Modelling Imperfect Product Line Requirements with Fuzzy Feature Diagrams

Joost Noppen
Computing Department
University of Lancaster
Infolab21, Southdrive, Lancaster
LA1 4WA, United Kingdom
j.noppen@comp.lancs.ac.uk

Pim van den Broek
Department of Computer Science
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
pimvdb@ewi.utwente.nl

Nathan Weston
Computing Department
University of Lancaster
Infolab21, Southdrive, Lancaster
LA1 4WA, United Kingdom
westonn@comp.lancs.ac.uk

Awais Rashid
Computing Department
University of Lancaster
Infolab21, Southdrive, Lancaster
LA1 4WA, United Kingdom
marash@comp.lancs.ac.uk

Abstract

In this article, we identify that partial, vague and conflicting information can severely limit the effectiveness of approaches that derive feature trees from textual requirement specifications. We examine the impact such imperfect information has on feature tree extraction and we propose the use of fuzzy feature diagrams to address the problem more effectively. This approach is then discussed as part of a general research agenda for supporting imperfect information in models that are used during variability analysis.

1 Introduction

With the increasing demand for software systems in a large variety of environments, software companies need to minimize development time and effort to remain competitive in the market. Over the years, software product lines (SPL) have become a popular means to attain these goals. In particular when families of products with overlapping functionalities must be provided, production time and cost of development can be significantly reduced by setting up an SPL infrastructure [17].

One of the most critical steps in SPL development is the identification of variable and common elements in the products that are to be supported. This stage determines which reusable assets and variability mechanisms will be included and the effectiveness of the SPL in supporting the product family. To aid software architects, many approaches have been proposed to describe and model variability and commonality of SPLs of which, arguably, feature modelling is the most well-known.

In spite of these efforts, deriving an accurate variability model from requirement documents remains a hard and complex activity. In [14] it has been identified that the vast majority of all requirement documentation is described in natural language. And as natural language is inherently inaccurate, even standardized documentation will contain ambiguity, vagueness and conflicts. Moreover, the requirements documentation of SPLs does not solely consist of standardized specifications. Rather, it consists of a range of documents that complement traditional requirement documents, such as business plans, marketing studies and user manuals.

This fragmentation of requirement documentation is already considerable for *pro-active* SPL design, when the various common assets and variations in a product line are designed upfront based on assumptions and estimates of foreseeable demands. When the variations are built into the product line incrementally driven by market needs (*reactive* SPL design) or when the product line is engineered by tailoring an existing product that is used as a baseline (*extractive* SPL design), requirements documentation becomes even more fragmented and inaccurate. Requirements can originate from various unrelated sources and can initially be specified without an SPL context in mind. The specifications that result generally are unfit for accurate SPL development as the ambiguous and unclear definitions hinder

the accurate representation and analysis of variability and commonality.

In this article, we examine the influence such *imperfect information* has on the definition of variability models and its consequences during SPL development. In particular, we examine the stage that tries to derive feature trees from requirement documentation. Based on the identified problems, we define an approach and research agenda for dealing with imperfect information during feature tree definition more effectively. To illustrate these results, an approach [2] that derives feature trees by clustering requirements based on their similarity is taken as a case study.

The remainder of this article is as follows. In Section 2, the problem of imperfect information during SPL development is defined. Section 3 explores the impact of imperfect information on approaches that cluster requirements to form feature trees and analyses how imperfect information influences their effectiveness. Our approach for capturing imperfection during feature tree derivation is described in Section 4 and in Section 5 we discuss and define the research agenda for imperfect information support in variability modelling. Related work is discussed in Section 6 and Section 7 concludes.

2 Imperfect Information in SPL Development

2.1 Introduction

Successful SPL development requires a considerable amount of information on, for example, the products to be supported, expectations of the product line architecture, etc.. But as we have identified in Section 1, the requirements documentation generally contains vagueness, ambiguities, conflicts or information is missing completely. For this reason, the available information might not be fit as input for the design of an SPL.

To demonstrate the impact imperfect information can have on SPL development, consider a software product line of home automation systems. These systems are designed to automatically co-ordinate various devices present in the users home in order to regulate lighting, heating, fire control and various other concerns via centralised control by the inhabitant. The following excerpt originates from the requirement definition that originally appeared in [1].

1. Access control to some rooms may be required. The inhabitants can be authenticated by means of a PIN introduced in a keypad, passing an identification card by a card reader, or touching a fingerprint reader.
2. If the opening of a window would suppose a high energy consumption, because, for instance, the heater is trying to increase the room temperature and it is too cold outside, the inhabitants will be notified through the GUI.

