Cross Product Line Feature Analysis

Ora Wulf-Hadash
Department of Information Systems,
University of Haifa, Haifa 31905, Israel
972-4-9590805
orawulf@gmail.com

Iris Reinhartz-Berger
Department of Information Systems,
University of Haifa, Haifa 31905, Israel
972-4-8288502
iris@is.haifa.ac.il

ABSTRACT
Software Product Line Engineering (SPLE) promotes the development and maintenance of artifacts that can be reused in families of related software-intensive systems. As product lines are not necessarily disjoint and the development of product line artifacts is a demanding task, utilization of “related” or “similar” product line artifacts for creating or improving the artifacts of a particular product line may be desirable for reducing time and effort and improving quality. Commonly product line artifacts are represented as feature models. However, current feature analysis methods concentrate on individual product lines, neglecting the ability to share the knowledge and experience gained from developing other related product lines. In this work we offer adopting similarity measurements and text clustering techniques in order to enable cross product line feature analysis. The benefits of this type of analysis are two-folded. First, such analysis will enable extension and validation of existing feature models aiming at strengthening the advantages of existing product lines and reducing their limitations with respect to other related product lines. Second, extraction of domain knowledge, analyzing a set of artifacts representing different product lines in the domain, may yield new insights relevant to the entire domain, calling for formalization of these insights and their use when creating new products in the same domain. Preliminary results reveal that the suggested method helps systematically analyze the commonality and variability between related software product lines, providing insights regarding the specific product lines and the entire domain.

Keywords
Feature Analysis, Feature Diagrams, Feature Clustering, Feature Similarity, Empirical Evaluation

1. INTRODUCTION
Software Product Line Engineering (SPLE) [9] promotes the development and maintenance of artifacts that can be reused in families of related software-intensive systems. The primary goal of SPLE is to develop tools, techniques, and methods to support identification, analysis, and management of the common and variable aspects of software products that belong to the same family. In order to handle product lines effectively and efficiently, highly reusable artifacts are developed, maintained, and utilized. These artifacts, which are commonly termed core assets [9] or domain artifacts [22], specify and implement the common aspects of a product line and support and guide the specification and implementation of variable aspects. These artifacts are further used for developing and assembling specific products in the line, aiming to improve the quality of specific software products, to reduce development time and development costs, to reduce the risk involved in product development and deployment, and to increase customer’s satisfaction from the deliverables [9]. A common way to specify a core asset is a feature model, which captures the characteristics of a given product line, as well as the relationships and constraints (dependencies) among these characteristics (features) [22].

The development of core assets in general and feature models in particular is a demanding task. It requires identification of common aspects, analysis of possible variations, and development of flexible, yet structured and well-defined, artifacts. These activities consume a lot of resources and involve many product- and production-related constraints [9]. Thus, various studies have been suggested for supporting core assets development and maintenance, especially employing feature models, e.g., [18], [19], and [15]. Currently, most studies introduce methods that use either the products in the product line or the knowledge gained in the domain of the product line for creating core assets. Some of these methods support automated analysis of feature models. Benavides et al. [4] reviewed 53 papers that refer to such methods. Based on this corpus of literature, they presented a catalogue of 30 analysis operations. All these operations focus on “inner feature analysis”, namely analysis of feature models within a specific software product line. While this kind of analysis is very important, we call in this work for “cross product line analysis”, meaning conducting commonality and variability analysis of related product lines. In particular, as product lines are not necessarily disjoint and different types of overlapping may exist, we suggest utilizing core assets from “similar” or “related” product lines for creating and improving core assets of a particular product line. In order to formally define the terms “similar” and “related”, we require that all the product lines we use will be over the same domain, namely the product lines will use, at least to some extent, similar terms and concepts.

The suggested method proposes an algorithm for analyzing the commonality and variability among related feature models aiming to provide the software product line engineers with new insights and knowledge regarding the entire domain. This new knowledge can be then used for evaluation of the product line completeness and correctness and thus may improve the quality of the developed core assets. In other words, the suggested method will enable the extension (identifying potentially missing features) and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

©2013 ACM 1-58113-000-0/00/0010 …$15.00.

