Ambiguity Defects as Variation Points in Requirements

Alessandro Fantechi
Dipartimento di Ingegneria dell'Informazione
Università di Firenze
Firenze, Italy
alessandro.fantechi@unifi.it

Stefania Gnesi
Istituto di Scienza e Tecnologie dell'Informazione "A.Faedo"
Consiglio Nazionale delle Ricerche, ISTI-CNR
Pisa, Italy
stefania.gnesi@isti.cnr.it

Laura Semini
Dipartimento di Informatica
Università di Pisa
Pisa, Italy
semini@di.unipi.it

ABSTRACT

Software requirements are generally expressed in Natural Language. NL is intrinsically ambiguous, and this is seen as a possible source of problems in the later interpretation of requirements. However, ambiguity or under-specification at requirements level can in some cases give an indication of possible variability, either in design choice, in implementation choices or configurability. Taking into account the results of previous analyses conducted on different requirements documents with NL analysis tools, we attempt a first classification of the forms of ambiguity that indicate variation points, and we indicate an approach to achieve automated support to variability elicitation.

CCS Concepts

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

Keywords

Ambiguity in requirements, variability in software product lines.

1. INTRODUCTION

In the early phase of software and system development, requirements are defined. This process aims to obtain a description of functional requirements (what the program should do) and non-functional requirements (how the software will do it). Even though this process is to some extent systematic, the identification of requirements is usually intuitive and informal, and the requirements are often expressed through natural language (NL) sentences. However NL is intrinsically ambiguous, and this is seen as a possible source of problems in the later interpretation of requirements. Nevertheless NL is still the most common way to express software requirements.

VaMoS '17, February 01-03, 2017, Eindhoven, Netherlands

DOI: http://dx.doi.org/10.1145/3023956.3023964

The analysis of software requirements with respect to problems related to their ambiguity, inconsistency and lack of correctness has been extensively studied in recent years [11]. A purpose of this analysis is to evaluate the quality of requirements with respect to their expressiveness that may be captured looking at characteristics like:

- **Unambiguity**: the capability of each requirement to have a unique interpretation.
- **Specification Completion**: the capability of each requirement to uniquely identify its object or subject.
- **Understandability**: the capability of each requirement to be fully understood when used for developing software.

In order to capture defects due to ambiguity that may cause lack of expressiveness, some syntactic or structural aspects of the requirements have been studied in NL literature [6, 7] and are listed below:

- **Vagueness**: the sentence contains items having a no uniquely quantifiable meaning.
  Vagueness-revealing words: adequate, bad, clear, close, easy, efficient, far, fast, good, in front, near, recent, significant, slow, strong, suitable, useful, ...
- **Optionality**: the sentence contains an optional part (i.e. a part that can be considered or not).
  Optionality-revealing words: possibly, eventually, in case, if possible, if appropriate, if needed, ...
- **Multiplicity**: The sentence has more than one main verb or more than one direct or indirect complement that describes the sentence.

  Multiplicity-revealing words: and, and/or, or, ...

Tools have been also developed to support the analysis of NL requirement specification to enlighten potential defects due to the above aspects [13, 1, 14]. However, ambiguity or under-specification at requirement level can in some cases give an indication of possible variability, either in design choice, in implementation choices or configurability. This paper is focused on those cases in which ambiguity in requirements is particularly due to the need to postpone choices for later decisions in the implementation of the system. Ambiguity becomes hence a means to enlighten possible variation points in an early phase of software and system

development. In this paper we investigate this possibility by analysing the outcomes of automated ambiguity detection applied to some set of requirements, and we outline a process to elicit variability starting from system or software requirement documents.

Structure of the paper. We start by describing some related work in Section 2. Some background details are given in Section 3, and the relation between ambiguity and variability is discussed in Section 4. In Section 5 we sketch a process for variability elicitation according to the introduced principles, and we give hints towards automatic support for such a process. Section 6 and Section 7 discuss, respectively, the application of the variability elicitation to a real world document of requirements, conclusion and future work.

2. RELATED WORK

In [3] a systematic literature review of the state-of-the-art approaches to feature extraction from NL requirements for reuse in SPLE has been presented; this review reports on a mixture of automated and semi-automated feature clustering approaches, from data mining and information retrieval, that have been used to group common features.

