrix. This additional association rule mining phase requires the features in the dataset to be replaced by the clusters they belong to. The mined association rules indicate how to connect the artificial roots of the FD together with directed edges. Edmonds' algorithm is applied again to the overall graph in order to reduce it to a tree. In other words, connecting different FDs can be seen as the construction of a tree structure between the abstraction nodes: first the tree structure is formed between the abstract nodes by mining association rules and by applying Edmonds' algorithm. Next, for each abstract node, the FD containing its successors' is added to the tree.

Finally, nodes that were previously merged as conjunctions of features (thus forming AND-groups) can now be unfolded. To accomplish this, one of the conjunctions must be chosen as the parent of the group. Given an AND-group, we define the parent feature as the feature that maximizes the minimal co-occurrence with other features of the group in the dataset. Formally, the parent feature f maximizes the number of configurations involving f and any other feature of the AND-groups. For example as shown in Figures 4 and 5, *anti-spyware* is selected as parent of the group.

Our clustering approach aims to ensure that features covering similar aspects of the domain are close to each other in the FD. Furthermore, our initial informal observations of several variants of these algorithms suggested that the automatic creation of abstraction nodes reduces the cognitive complexity of the final model.

5.4 Recovery of CTCs and OR-groups

At this point, a tree structure showing the relationships between features and abstraction nodes has been constructed. In addition, mandatory relationships have been synthesized and an FD without feature groups has been created. Some of the disjunctive rules that have previously been mined can be represented as OR-groups in the FD. These are the disjunctive rules for which the antecedent feature is the parent of all its consequent features in the FD : $f_1 \rightarrow f_2 \vee f_3 \vee ... \vee f_n$ where f_1 is the parent of f_2 , f_3 , ... , f_n. Unfortunately, for a large dataset, computing prime implicates can become infeasible in practice. However, if we are primarily interested in finding OR-groups in the FD instead of looking for all disjunctive rules from the dataset, then performance can be increased by reducing the task to consider only features sharing a common parent in the hierarchy [6]. Algorithm 1 shows the computation of prime implicates after elicitation of the feature hierarchy, at line 20. When the scope of OR-groups mining is reduced, it becomes feasible to compute prime implicates in a brute-force manner - i.e., by counting co-occurrences between the parent feature and disjunctions of its children in the dataset in order to find OR-groups of minimal size. Furthermore, once the implication graphs and the FD are known, the cross-tree constraints can be easily deduced. The FD coupled to the conjunction of cross-tree constraints form a FM, as defined in Section 3.

6. EVALUATION

To evaluate the quality of the generated FMs we first explored the possibility of creating a "golden answer set" and then comparing our FM against this standard. For this approach to be viable it would be necessary for multiple users to manually construct similar FMs given an initial group of features. As an initial feasibility study we asked two distinct pairs of users to construct an FM for the *antivirus* domain. Each pair was given a brief introduction to feature modeling, so that they had a solid understanding of the expected hierarchy, optional and mandatory relationships, as well as OR-groups. In separate sessions, each pair was given the same list of features (taken from our product-by-feature matrix) and was asked to organize these features into an FM using a whiteboard. The participants were not required to create cross-tree constraints. Each pair of users took approximately four hours to complete their task.

While we did not ask the users to to follow any specific process, both pairs approached the problem by creating clusters of features related to similar topics, and then creating a hierarchy that connected these clusters. While there were several similarities in the way the two pairs created clusters, the final hierarchical organization of the two FMs was quite

