The Journal of Systems & Software 195 (2023) 111540

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

# VIBE: Looking for Variability In amBiguous rEquirements☆

Alessandro Fantechi [a,b], Stefania Gnesi [b], Laura Semini [b,c,*]

[a] Dipartimento di Ingegneria dell'Informazione - Università di Firenze, Italy
[b] Istituto di Scienza e Tecnologie dell'Informazione "A.Faedo", Consiglio Nazionale delle Ricerche, ISTI-CNR, Pisa, Italy
[c] Dipartimento di Informatica, Università di Pisa, Italy

ARTICLE INFO

*Article history:*
Received 8 February 2022
Received in revised form 29 September 2022
Accepted 7 October 2022
Available online 17 October 2022

*Keywords:*
Natural language requirements documents
Ambiguity
Natural language processing tools
Software product lines
Variability detection

ABSTRACT

Variability is a characteristic of a software project and describes the fact that a system can be configured in different ways, obtaining different products (variants) from a common code base, accordingly to the software product line paradigm. This paradigm can be conveniently applied in all phases of the software process, starting from the definition and analysis of the requirements. We observe that often requirements contain ambiguities which can reveal an unintentional and implicit source of variability, that has to be detected.

To this end we define VIBE, a tool supported process to identify variability aspects in requirements documents. VIBE is defined on the basis of a study of the different sources of ambiguity in natural language requirements documents that are useful to recognize potential variability, and is characterized by the use of a NLP tool customized to detect variability indicators. The tool to be used in VIBE is selected from a number of ambiguity detection tools, after a comparison of their customization features. The validation of VIBE is conducted using real-world requirements documents.



## 1. Introduction

Software Product Line Engineering (SPLE) is a paradigm that has been proposed to support the development of a set of different, but similar, software products (Clements and Northrop, 2002; Pohl et al., 2005; Apel and Kästner, 2009; Northrop and Jones, 2010). At all abstraction levels in a software product line (SPL), a product family description is composed of a common part and a variable part. Product *variants* can be derived from a product family specification and dealing with *variability* is the key issue that has caused SPLE to mature into a specific software engineering discipline.

The design of a SPL begins with the identification of the possible *variation points*, i.e. variability, in the requirements documents. To this end, and given the prevalent trend in industry to express requirements using Natural Language (NL) statements, several researches focused on exploiting Natural Language Processing (NLP) techniques and tools[1] to identify the requirements that can be used as a source from which variability-relevant information can be elicited (Itzik et al., 2016; Bakar et al., 2015; Fantechi et al., 2018a, 2017, 2018b, 2019; Nasr et al., 2017; Li et al., 2017; Ferrari et al., 2013; Fantechi et al., 2021). This is done by searching the linguistic constructs specifically relevant to detect the variability contained in NL requirements documents.

In previous works (Fantechi et al., 2018a, 2017, 2018b), we initiated a line of research to extract possible variation points from NL requirements documents using an existing NLP technique intended to disclose *ambiguity* flaws, showing that ambiguity detection can be helpful to detect variability and stimulating a more systematic and deep experience.

Ambiguities normally cause inconsistencies between the customer expectation and the developed product, and often lead to unwanted rework of artefacts. However, an ambiguous term can also be used as a means of postponing a decision for a later time in software development. With this insight, we also realized that existing NLP tools for ambiguity detection can be used to capture hidden aspects of variability in requirements.

The contribution of this paper is to exploit, combine and extend the preliminary results of our research to define an overall variability elicitation process (VIBE — Variability In amBiguous rEquirements), based on the analysis of ambiguity in requirement documents, with the automated support offered by an NLP tool, QuARS (Quality Analyser for Requirements Specifications) (Fabbrini et al., 2001), specifically tailored to ambiguity detection in this kind of documents.

What is new compared to related work in SPLE is the search for variability through ambiguity, exploiting established tools and

☆ Editor: Prof. Raffaela Mirandola.

* Corresponding author at: Dipartimento di Informatica, Università di Pisa, Italy.
*E-mail addresses:* alessandro.fantechi@unifi.it (A. Fantechi), stefania.gnesi@isti.cnr.it (S. Gnesi), laura.semini@unipi.it (L. Semini).

[1] NLP is a linguistic activity focused on processing and analysing written natural language texts, with the purpose of identifying, recognizing, and classifying the terms, and recovering the structure, of a discourse (Chowdhury, 2003).

https://doi.org/10.1016/j.jss.2022.111540
0164-1212/

achievements from the area of requirements engineering and customizing them to identify variability.

Specifically, in this paper we investigate the following issues:

- Relevance of ambiguity in requirements document to detect variability: we evaluate whether ambiguities in NL requirements can be considered as potential variability;
- Identification of the ambiguity classes and indicators that appear as more suited to detect variability;
- Identification of other text characteristics than ambiguities that can be variability indicators;
- Assessment of the extent to which the variability identification process can be automated with an ambiguity detection tool;
- Comparison of off-the-shelf NLP tools with respect to their suitability to extract variability, to evaluate the characteristics that an NLP tool should have for this aim.

The investigation of these issues is supported by a quantitative analysis on six requirements documents publicly available and referring to different domains and systems to be built.

Looking at requirements documents from a SPL perspective we can note that they can exhibit different awareness of the editor about dealing with a product line: from this point of view three classes of requirements documents can be identified:

1. Product requirements documents: these are the documents that explicitly refer to a single product;
2. Family requirements documents: they specifically envisage a family of products. They are structured or use formalisms to describe variability and make explicit a mapping between requirements and variants, for example by saying in which variant a requirement is mandatory or not.
3. Requirements documents that do not belong to the previous classes, because it has not yet been determined whether they define a single product or a family, or because they contain parts not specified in all details. These documents are open to become the requirements of a single product or of a family.

VIBE proves useful with the requirements of the third class, since they may contain an implicit source of variability that has to be detected. The documents of the first class are clearly not candidates to be processed by VIBE: being explicitly aimed at the definition of a single product, possible variabilities, even if detected, would be ignored or dealt with as ambiguities. The documents of the second class already include instead the explication of variability, and we can assume that they are not subject of residual ambiguity that hides variability, see for instance the car crash example in Capozucca et al. (2011) and Shahin et al. (2019).

The paper is structured as follows. We define the background in Section 2. In Section 3 we use two simple examples to illustrate the relevance of ambiguity classes for detecting variability. In Section 4 we define new indicators that are not ambiguities but are nevertheless indicators of potential variation points and that can be detected in a similar way. In Section 5 we show how variability detection can be aided by NLP tools, within an integrated analysis workflow that constitute the VIBE process. Quantitative analysis and validation data on large requirements documents are presented in Section 6. In Section 7 we develop further the process by defining a mapping that indicates to which variation point a detected ambiguity can correspond. In Section 8 we discuss threats to validity and in Section 9 we present related work. Section 10 concludes the paper.

## 2. Background

In this section we will give some background information: sources of ambiguity in natural language requirements and the notions of features and variation points in SPLs.

### 2.1. Ambiguity in NL requirements

Software requirements must comply with quality standards well known to developers, in particular requirements must avoid ambiguities in order to be unequivocal and verifiable, that is, they must be interpreted in a single way by all expected readers and it must be possible to demonstrate that the implementation meets them. On the other hand, requirements are often expressed in natural language, which is inherently ambiguous, and this is a recognized cause of interpretation problems of requirements documents.