In [10] the authors suggest to employ a natural language processing approach based on contrastive analysis to identify commonalities and variabilities from the textual documents describing a specific product in the railway industry (Communications-Based Train Control (CBTC) systems) in order to derive a global CBTC model, represented as a feature diagram from which specific product requirements for novel CBTC systems can be derived. The proposed method takes the brochures of different vendors as input, and identifies the linguistic expressions in the documents that can be considered as terms. In this context, a term is defined as a conceptually independent expression. The domain-specific terms that are common among all the brochures are considered as commonality candidates. On the other hand, those domain-specific terms that appear solely in a subset of the brochures are considered as variability candidates.

Another interesting proposal is in [4] where techniques capable of synthesizing feature attributes and relations among attributes have been proposed. In particular, the authors introduced an algorithmic and parametrizable approach for computing a legal and appropriate hierarchy of features, including feature groups, typed feature attributes, domain values and relations among these attributes starting from real-world examples.

A different approach applies product comparison matrices to identify variability patterns like optionality, multiplity, and vaguenes in tabular data [18, 5].

Finally, in [16] a different technique to analyze variability of behaviors as described in functional requirements has been presented. The approach, called semantic and ontological variability analysis (SOVA), uses ontological and semantic considerations to automatically analyze differences between initial states (preconditions), external events (triggers) that act on the system, and final states (post-conditions) of behaviors. The approach generates feature diagrams to model variability.

We notice however that the cited approaches are aimed at feature elicitation starting from a set of requirement documents (or other technical documentation, such as brochures), each referring to a possible variant or product, by making a comparative analysis in order to figure out common parts, assuming that the parts not in common constitute the variability.

Our idea is instead that looking at a single requirement document, the ambiguity that is present in it can be considered not as a defect any more, but as a placeholder for different choices, indicating a range of different products; hence ambiguity is used to identify possible variation points. The positive role of ambiguity, when related to the need to provide a concise description of the requirements, which abstracts from irrelevant details, is discussed also in [12]: that analysis of the role of ambiguity in requirements shows how it is intimately linked to two phenomena, abstraction and absence of information, which in our view become the indicators of a possible variability.

For this purpose, a careful identification of ambiguous requirements is needed, hence we resort to a NLP based analysis, by adopting tools that have been defined for ambiguity detection in NL requirements, in order to give a preliminary classification of the ambiguity forms that can indicate variability, leaving to further work a more systematic classification study.

3. BACKGROUND

Requirements are an abstract description of the system needs that is inherently open to different interpretations. This openness is emphasized by the use of natural language (NL) which is intrinsically ambiguous, even though it is commonly used to express requirements. Indeed, NL is the most widely used communication code, since it easily supports the exchange of knowledge among different stakeholders with heterogeneous backgrounds and skills. As the requirements process progresses, requirements are expected to be sufficiently clear to be interpreted in a univocal way by the stakeholders who deal with them [11].

A solution found within the RE community is to employ NLP tools that make the editors aware of the ambiguity in their requirements [13, 19]. Ambiguities normally cause inconsistencies between the expectation of the customer and the product developed, and possibly lead to undesirable reworks on the artifacts. However, ambiguity can also be used as a way to capture variability aspects to be solved later in the software development.

3.1 QuARS

QuARS (Quality Analyzer for Requirements Specifications) was introduced as an automatic analyzer of requirement documents [13]. **QuARS** performs an initial parsing of NL requirements for automatic detection of potential linguistic defects that can determine ambiguity problems impacting the following development stages. **QuARS** performs a linguistic analysis of a requirements document in plain text format and points out the sentences that are defective according to the expressiveness quality model described in [7]. The defect identification process is split in two parts: (i) the "lexical analysis" capturing *optionality, subjectivity, vagueness and weakness* defects, by identifying candidate defective words that are identified into a corresponding set of dictionaries; and (ii) the "syntactical analysis" capturing *implicitness, multiplicity and under-specification* defects. In the same way, detected defects may however be *false defects*. This may occur mainly for three reasons: (i) a correct usage of a candidate defective word, (ii) a usage of a candidate de-

fective wording which is not usually considered a defect in the specific system or domain, and (iii) a possible source of ambiguity inserted on purpose to give more freedom to implementors. For this reason, a false positive masking feature is also provided by the tool.

Other functionalities are offered by **QuARS** like requirements clustering, metrics derivation for evaluating the quality of NL requirements and view derivation, to identify and collect together those requirements belonging to given functional and non functional characteristics. We do not present them in detains here since they are not related to the aim of this paper.