1 Note that in some sources, such as [19], there is no distinction between product lines and domains. In these sources, product lines are termed domains, while products are called applications. Similarly to [9], we distinct in the current work between three abstraction levels, namely domain, product line, and product, in order to support cross product line analysis.
3. Feature Models, Formalisms, and Analysis

3.1 Feature Models and Feature Diagrams

A large group of SPL methods that concentrate on variability representation and management use feature-oriented notations [8]. These notations, known as feature models, support representing characteristics of families of software products and the relationships between them [18]. In other words, a feature model represents the common and variable aspects of products within a specific product line [19]. Feature models are usually composed of feature diagrams and descriptions. A feature diagram is a graphical notation for describing features and their relationships and dependencies [27]. It is usually represented as a tree or a graph, where the nodes denote features and the edges – relationships and constraints (dependencies). Feature descriptions add information regarding the intention of the feature, its name, and its possible synonyms [18]. Feature descriptions may further refer to various aspects of the features, such as trade-offs, rationale, and justifications for feature selection [19]. Although feature descriptions add important information that is not shown in the diagrams, feature diagrams are the most utilized specification aids in SPL, probably due to their simplicity.

Four types of relationships are commonly defined in feature diagrams: (1) mandatory features which must be selected for any product in the line, (2) optional features which have some added value when selected, but not all products in the line must exhibit them, (3) alternative features which cannot appear together in the same product, and (4) ‘or’ features which support any combination of the features that includes at least one feature.

A feature diagram may further constrain the selection of features utilizing the following two types of constraints: (1) A requires B, meaning that a product cannot exhibit feature ‘A’ unless it also exhibits feature ‘B’, and (2) A excludes B, meaning that both features cannot appear together in the same product, i.e., either one of them appears or they are both excluded.

3.2 Feature Diagram Formalism

We will use the following formal definition of a feature diagram, adapted from [1].

Definition 1 (Feature Diagram): a feature diagram is defined as

\[ FD = <G, r, E_{MAND}, G_{XOR}, G_{OR}, I, EX> \], where:

- \( G = (F, E) \) is a rooted tree; \( F \) is a finite set of features; \( E \subseteq F \times F \) is a finite set of edges.
- \( r \in F \) is the root of \( G \).
- \( E_{MAND} \subseteq E \) is a set of edges that define mandatory features with their parents.
- \( G_{XOR} \subseteq F \) define alternative and “or” features respectively; they are described as pairs of child features and their common parent feature.
- \( I, EX \) are sets of feature pairs related with “requires” (a.k.a. “implies”) and “excludes” constraints, respectively; a “requires” constraint is of the form \( \neg A \rightarrow B \) and an "excludes" constraint is of the form \( \neg A \rightarrow \neg B \), where \( A, B \in F \).

As an example of a feature diagram, consider the diagram in Figure 1 that (partially) represents the features in a product line of mobile phones for the tablets, could also benefit from the analysis, whereas in the second example each one of the companies could learn and adjust its core assets according to the generated “generic” messaging function.

---

\[ \text{We removed } G_{MUTEX} \text{ and BI that appeared in the original definition [1], since in the current stage we refer to the most simplified version of feature diagrams that only includes OR and XOR relationships and “requires” and “excludes” constraints.} \]
mobile phones. Part (a) is the graphical representation of this product line, while part (b) is its formal representation.

![Feature Diagram](image)

### Feature Analysis

Feature analysis refers to the activities for identifying product features, classifying them, and organizing them as a set of coherent feature models [19]. Reviewing 53 papers, published between the years 1990 and 2009, Benavides et al. [4] found about 30 analysis operations on feature models or diagrams. The automated support that has been proposed for these analysis operations was further classified by Benavides et al. into four different groups according to the used paradigm or method.

The first group of methods utilizes propositional logic for enabling automated analysis of large software product lines that include many potential products. A propositional formula is a set of primitive variables and propositional logic predicates (namely, $\land$, $\lor$, $\rightarrow$, and $\iff$) [3]. When mapping a feature diagram into a propositional logic each feature is mapped into a variable and each relationship is mapped into one or more small formulas depending on the type of relationship. Examples of methods utilizing propositional logic for feature analysis are [10] that supports the identification of void feature models and [3] which use propositional logic for checking product validity.