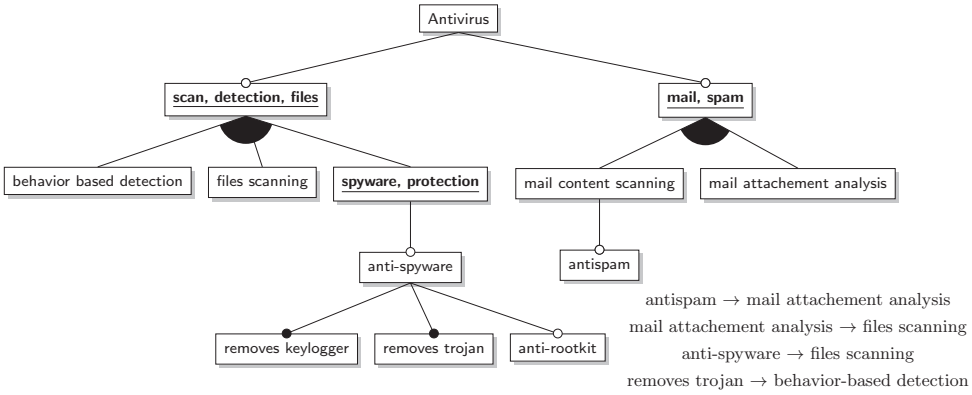


Figure 5: Resulting FM

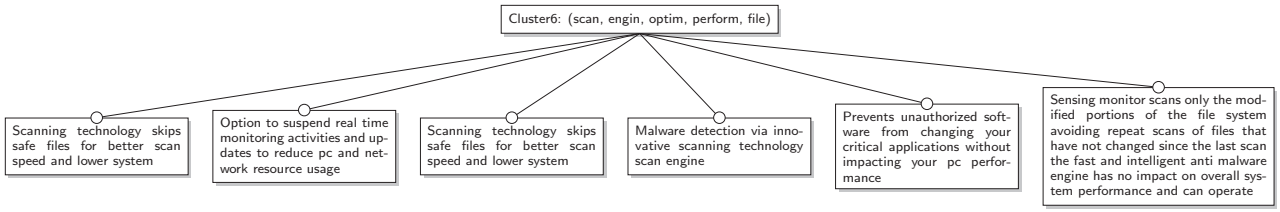


Figure 6: Group of features shown in the evaluation

different. As a result of these observations we decided that it would not be appropriate to compare our approach against a single golden-standard.

We therefore adopted a more direct evaluation technique in which we asked several graduate students familiar with feature modeling to critically evaluate parts of four different FMs. These FMs included one FM automatically generated using only association rules and the computation of a maximal tree based on the conditional probabilities between features occurrences (which we refer to as the *probabilistic* approach), the two manually constructed ones, and the one generated by our algorithm (which we refer to as the *clustered* approach. We created four surveys and two participants were assigned to each one. Each survey included questions taken from each of the FMs; however the participants were not informed of the source of each question. Each question was asked in the context of a specific parent-child relation or the grouping of features, such as the automatically generated features depicted in Figure 6). The study included 25 questions and was completed by 8 users at an average completion time of approximately one hour.

The first group of questions in the survey were designed to evaluate the quality of groups of features, and read: "*On a scale of 1-5 with 5 being the HIGHEST, please provide an overall rating for the group as a whole. A score of 5 means that the group is cohesive i.e. all features belong together in this group, while a score of 1 means that the group is not very meaningful at all*". Results from this question for the three types of FM are reported in Figure 7(a) and show that the evaluators assigned an average rating of 4.05 to groupings in the manually constructed FMs, 3.54 to the clustered FMs (our approach), and only 2.94 to the probabilistic FMs.

The second group of questions were designed to evaluate the quality of the parents for each feature group. The question read: "*On a scale of 1-5 with 5 being the HIGHEST, please provide an overall rating for the parent of the group. A score of 5 means that the parent captures the essence of the features in the group and a score of 1 means that there is no obvious relationship between the parent and its child nodes*". Results are reported in Figure 7(b) and show that the evaluators assigned an average rating of 4.10 to the manually constructed FMs, 3.46 to the clustered FMs, and only 3.02 to the probabilistic FMs.

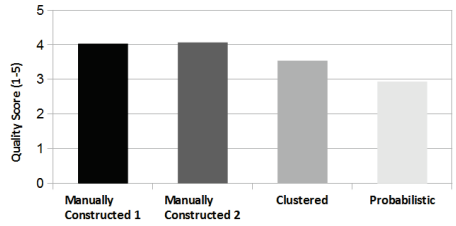
The third group of questions were designed to individually evaluate parent-child relationships. For each presented feature group, all parent-child associations were listed and evaluators were asked to mark the correct associations. Results are reported in Figure 7(c) and show that the average percentage of correct associations is 85.14% in the manually constructed FMs, 69.885% in the clustered FMs, and only 58.25% in the probabilistic FMs.

These results show that while the feature groupings in the clustered FM generated by our approach do not reach the same level of quality achieved in manually constructed FMs, they outperform the groupings of the probabilistic FMs.

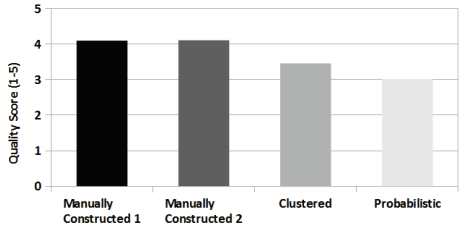
A careful comparison of the FM generated by the two automated techniques showed that in several cases, the clustered approach identified key abstractions which were not identified by the probabilistic approach. For example, Figure 6 shows a group of features that the clustered FM grouped together under an abstraction node ("Cluster6") whereas in the FM obtained by the probabilistic approach, these six features were scattered across five unrelated groupings.