3.2 Feature Models

A feature model is a compact representation of all the products of the Software Product Line (SPL) in terms of "features". Feature models are visually represented by means of feature diagrams, introduced in [15] as a graphical formalism, *and/or* hierarchy of features, to describe in a uniform way a variety of different possible implementations of a system. The variability depends on which features are included and which ones are not. The features are represented as the nodes of a tree, with the product family being the root. Features come in several flavours, (Figure 1 illustrates the graphical constructs):

- **Optional** features may be present in a system only if their parent is present;
- **Mandatory** features are present in a system if and only if their parent is present;
- **Alternative** features are a set of features among which one and only one is present in a system, provided their parent is present.
- **Or** features are a set of features among which at least one is present in a system, provided their parent is present.

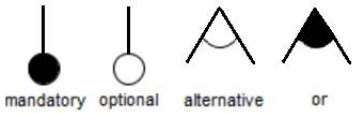

Figure 1: Basic constructs of a feature model

4. AMBIGUITY VERSUS VARIABILITY

The novelty we introduce here is that the ambiguity defects that are found in a single requirement document may be due to, intentional or unintentional, references made in the requirements to issues that may be solved in different ways, possibly envisioning a family of different products rather than a slngle product. We therefore use the analysis ability of QuARS to elicit the potential variability hidden in a requirement document.

4.1 Running example

To illustrate the contribution of this paper we use a simple running example, namely a family of (simplified) coffee machines, for which we consider the following requirements (which simplify those given for a similar running example in [9]):

1. R1: After inserting a suitable coin, the user shall choose a beverage and whether to add sugar.
2. R2: The offered beverages are Coffee and/or Cappuccino and/or Tea.
3. R3: Coffee shall be offered by all products of the family.
4. R4: A ring tone shall be possibly rung after delivering a beverage.
5. R5: After the beverage is taken, the machine returns idle.

Analyzing this sets of requirements according to the classes of ambiguity identified in the Introduction we can notice that R1 is vague (*suitable* coin), R1 and R2 contain a multiplicity (*and/or*) and R4 is optional (*possibly*).

4.2 Variability due to Vagueness

Vagueness occurs whenever a requirement admits borderline cases, e.g., cases in which the truth value of the sentence cannot be decided.

Example 1. Vag1: After inserting a suitable coin, the user shall choose a beverage and whether to add sugar.

"Suitable" is a vague word, that indicates a variability about the different coins that can be used in the vending machine. In this case it refers to the currencies used in the countries in which the products are marketed. For example, the targeted countries could be Europe and Canada, with the related currencies.

A vague word abstracts from a set of instances, that are considered as the "suitable" (or adequate, bad, etc.) ones, and the process of requirement refinement will make these instances explicit. In the general case, these instances are alternative one to the other: either insert a Euro or a Canadian Dollar, as illustrated in Figure 2.

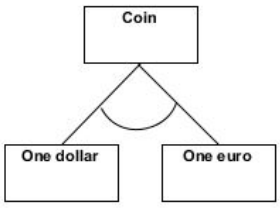

Figure 2: Feature model for resolved vagueness

4.3 Variability due to Optionality

Optionality occurs when a requirement contains an optional part, i.e. a part that can or cannot be considered.

Example 2. Opt1: A ring tone shall be possibly rung after delivering a beverage.

Optionality of a requirement is naturally expressed in a feature diagram with an optional feature. In the example, it has been recognized that the ring tone is an optional feature of a delivery unit, as shown in Figure 3.

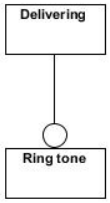

Figure 3: Feature model for optionality

4.4 Variability due to Multiplicity

Another indicator of possible variability is the usage of an and/or-construct among two or more different alternatives. In particular, we have seen that the multiplicity-revealing words can be *and, and/or, or*, but these have different meanings, and this normally requires some expert judgment in order to define the right variability description.

4.4.1 Multiplicity due to or-constructs

Again, a requirement with an or construct is not precise, since it leaves several possibilities open, and hence different products compliant to such requirement can choose different alternatives.

A disjunction may come in different flavors:

- implicitly exclusive or: the alternatives are mutually exclusive. In this case, the corresponding features are declared as "alternative", with a partial diagram as the one used for vagueness.
- weak or: all the alternatives are optional.
- or: at least one of the alternatives should be present, but it is irrelevant which one.
- and/or: as above, at least one of the alternatives should be present, but in this case there is an implicit assumption of which of the alternatives should be present in a product. This can be resolved by looking at some other requirement.