When examining these descriptions, it can be seen that the information provided can hinder successful SPL development. There are a number of statements that contain ambiguity. For example, requirement 1 states that "some" rooms "may" require access control. They word *may* in this context can mean that it is optional to have access control for a *product* of the SPL, but it can also mean that it is optional to be included in SPL all together. The second requirement states that a check should be made on whether opening a window leads to "high" energy consumption. It does not specify what *high energy consumption* exactly means and whether this is defined by an inhabitant or it should be determined by the Smart Home.

When this information is used during SPL development, the software architects will use the interpretations that seem most logical. However, the interpretation that is chosen will make a huge difference for the resulting SPL. When high energy consumptions is to be determined by the inhabitant, it will influence user interaction elements. However, when this is to be determined by the Smart Home, it will require changes to sensors and reasoning elements.

The traditional way of solving imperfection is to resolve it together with the stakeholders [5]. However, such kind of perfective modifications require assumptions on what is actually meant by the imperfect information. When these assumptions can be justified, this is a valid step during the design. When such assumptions cannot be justified, however, the imperfection can only be resolved by making an arbitrary choice between the possible interpretations.

Nonetheless, the choice of one interpretation over the other can have considerable impact on the SPL and its effectiveness. In particular the phase in which requirements are clustered into features (whether manually or by a tool), the consequences can be severe. The choice of interpretations directly impacts the structure of the resulting feature tree which is used to determine the common assets and variability mechanisms of the SPL, two elements that are key for the success of the SPL.

2.2 Definition of Terms

To establish a uniform terminology, we define the concept of imperfect information in terms of information being *sufficient* or *insufficient* with respect to the context in which it is used:

- Information is considered to be *sufficient* with respect to a particular context of use when it is possible to

come to an optimal decision in this context with only this information.

- Information is considered to be *insufficient* with respect to a particular context of use when the information is not sufficient for this context.

The concept of sufficiency relates information to the context in which it is to be used. Depending on decisions to be taken, the available information can or can not suffice to resolve the situation. For example, when the uncertain occurrence of an undesired event is given by a probabilistic model, this information is sufficient to perform a risk analysis. However, the probabilistic model does not provide sufficient information to answer the question *"Will the event occur tomorrow?"* with a yes or no. Whenever information is insufficient for particular context of use, this can be resolved in two ways: perfecting the information to make it sufficient for its context or adjusting the context so the information becomes sufficient.

Based on these definitions, perfect and imperfect information can be defined as follows:

- Information is considered to be *perfect* when it is sufficient in any context of use.
- Information is considered to be *imperfect* when it is not perfect.

3 Feature Trees from Imperfect Textual Requirements

3.1 Requirements Clustering

Since its first introduction [10], feature-oriented domain analysis has become one of the most widely used variability-modelling approaches. Over the years, many extensions have been proposed to this initial model, such as FeatuRSED [8], and it has been successfully applied in the design of software product lines. With increased use, the formal semantics and correctness of feature diagrams has also received increasing attention [3, 19]. But while the expressiveness and correctness support for feature diagrams has significantly increased, systematic approaches for deriving feature trees from requirement specifications have been few and far between.

Rather than a systematic process, the derivation of an initial feature tree from the provided requirement descriptions has remained something of a black art. The successful definition of a feature tree that accurately represents the information in the requirement specifications still depends heavily on the intuition and experience of the software architect. Assistance for this process has been proposed, but this mostly consists of guidelines and heuristics [13, 6]. Nonetheless, these approaches acknowledge the difficulty of this step as the vagueness in both requirement descriptions and the understanding of what exactly constitutes a feature severely hinders the systematic derivation of feature diagrams.

Generally, the initial feature tree structure is determined by performing a clustering on the requirements. Requirements that relate to the same concepts are grouped together in clusters. The clustering that results forms the basis for the features in the feature tree. However, clustering-based approaches (implicitly) expect requirement specifications to be clear, structured and unambiguous. Requirements can only be clustered accurately and effectively if they are accurate and free of conflicting and ambiguous information. In practice, however, the provided requirement specifications seldom exhibit these properties.

Ambiguous textual specifications can lead to different clusterings depending on how the requirement is interpreted. It can be said therefore, that such requirement specifications provide *insufficient* information for accurate clustering and feature tree derivation. Most times, this problem is "resolved" by making explicit assumptions on the meaning of the imperfection information, even when this can not be justified. As a result, the resulting feature trees are based on unjustifiable information and can have a significantly different structure than would have been the case otherwise.

Nonetheless, the resulting feature trees are used for decisions that are critical for the success of the SPL, such as the choice of core assets and variation mechanisms. And as an SPL has a significantly longer lifecycle than traditional software systems, the consequences of wrong decisions will also be felt longer. It is therefore vital that imperfect information in requirement specifications is identified and its influence is well understood. Rather than replacing imperfection with unjustifiable assumptions, the nature and severity of the imperfection should be modelled and considered during the definition of feature trees.