7. THREATS TO VALIDITY

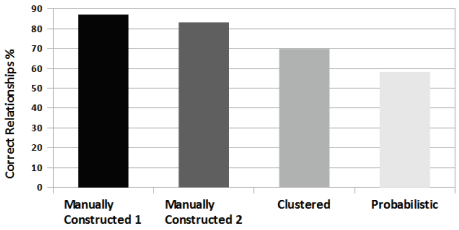
There are several threats to validity for our study. The primary threat is our assumption that extracting feature information from large numbers of partially complete product descriptions yields a representative set of features from the entire domain. We do not fully explore this assumption in this paper, apart from ensuring that (1) the domain we chose



(a) User evaluation of the quality of groups of features



(b) User evaluation of the quality of the parent for each group of features



(c) User evaluation of the quality of the individual parent-child relationships

Figure 7: A comparison of our approach: clustered *versus* manually created *versus* probabilistic approach for FM construction

for experimental purposes contained a large and varied representation of products, and (2) clarifying that it is clearly a constraint of our method (and that of most machine learning techniques), that we can only learn from available data. Applying our approach to a domain with a more constrained set of product descriptions, is highly unlikely to produce good coverage of features in the domain.

A second threat to validity lies in the scope of our study, which focused on only the single domain of antivirus software. The primary reason for this is that evaluating the quality of the generated FM is time-consuming and involves manually creating one or more FMs for the domain, and then asking human users to evaluate the quality of the product lines in a blind study. We leave the evaluation of our approach across a broader set of domains with a larger group of evaluators to a future study.

A third threat is the computation of SCCs to identify AND-groups for a set of partially complete product descriptions. As explained in section 5.2, Andersen et al. [6], compute strongly connected components in implication graphs to detect AND-groups. The mapping from SCCs to AND-groups is motivated by the fact that the implications between the features in the implication graph are transitive. Because there is a path between any two features in a SCC, each presence of a feature of a SCC in a configuration implies the presence of every other feature of the SCC in this configuration. However, the implication graph described in section 5.2 is not made of implications but probabilistic association rules which are not transitive. It follows that features that do not occur often together in configurations may be considered as parts of the same AND-group. So, in case of an implication graph using probabilistic association rules, AND-groups can be computed from SCCs to offer the user an approximation of potential AND-groups but these groups must be manually reviewed.

The final threats we discuss here relate to the evaluation process. It was impossible to entirely separate out the task of evaluating the quality of each feature's name, with the quality of the associations established in the FM. For example, if we asked a user to evaluate whether P was a correct parent of child C, then the study participant might be influenced by both the essence of the feature (i.e., what it truly represented) as well as the name of the feature. Nevertheless feature naming was essential for human cognition purposes. To mitigate this problem, our study presented both the generated feature name and the bag of words representing the primary theme(s) of the underlying feature descriptors to the user. While our evaluation was conducted by only 8 graduate students, we deliberately chose the domain of antivirus products as our evaluators were knowledgeable about this domain and therefore served as valid stakeholders. Finally, while we did perform an initial analysis of the generated FM to determine that it modeled valid (and sensible) product configurations, we did not evaluate this in a formal way. We leave this for future work.

8. RELATED WORK

Synthesizing FMs. Several techniques for synthesising an FM from a set of configurations or constraints (e.g., encoded as a propositional formula) have been proposed [19, 33, 6, 1, 24]. These techniques cannot be applied in our context, since we cannot assume the availability of formal and complete descriptions of configurations or constraints. Therefore we develop new extraction and synthesis techniques to deal with the informal and textual nature of product descriptions.

An important limitation of prior work is the identification of the feature hierarchy. In [19, 6], the authors calculate a diagrammatic representation of *all possible* FMs. However they did not address the problem of selecting a unique FM with a meaningful hierarchy. Similarly, the algorithm proposed in [24] does not control the way the feature hierarchy is synthesized in the resulting FM. In [33], She et al. proposed heuristics for identifying the likely parent candidates for a given feature in order to assist users in selecting a hierarchy. However the heuristics are specific to the targeted systems (Linux, FreeBSD, eCos both from the domain of operating system) and therefore hardly apply to our case. Furthermore She et al. reported that their attempts to use clustering techniques did not produce a single and desirable hierarchy. They gave a possible reason, arguing that *"there is simply not enough information in the input descriptions and dependencie"* for the kinds of artefacts they considered. Acher et al. [1] proposed a procedure that processes *user-specified knowledge* for organizing the hierarchy of features. Compared to [33, 1], our approach does not require user

intervention. Another key difference is that we integrate feature mining and clustering techniques for building and exploiting the implication graph.