There are actually many different sources of ambiguity that can compromise the quality of the requirements. They are well known in requirements engineering: we list them in Table 1, where we report the main ambiguity classes in the left column and examples of indicators in the right one. We refer to Berry et al. (2003a), Gervasi et al. (2019) and INCOSE (2019) for a detailed discussion.

Detecting ambiguity requires a text analysis that can be *lexical*, when specific terms (named *indicators*) are searched over the text, or *syntactical*, when specific syntactic structures are searched for. As we will see in the following, lexical analysis reveals to be very effective in pointing to possible ambiguity defects and in the right column of Table 1 we list some of the relevant indicators that may reveal the related source of ambiguity.

System engineers and project managers can use NLP techniques and tools as a support to analyse and correct requirements in natural language. NLP provides a help speeding up the task of finding possible defects (Chowdhury, 2003); a review of the results provided by the tools by domain experts is then needed to correct the found deficiencies.

### 2.2. Variation points

In the SPL context, a *feature* is usually considered as an abstraction of a concrete artefact (Apel and Kästner, 2009; Apel et al., 2013) and a feature is typically defined as a unit of functionality or component, also physical, of a variant-rich software system. The variability depends on which features are included and which are not in each variant (product). The decision between inclusion or not is located at specific places in design artefacts, named *variation points*, and features can be classified as:

**Optional:** features that may be present in a product or not;

**Mandatory:** features that must be present in all products;

**Alternative:** features among which one and only one is present in each product;

**Or:** features among which at least one is present in each product.

Binary relations can also be defined among features:

**Requires:** unidirectional relation between two features indicating that the presence of one feature in a product requires also the presence of the other;

**Excludes:** mutual exclusion relation between two features: no product may contain the two features at the same time[2]

[2] Note that *excludes* is weaker than *alternative* admitting products in which none of the mutually exclusive feature is present.

**Table 1**
Ambiguity classes and indicators (Berry et al., 2003a; Gervasi et al., 2019; INCOSE, 2019).

| Ambiguity classes | Indicators |
|---|---|
| **Homonymy** and **polisemy** occur when a term can have different meanings, having different (homonymy) or one (polisemy) etymology | some examples are: *bank, can, bat... (homonymies), left, right, fall, minute, …(polysemies)* |
| **Analytical, attachment, coordination ambiguities** occur when a sentence admits more than one grammatical structure, and different structures have different meanings | syntactic analysis: the sentence admits two or more syntactic trees |
| **Anaphora** occurs when an element of a sentence depends for its reference on another, antecedent, element and it is not clear to which antecedent it refers | relative and demonstrative pronouns: *that, which, their, it, them, they, both, …* |
| **Vagueness** occurs when it is not possible to interpret a sentence in a unequivocal way | *clear, easy, strong, good, bad, adequate, tall, short, various, completed, similar, similarly, accordingly, …* |
| **Comparatives & superlatives** occurs when the term of comparison or the universe of discourse are missing | *better, easier, worst, faster, bigger, biggest, …* |
| **Disjunctions** occurs when a sentence admits different models in which the first, the second or both disjuncts are true | *or, and/or, …* |
| **Escape clauses** occurs when a sentence admits different models, containing or not the object the escape clause | *case, possibly, if possible, if appropriate, where permitted, among others, as a minimum, when requested by, when required, …* |
| **Weakness** occurs when the sentence contains weak verbs | *may, can, could, …* |
| **Quantifiers** in presence of quantifiers, ambiguities are due to the scope or to the universe of quantification | *a, all, always, every, any, nothing, …* |
| **Under-specification** occurs when the sentence contains terms that need to be instantiated or qualified | *information, interface, attack, button, channel, component, procedure, process, report, session, …* |
| **Passive voice** occurs when the subject of the passive sentence is not be revealed | auxiliary *to be* with a past participle and no agent specified *(by)* |

## 3. Relevance of ambiguity to detect variability

Ambiguity is a potential cause of inconsistencies between the customer's expectations and the developed product, and leads to unwanted reworkings on the artefacts. However, if we think about how an ambiguity arises, we realize that in many cases the use of an ambiguous term is due to the need, deliberate or unconscious, to postpone choices for later decisions in the implementation of the system. From this perspective, ambiguity can also be seen as a way to abstract aspects of variability to be resolved later in software development.

Identifying the ambiguities in a requirement document is a way to find the points of variability and to revise the document to specify the requirements of a family of products.

Now, we can begin to formulate hypotheses, namely that ambiguity is likely to hide some variability and that some classes of ambiguity indicators are more suitable than others for looking for potential variability. We here consider all the ambiguity classes presented in Table 1 and we discuss which are the most promising candidates, evaluating whether it is plausible or not to envision different products for the different interpretations of an ambiguous requirement.

In the following, we use ✓ to denote plausible candidates, ✗ for the implausible ones, and **?** when we cannot anticipate an opinion of plausibility.

✗ **Homonymy and polisemy:** indicators are all the English words that have more than one meaning. These classes of ambiguities are not relevant for variability: the possible interpretations of ambiguous words are in these cases too different and unrelated to denote potential variants. As an example consider the term *left* which is either a verb or a noun, with complete different meanings. A further example is *bank* which can be the institution where people keep their money or the elevated areas of land along the edge of a river: there is no relation with variability issues.

✗ **Analytical, attachment, and coordination:** also in this case we are in the presence of pure ambiguities as we show with the following examples. "Software fault tolerance", can be tolerance to software faults or fault tolerance achieved by software (*analytical*); "The new users can buy items with a discount" means either that "The new users can only buy discounted items" or that "The new user can have a discount when buying items" (*attachment*); "The machine shall offer, as beverages, coffee and cappuccino or tea", that can be interpreted as "The machine shall offer, as beverages, (coffee and cappuccino) or tea" or as "The machine shall offer, as beverages, coffee and (cappuccino or tea)" (*coordination*). The latter, coordination, can lead to variability in the presence of an *or*, but in this case variability is more related to disjunction than to coordination.

In all three examples we can associate two syntactic trees with each sentence, and the different interpretations obtained manifest only an ambiguity problem.

✗ **Anaphoras:** also in this case the different meanings that a requirement can have are unrelated and we are in the presence of pure ambiguity defects. In fact, anaphoras are another category of syntactic ambiguity, related to the use of pronouns, where different interpretations lead to completely different sentences. As an example consider: "The coffee machine adds sugar to the beverage but allows the user to exclude it". In one interpretation the user can exclude the sugar, in the other the beverage.

✓ **Vagueness:** a requirement containing a vague term may represent an abstraction (e.g. *adequate, various*), which is made concrete by a set of possible instances. In this case the process of disambiguation and refinement of the requirements will make the instances explicit and these instances can actually correspond to several variants.

✗ **Comparatives & Superlatives:** also in this case we are in the presence of pure ambiguity defects that do not hide variation points. Comparatives and Superlatives may actually indicate some non-functional feature differentiators and can be used to attach attributes to features in an *attributed feature model* (Bak et al., 2016). This issue is out of scope in this paper.

✓ **Disjunctions:** a requirement that contains a disjunction between two options can envision three different products implementing the first, the second, and both options, respectively.

✓ **Escape clauses:** this ambiguity may hide the fact that a decision has not yet been made on whether or not an option should be implemented by the system: it may be reasonable to envisage two different products, one that incorporates the feature in question and the other that does not.

✓ **Weakness:** similarly to escape clauses, a requirement containing a weak verb, such as *may* or *can*, normally indicates an option, paving the way to two different products.