Example 3. Mul1: The offered beverages are Coffee and/or Cappuccino and/or Tea.

In this case, we can note that R3 tells that Coffee should be mandatory, while the other alternatives are optional (see Fig. 4).

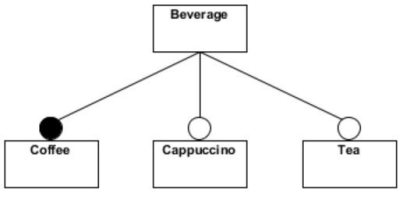

Figure 4: Feature model for multiplicity: Mul1

4.4.2 Multiplicity due to and-constructs

A conjunction, although recognized as a multiplicity, simply shows that all alternatives are mandatory: hence it is not really a variability indication, although it can be modelled with a feature diagram as well (see Fig. 5).

Example 4. Mul2: After inserting a suitable coin, the user shall choose a beverage and whether to add sugar.

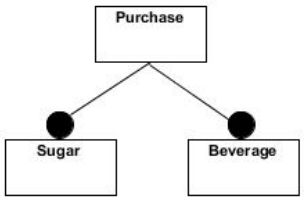

Figure 5: Feature model for multiplicity: Mul2

5. TOWARDS A PROCESS TO AUTOMATIZE VARIABILITY ELICITATION

The presented ideas have inspired a variability elicitation process to be applied on requirement documents, that can, in a future work, be partially automated. The process, after a first requirement elicitation phase and the consequent writing of the requirements in NL, follows the steps detailed in the activity diagram of Fig. 6, and described in the following:

1. Run one of the tools that have been defined for ambiguity detection in NL requirements. We have used QuARS for this purpose.
2. Acquire expert judgment of the outcome of the tool to distinguish among:
   (a) False positives. A false positive is a sentence recognized as defective by the tool but considered acceptable by expert judgment. Removal of false positives from the blacklist is decided by an expert user and handled by the masking features provided by QuARS.
   (b) Actual ambiguity defects that require a clarification and therefore a cycle back to requirement correction. The defects can be, for instance, words or sentences that can be understood differently by different readers: a *recent* book, besides being vague, may have a subjective interpretation (think of a computer scientist vs. a historian). In this case, the ambiguous requirement must be substituted by a more precise one.
   (c) After resolving the actual ambiguities, and removing the false positives, the intentional (or sometimes unintentional, but positive) ambiguities, due to the need of abstracting from finer grained details, are left. These are candidate variability indicators, which need to be categorized in different defect types.