Extraction of FMs. Acher et al. [2] proposed a semi-automated procedure to support the transition from product descriptions (expressed in a tabular format) to FMs. Ryssel et al. developed methods based on Formal Concept Analysis and analyzed incidence matrices containing matching relations [31]. A common assumption is the availability of formal and complete descriptions of products, which is not the case in our context. The two works also assume a certain structure in the product description or other knowledge that is exploited to hierarchically organize the features.

Probabilistic FMs (PFMs). In [18], Czarnecki et al. introduced PFMs. Soft constraints are formally described and indicate the conditional probabilities between the presence of features in configurations which enable reasoning about preferences between configuration choices. While hard constraints express configuration rules that must be obeyed by all the configurations, soft constraints should be respected by most configurations but can be violated by some of them. The authors extended their prior work [19] by proposing an extraction procedure that relies on association rule mining. The prior work used association rules with a confidence of 100% to build the hierarchy and association rules with confidence less than 100% but above a predefined threshold to add extra information to the FM. In our approach, the product-by-feature matrices are built from natural language text and can be very disparate. Therefore, all the mined soft constraints above a defined confidence threshold are used to build the hierarchy. Our approach also integrates clustering techniques to derive a desirable hierarchy.

Clustering techniques. Alves et.al. [5] conducted an exploratory study based on the use of the Vector Space Model (VSM) and Latent Semantic Analysis (LSA) to determine the similarity between requirements. The variability information is out of the scope of their study though. Weston et al. [39] proposed an extension to the framework proposed by Alves et al. and developed a tool which creates FMs from natural language requirements specifications. First, they divided the specifications in fragments and then used clustering techniques to identify features. The size of the segments can be parameterized by the user. They parsed the specifications to identify variabilities by detecting grammatical pattern-based structures and words from a vocabulary lexicon. In our approach, we build FMs by both using the information about co-occurrence of the features in the products and the content of features without assuming the presence of specific grammatical patterns or the presence of words from a lexicon. Chen et al. [13] proposed an approach to build FMs from applications specifications. The authors introduce a classification of relationships between requirements. For each application, the procedure first elicited a set of functional requirements and modeled the relationships between them in an undirected graph. Features were then identified by clustering the functional requirements. Finally, the resulting FMs were merged as one. Instead of merging as many models as there are configurations, we directly mine an FM from the set of all the configurations. The determination of the feature hierarchy is fully-automated and based on features co-occurrences in the dataset, without predefining any type of relationship. Niu and Easterbrook [28] provided semi-automatic support for analyzing functional requirements, denoted FRPs, in a product line. The FRPs and their attributes were extracted manually, thus increasing the user effort.

It should be noted that several other techniques [28, 13, 5, 39] assume existing requirements specifications that provide a deep and rather complete description of the products. In our context, we have to infer FMs through analyzing hundreds of informal product specifications.

9. CONCLUSION AND FUTURE WORK

In this paper we have presented a novel algorithm for automating the generation of an FM from a set of informal and incomplete product descriptions. The results from the reported evaluation show, that in the case of the antivirus software, utilizing clustering techniques to augment association rule mining led to marked improvements in the quality of the generated FM. As such, the findings reported in this paper make a significant contribution to the ongoing research goal to automate the generation of FMs.

Furthermore, one of the major advantages of our approach is that product descriptions are publicly available for many different kinds of products, which means that our approach can be used in practice even if an organization has not previously developed software for the targeted domain. A quick perusal of SoftPedia shows products such as authoring tools, desktop enhancements, file managers, ipod tools, networking tools, and office tools, to name a few. Future work will involve applying and then evaluating our approach on a far broader set of products, and also exploring other sources of informal product descriptions.

Given the probabilistic nature of our approach, the generated FM must either be further refined by a human analyst into a more formal FM or else used informally during domain analysis to provide ideas for features to implement in a product. In future work, we plan to explore the utility of the generated FM for supporting specific tasks related to domain analysis and the design of a software product line.

10. ACKNOWLEDGMENTS

The work in this paper was partially funded by US National Science Foundation Grant III-0916852.

11. REFERENCES

[1] M. Acher, B. Baudry, P. Heymans, A. Cleve, and J.-L. Hainaut. Support for reverse engineering and maintaining feature models. In *Proceedings of VaMoS'13*. ACM, 2013.

[2] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. *Proceedings of VaMoS '12*, pages 45–54, 2012.