? **Quantifiers:** this ambiguity class does not seem to reveal any variability. We postpone any further consideration at this regard to Section 6.

? **Under-specification:** occurs when a generic term is used that needs a specifier to be correctly interpreted. This defect does not seem to be related to variability and the generic terms in Tables 1 do not provide any clue to anticipate a hypothesis as to whether this class can help detect variability or not. We postpone any consideration to Section 6.

? **Passive voice:** the disambiguation job here is to put the requirement in active form and decide who the subject is. If different product variants are envisaged for different subjects, then there might be a variation point. We postpone any further consideration to Section 6.

In Section 6 we will validate the candidate (✓) and the uncertain (?) indicators by testing them on real case studies, while we will ignore the implausible ones (✗).

For now, we introduce two simple examples with the purpose of presenting how ambiguity indicators can help in detecting variability.

The final judgement if ambiguity indicators are actual variability detectors can be considered in real life as a capability of a senior figure such as a product manager. In the following, we are actually impersonating this role, so discriminating actual variability cases from *true ambiguities* (ambiguities that are not variabilities) and false positive cases (not even ambiguities).

### 3.1. Example 1: Basic coffee vending machine

We consider the requirements of a simple coffee vending machine (Fantechi et al., 2017) defined to illustrate the relevance of some ambiguity classes for variability.

**Rc1** After inserting a suitable coin, the user shall choose a beverage and select the amount of sugar.
**Rc2** The machine shall offer, as beverages, coffee and cappuccino or tea.
**Rc3** The machine shall always offer coffee.
**Rc4** A ringtone possibly has to be played after beverage delivery.
**Rc5** After the beverage is taken, the machine returns idle.
**Rc6** The British market requires tea and excludes any ring tone.

We begin the analysis of these requirements by enlightening the ambiguities revealed according to the indicators in Table 1. After this, we conduct a second analysis to extract, from the detected ambiguities, the potential variabilities.

We report the outcome in Table 2, where we the ambiguous terms are highlighted in blue and we underline the terms that reveal a variability according to the following analysis.

We can notice that Rc1 is vague (*suitable* coin), the meaning of *suitable* must be clarified to indicate which coins are accepted.

**Table 2**
Coffee vending machine example.

| | |
|---|---|
| **Rc1** | After inserting a suitable coin, the user shall choose a beverage and select the amount of sugar. |
| **Rc2** | The machine shall offer, as beverages, coffee and cappuccino or tea. |
| **Rc3** | The machine shall always offer coffee. |
| **Rc4** | A ringtone possibly has to be played after beverage delivery. |
| **Rc5** | After the beverage is taken, the machine returns idle. |
| **Rc6** | The British market requires tea and excludes any ring tone. |

The ambiguity might be referred to the denomination of the accepted coins (10c, 20c, …). On the other hand, it can also refer to different currencies (dollar, euro, pound, …). If we assume that the product manager considers producing the coffee machine for several markets, then *suitable* hides a variability and dollar, euro, pound, …correspond to different variants.

Rc2 contains an interweaving of (*and*) and (*or*), i.e. a coordination ambiguity: it can be interpreted either as *coffee and (cappuccino or tea)* or as *(coffee and cappuccino) or tea*. The correct interpretation of this requirement is made clear in Rc3. Indeed, Rc3 says that coffee must always be offered, indicating that the first interpretation of Rc2 is the correct one. The disjunction *or* is another ambiguity in Rc2, and in this case it hides a variability, since we can envision three possible products, offering respectively: coffee + tea, coffee + cappuccino, coffee + tea + cappuccino.

Rc3 contains a quantifier that is not even an ambiguity.

Rc4 contains two ambiguities. The first one is an escape clause, (*possibly*). In principle, this requirement could be interpreted in two ways: (1) in some cases, for example if the beverage is not collected within a time limit, a ringing sound is played, (2) some machines are ringing and others are not. The overall analysis of the set of requirements, in particular Rc6, shows that the second option is true: the ringer may or may not be in a machine, and this indicates a variability. The other ambiguity in Rc4 is a passive voice that is not a variability indicator since the only reasonable player is the coffee machine itself.

Rc5 contains a passive voice too, and also in this case it is not a variability indicator.

Rc6 contains *any* which does not indicate variability either.

### 3.2. Example 2: E-shop

The second example is a (simplified) e-shop (Fantechi et al., 2019) inspired by the example in Itzik et al. (2016), for which we consider the requirements in Table 3. As above, we highlight in blue those terms that are indicators of ambiguity and, among these, we underline those that indicate variability.

Details of the variability found in the e-shop will be discussed in Section 7, here we observe that the quantifiers (*all, a, an*), and anaphoras (*his*) are not detectors of variability, while vagueness (*various*), disjunctions (*or*), weakness (*may*), and escape clauses (*possibly*) are. In the e-shop there is also an under-specification (*screen*) that is not a variability. There are three passive voices: two of them (*is found* and *be shipped*) are false positives, while the one in Re1 (*to be entered*) can be a source of variability and requires further investigation.

To sum up, in Table 4 we recap the ambiguity indicators found in the coffee machine and e-shop requirements and link them to their ambiguity classes.

**Table 3**
E-shop example.

| | |
|---|---|
| **Re1** | The system shall enable the search text to be entered on the screen. |
| **Re2** | The system shall display all the matching products based on the search. |
| **Re3** | The system possibly notifies with a pop-up the user when no matching product is found on the search. |
| **Re4** | The system shall allow a user to create his profile and set his credentials. |
| **Re5** | The system shall authenticate user credentials to enter the profile. |
| **Re6** | The system shall display the list of active and/or the list of completed orders in the customer profile. |
| **Re7** | The system shall maintain customer email information as a required part of customer profile. |
| **Re8** | The system shall send an order confirmation to the user through email. |
| **Re9** | The system shall allow a user to add and remove products in the shopping cart. |
| **Re10** | The system shall display various shipping methods. |
| **Re11** | The order shall be shipped to the client address or to an associated store, if the click&collect service is available. |
| **Re12** | The system shall enable the user to select the shipping method. |
| **Re13** | The system may display the current tracking information about the order. |
| **Re14** | The system shall display the available payment methods. |
| **Re15** | The system shall allow the user to select the payment method for order. |
| **Re16** | After delivery, the system may enable the users to enter their reviews or ratings. |
| **Re17** | In order to publish the feedback on the purchases, the system needs to collect both reviews and ratings. |
| **Re18** | The click&collect service excludes the tracking information service. |

## 4. Ad hoc variability indicators

In the previous section we have observed that some of the ambiguity classes can be used to detect variability. Here, we wonder whether there might be other terms or sentence fragments that are not ambiguities but are nevertheless indicators of potential variation points. Discovering such indicators may support a more complete analysis. The experience gained from manual analysis of many requirements documents led to the definition of the following two classes:

**Variability:** The first set of indicators contains terms and constructs that explicitly refer to variability. These are terms like: *configuration, feature, range, …* or dependent clauses containing both a conjunction *if, where, when, whether, …* and a verb like *available, provided, implemented*.

**Inter-feature constraints:** A requirement may indicate some constraints expressing implication or mutual exclusion relations between features. Indicators are terms like *imply, require, entail, implicate, demand, exclude, rule out, mutually exclusive, need, …*

In Table 5 we underline these terms in the requirements of our two examples:

**Rc6** introduces two inter-feature constraints: a *requires* and an *excludes*.

**Re11** tells that not all e-shops need to provide a click&collect service.

**Re17** introduces a *requires*: reviews collection and ratings collection are mandatory if the feedback feature is present.