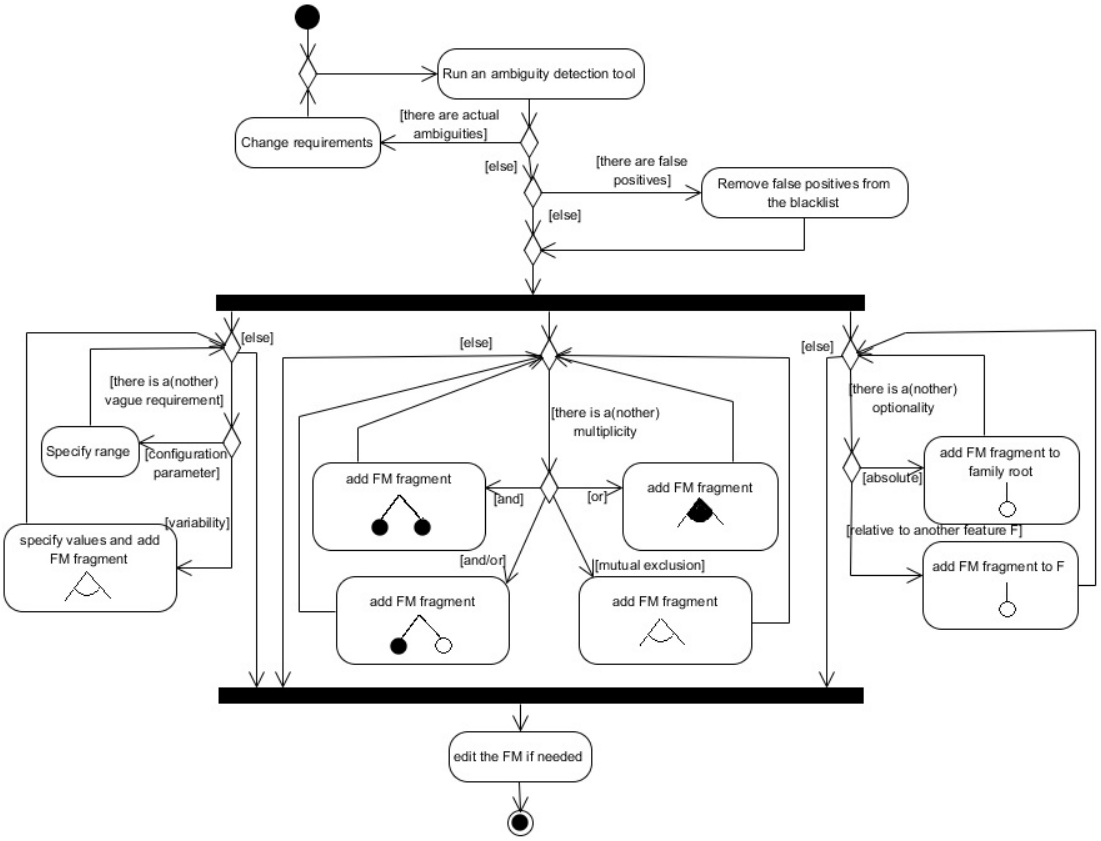

Figure 6: Activity diagram of the variability elicitation process

3. Based on the different defect types, and according to the description done in Sections 4.2 to 4.4, a feature model capturing the identified variation points is built, possibly requiring input by the expert.

The process might be automated by an integrated tool supporting the various steps according to the flow given in Fig. 6; in particular the third step of the process could be supported by an interactive tool as in the following:

For each vague requirement, the tool asks the user for the set of intended values, e.g. $\{Euro, Dollar\}$ in our case study, and builds a feature model fragment. As a default choice, the fragment is built having as root the noun over which the vagueness-revealing word predicates, coin in our case, and the determined values as sub-features.

Sometimes, this variability case can be better resolved by introducing a numeric configuration parameter instead, without resorting to a description in a feature model. Take for example a controller of a heating plant, for which the requirements ask for a *suitable* room temperature. This can be recognized as a variability issue depending on the final deployment of the plant, but the expert judgment can make the requirement more precise by considering a configurable "suitable" temperature being between 18 and 22 Celsius. The tool should help the user in specifying the range values for the identified configurable parameters.

For multiplicity, the tool can ask the user whether it is: a pure *and*, i.e. all alternatives are compulsory; a pure *or*, with all alternatives optional; an *and/or*, where some variants are compulsory, other optional, and build the feature model accordingly. (Hidden) constrains of mutual exclusion can be highlighted too, and added to the model.

To determine the root of the multiplicity fragment, the tool can help the user with a default choice, proposing him the subject of the sentence.

When the tool encounters an optionality word, it can prompt the user if:

- the feature is actually optional, i.e. to be included in only some configurations of the system at hand. In this case, the feature model under construction is enriched with this optional fragment, with a branch from the family root.
- the feature is an optional sub-feature of another feature. In the case study, the ring tone is an optional sub-feature of the delivery unit.

Systematically following the proposed process, we have built the full feature model for our case study (Fig. 7), where all the extracted fragments are combined as sub-features of the root feature representing the coffee machine.

A tool automating the given process should in the end give the possibility to the user to edit the feature model in