In the following section, we examine the impact of imperfect information for a clustering approach that derives feature trees from textual requirements documentation by clustering requirements based on their similarity. We identify where imperfect information can manifest itself and how it influences the effectiveness of the approach. In Section 4, we propose an approach for handling imperfection in textual requirements and representing it accordingly in feature trees.

3.2 Similarity-based Requirements Clustering

As indicated in the previous section, clustering of requirements is used as a basis for the identification of features that make up a feature tree. Naturally, the relevancy of

the resulting feature tree heavily depends on how requirements are clustered into features. In [2], it was argued that such clustering can be based on the measure of similarity between requirements, as similar requirements typically relate to the same concepts and therefore likely belong to the same feature. The continued work of [2], called Arborcraft, extends on this notion by defining an approach that clusters textual requirement specifications based on similarity of requirements.

3.2.1 Overview of the Approach

The Arborcraft approach clusters requirements together based on the similarity these requirements exhibit from their natural language descriptions. From the clustering that results, a feature tree is derived. In Figure 1, the phases of Arborcraft are depicted.

In stage I, the similarity of requirements expressed in natural language is determined using *latent semantic analysis* (LSA) [20]. Without going into detail, LSA considers texts to be similar if they share a significant amount of concepts. These concepts are determined according to the terms they include with respect to the terms in the total document space. As a result, for each pair of requirements a *similarity measure* is given, represented by a number between 0 (completely dissimilar) and 1 (identical). In the figure, this result is represented by the block *Requirements Similarity Results*.

Stage II uses the *Requirements Similarity Results* and a hierarchical clustering algorithm to cluster requirements that are semantically similar to form features. Small features are clustered with other features and requirements to form parent features. The similarity measure is therefore used to build up the hierarchy of the feature diagram.

Finally, in stage III Arborcraft provides the possibility to identify variability and crosscutting concerns in the requirements documents. Based on these results, the feature tree can be refactored by, for example, relocating requirements or introducing aspectual features.

Consider the similarity analysis result in Table 1. In this table, the similarity of four requirements has been determined, R_1, R_2, R_3 and R_4. After the clustering stage of Arborcraft, the feature tree of Figure 2 results. Requirements R_1, R_2 and R_3 are clustered together on the second level of the feature tree, as they are the most similar. As a result of the lower similarity, R_4 is clustered with the other requirements only on the highest level. Due to a lack of space, the refactorings of the final stage have been omitted.

3.2.2 Imperfect Information in Arborcraft

Arborcraft provides a comprehensive approach for extracting feature trees from textual requirements specifications. However, early experimentation has indicated that small, well-structured and clear documents produce considerably

Table 1. Requirement Similarity Values

	R_1	R_2	R_3	R_4
R_1	1	0.9	0.6	0.4
R_2	0.9	1	0.8	0.6
R_3	0.6	0.8	1	0.4
R_4	0.4	0.6	0.4	1

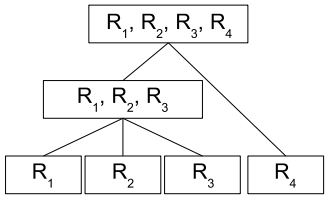

Figure 2. Feature Tree

better results than large, unstructured documents containing vagueness and ambiguity. This supports our claim in Section 3.1 that such insufficiency can severely hinder the effectiveness of clustering-based derivation of feature diagrams. The impact of imperfect information on the Arborcraft approach can be identified in the following areas:

Requirements clustering
Imperfect information influences the step of clustering requirements. The similarity of requirements is determined by evaluating the amount of concepts that are shared between them. But as indicated before, natural language naturally contains ambiguities and depending on which interpretation is compared to the other requirements, the results of the similarity analysis will differ. It is therefore vital that the similarity analysis is provided with clearly defined specifications if it is to provide accurate results.

Feature tree derivation from clusters
For the step from clusters to feature trees, there is no direct influence of imperfect information. However, this step uses the clustering result of the previous step and assumes that these results are accurate and reliable. Moreover, this stage aims to come up with a single feature tree, which is realistically speaking neither possible nor desirable given the presence of imperfect information. By having to commit to one particular feature tree, the software architect is forced to commit to the chosen interpretations for the identified imperfection in the clustering step.