**Re18** introduces an *excludes*: features click&collect service and tracking information service are mutually exclusive.

**Table 4**
Ambiguity indicators found in the examples: we separate those that indicate variability from those that do not (pure ambiguities or false positives).

| Ambiguity classes | Variability | Non variability |
|---|---|---|
| Attachment | | his |
| Coordination | | and…or |
| Vagueness | suitable, various | |
| Disjunctions | or, and/or | |
| Quantifiers | | always, any, all, a, an |
| Escape clauses | possibly | |
| Weakness | may | |
| Passive voice | | be played, is taken, be entered, is found, be shipped |
| Under-spec | | screen |

**Table 5**
New ad hoc indicators found in the case studies.

| | |
|---|---|
| | Coffee vending machine |
| **Rc6** | The British market requires tea and excludes any ring tone. |
| | E-shop |
| **Re11** | The order shall be shipped to the client address or to an associated store, if the click&collect service is available. |
| **Re17** | In order to publish the feedback on the purchases, the system needs to collect both reviews and ratings. |
| **Re18** | The click&collect service excludes the tracking information service. |

## 5. Automatic detection of variability: The VIBE process

VIBE (Variability In amBiguous rEquirements) is a tool-supported process for variability identification to be applied in the requirement phase definition for SPLs. The idea is to use NLP tools for ambiguity detection, customized to identify variability. Customization includes the use of some experimentally meaningful indicators of ambiguity (see Section 6) and the addition of specific indicators of variability.

The process is not tied to a particular NLP tool. However, we consider more sensible to use a rule-based NLP tool than an AI-based approach due to the typical lack of ground-truth learning information experienced in the requirement analysis domain (Ferrari et al., 2017b).

Fig. 1 illustrates the VIBE process. The first step is to automatically analyse the requirements by running a customized NLP tool. The outcome of the tool is manually analysed and assessed to disambiguate the requirements and to remove the false positives from the check list. The remaining terms and text fragments identify variabilities: the (manual but systematic) association with variation points in a product family specification completes the process.

### 5.1. A NLP tool for VIBE

Tools based on NLP techniques are able to detect ambiguities in natural language requirements and can also be used to detect variability in them. These tools return the potential ambiguity sources due to linguistic defects, that are then reviewed by domain experts to distinguish between false positives, pure ambiguities and variabilities.

We present in this section a set of existing NLP tools used in requirements analysis, that we consider as possible candidates to be employed in VIBE.

We are looking for a tool that can detect the indicators of potential variability identified in Section 3. Specifically, we are interested in a tool able to filter out non interesting ambiguity indicators and at the same time to be extended to capture the ad

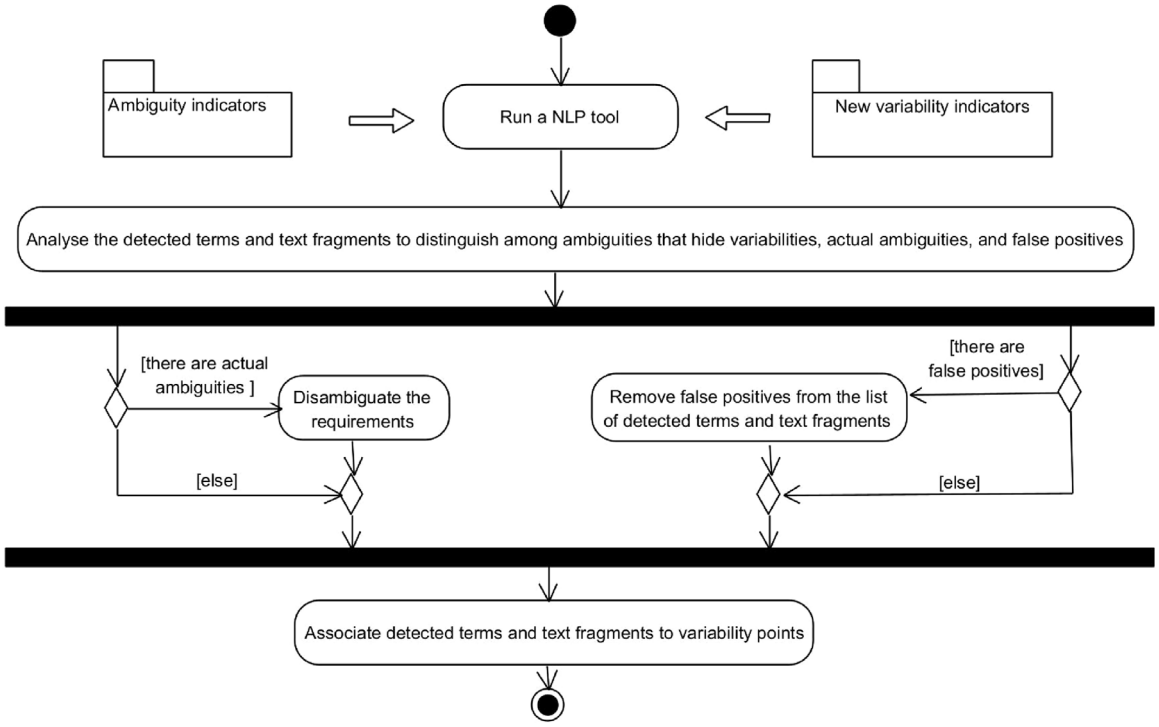



**Fig. 1.** The VIBE tool-supported process for variability identification.

hoc variability indicators. For the time being, we focus on tools dealing with the English language.

To this end, we consider the academic tools: QuARs (Fabbrini et al., 2001) and TIGER-PRO (Kasser, 2022), and the commercial ones: Requirements Scout developed by Qualicen GmbH (Femmer, 2018); QVscribe by QRA (Kenney and Cooper, 2020); RAT by Reuse Company (Reuse Company, 2022) and RQA by IBM (2022). The choice has been based on a search in the literature and on tools availability. We present and compare those that we have identified as being closest to our needs. To the best of our knowledge this set, an extension of the one analysed in Arrabito et al. (2020), represents a sufficiently complete selection of available NLP tools specifically focused on NL requirements and ambiguity.

The tools we consider have some common characteristics: they perform a linguistic analysis of a requirements document in text format and flag potentially defective sentences. Most of the tools are based on a specific NLP technique, namely lexical or syntactical analysis, and are aimed at the detection of particular defects of the requirements, like for instance ambiguity and incompleteness. Some of them refine the analysis using machine learning techniques.

Using the documentation, we compare the tools in their coverage of the ambiguity classes that more relate to variability as discussed in Sections 3 and 4

Each of these tools has its set of ambiguity indicators and a classification for them, stored in built-in *dictionaries* and these indicators may not be classified exactly as in Table 1. In the following, we present the dictionaries of each tool, when publicly available.

Other families of tools exist for analysing NL requirements, that exploit ontologies (Dermeval et al., 2015). Ontologies are usually applied in two ways: to disambiguate polysemies and to find homonyms, and this use is not related to variability; to relate conceptually close words, which can help to complement our approach, as discussed in Section 9.