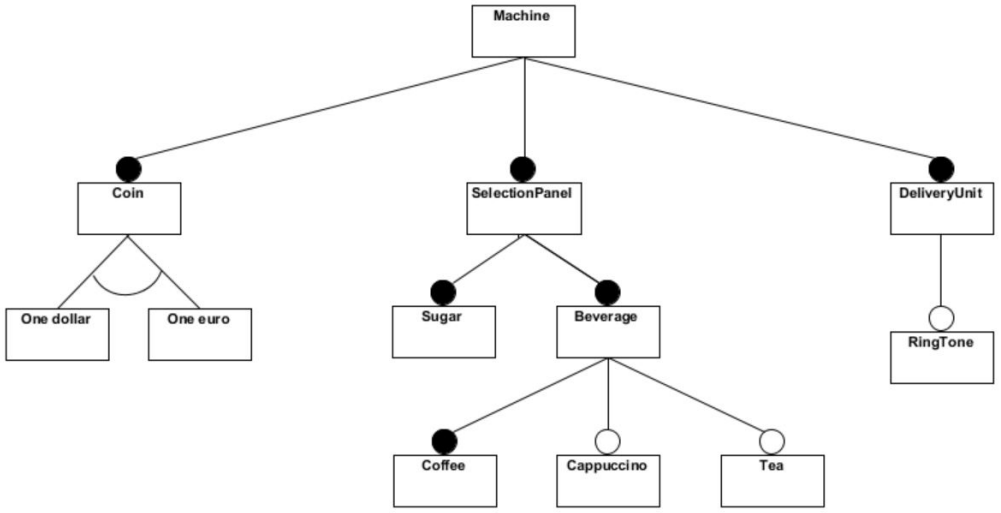

Figure 7: Composed feature model for the coffee machine family

order to refine the hierarchical structure of the features and to add possible crosscut constraints.

6. REAL WORLD EXPERIMENTS

The above variability identification approach should be validated on a significant set of real requirements documents. As an initial effort in this direction, we have considered the requirements from the real datasets analyzed in [17, 8].
This preliminary analysis has confirmed the importance of the expert judgment needed to distinguish actually defective requirements from false positives firsts, and then from variability indicators.
Just to report a few numbers on this preliminary analysis, out of the 1866 requirements of one of the data sets analyzed in [17], 116 have been classified as vague, with many false positives, and among them, 10 vagueness defects have been classified as variability indicators.

In the following we report a few other examples taken from the cited datasets, in order to show how the variability indicators actually appear in real requirements and how expert judgment is needed to resolve variability.

Example 5. Vag2: An appropriate device shall indicate the driver or the train crew that all the doors are closed.

This requirement is pointed out as defective because it contains the vague term "appropriate" and because of a disjunction. We first address vagueness. As for the example in Section 4.2, this requirement must undergo an analysis process to understand which devices can be considered as appropriate. An answer can be found in the documentation or be the outcome of an interview to the stakeholders. Possibly, the identified devices have different costs and impact on the project, calling for a negotiation process to make a choice. Another approach to the problem is to leave the choice open and introduce a variation point. The enriched expressive power of *attributed feature models* [2], where quantitative, non-functional characteristics of features are captured by attributes that are assigned to each feature, permits to document the cost of each alternative in the model, for a successive decision-making process.

Example 6. Mul3: An appropriate device shall indicate the driver or the train crew that all the doors are closed.

As pointed out in section 4.4.1 an or-construct needs an expert judgment to clarify its nature. In this case, it was made clear that it is a classical "or": at least one, no matters which. This requirement can be modeled with the corresponding construct in the feature model.

Example 7. Vag3: Evaluated events shall be saved in event lists in order to be available for a certain period of time before being saved in some other medium.

Also in this case the requirement is pointed out as defective because it contains the vague term "certain". Differently from the example Vag1, in which two particular currencies were identified as different options to resolve the ambiguity, the intended period of time is better described using a numeric configuration parameter as for the case of the room temperature in section 5.

Example 8. Mul4: It should also be possible to dump views to a file, in jpeg format or equivalent. It shall be possible for the file to be collected or dispatched to another system.

This requirement, containing two *or* constructs, indicates two variation points: one related to the file format (with an opening to different formats), and one related to the choice of whether to locally store the files or to dispatch them to another system. Both cases of multiplicity can be expressed as an alternative between the various options.

7. CONCLUSIONS

We have defined an approach to extract variability issues from a requirements document using Natural Language analysis tools. These tools are aimed at revealing the ambiguity defects of the NL sentences in the requirements document. We have focused on those cases in which ambiguity in requirements is particularly due to the need to postpone

choices for later decisions in the implementation of the system. Hence, ambiguity becomes a means to enlighten possible variation points in an early phase of software and system development.

Other attempts have been done to use NLP to extract variability indications from a set of requirement documents (or other technical documentation, such as brochures), each referred to a possible variant, by making a comparative analysis in order to figure out common parts, assuming that the parts not in common constitute the variability. Our idea is instead that looking at a single requirement document, the ambiguity that is present in it can be used to identify possible variation points, where ambiguity is not a defect but points to different choices that can give space for a range of different products.