Variability and crosscutting analysis
The final stage of Arborcraft searches for variability and crosscutting by analysing the requirements based on their clustering in the feature tree. This stage uses semantic anal-

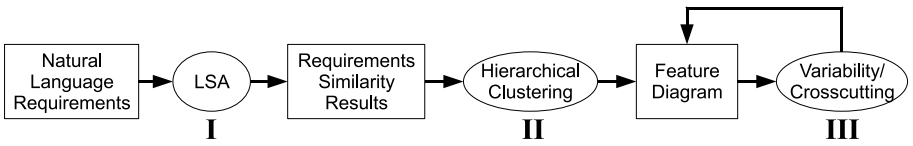

Figure 1. The stages of Arborcraft

ysis of the textual requirement descriptions, which means the imperfection in natural language naturally influences the effectiveness of the result. Again, unjustifiable assumptions can be required to come up with refactorings for the feature tree. Moreover, this stage does not ensure that for both the clustering and the proposed refactorings the same interpretations and assumptions have been considered.

4 Imperfection Support for Feature Tree Derivation

4.1 Introduction

We propose a coherent approach for handling imperfect information during the derivation of feature trees from textual requirements. The main element of the approach is the *fuzzy feature diagram*, an extension to traditional feature diagrams that can capture ambiguous requirements and describe how they can be part of multiple features simultaneously. This extension is then used to accommodate the influence of multiple interpretations for ambiguous information in the textual requirement documents. Our approach consists of three steps:

1. Model imperfection in textual requirements
2. Clustering of requirements including imperfection
3. Derivation of a *fuzzy feature diagram*

Our proposed approach focuses on *ambiguous* information, i.e. information that can be interpreted in multiple ways. The first two steps are aimed at handling the multiple interpretations of ambiguous informationthat is found in textual requirement documents. The third step uses *fuzzy feature trees* to describe the clustered ambiguous information, rather than attempting to describe this information using traditional feature diagrams. In the following sections, we describe these steps in more detail.

4.2 Modelling Ambiguous Requirements

The first extension is explicit modelling of ambiguous information in the textual requirement specifications. For all identified ambiguities, instead of considering a single interpretation, all relevant interpretations are described. Moreover, each interpretation is attributed with a *relevance value*, a number between zero and one that indicates the perceived relevance of the interpretation. The single imperfect requirement therefore is replaced by a *fuzzy set*[1] of interpretations.

Requirements that are defined using fuzzy sets of interpretations are called *fuzzy requirements* and were first proposed in [15]. The identification of the interpretations and the definition of their relevance values will be done in close cooperation with the stakeholders. In subsequent steps, all identified interpretations are included in the development process as normal requirements. Note that, naturally, ambiguities need to be identified before they can be modelled, but this goes beyond the scope of this work.

To illustrate this step, in Section 2.1 we indicated that the *high energy consumption* defined in the Smart Home requirements can mean "pre-defined by the habitant", but can also mean "a measurement result from the system". With the extension proposed above, this imperfect requirement is replaced with $\{p$/*"exceeds a pre-set energy consumption level", q/"when the system determines a high energy consumption level"*$\}$, where p and q are the respective relevancy values of the interpretations. The ambiguous statement is now refined to two explicit interpretations, which both can be considered in subsequent steps.

4.3 Clustering of Ambiguous Requirements

In Section 3.1, we have established that due to the use of natural language generally provides insufficient information to determine a single best clustering of requirements. With the concept of fuzzy requirements, this problem can now be resolved. By considering all interpretations as tra-

[1] Fuzzy sets allows elements to be a partial member of a set, which can be used to describe imperfect information. The partial membership of an element x in a fuzzy set is given by the membership value $\mu(x)$, where μ is a function that maps the universe of discourse to the interval $[0, 1]$. This value is the degree to which x is an element of the fuzzy set, where 1 means "completely a member" and 0 means "completely not a member". By considering membership degree during manipulations, the risk and impact of the modelled imperfect information can be assessed more accurately. Due to the lack of space a more elaborate introduction is omitted, but the interested reader is forwarded to [12].

ditional requirement, the clustering of requirements can be performed in much the same way as before. However as a result, different interpretations can end up in different clusters, even when they originated from the same fuzzy requirement. This essentially means that the fuzzy requirement has become a member of multiple clusters simultaneously, which is not possible with traditional clustering of requirements.

We therefore propose to use to group requirements into fuzzy clusters (fuzzy sets) instead of traditional, crisp clusters. Requirements can be part of multiple clusters at the same time and to differing degrees. We define the degree of membership of an interpretation in a fuzzy cluster to be the relevancy degree of that interpretation in the fuzzy requirement. A fuzzy clustering is then achieved by first clustering all crisp requirements and interpretations using a traditional clustering method. Then, in all the clusters that contain interpretations, these interpretation are replaced by the original imperfect requirement and they are tagged with the relevancy degree of the interpretation that was replaced, thus creating fuzzy sets.