**Table 6**
Indicators of QuARS: ambiguity classes relevant for variability.

| Ambiguity classes | Indicators |
|---|---|
| Vagueness | *clear, easy, strong, good, bad, adequate...* |
| Disjunction | *or, and/or, or/and* |
| Optionality | *possibly, eventually, case, if appropriate...* |
| Weakness | *can, could, may, might, …* |
| Quantifiers | *all, always, every, any, …* |
| Under-specification | *channel, component, document, interface, message, number, unit …* without a qualifying adjective (syntactic analysis) |

### 5.1.1. QuARS

QuARS (Quality Analyser for Requirements Specifications) (Fabbrini et al., 2001) is a requirements analysis tool developed in our lab and has been applied to a variety of documents, many from industry, pertaining to different domains (Bucchiarone et al., 2005; Lami et al., 2019).

The requirements analysis process is split in two parts: (i) the lexical analysis that captures disjunction, optionality, vagueness, and weakness defects. This step identifies candidate terms in requirements that match the indicators stored into a set of built-in dictionaries as lists of terms; and (ii) the syntactical analysis to capture under-specification.

The user uploads a document and chooses the desired type of analysis and a corresponding dictionary or syntactical rule (see Table 6). QuARS returns the list of ambiguous requirements for that analysis and highlights the indicators that led to the red flag. Moreover, dictionaries are editable and can be easily modified by the user, to add or remove indicators. The user also can exclude a dictionary during the analysis or define and add new dictionaries.

### 5.1.2. TIGER-PRO

TIGER-PRO (Kasser, 2022) is an educational tool developed by Joseph Kasser with the purpose of helping students to write high quality requirements. TIGER-PRO analyses requirements with a

**Table 7**
Indicators of TIGER-PRO: ambiguity classes relevant for variability.

| Ambiguity classes | Indicators |
|---|---|
| Possible multiple requirements | *and, or, and/or, /, either, both , …* |
| Unverifiable terms | *same, each, those, all, include, minimum, up to, any, brief, clear, affect, sufficient, …* |
| Wrong word | *should, will* |

**Table 8**
Indicators of QVscribe: ambiguity classes relevant for variability.

| Ambiguity classes | Indicators |
|---|---|
| Optional escape clauses | *possibly, may, optionally, if necessary, can, …* |
| Vague words | *various, completed, a lot of, almost, bad, …* |
| Passive voice | *based, found, shipped, …* |
| Immeasurable quantification | *abundant, far, always, all, large, …* |

**Table 9**
Indicators of Requirements Scout: ambiguity classes relevant for variability.

| Ambiguity classes | Indicators |
|---|---|
| Passive voice | *done, found, sent, …* |
| Universal quantifiers | *all, always, every, any, nothing, …* |
| Imprecise phrases | *possibly, various, small, if possible, general…* |
| Occurrence of will or may | *will, may, can, …* |
| Dangerous slash | */* |

testing perspective and highlights requirements that may be difficult to verify, or that are written in a way that complicates testing. The analysis is lexical and it is based on dictionaries of indicators that we report in Table 7.

### 5.1.3. QVscribe

QVscribe is a tool developed by QRA (https://qracorp.com) (Kenney and Cooper, 2020), it analyses the quality of the requirements, highlighting ambiguity, inconsistencies and possible similarities. In addition, it allows the generation of detailed reports that can be used to increase clarity and consistency of requirements. The classes of defects of interest for variability that are detected by the tool and the related indicators can be classified according to Table 8. Also in this case, indicators are stored in dictionaries that can be edited and modified by the user.

### 5.1.4. Requirements Scout

Requirements Scout is developed by Qualicen GmbH, (https://www.qualicen.de/en/) (Femmer, 2018) to analyse requirements specifications. Requirements Scout, besides analysing NL requirements with the aim of identifying the defects, also allows the analyst to keep track of different versions of the requirements, creating a complete history of the detected defects: as soon as the requirements are updated, the tool re-analyses the modified parts and shows whether the update has eliminated existing defects or has introduced new ones.

The ambiguity classes and indicators of Requirements Scout relevant to variability are shown in Table 9. They are stored in non-modifiable built-in dictionaries.

### 5.1.5. IBM Engineering Requirements Quality Assistant (RQA)

RQA, produced by IBM, is a requirements quality analysis tool that integrates a standard rule based analysis with Watson AI, the IBM solution for AI and machine learning (IBM, 2022). RQA is based on the INCOSE guide (INCOSE, 2019) and catches the following issues: *Unclear actors or users, compound requirements, negative requirements, escape clauses, missing units, missing tolerances, ambiguity, passive voice, incomplete requirements and unspecific quantities,…*. The list of indicators for each ambiguity class is not available for RQA.

The analysis assigns a quality measure to each requirement and, if the requirement contains an ambiguity, RQA indicates it. The tool exploits machine learning capabilities to learn when the user discards some detected ambiguities as false positives. It also permits to manually exclude some ambiguity classes when analysing the document, but not to add new ones.

**Table 10**
Comparison of NLP tools, T1_coverage: ambiguity classes supported by the tools.

| Ambiguity classes | QuARS | TIGER | QVscribe | ReqSco | RQA | RAT |
|---|---|---|---|---|---|---|
| Vagueness | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Disjunction | ✓ | ✓ | | ✓ | ✓ | |
| Escape clauses | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Weakness | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Quantifiers | ✓ | ✓ | ✓ | ✓ | | ✓ |
| Passive voice | | | ✓ | ✓ | ✓ | ✓ |
| Under-specification | ✓ | | | | | |

**Table 11**
Comparison of NLP tools: licencing, T2_Filtering capabilities, and T3_Extensibility capabilities (addition of new dictionaries or editing existing ones).

| | QuARS | TIGER | QVscribe | ReqSco | RQA | RAT |
|---|---|---|---|---|---|---|
| Free licence | ✓ | ✓ | | | | |
| Filtering | ✓ | | ✓ | | ✓ | ✓ |
| Addition | ✓ | | | | | ✓ |
| Editing | ✓ | ✓ | | ✓ | | ✓ |

### 5.1.6. Requirements Analysis Tool (RAT)

RAT is a tool produced by the Reuse Company (Reuse Company, 2022). It supports the analyst while writing the requirements and also performs a quality analysis. Indeed, in RAT it is possible to define domain ontologies, based on which the tool proposes to the writer possible words to complete a requirement. On the analysis side, RAT is inspired by the INCOSE guide (INCOSE, 2019) and looks for ambiguities like: *passive voice, vague adverb, vague adjective, imprecise quantification, pronouns, ambiguous sentences, too long requirements, ambiguous temporal expressions, wrong unit of measurement,…*. The list of indicators for each ambiguity class is not available for RAT.

## 5.2. Which ambiguity detection tools are more suitable to extract variability?

We compare here the tools presented above and answer the following questions:

**T1_Coverage:** which of the ambiguity classes that are candidate to detect variability, (as discussed in Section 3) are detected by each tool?