The second group of methods, constraint programming, includes methods that deal with Constraint Satisfaction Problems (CSP), namely problems whose solutions require finding states (values for variables) in which all constraints are satisfied [26], [6]. Contrary to propositional formulas, CSP solvers can deal not only with binary values (true or false), but also with numerical values such as integers or intervals. An example of a method in this category is [5] which supports calculating the number of products and analyzing the commonality within a product line.

The third group of methods uses description logic, which was originally developed as a formalism for representing knowledge [2]. A problem described in terms of description logic is usually composed of a set of concepts, a set of roles and a set of individuals. The reasoner receives a problem described in description logic and provides facilities for consistency and correctness checking [4]. Examples of methods in this category are [32] which identifies dead features and [29] and [28] which supports checking product validity and identifying void features.

Finally, the fourth group contains methods that cannot be classified into the other groups, including ad-hoc solutions, algorithms, and paradigms.

All surveyed feature analysis operations refer to a specific product line and its products. Extending the scope of feature analysis to several software product lines will enable sharing the knowledge and experience gained in those related product lines and may lead to development of improved core assets. Note that sharing the knowledge between different software product lines will not affect the uniqueness of each one of them by its own, but may greatly contribute to the overall quality of core assets, and sequentially to the quality of products.

### 4. The Cross Product Line Analysis Method

#### 4.1 Method Overview

The input to the cross product line feature analysis method is a set of feature diagrams, (FDi), e.g., in the Simple XML Feature Model format (SXFM) [24] that follows the formalism provided in Definition 1. Each diagram represents a different product line, however, in order to get meaningful output, all feature diagrams need to be over the same domain.

The input is processed in three main steps (see Figure 2). First, during the Feature Similarity Analysis step, the set of feature diagrams is analyzed using linguistic and structural techniques for finding similar features and relationships that can serve as anchors for the next step of clustering. In the second step, feature clustering, an agglomerative clustering technique is used for creating groups (clusters) of similar features that may represent variants of the same features. Finally, in the Output Generation step, the clusters from the previous step are analyzed to provide insights regarding the commonality and variability of the examined feature diagrams.

#### 4.2 Feature Similarity Analysis

The definition of a set of products as a family or a product line mainly relies on the similarity of the product’s features: products which share a small number of similar features cannot be considered as a family or a line. However, the different features are the ones that make the products distinguishable. These observations are valid when analyzing different product lines. Here again the similarity between the features of these product lines makes them a group, this time called a domain, while the differences make them distinguishable product lines. Nevertheless, in this case further difficulties are introduced, namely different product lines may name the same or similar features differently. Thus, in order to measure the amount of commonality and variability between the product lines in the given set, we will measure the similarity in their feature names and the similarity in their context (i.e., where they appear in the feature diagrams).

---

3 Note that the method will work even if the feature diagrams are from different domains, but then the low degree of similarity will prevent the method from suggesting improvements.
4.2.1 Similarity of Feature Names

We utilize linguistic measurements for calculating the distances between terms and sentences. Many of the linguistic measurements (see [7], for example) use WordNet, which is a large lexical database of English [30]. The benefits of WordNet are that it is large, rich, freely available online, and general-purpose; hence, it can be used for different domains. Note, however, that WordNet also has shortcomings in the context of cross product line feature analysis. The features in technological domains can be represented as abbreviations or commonly known acronyms which are sometimes not recognized as meaningful words for WordNet. In other cases, the same word may have different meanings depending on the domain and the context. To overcome these deficiencies, we currently added the ability to import user-defined acronyms for certain domains. In the future, we intend to improve this step with Wikipedia-based semantic analysis methods, such as the one proposed in [14].

For measuring the similarity of two features, we adopted Dao and Simpson’s similarity measurement between two sentences (or phrases) [11], which is a simple and straightforward metric that does not require a large corpus of statistics. The following formula defines feature name similarity.