[3] M. Acher, P. Collet, P. Lahire, and R. France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming (SCP) Special issue on programming languages*, page 22, 2013.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Intn'l Conf. on Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, September 1994.

[5] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. An

exploratory study of information retrieval techniques in domain analysis. In *SPLC*, pages 67–76. IEEE Computer Society, 2008.

[6] N. Andersen, K. Czarnecki, S. She, and A. Wasowski. Efficient synthesis of feature models. In *Proceedings of SPLC'12*, pages 97–106. ACM Press, 2012.

[7] S. Apel and D. Beyer. Feature cohesion in software product lines: an exploratory study. In *Proceedings of ICSE'11*, pages 421–430, New York, NY, USA, 2011. ACM.

[8] S. Apel and C. Kästner. An overview of feature-oriented software development. *Journal of Object Technology (JOT)*, 8(5):49–84, July/August 2009.

[9] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, 2010.

[10] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wąsowski. A survey of variability modeling in industrial practice. In *Proceedings of VaMoS'13*. ACM, 2013.

[11] BigLever – Gears. `http://www.biglever.com/solution/product.html`.

[12] P. Borba, L. Teixeira, and R. Gheyi. A theory of software product line refinement. *Theor. Comput. Sci.*, 455:2–30, 2012.

[13] K. Chen, W. Zhang, H. Zhao, and H. Mei. An approach to constructing feature models based on requirements clustering. In *Proceedings of RE'05*, pages 31–40, 2005.

[14] A. Classen, Q. Boucher, and P. Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Sci. Comput. Program.*, 76(12):1130–1143, 2011.

[15] A. Classen, P. Heymans, and P.-Y. Schobbens. What's in a feature: A requirements engineering perspective. In *Proceedings of FASE'08*, volume 4961 of *LNCS*, pages 16–30, 2008.

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

[17] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wąsowski. Cool features and tough decisions: a comparison of variability modeling approaches. In *Proceedings of VaMoS'12*, pages 173–182, New York, NY, USA, 2012. ACM.

[18] K. Czarnecki, S. She, and A. Wasowski. Sample Spaces and Feature Models: There and Back Again. In *Proceedings of SPLC'08*, pages 22–31. IEEE, 2008.

[19] K. Czarnecki and A. Wasowski. Feature Diagrams and Logics: There and Back Again. In *Proceedings of SPLC'07*, pages 23–34. IEEE, 2007.

[20] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Journal of Machine Learning*, 42:143–175, January 2001.

[21] D. Dhungana, P. Grünbacher, and R. Rabiser. The dopler meta-tool for decision-oriented variability modeling: a multiple case study. *Autom. Softw. Eng.*, 18(1):77–114, 2011.

[22] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. *Proceedings of ICSE '11*, page 181, 2011.

[23] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of SIGMOD'00*, New York, NY, June 2000.

[24] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed. On extracting feature models from sets of valid feature combinations. In *Proceedings of FASE'13*, pages 53–67, 2013.

[25] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proceedings of KDD'04*, pages 168–177, New York, NY, USA, 2004. ACM.

[26] S. E. Institute. Software product lines, http://www.sei.cmu.edu/productlines/.

[27] B. Liu, W. Hsu, and Y. Ma. Mining association rules with multiple minimum supports. *Proceedings of KDD'99*, pages 337–341, 1999.

[28] N. Niu and S. M. Easterbrook. Concept analysis for product line requirements. In K. J. Sullivan, A. Moreira, C. Schwanninger, and J. Gray, editors, *AOSD*, pages 137–148. ACM, 2009.

[29] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[30] pure::variants. `http://www.pure-systems.com/pure_variants.49.0.html`.

[31] U. Ryssel, J. Ploennigs, and K. Kabitzsch. Extraction of feature models from formal contexts. In *Proceedings of SPLC'11 (volume 2)*, pages 1–8. ACM, 2011.

[32] K. Schmid and M. Verlage. The economic impact of product line adoption and evolution. *IEEE Software*, 19(4):50–57, 2002.

[33] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki. Reverse engineering feature models. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 461–470, New York, NY, USA, 2011. ACM.

[34] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques. *Softw., Pract. Exper.*, 35(8):705–754, 2005.

[35] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[36] R. E. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

[37] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. Featureide: An extensible framework for feature-oriented software development. *Science of Computer Programming (SCP)*, 2012.

[38] D. Weiss and C. Lai. *Software product-line engineering: a family-based software development process*. Addison-Wesley, 1999.

[39] N. Weston, R. Chitchyan, and A. Rashid. A framework for constructing semantically composable feature models from natural language requirements. *Proceedings of SPLC'09*, pages 211–220, 2009.