**T2_Filtering:** which filtering capabilities, represented by the ability to select only certain dictionaries, are offered by the tools? (this feature permits to reduce the number of false positives)

**T3_Extensibility:** which extensibility capabilities, represented by the possibility of adding new dictionaries or new indicators to existing dictionaries or of adding new syntactic rules, are offered by the tools? (this feature permits to reduce the number of false negatives)

To answer to these questions, we refer to Tables 10 and 11. We first consider the academic tools, QuARS and TIGER-PRO. TIGER-PRO does not allow to select the dictionaries to be used

```
-------------  QuARS [Lexical] vagueness ANALYSIS  -------------
The line number:
 1. after inserting a suitable coin, the user shall choose a beverage and
  select the amount of sugar.
  is defective because it contains the wording: suitable
The line number:
  6. the British market requires tea and excludes any ring tone.
  is defective because it contains the wording: any
-------------  QuARS [Lexical] disjunction ANALYSIS  -------------
The line number:
 2. the machine shall offer, as beverages, coffee and cappuccino or tea.
  is defective because it contains the wording: or
-------------  QuARS [Lexical] optionality ANALYSIS  -------------
The line number:
  4. a ringtone possibly has to be played after beverage delivery.
  is defective because it contains the wording: possibly
-------------  QuARS [Lexical] quantifiers ANALYSIS  -------------
The line number:
  3. the machine shall always offer coffee.
  is defective because it contains the wording: always
```

**Fig. 2.** Outcome of QuARS when looking for ambiguity indicators in the Coffee vending machine example.

during the analysis while QuARS does it (T2). Both tools permit editing the dictionaries (partly answering to T3). In addition, QuARS dictionaries are more complete, i.e. more defects are found (T1) and new dictionaries can be easily added (T3). For instance, it has taken few minutes to add the *quantifiers* dictionary and the new dictionaries for variability, which were not in the original version of the tool: just press the add button, give a name to the new dictionary, populate it with the terms of interest and save. A limitation of both tools is that the user cannot add new syntactic checks, e.g. as the one needed to find passive voices. Therefore, given the three terms of comparison we have considered, QuARS performs significantly better than TIGER-PRO.

We now examine the commercial tools: QVscribe and ReqScout have very good performances with respect to coverage (T1). Again, for the purposes of identifying points of variability, we have to keep in mind the ability to select dictionaries (T2) and possibly add the new ad hoc indicators (T3): QVscribe only permits editing, and Requirements Scout allows the user to select the dictionaries during the analysis.

RAT by Reuse and IBM-RQA are the most sophisticated tools, the former can be configured by the user, and the latter integrates the Watson system, but they come with a high cost. RAT permits to select (T2), add and edit the dictionaries (T3), while in the documentation of RQA these features are not mentioned. Indeed, RQA learns only through examples and cannot be tuned with dictionaries and syntactic rules, so the extensibility (T3) does not apply.

To conclude, we can say that most of the tools considered can be used to find variabilities in a requirements document because they detect the ambiguities that most likely hide a variability. QuARS has the following advantages: it is free, offers built-in dictionaries that are reasonably complete, offers the ability to select the dictionaries to be used and the ability to edit them and to add new ones.

In the next sections we will use QuARS to exemplify and to validate the automatic detection step of the VIBE process. QuARS can be downloaded from https://github.com/Vibe-NLP/RequirementsForValidation.

### 5.3. Application of a NLP tool to the coffee machine example

We exemplify the application of a NLP tool to detect variability by showing the outcome of the QuARS tool when applied to the coffee vending machine example (Fig. 2).

**Table 12**
Summary the utilized requirement documents: number of requirements, number of words, authorship and characteristic of the system to be.

| | reqs | Words | Issued by | Characteristics |
|---|---|---|---|---|
| Eirene | 475 | 13 009 | Public auth. | Control system |
| Library | 94 | 1815 | Company | Information system |
| Blit | 55 | 1194 | Company | Information system |
| ERTMS | 49 | 1166 | Public auth. | Control system |
| People By T. | 185 | 3749 | Academia | Social |
| DigitalHome | 112 | 1121 | Academia | Control system |

QuARS detects all the ambiguities useful to reveal variability that we have shown in Table 2. It also detects a false positive, the quantifier *always*. The fact that manual and automatic analysis return nearly the same results is not surprising, the coffee machine example was defined to illustrate the approach, not to validate it.

Moreover, we have taken advantage of the editing ability offered by QuARS, to add a new dictionary for each of the new variability classes defined in Section 4, so improving the variability detection capabilities of the tool. Fig. 3 reports the additional output produced by QuARS when analysing the coffee machine requirements with the new dictionaries.

## 6. Validation

The adequacy of VIBE to identify variation points needs to be validated against a set of real world requirements documents. In particular, we want to address the following issues:

**V1_Narrowing:** Which are the ambiguity classes and indicators that are most relevant to detect variability?

**V2_Widening:** Are the new ad hoc variability indicators of help?

We consider six third-party requirements documents. These documents are very different from each other in terms of domain, system characteristics, and background and experience of their authors. The documents are available at https://github.com/Vibe-NLP/RequirementsForValidation and are described below (see also Table 12).

**Eirene** (European Integrated Railway Radio Enhanced Network) has been issued by the GSM-R Functional Group and it specifies the functional requirements for a digital radio standard for the European railways.

```
  -------------  QuARS [Lexical] variability ANALYSIS  -------------
The line number:
  1. after inserting a suitable coin, the user shall choose a beverage and
  select the amount of sugar.
  is defective because it contains the wording: choose
  is defective because it contains the wording: select
-------------  QuARS [Lexical] crossConst ANALYSIS  -------------
The line number:
  6. the British market requires tea and excludes any ring tone.
  is defective because it contains the wording: requires
  is defective because it contains the wording: excludes
```

**Fig. 3.** Outcome of QuARS when looking for the new ad hoc indicators in the Coffee vending machine example.

**Library** was prepared by the Galecia Group, a company specialized in libraries and organizations supporting libraries. It describes the functional and nonfunctional requirements for the System Administration Module of the Integrated Library System of a urban library system.

**Blit** is a draft of the functional specification of the requirements of a business project management tool, required by a company for re-writing its core Laboratory Information System to improve the performance. The authors of the document are anonymous.

**ERTMS** defines the functional requirements of ERTMS/ETCS (European Rail Traffic Management System/European Train Control System), issued by the European Railway Agency in June 2007. The document includes the requirements of a control system that provides the driver with information needed for the safe driving of the train, and it is able to supervise train and shunting movements.

**People by Temperament** is a document coming from the first edition of Plat_Forms, an international academic-industrial programming contest. The system to be built is a simple community portal where members can find others with whom they might like to get in contact, according to compatible tastes or personality types.

**DigitalHome** specifies requirements for developing a domotic system that allows a resident to manage the devices that control the home's environment. The document was developed by a team of 5 students in an academic context.

### 6.1. Data collection

The *data collection* procedure, for each document, consists of the following steps:

**Automatic Detection:** The document in textual format is given as input to QuARS, that produces a set of potentially ambiguous sentences, together with the terms or expressions that are the source of the ambiguity. We have used the following dictionaries and syntactic analysis for ambiguity: *Disjunction, Optionality, Vagueness, Weakness, Quantifiers*, and *Under-specification* and the new dictionaries *Variability* and *Inter-feature constraints*.

**Review:** The output of QuARS is reviewed independently by the authors of this paper, who at this stage assume the role of the product manager. Each defect revealed by the tool has been classified as: false positive, variability, or ambiguity. Our knowledge of the domains helped us make the choices.

**Assessment:** The output of the review phase is collegially discussed by the authors to analyse the discrepancies that emerged in the independent work and to seek a consensus. The data collected in this phase are reported in the first 4 rows of each ambiguity class block of Table 13 where we report, for each case study and for each ambiguity class:

- *fnd*: the number of defects found by QuARS in the Automatic Detection phase;
- *fp*: the number of false positives among *fnd*;
- *amb*: the number of ambiguities among *fnd*, (that do not indicate a variation point);
- *var*: the number of variation points among *fnd*;

In the last column of Table 13 we show aggregated data, obtained as illustrated in the table's caption.