**Definition 2 (Feature name Similarity).** Let \( f_1 \) and \( f_2 \) be two features (from two different feature diagrams or from the same feature diagram). Feature name similarity, \( NSim \), is calculated as follows:

\[
NSim(f_1, f_2) = \frac{\sum_{i=1}^{m} \max_{j=1}^{n} l_{i_j} + \sum_{j=1}^{n} \max_{i=1}^{m} l_{i_j}}{m + n}
\]

Where:
- \( l_{i_j} = \frac{2^N_i}{N_i + N_j + 2} \)
- \( N_i \) is the number of nodes on the shortest path from \( t_i \) to LCS in WordNet
- \( N_j \) is the number of nodes on the shortest path from \( u_j \) to LCS in WordNet
- LCS is the least common super-concept of \( t_i \) and \( u_j \) in WordNet

Figure 3. Calculating similarity between terms that are hierarchically related

As an example of calculating the name similarity of two features, consider the feature 'Short Message Service' (SMS), which appears in Figure 1, and a feature named 'text message' that can appear in another mobile phones product line. Table 1 summarizes the pair-wise name similarity values of these features, while the formula below calculates their feature name similarity.

\[
NSim('short message service', 'text message') = \frac{(0.52 + 1.00 + 0.55) + (0.62 + 1.00)}{3 + 2} = 0.74
\]

<table>
<thead>
<tr>
<th>text</th>
<th>short</th>
<th>message</th>
<th>service</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.52</td>
<td>0.62</td>
<td>0.55</td>
<td></td>
</tr>
<tr>
<td>0.43</td>
<td>1.00</td>
<td>0.46</td>
<td></td>
</tr>
</tbody>
</table>

4.2.2 Similarity of Feature Context

So far we considered similarity in the feature names. However, the inputs of our method are feature diagrams, which are structured trees (or graphs) of features and not plain lists. The context of a feature in the product line is highly important for determining its role. Note that similar structures of completely different features may not indicate on their potential relatedness. However, the similarity of features whose names and structures are similar should be higher than the similarity of features which only share similar names.

At the current stage, we focus on relationships between parent and child features, without referring to the different relationship types (e.g., mandatory vs. optional features). In the future we will need to examine the different impacts of the various relationships and dependencies on feature similarity.

Let \( f_1, f_1', f_2, \) and \( f_2' \) be features, such that \( f_1 \) and \( f_1' \) are connected via a relationship (\( f_1 \) is the parent) and \( f_2 \) and \( f_2' \) are connected via a relationship (\( f_2 \) is the parent) (see Figure 4). If \( f_1 \) and \( f_2 \) are similar (considering both their names and context with respect to their children), then the similarity of \( f_1 \) and \( f_2' \) should increase. Note that the increase in the similarity is percolated from the leaves of the feature diagram to its root. The following definition defines feature similarity taking into consideration both feature names and context.

**Definition 3 (Feature Similarity).** Feature similarity of features \( f_1 \) and \( f_2 \) in \( F \) is calculated using the following formula:

\[
Sim(f_1, f_2) = \frac{\frac{NSim(f_1, f_2)}{m + 1} + \sum_{(f_1', f_2')} \text{Sim}'(f_1', f_2')}{m + 1}
\]

Where \( \text{Sim}'(f_1', f_2') = \text{Sim}(f_1', f_2') \) if \( \text{Sim}(f_1', f_2') > \text{threshold} \) and \( \text{Sim}'(f_1', f_2') = 0 \) otherwise; \( m \) is the number of pairs \( (f_1', f_2') \) satisfying this condition, i.e., \( \text{Sim}(f_1', f_2') > \text{threshold} \).

In order to determine the threshold for similar features, different algorithms may be used. As will be explained and demonstrated in Section 5, we chose to use AdaBoost [12], which is a machine learning, adaptive algorithm, for this purpose.

Three important characteristics of the above formula are: (1) the value of similarity is always between 0 and 1; (2) the similarity of parent features increases proportionally to the degree of similarity of their children; and (3) the similarity of parent features increases proportionally to the number of similar children.

As an example of feature similarity calculation, consider two portions of product lines in the mobile phones domain. The first product line, depicted in Figure 5(a), is taken from Figure 1. The second feature diagram, shown in Figure 5(b), refers to the feature ‘sending utility’ and its mandatory child – ‘text message service’.

![Figure 4. Percolating similarity through relationships](image-url)
Feature name similarity calculation reveals that the child features SMS, MMS, and ’text message service’ are similar (their similarity values are greater than 0.88, see Table 2), while the parent features, namely, ’sending utility’ and ’messaging’ are less similar (their feature name similarity is about 0.64).