Consider requirements R_1, R_2 and R_3, where the imperfect requirement R_3 is replaced with the fuzzy requirement $x/R_{3.1}$, $y/R_{3.2}$. A traditional clustering of crisp requirements and interpretations has resulted in the clusters R_1, $R_{3.1}$ and R_2, $R_{3.2}$. Here, the requirement R_3 has become part of two clusters due to two different interpretations. This clustering is transformed into a fuzzy clustering by replacing the interpretations in the clusters with the initial imperfect requirement: R_1, x/R_3 and R_2, y/R_3. As indicated, the membership values correspond to the relevancy degrees of the respective interpretation.

4.4 Fuzzy Feature Trees

Where traditional, crisp clustering leads to the definition of a feature diagram, with the proposed fuzzy clustering this is no longer possible. Therefore, our third proposed extension is the use of *fuzzy feature trees*. A traditional feature tree forces the software architect to precisely nail down variability and hierarchical structure for the software product line. A fuzzy feature tree does not expect this kind of precision.

In a fuzzy feature tree, requirements are clustered into features to a certain *degree*, which means that features in a fuzzy tree are *fuzzy clusters*. Moreover, a fuzzy feature tree imposes no restrictions on features or relationships between them, even when this would be invalid in traditional feature trees. For instance, multiple features can contain the same requirements and it is possible for a feature to have multiple parents.

In Figure 3, a fuzzy feature tree is depicted that is a modification of the diagram in Figure 2. In this picture, the requirement R_3 initially was identified as being ambiguous and two interpretations were been identified, say $R_{3.1}$ and $R_{3.2}$, with a respective relevancy degree of x and y. In the fuzzy clustering that results, there are two overlapping requirement clusters, $\{R_1, x/R_3\}$ and $\{R_2, y/R_3\}$, in which R_3 has differing degrees of membership. This fuzzy feature tree now describes the best clustering that could be achieved in the feature tree based on the ambiguous information that was provided. Note that as a result of including these clusters, other features have multiple parent features.

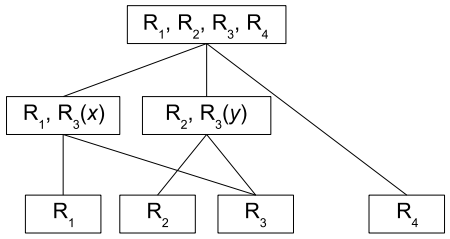

Figure 3. A Fuzzy Feature Diagram

As a fuzzy feature diagram is a generalization of feature diagrams, it essentially describes multiple feature diagrams simultaneously. By removing elements, such as features with overlapping requirements, a fuzzy feature diagram can be transformed into a traditional feature diagram. Depending on how the membership degrees are used during this *defuzzification* step, a number of alternative feature diagrams can be proposed to the software architect. Ideally, however, a fuzzy feature diagram is maintained throughout SPL development so more detailed information can arrive at a later stage to resolve the imperfection. Also, by extending approaches that operate on traditional feature diagrams, such as the variability/crosscutting analysis of Arborcraft, they can assess the risks that come with specific decisions by considering the imperfection described in fuzzy feature diagrams.

4.5 Application to Arborcraft

When this proposal is applied to the Arborcraft approach, this results in the picture of Figure 4. The new elements in the picture when compared to Figure 1 are indicated in grey. In step (I), first the ambiguous statements are modelled as fuzzy requirements. The resulting requirements are then clustered with the standard techniques from Arborcraft (steps II and III). The feature tree that results is restructured to a fuzzy feature tree in step IV. If required, in step VI the fuzzy feature tree is defuzzified to a number of crisp feature trees. The software architect can then select the most appropriate alternative. The refactorings of Arborcraft with respect to variability and cross-cutting (step V) can be applied to both fuzzy feature tree as well as the

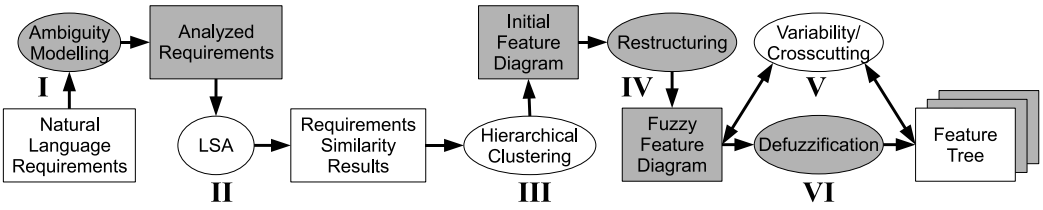

Figure 4. Arborcraft with support for ambiguous requirements

defuzzified, crisp feature trees. Naturally, to handle fuzzy feature trees the refactoring mechanisms would need to be enhanced.