The proposed approach needs to be validated on an extensive set of requirement documents, also in order to refine the definition of a possible supporting tool. The validation should address three issues:

- Scalability of the approach to large requirement documents.
- Representativeness of the set of required documents to actually define product families. The approach can be validated using those requirement document that actually generated a product family, to check if the approach identifies the ambiguities that were sources of variation points.
- Number of false positives, and cost of their analysis.

All these issues are the subject of the research activity started with the preliminary actions presented in this paper.

8. REFERENCES

[1] V. Ambriola, V. Gervasi, On the Systematic Analysis of Natural Language Requirements with CIRCE, Automated Software Engineering, Volume 13, Issue 1, pp 107-167, 2006.

[2] M. Antkiewicz, K. Bąk, A. Murashkin, R. Olaechea, J. H. Liang, and K. Czarnecki. Clafer Tools for Product Line Engineering. 17th International Software Product Lines Conference, *SPLC* 2013, volume 2, pages 130–135.

[3] N. H. Bakar, Z. M. Kasirun, and N. Salleh, Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review, J. Syst. Softw., vol. 106, pp. 132-149, 2015.

[4] G. Bécan, R. Behjati, A. Gotlieb, M. Acher, Synthesis of attributed feature models from product descriptions, 19th International Software Product Lines Conference, SPLC 2015: pp. 1-10.

[5] G. Bécan, N. Sannier, M. Acher, O. Barais, A. Blouin, B.Baudry, Automating the formalization of product comparison matrices, ACM/IEEE International Conference on Automated Software Engineering, ASE, 2014, pp. 433-444.

[6] D.M. Berry, Ambiguity in Natural Language Requirements Documents, Lecture Notes in Computer Science 5320, 2008, pp. 1-7.

[7] D. M. Berry, A. Bucchiarone, S. Gnesi, G. Lami , G. Trentanni, A new quality model for natural language requirements specifications . 12th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ) 2006, pp. 115-128.

[8] A. Bucchiarone, S. Gnesi, A. Fantechi, G. Trentanni, An experience in using a tool for evaluating a large set of natural language requirements. ACM Symposium on Applied Computing (SAC) 2010: pp. 281-286.

[9] A. Fantechi, S. Gnesi, Formal Modeling for Product Families Engineering. 12th International Software Product Lines Conference, SPLC 2008, pp. 193-202.

[10] A. Ferrari, G. O. Spagnolo, and F. Dell'Orletta, Mining commonalities and variabilities from natural language documents, in Proc. 17th International Software Product Lines Conference, SPLC 2013, pp. 116-120.

[11] A. Ferrari, F. Dell'Orletta, A.Esuli, V. Gervasi, S. Gnesi, Natural Language Requirements Processing: A 4D Vision, IEEE Software, to appear.

[12] V. Gervasi, D. Zowghi, On the Role of Ambiguity in RE, Requirements Engineering: Foundation for Software Quality: 16th International Working Conference, (REFSQ), LNCS 6182, 2010, pp. 248-254.

[13] S. Gnesi, G. Lami, G. Trentanni, An automatic tool for the analysis of natural language requirements, Comput. Syst. Sci. Eng., 20, 1, 2005.

[14] B. Gleich, O. Creighton, and L. Kof, Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources, Requirements Engineering: Foundation for Software Quality, (REFSQ) LNCS 6182, Springer, 2010, pp. 218-232.

[15] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Tech. rep., Carnegie-Mellon University Software Engineering Institute , 1990.

[16] N. Itzik, I. Reinhartz-Berger, Y. Wand, Variability Analysis of Requirements: Considering Behavioral Differences and Reflecting Stakeholders Perspectives IEEE Transactions on Software Engineering, 42, 7, July 2016, pp. 687-706.

[17] B. Rosadini, A. Ferrari, G. Gori, A. Fantechi, S. Gnesi, I. Trotta, and S. Bacherini, Using NLP to Detect Requirements Defects: an Industrial Experience in the Railway Domain, to appear on Proceedings Requirements Engineering: Foundation for Software Quality: 23rd International Working Conference, (REFSQ),2017.

[18] N. Sannier, M. Acher, B. Baudry: From comparison matrix to Variability Model: The Wikipedia case study. 28th IEEE/ACM International Conference on Automated Software Engineering, ASE, 2013, pp. 580-585.

[19] S.F. Tjong, and D.M. Berry, The Design of SREE–A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned, Requirements Engineering: Foundation for Software Quality, (REFSQ), Lecture Notes in Computer Science 7830, 2013, pp 80-95.