Table 2. Examples of feature name similarity

<table>
<thead>
<tr>
<th>feature1</th>
<th>feature2</th>
<th>feature name similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>short message service</td>
<td>multimedia message service</td>
<td>0.913943355</td>
</tr>
<tr>
<td>short message service</td>
<td>text message service</td>
<td>0.898860399</td>
</tr>
<tr>
<td>short message service</td>
<td>sending utility</td>
<td>0.778038278</td>
</tr>
<tr>
<td>multimedia message service</td>
<td>messaging</td>
<td>0.583333333</td>
</tr>
<tr>
<td>text message service</td>
<td>sending utility</td>
<td>0.886877828</td>
</tr>
<tr>
<td>text message service</td>
<td>messaging</td>
<td>0.79214748</td>
</tr>
<tr>
<td>sending utility</td>
<td>messaging</td>
<td>0.687719298</td>
</tr>
<tr>
<td>text message service</td>
<td>messaging</td>
<td>0.758038278</td>
</tr>
<tr>
<td>text message service</td>
<td>messaging</td>
<td>0.644444444</td>
</tr>
<tr>
<td>text message service</td>
<td>messaging</td>
<td>0.516666667</td>
</tr>
<tr>
<td>sending utility</td>
<td>messaging</td>
<td>0.644444444</td>
</tr>
</tbody>
</table>

Figure 5. Two portions of feature diagrams in the mobile phones domain

If we take into consideration the similarity between the children in these diagrams, then the overall similarity between ’sending utility’ and ’messaging’ will increase as follows:

\[ \text{Sim}('\text{sending utility}', '\text{messaging}') = \frac{\text{NSim(SU,M)}}{\text{Sim(SMS,TMS)} + \text{Sim(MMS,TMS)}} = 0.81 > 0.64 \]

Note that the values of both name similarity and overall similarity have no absolute meaning, but only relative ones (“more similar than”, “less similar than”). Thus, other techniques are required for better understanding the degree of similarity of the different feature diagrams or product lines that they represent. For this purpose, we utilize feature clustering, as described next.

4.3 Feature Clustering

Clustering is the process of grouping a set of objects into classes of similar objects [21]. In our research, the objects are features that are represented via their names and relationships. Thus, document or text clustering techniques may be relevant [25], [16].

The suggested cross product line feature analysis uses a variation of the agglomerative hierarchical clustering technique. This technique [20] is a bottom-up clustering approach, which gets as a parameter the number of expected clusters and starts with putting each object in a separate cluster. Then, in each iteration, the algorithm agglomerates (merges) the closest pair of clusters by calculating the distance between different clusters. The algorithm continues until the number of expected clusters is reached.

We chose this algorithm because of the following reasons. First, this algorithm is known as one of the most accurate clustering techniques [25], [20]. Since clustering quality has a great impact on the feature analysis results, accuracy is very important in our case. Second, the distance between two clusters reflects the degree of similarity between their features. Starting with each feature in a different cluster will prevent grouping features when they are not similar enough. However, the agglomerative hierarchical clustering algorithm requires determining the number of clusters a-priori. This number cannot be determined in our case as it varies depending on the size of the product lines and their degree of variability. Therefore, we modified the stopping criterion of the algorithm to merge two closest clusters as long as the distance between them is not bigger than a pre-defined threshold. This way we ensure that too different features will not be put in the same cluster.

Three types of distances between clusters are commonly mentioned in the literature [23], [17] (see Figure 6):

- **Single-link**: the distance between the two closest features of the clusters.
- **Complete-link**: the distance between the two farthest features of the clusters.
- **Average-link**: the average of pair-wise distances between features in the two clusters.

Three types of distances between clusters are commonly mentioned in the literature [23], [17] (see Figure 6):

- **Single-link**: the distance between the two closest features of the clusters.
- **Complete-link**: the distance between the two farthest features of the clusters.
- **Average-link**: the average of pair-wise distances between features in the two clusters.

Our algorithm for feature clustering supports the three aforementioned types of distances and the user can choose the preferable type. In cases where the domain is narrow and there are many similar features, for example, the ’complete-link’ distance may be more appropriate to create a refined division to clusters (namely, more clusters where each cluster includes fewer but more similar features). If the domain is wide, on the other hand, the ’single-link’ may be better to avoid stopping cluster merging too early. After selecting a distance type, the algorithm calculates all distances between clusters according to the selected type of distance.