5 Discussion

5.1 Discussion of the Approach

In the previous section, we have sketched an approach for coping with imperfect information during the definition of feature diagrams. Nonetheless, this approach leaves a number of questions unanswered. In this section we examine some of these questions.

5.1.1 Maintaining Imperfect Information during SPL Development

One of the key properties of the proposed approach is the possibility to maintain imperfection during multiple stages of SPL development. With this property, software architects do not need to make unjustifiable assumptions which can hinder development at later stages. However, the inclusion of alternative interpretations creates extra overhead and introduces new model elements that need to be considered.

However, the benefit of maintaining imperfect information during SPL development is two-fold. First, the presence of imperfect information poses a danger to effective software development. If the SPL architecture is built based on imperfect information, it is likely that at later stages some sort of redesign will be required. By including alternative interpretations and considering the influence of imperfection, the design becomes more resilient to such changes. In essence, design becomes a defensive activity as a number of likely scenarios are already considered. Moreover, the modelled imperfection offers the opportunity to examine risks that come with design decisions.

The second benefit lies in the fact that insufficient information might not stay insufficient indefinitely. A traditional approach forces the architect to make explicit assumptions that at a later stage can turn out to be correct or false. With the support for imperfect information, the architect does not have to commit to a single assumption. Rather, the design will consider and support a number of alternative assumptions throughout the design stages. This creates a larger window of opportunity for the imperfection to be resolved and it will require considerably less refactoring if the design already contains what turns out to be the correct information.

Nonetheless, the concept of maintaining imperfect information introduces a trade-off of effort during the development process. By including all kinds of potential interpretations and design alternatives, software architects can be faced with a considerable increase in effort. It is therefore vital that the software architect can control and manage the extra effort that is created. This can be achieved by, for instance, focusing only on the most relevant interpretations and removing excess information whenever possible. To perform this kind of operations, support for (partial) removal of imperfect information from design is required.

5.1.2 Identifying Imperfect Information

One of the most important problems to be solved is the identification of imperfect information in textual requirement specifications. More specifically, the sufficiency of available information needs to be determined for the design step in which it will be used. As the reason for information to be insufficient is defined by this context, imperfection not only needs to be identified, but also the nature and consequences of its insufficiency.

NLP techniques can assist in identifying imperfect information in textual requirement specifications. Specific approaches have been proposed for the identification and management of imperfection in requirement specifications, such as [9, 11]. With the addition of specific lexicons and vocabulary for typical imperfection in software specifications, semantic analysis approaches can be extended to aid in this goal.

Ideally, at the moment imperfection is identified the stakeholders are consulted and additional information is acquired to resolve the situation. However, as we identified in Section 2, this is not always possible due to a lack of knowledge or insight. In this case, stakeholders can be consulted to describe the actual imperfection. For example, in our extension for requirements clustering we propose alternative

interpretations to be given for ambiguous statements. While NLP cannot derive this kind of information, the automatic identification of potential ambiguities provides a valuable first step.

5.1.3 Removing Imperfection Models from SPL Development

As mentioned in the previous section, it can be desired at given points during development to remove imperfection models. This can for instance be the case when subsequent stages no longer support imperfect information or when the additional effort for maintaining it are no longer feasible. This warrants the question how these models can be removed from the design when the need arises.

As the proposed extensions, such as the fuzzy feature diagram, essentially describe a number of traditional models using a single generalized model, the removal of imperfection corresponds to determining the best traditional model from the generalized model. The numerical information (such as the membership values of clustered requirements) can be used for this purpose. By identifying the traditional, crisp model that best fits the numerical information and the restrictions the model should adhere to, the imperfection can be removed. This is in essence an optimization problem where all traditional models that can be derived from a generalized model are ranked and the best one selected.

5.1.4 How do Imperfection Models affect existing Approaches?

The introduction of fuzzy feature trees in this paper, and imperfection models in general, has a direct impact on all approaches to operate directly on traditional feature trees. As these approaches do not consider the typical properties of fuzzy feature trees, they cannot be applied in the same manner during SPL development.

This can be resolved in two possible ways: first of all, the imperfection can be removed at the moment a design activity is to be undertaken that does not support imperfection models. This can be done by the earlier mentioned defuzzification techniques and the selection of one of the resulting alternative feature trees. However, as identified in Section 5.1.1, it is desirable to maintain unresolved ambiguity as long as possible. Therefore, the second way is to extend these approaches to support fuzzy feature trees. Naturally, the effort required for realising such support must be aligned with the benefits during development.

5.2 A Research Agenda for Supporting Imperfect Information

In Section 2, we have identified that imperfect information in textual requirement specifications can severely hinder variability/commonality analysis. And with the proposal of fuzzy similarity values, overlapping clustering and fuzzy feature trees, we have sketched a first direction for imperfection support in Arborcraft. In this section, we define a research agenda with key problems of imperfect information in feature tree derivation and the general direction on how these problems should be addressed.