An exception to this procedure has been followed for the *passive voices*, since QuARS is not able to detect this type of ambiguity. We conducted a semi-automatic detection, using QuARS to search for potential passive items (i.e. occurrences of *is, are, was, were*) and manually filtering out the active forms. At this point we have continued as above with the review and assessment steps, classifying passive voices as: false positive, variability, or ambiguity. Due to the significant manual workload for passive voices detection, we fully analysed the shorter documents (Library, Blit, ERTMS, DigitalHome), and in the case of Eirene and People by Temperament we decided to limit ourselves to the first 100 requirements (out of 475 and 185, resp.), considering this a reasonable threshold.

The review and assessment steps, which highlight variation points, are based on the criteria introduced above, and the analysis of the requirements found ambiguous by the tool is guided by the general question "Can we envision different products?" More concrete review guidelines may depend on the indicators. For example, with *under-specification* and *vagueness* the criterion for identifying a variability is the existence of more than one possible instance of the defective term. The *disjunction* usually indicates a variability if the coordinating particle "or" ("and/or") relates two nouns, whereas it is likely to be a false positive if the particle connects two sentences or two adjectives. *Weakness* and *optionality* are inherently associated to variation points, especially when they appear in functional requirements (Fantechi et al., 2018a).

### 6.2. Data analysis

Once we have collected the data, we can measure the adequacy of VIBE to identify variation points. As a measure we use *precision*, i.e. the fraction of relevant instances among the retrieved instances. Precision general formula is:

$$prec = \frac{tp}{fnd}$$

where $fnd = tp+fp$ is the sum of true positives and false positives (resp.), and $tp = amb + var$. Precision measures the relevance of the indicators for variability detection. We consider the following precision measures:

- *p_amb* denotes the precision in identifying ambiguities:

$$p\_amb = \frac{amb}{fnd}$$

**Table 13**
Detailed results of the quantitative analysis: ambiguity classes and their relevance for variability extraction. In the first column: fnd=number defects found; fp=number of false positives; amb=number of pure ambiguities; var=number of variation points. The aggregate values for *fnd*, *fp*, *amb*, and *var* in the rightmost column are the sum of the values in the row. The aggregate derived measures are calculated on these sums.

| | LIBRARY 94 reqs | EIRENE 475 reqs | BLIT 49 reqs | ERTMS 55 reqs | PeopleByT 185 reqs | DigHome 112 reqs | Aggregate |
|---|---|---|---|---|---|---|---|
| **Disjunction** | | | | | | | |
| fnd | 25 | 91 | 3 | 10 | 23 | 30 | 182 |
| fp | 14 | 71 | 2 | 6 | 13 | 17 | 123 |
| amb | 0 | 7 | 0 | 0 | 1 | 0 | 8 |
| var | 11 | 13 | 1 | 4 | 7 | 13 | 49 |
| p_amb | 0,000 | 0,077 | 0,000 | 0,000 | 0,043 | 0,000 | 0,044 |
| p_var | 0,440 | 0,143 | 0,333 | 0,400 | 0,304 | 0,433 | 0,269 |
| rp_var | 1,000 | 0,650 | 1,000 | 1,000 | 0,875 | 1,000 | 0,860 |
| **Escape clauses (optionality)** | | | | | | | |
| fnd | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Vagueness** | | | | | | | |
| fnd | 24 | 163 | 7 | 2 | 41 | 35 | 272 |
| fp | 5 | 113 | 3 | 2 | 36 | 24 | 183 |
| amb | 18 | 44 | 4 | 0 | 3 | 4 | 73 |
| var | 1 | 6 | 0 | 0 | 2 | 7 | 16 |
| p_amb | 0,750 | 0,270 | 0,571 | 0,000 | 0,073 | 0,114 | 0,268 |
| p_var | 0,042 | 0,037 | 0,000 | 0,000 | 0,049 | 0,200 | 0,059 |
| rp_var | 0,053 | 0,120 | 0,000 | -- | 0,400 | 0,636 | 0,180 |
| **Weakness** | | | | | | | |
| fnd | 38 | 48 | 2 | 4 | 32 | 10 | 134 |
| fp | 26 | 32 | 2 | 0 | 13 | 4 | 77 |
| amb | 8 | 2 | 0 | 0 | 6 | 5 | 21 |
| var | 4 | 14 | 0 | 4 | 13 | 1 | 36 |
| p_amb | 0,211 | 0,042 | 0,000 | 0,000 | 0,188 | 0,500 | 0,157 |
| p_var | 0,105 | 0,292 | 0,000 | 1,000 | 0,406 | 0,100 | 0,269 |
| rp_var | 0,333 | 0,875 | -- | 1,000 | 0,684 | 0,167 | 0,632 |
| **Quantifiers** | | | | | | | |
| fnd | 19 | 106 | 17 | 6 | 33 | 19 | 200 |
| fp | 13 | 92 | 11 | 2 | 22 | 13 | 153 |
| amb | 6 | 12 | 5 | 4 | 11 | 5 | 43 |
| var | 0 | 2 | 1 | 0 | 0 | 1 | 4 |
| p_amb | 0,316 | 0,113 | 0,294 | 0,667 | 0,333 | 0,263 | 0,215 |
| p_var | 0,000 | 0,019 | 0,059 | 0,000 | 0,000 | 0,053 | 0,020 |
| rp_var | 0,000 | 0,143 | 0,167 | 0,000 | 0,000 | 0,167 | 0,085 |
| **Under-specification** | | | | | | | |
| fnd | 6 | 47 | 1 | 0 | 1 | 10 | 65 |
| fp | 6 | 41 | 1 | 0 | 1 | 9 | 58 |
| amb | 0 | 5 | 0 | 0 | 0 | 1 | 6 |
| var | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| p_amb | 0,000 | 0,106 | 0,000 | -- | 0,000 | 0,100 | 0,092 |
| p_var | 0,000 | 0,021 | 0,000 | -- | 0,000 | 0,000 | 0,015 |
| rp_var | -- | 0,167 | -- | -- | -- | 0,000 | 0,143 |
| **Passive voice** | | | | | | | |
| fnd | 8 | 6 | 4 | 6 | 9 | 10 | 43 |
| fp | 4 | 6 | 3 | 6 | 8 | 7 | 34 |
| amb | 4 | 0 | 1 | 0 | 1 | 3 | 9 |
| var | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| p_amb | 0,500 | 0 | 0,250 | 0,000 | 0,111 | 0,300 | 0,209 |
| p_var | 0,000 | 0 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 |
| rp_var | 0,000 | -- | 0,000 | -- | 0,000 | 0,000 | 0,000 |

- *p_var* denotes the precision in identifying variabilities:

$$p\_var = \frac{var}{fnd}$$

- *rp_var* denotes the relative precision of variability vs. ambiguity, i.e. it measures to what extent the NLP tool returns results that, when *true positives*, are relevant for variability:

$$rp\_var = \frac{var}{tp}$$

These values are reported in the 5th, 6th, and 7th row of each case study in Table 13.

We can now analyse the data in Table 13 that addresses the *V1_Narrowing* issue. To support our argumentation, we provide examples taken either from the running case studies or from the six third party requirements documents.

When considering *disjunction*, there are many false positives. However, once discarded the false positives, we have a high number of variabilities, with an average relative precision *rp_var* of 0.86 telling that disjunction is more likely to be an indicator of variability than ambiguity. During the validation, we have also observed that disjunctions between noun phrases (as in Ex1 below) are more likely a variability than disjunctions between verb phrases (Ex2).