Let \( \text{FeatSim} \) be a two dimensional matrix, where the cell \( i, j \) represents the (overall) similarity between feature \( i \) and feature \( j \). The feature clustering algorithm starts with creating a hash of clusters, each holding one feature that appears in \( \text{FeatSim} \). Then, iteratively, the algorithm measures the distance between clusters and merges the closest clusters as long as their distance is greater than some threshold – \( \theta \).

\begin{verbatim}
FeatureClustering(FeatSim){
    // Initialization a hash of the clusters, each holding a
    // single feature from FeatSim
    FeatClst = InitializeHash (FeatSim)
    Do { // A merging iteration
        // find the two closest clusters based on the
        // selected distance method
        (i,j) = FindClosestClusters(FeatClst, FeatSim)
        If Distance (FeatClst[i],FeatClst[j], FeatSim) ≥ \theta
            //merge two closest clusters
            merge(i, j, FeatClst)
        }
    // merge as long as there are close clusters
    While Distance (FeatClst[i],FeatClst[j], FeatSim) ≥ \theta
    Return FeatClst
}
\end{verbatim}
To demonstrate the feature clustering step, consider another feature diagram of a mobile phones product line (Figure 7). The clusters that can be generated for the two examples of mobile phones product lines (Figure 1 and Figure 7) using a threshold of 0.88 (whose choice will be elaborated in Section 5) are:

Cluster 1: mobile phone
Cluster 2: voice call, calls
Cluster 3: messaging, EMS, MMS, SMS
Cluster 4: High resolution, Low resolution, color, colour, basic
Cluster 5: screen, display
Cluster 6: media, extras
Cluster 7: camera, mp3, mp4
Cluster 8: utility functions

Figure 7. Another feature diagram of a mobile phones product line

### 4.4 Output Generation

Once the clusters are created, we can analyze intra- and inter-cluster relationships in order to get insights regarding the commonality and variability of the examined feature diagrams.

When analyzing features, we distinguish between features within the same cluster and features that belong to different clusters. Several reasons may cause features to fall into the same cluster: (1) the features are identical or almost identical (i.e., the values of the similarity measurement are almost 1). Examples of such a case are the features ‘color’ and ‘colour’ in our example; (2) the features are not identical but “similar enough”, potentially justifying their consideration as different specializations or variants of the same abstract feature. Examples of such a case are the features EMS, MMS, and SMS in our example; and (3) the names of the features are different but the context in which they are used is so similar that their overall similarity has increased. Examples of such a case are the features ‘media’ and ‘extras’ in our example. In all these cases we may wish to learn from the context of a feature about potential contexts of similar features, including their decomposition into children and reference to other features. For example, we may wish to add the feature mp4 below the feature media in the second example (Figure 7).

Another interesting observation may be examining whether the clusters include features from different feature diagrams or from the same feature diagram. Inclusion of features from different feature diagrams, representing different software product lines, may indicate on a high degree of similarity between the product lines, potentially calling for mutual learning between the product lines. However, sometimes different representations of the same fact may use complementary features. For example, consider cluster 4 (including ‘high resolution’ and ‘low resolution’) and cluster 7 (including ‘camera’) in our example. In both feature diagrams camera requires high resolution and no low resolution. However, this constraint is differently expressed: in Figure 1 it is expressed as ‘(camera ∧ high resolution)’, while in Figure 7 – as ‘(¬camera ∨ ¬low resolution)’. In the future, we will employ mining techniques to automatically detect such situations.

The relationships between features after performing the feature clustering step can be divided into intra- and inter-cluster relationships. Furthermore, the strength of inter-cluster relationships can be determined according to the number of relationships between features of the clusters, the types of relationships, and the number of involved software product lines. A weak relationship or a relationship that involves only a few software product lines may indicate on an arbitrary relationship that cannot be generalized. A strong (tight) relationship, on the other hand, that involves several different feature diagrams may point on a common knowledge that is relevant to product lines in the domain. This type of relationship should be further analyzed and possibly generalized to improve the domain knowledge. In our example, the relationship between Clusters 6 and 7 (possible media or extras in mobile phones) and the relationships between Clusters 1 and 2 (supporting calls in mobile phones) can be considered tight.