A formalized procedure for deriving feature diagrams
One of the most important problems is a complete understanding of the process that turns textual requirement specifications into actual feature diagrams. At the moment, this step still relies considerably on the intuition, knowledge and experience of the software architects. To understand how imperfect information influences the decisions that define the feature tree, they need to be understood in an unambiguous and uniform manner.

With a formal model of the derivation of feature trees, the way information is used will become well-understood. Moreover, it becomes possible to analyse the problems that imperfect information causes during this process. NLP-baed approaches such as Arborcraft actually define a (semi-)formal approach for going from textual requirement specifications to feature diagrams. The proposed approach in this article utilizes this by extending its capabilities to support imperfection.

A taxonomy of types of imperfect information
A second important problem to be solved is understanding the nature of the imperfection that can occur in requirement specifications. Many different types of imperfection exist, such as conflict, ambiguity and vagueness, and each of them influences the usability of the information in a different manner. By having a standardized categorization of imperfection types, potentially hazardous elements in specifications can be identified and its impact assessed.

In particular for NLP-based approaches, the definition of an imperfection taxonomy would be very useful. As many NLP approaches utilize a semantic lexicon (e.g. EA-Miner uses variability lexicons), an imperfection lexicon based on this taxonomy can aid in the automatic identification of imperfect information. Moreover, a well-defined terminology will aid in communicating about imperfect information and creating awareness about this phenomenon.

Modelling and reasoning support for imperfection
The core element of any approach for dealing with imperfect information is the ability to model the imperfection and reason with this model in subsequent steps. By capturing the identified imperfection with models such as probability theory and fuzzy set theory, the nature and risk of such information can be quantified. By extending the subsequent design steps to consider these models, the influence of the

imperfection can be considered during decision making activities.

Resolving this problem requires the first two problems of this research agenda to be resolved. Formalizations need to be extended with techniques from probability and fuzzy set theory to support reasoning with models for imperfection. Moreover, only when the type and nature of the imperfection is known is it possible to identify the appropriate model to quantify it accurately.

Removal of imperfection models from development
The final problem is the systematic removal of imperfection models from development. The first three research problems are targeted at introducing models for imperfection. Conversely, at particular stages and situations it can be required to remove these models because of new insights or because of a lack of budget to maintain all the extra information.

The removal requires an approach that can determine which elements of the imperfection models are no longer relevant. Moreover, it can also require the selection of the most relevant interpretations that have been included. As identified in Section 5.1.3, such removal activities are in essence optimization problems so the body of knowledge in optimization theory offers a promising starting point.

6 Related Work

This paper focuses on the problem of supporting imperfect information in feature derivation and relates to the areas of requirements engineering, SPL development and imperfection modelling and support for feature diagrams. In this section, we give a short overview of related work in these fields.

Extensions to feature diagrams based for imperfect information have been proposed before. In [18], features in feature diagrams are attributed with fuzzy weights that describe the preference of specific customers. The weights subsequently are used by a fuzzy logic-based expert system that can determine typical product configurations for specific customer profiles. The approach described in [16] extends on this approach by introducing fuzzy probabilistic descriptions of market conditions that can influence the expert system. In [7], soft constraints are introduced that specify the conditional probability that feature will be part of a configuration when another feature already is part of it. This information can then be used to identify product parts that must be supported by a particular platform or to understand how these products utilize a particular platform.

These approaches capture a particular type of imperfection when dealing with feature diagrams. However, the goal of these approaches is distinctly different from the problem we have identified in this article. The imperfection that these approaches support originates from an uncertain preference of customers for particular configurations. The imperfection support in our approach addresses unresolvable imperfection in requirement specifications. Moreover, our approach is aimed at supporting the design steps that lead up to an actual feature diagram.

In this work, a fuzzy extension is proposed for deriving feature diagrams from textual requirement specifications. In [4], an approach is proposed that derives feature trees by performing clustering of textual requirements definitions. Our approach is a generic extension that can be integrated in this and other similar approaches, like Arborcraft. Imperfection support for feature tree extraction, to the best of our knowledge, has not been proposed before.

The influence of imperfect information on feature diagrams is well recognized [18, 16]. Kamsties identifies in [9] that ambiguity in requirement specifications needs to be understood before any subsequent design can be undertaken. With support for feature tree definition being largely heuristic [13, 6], systematic support for imperfect information is all but completely absent. Our approach defines a first step by proposing models and design steps to support these models.