Intra-cluster relationships may occur when the refinement of features is fine-grained, so that, for example, the parent features and their children are similar enough to be included in the same cluster. As an example to such a case consider Cluster 3 in our example that includes the features ‘messaging’, EMS, MMS and SMS. The last three features can be considered specializations or possible variants of the first feature – ‘messaging’.

### 5. PRELIMINARY RESULTS

We started evaluating the suggested method on feature diagrams taken from S.P.L.O.T, an academic feature diagrams repository [24]. This repository includes about 220 feature diagrams, which were extracted from academic publications and other relevant sources. The size of the diagrams range from 9 to 290 features and the domains vary and include insurance products, mobile phones, electronic shopping, and more. The criteria for including feature diagrams in that repository are listed in S.P.L.O.T web site: (1) Consistency: All models are guaranteed to be consistent (contain at least one valid configuration); (2) Correctness: None of the models contain dead features; and (3) Transparency: All models identify their authors (or related literature) and provide some contact information.

We examined the domains that are included in S.P.L.O.T repository and selected the mobile phones domain, due to the existence of seven different feature diagrams in the repository. We modeled two additional feature diagrams based on the supplement material, we found on this domain and added some challenges to better evaluate our method. In particular, we added synonyms and antonyms and we modeled the hierarchies of features using different nesting structures. Table 3 lists the nine feature diagrams we used in the evaluation. We used the number of features (marked #F in the table) and the number of levels (marked #L) as indicators to the complexity of the diagrams. As can be seen, the diagrams are quite simple, since we wanted to be able to examine the method outputs manually. In the future we will evaluate the method on more complicated feature diagrams.

We run the suggested method on the above set of feature diagrams. In order to determine the threshold for similar features and cluster merging we used AdaBoost [12], which as noted is a machine learning, adaptive algorithm. We chose AdaBoost because of the following advantages [13]: it is fast, simple and easy to program; it has no parameters to tune (except for the number of rounds); and it requires no prior knowledge about the weak learner and thus it can be flexibly combined with any method for finding weak hypotheses. In addition, often boosting
does not suffer from over-fitting. Therefore, the classifier found on a specific domain can be generalized and used as a pre-defined threshold for the clustering algorithm.

Table 3. The feature diagrams used in the evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>Creator</th>
<th>#F</th>
<th>#L</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile Phone</td>
<td>ISA-RDA</td>
<td>10</td>
<td>3</td>
<td>model_20120110_139114401.xml</td>
</tr>
<tr>
<td>Mobile Phone</td>
<td>Sergio Segura</td>
<td>20</td>
<td>4</td>
<td>model_20100322_955726153.xml</td>
</tr>
<tr>
<td>Mobile Phone</td>
<td>AE UNB</td>
<td>25</td>
<td>4</td>
<td>model_20101119_1472596180.xml</td>
</tr>
<tr>
<td>Phone</td>
<td>m</td>
<td>10</td>
<td>3</td>
<td>model_20101111_1790887308.xml</td>
</tr>
<tr>
<td>Mobile Phone</td>
<td>Lenita ETSH</td>
<td>10</td>
<td>3</td>
<td>model_20120110_1719396361.xml</td>
</tr>
<tr>
<td>Example</td>
<td>Rayco</td>
<td>10</td>
<td>3</td>
<td>model_20120110_1094246588.xml</td>
</tr>
<tr>
<td>Cell Phone</td>
<td>Sebastian Oster</td>
<td>15</td>
<td>4</td>
<td>model_20100308_1032655961.xml</td>
</tr>
<tr>
<td>Mobile Phone</td>
<td>Self</td>
<td>21</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Mobile Phone</td>
<td>Self</td>
<td>21</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

We trained the algorithm on 328 pairs of features and their similarity scores as indicated by six human graders and Dao and Simpson’s similarity measurement (with Wu and Palmer’s formula for comparing two words). The human graders, who have strong technical background and experience in mobile devices architecture, were required to grade each pair of features on a scale that ranged from 1 to 10. After further interviews with the graders, we identified four similarity categories: completely different features (grades 5-7), similar features (grades 8-9), and identical features (grade 10). We set the threshold to 0.88, as the similarity measurement of most pairs categorized by humans as different (grades 1-7) were below this threshold.