7 Conclusions

In this article, we have taken a first step towards supporting imperfect information in the definition of feature trees. We have identified that the effectiveness of approaches that derive feature trees from textual requirement specifications can be severely compromised by imperfect information. And as imperfection is naturally part of requirements that are specified in natural language, its influence on such approaches can not be ignored. We established that the main cause is that most approaches require perfect information for the definition of an accurate feature trees. As a result, any imperfections need to be resolved even when specific assumptions can not (yet) be justified.

To illustrate the impact of imperfect information, we explored approaches that derive feature trees from requirements specifications by clustering related requirements. As the clustering mechanisms used in these approaches do not explicitly consider imperfection, the clustering that results is influenced by vagueness and ambiguity in natural language. Nonetheless, subsequent stages use the feature tree that results as input while assuming these results to be accurate.

To address these problems, we have proposed an approach that captures ambiguity in requirement descriptions using techniques from fuzzy set theory. In particular, we proposed the consideration of multiple interpretations when ambiguity can not be resolved, fuzzy clusters to extend the clustering of requirements and a fuzzy extension for feature

diagrams that captures the imperfection. These proposals have been generalized to form a research agenda for imperfection support in feature diagram derivation.

As future work, we want to formalise the steps for the derivation of fuzzy feature trees from ambiguous requirements. Also, we want to integrate with existing approaches that can identify imperfect information using natural language processing and we want to extend the support for refactorings of feature diagrams. When this is completed, we plan to implement the approach as part of Arborcraft and evaluate it with an industrial case study.

8 Acknowledgements

This work is sponsored by the European Union as part of the AMPLE project (IST-33710) and the DISCS project (IEF-221280).

References

[1] M. Alvarez, U. Kulesza, N. Weston, J. Araujo, V. Amaral, A. Moreira, A. Rashid, and M. Jaeger. A metamodel for aspectual requirements modelling and composition. Technical report, AMPLE project deliverable D1.3, 2008.

[2] 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 '08: Proceedings of the 2008 12th International Software Product Line Conference*, pages 67–76, Washington, DC, USA, 2008. IEEE Computer Society.

[3] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, pages 491–503. Springer-Berlin, 2005.

[4] K. Chen, W. Zhang, H. Zhao, and H. Mei. An approach to constructing feature models based on requirements clustering. *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 31–40, 2005.

[5] A. Classen, P. Heymans, and P.-Y. Schobbens. What's in a feature : A requirements engineering perspective. In *LNCS 4961*, pages 16–30. Springer-Verlag, 2008.

[6] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[7] K. Czarnecki, S. She, and A. Wasowski. Sample spaces and feature models: There and back again. *Software Product Line Conference, International*, 0:22–31, 2008.

[8] M. L. Griss, J. Favaro, and M. d' Alessandro. Integrating feature modeling with the rseb. In *ICSR '98: Proceedings of the 5th International Conference on Software Reuse*, page 76, Washington, DC, USA, 1998. IEEE Computer Society.

[9] E. Kamsties. Understanding ambiguity in requirements engineering. In A. Aurums and C. Wohlin, editors, *Engineering and Managing Software Requirements*, pages 245–266. Springer-Verlag, 2005.

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

[11] N. Kiyavitskaya, N. Zeni, L. Mich, and D. M. Berry. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. *Requirements Engineering*, 13(3):207–239, 2008.

[12] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic, Theory and Applications*. Prentice Hall, 1995. Standard Reference for Fuzzy Sets and Fuzzy Logic ISBN: 0-13-101171-5.

[13] K. Lee, K. C. Kang, and J. Lee. Concepts and guidelines of feature modeling for product line software engineering. In *ICSR-7: Proceedings of the 7th International Conference on Software Reuse*, pages 62–77, London, UK, 2002. Springer-Verlag.

[14] L. Mich and P. N. Inverardi. Requirements analysis using linguistic tools: Results of an on-line survey. In *11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, pages 323–328, 2003.

[15] J. Noppen, P. van den Broek, and M. Aksit. Imperfect requirements in software development. In *Requirements Engineering: Foundation for Software Quality (REFSQ) 2007*, number 4542 in LNCS, pages 247–261. Springer-Verlag, 2007.

[16] A. Pieczynski, S. Robak, and A. Walaszek-Babiszewska. Features with fuzzy probability. *Engineering of Computer-Based Systems, IEEE International Conference on the*, 0:323, 2004.

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

[18] S. Robak and A. Pieczynski. Application of fuzzy weighted feature diagrams to model variability in software families. *Artificial Intelligence and Soft Computing*, LNCS 3070/2004:370–375, 2004.

[19] P.-Y. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51:456–479, 2007.

[20] A. Stone and P. Sawyer. Identifying tacit knowledge-based requirements. *IEE Proceedings–Software*, 153(6):211–218, 2006.