Running the cross product line feature analysis method on our set of (nine) feature diagrams, we got 29 clusters. Each cluster includes between 1 to 19 features. We analyzed these clusters and the relationships between them, as described in Section 4.4. Based on this analysis, we further created a list of 10 questions true/false regarding the commonality and variability within the domain of mobile phones. Examples of the questions are:

- Supporting call is an essential feature of mobile phones.
- SMS, MMS, and EMS are all types of messaging services.
- Media supplies extra features to mobile phones.

We divided the questions into questions that concentrated on features within the same cluster (fin), questions that focused on features in different clusters (fout), questions that concentrated on intra-cluster relationships (rin), and questions that focused on inter-cluster relationships (rout). Two people who are familiar with the domain of mobile phones were asked to answer the questions. Based on the answers, we concluded the following.

- **Features within the same cluster** may indeed indicate on the commonality between the software product lines calling for mutual learning and knowledge sharing. The features ‘extras’ and ‘media’, whose name similarity is relatively low, were clustered together and their role in mobile phones can indeed be considered similar.

**Features in different clusters** may indicate on the extent of the domain. A large number of clusters may indicate on a wide domain or on the selection of largely different product lines (e.g., product lines that belong to opposite boundaries of the domain). A cluster that includes many features may indicate on low standardization of terms in the domain (i.e., different product lines use very similar, but non-identical, terms for the same feature). For example, the features ‘screen’ and ‘display’ in our case can demonstrate such a situation.

**Intra-cluster relationships** may indeed indicate on refinement of features. As the features within the same cluster are similar to each other, a parent and a child that fall into the same cluster may indicate on fine-grained refinement of features in at least one product line. The features SMS, MMS, and EMS are all types of messaging service. The method indeed resulted in grouping all these features in the same cluster.

Depending on their strength, **inter-cluster relationships** may indicate on the commonality and the variability within the domain. A weak relationship may indeed indicate on an arbitrary relationship which cannot be generalized. An example to such a case is the relation between the features ‘extras’ and ‘mp4’. Although such a relationship is feasible in reality, it appeared only in one feature diagram in our example, preventing the method from generalizing this finding. A strong (tight) relationship, on the other hand, may point on a common knowledge that is relevant to the entire domain. This type of relationship should be further analyzed and possibly generalized to enrich the domain knowledge. Relationships between Cluster 1 (including the feature ‘mobile phone’) and Cluster 6 (including the features ‘media’ and ‘extras’) appeared in all involved software product lines, meaning that although these features are optional (and not mandatory), they are an essential part of the domain knowledge.

Although our results are promising, only further evaluation may indicate whether our results can be generalized to other (more complicated) cases.

6. SUMMARY AND FUTURE WORK

Analysis of the commonality and variability within families of software product lines is highly important for standardization and interoperability purposes. Currently feature analysis methods focus on specific product lines. In this research we call for conducting cross product line analysis in order to enrich the domain knowledge and to improve the correctness and completeness of existing core assets (product line artifacts) in the domain. To this end, we introduce a method that uses linguistic and structural similarity techniques for measuring feature similarity and an agglomerative hierarchical clustering technique for analyzing the commonality and variability of the examined product lines. Preliminary results indicate that the method may have potential for both extracting domain knowledge and improving product line artifacts.

Future research includes several directions. First, so far we mainly used WordNet for similarity calculations. However, as there are a wide variety of domains which require wide and updated corpuses of terms, additional techniques for measuring semantic similarity should be evaluated (e.g., the one that is based on Wikipedia [14]). In addition, the impact of the different similarity types, namely name and context similarity, on the overall similarity needs to be examined, especially for differently structured product lines in the same domain.

Second, currently the feature clustering is done based on the existence of relationships and not their types. However, mandatory features may indicate on a stronger relationship than
optional features, as the first kind points on essential characteristics of the product line, while the second kind may point “only” on possible added values. Dependencies should be analyzed as well.

Third, the usability of the method can be improved by representing recommendations for changes for each examined product line. This can be done by introducing mining techniques for automatic analysis of the clustering results.

Finally, additional evaluations of the method are required. In particular, additional, more complicated domains need to be examined, as well as different sources of feature diagrams (besides S.P.L.O.T repository) need to be explored.

REFERENCES