> Ex1. Alarm contact switches shall be used to monitor entry through a door or window when the switch is active.
> Ex2. A Master User shall be able to add a user account or change the default parameter settings.

In the considered documents, we did not find any *escape clause*, detected by QuARS *optionality* class. As a matter of fact, we have only taken into account final endorsed documents, where it is difficult to find such obvious defects as "if possible" or "if necessary". We consider however that, if present, escape clauses are an indicator of variability.

We recommend the inclusion of escape clauses among the indicators to be analysed by the NLP tool because escape clauses may be present in some requirements documents, for instance in those under development. This does not represent an overhead

**Table 14**
Detailed results of the quantitative analysis: new ad hoc indicators. In the first column: fnd=number defects found; fp=number of false positives; var=number of variation points; constraints=number of inter-feature constraints. The value of p_const is obtained as constraints/fnd.

| | LIBRARY 94 reqs | EIRENE 475 reqs | BLIT 49 reqs | ERTMS 55 reqs | PeopleByT 185 reqs | DigHome 112 reqs | Aggregate |
|---|---|---|---|---|---|---|---|
| **Variability** | | | | | | | |
| fnd | 4 | 31 | 2 | 0 | 0 | 1 | 38 |
| fp | 3 | 16 | 1 | -- | -- | 1 | 21 |
| var | 1 | 15 | 1 | -- | -- | 0 | 17 |
| p_var | 0,250 | 0,484 | 0,500 | -- | -- | 0,000 | 0,447 |
| **Constraints** | | | | | | | |
| fnd | 3 | 20 | 0 | 0 | 6 | 3 | 32 |
| fp | 3 | 19 | -- | -- | 6 | 2 | 30 |
| constraints | 0 | 1 | -- | -- | 0 | 1 | 2 |
| p_const | 0,000 | 0,050 | -- | -- | 0,000 | 0,333 | 0,063 |

**Table 15**
Aggregated data of Table 13: all indicators set vs. the *significant* subset {disjunction, vagueness, weakness, escape clauses}.

| | Indicators | fnd | fp | amb | var |
|---|---|---|---|---|---|
| Eirene | All classes | 461 | 355 | 70 | 36 |
| | Significant | 302 | 216 | 53 | 33 |
| Library | All classes | 120 | 68 | 36 | 16 |
| | Significant | 87 | 59 | 26 | 16 |
| Blit | All classes | 34 | 22 | 10 | 2 |
| | Significant | 12 | 7 | 4 | 1 |
| ERTMS | All classes | 28 | 16 | 4 | 8 |
| | Significant | 16 | 8 | 0 | 8 |
| People By T. | All classes | 139 | 93 | 22 | 22 |
| | Significant | 96 | 62 | 10 | 22 |
| DigitalHome | All classes | 114 | 74 | 18 | 22 |
| | Significant | 75 | 45 | 9 | 21 |

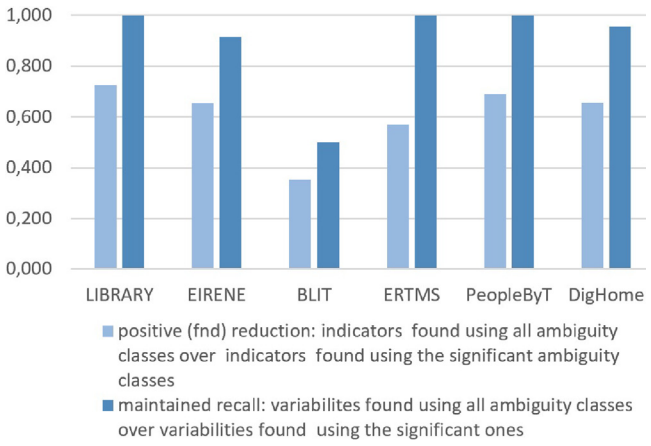



**Fig. 4.** Performance improvement by narrowing the ambiguity indicators: the number of found indicators occurrences decreases by ∼35% with a negligible loss of the variabilities detected. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

since, if no escape clause is present in a document, the very efficient automatic analysis returns an empty list of indicators and there is no cost for the manual analysis of the detected terms.

*Vagueness* is due to the presence of undetermined adjectives and adverbs and can mask a variability, albeit in a few cases. For instance terms like *similarly, accordingly,* or *completed* are true ambiguity indicators (Ex3) while terms like *suitable, various, clear* likely indicate variability (Ex4).

> Ex3. The digitalhome shall be equipped with various environmental controllers and sensors.
> Ex4. A visual confirmation […] shall be given to the driver when a […] network change has been completed successfully.

A further category is *weakness*: in this case, a good percentage of the defective sentences revealed by the tool contains a variation point. We also report that variability is due to *may* (Ex5) more than to *can*: indeed the latter is typically used in requirements beginning with "The user can …", indicating that the system must enable the user to perform a given action (Ex.6)

> Ex5. After delivery, the system may enable the users to enter their reviews or ratings.
> Ex6. The shunting leader can then choose the moment when he takes or rejects the call.

In the considered documents, *under-specification* defects are in most cases false positives and we only found one variability: in Ex7 below the buttons to adjust can vary.

> Ex.7 the driver shall be able to adjust the brightness of buttons […]

We finally consider the *passive voices*: the number of occurrences found is quite low, and among them none is an indicator of variability.

From the data analysis we observe that it is appropriate to exclude quantifiers, under-specification and passive voices from the search for variability because they have been shown not to be relevant. Also, the two latter are costly in terms of execution time, requiring syntactic analysis.

To conclude, we also examine the *V1_Narrowing* issue with a different data aggregation policy: in Table 15 we show, for each analysed requirement document:

– the total number of *fnd, fp, amb*, and *var*, considering all ambiguity classes,

– the total number of *fnd, fp, amb*, and *var*, considering only significant ambiguity classes: vagueness, disjunction, escape clauses, and weakness.

The comparison between the first and second row of each case study confirms the higher performance of these four ambiguity classes to detect variability. In Fig. 4 we graphically render this observation. Indeed, by narrowing down the analysis to the four significant ambiguity classes on the one side we can reduce significantly the number of ambiguity indicators found (light blue bar), on the other side we leave almost unchanged the number of variabilities detected (dark blue bar). The only exception is BLIT, where recall is 0,5 only. However, looking at the rightmost column of Table 13, this value is generated by the ratio between two numbers (2 and 1, resp.) that are not statistically relevant.

We now consider the *V2_widening* issue by taking into account the results relative to the new dictionaries, given in Table 14, where, for each indicator we report the number of: *fnd*, *fp*, *var* and inter-feature *constraints*.

**Table 16**
Variability classes and indicators.

| Classes | Indicators |
|---|---|
| **Vagueness** A vague term is likely a variability when it can have different instances | *clear, easy, strong, good, bad, adequate, various, completed, accordingly, …* |
| **Disjunctions** A requirement containing a disjunction potentially specifies different products: implementing the first, the second or both disjuncts | *or, and/or, or/and* especially between noun phrases |
| **Escape clauses** A requirement potentially specifies different products, implementing or not the object of the escape clause | *case, possibly, if possible, if appropriate, where permitted, among others, as a minimum, shall be included if required, when requested by, when required, …* |
| **Weakness** A requirement containing weak verbs potentially specifies different products, implementing or not the object of the weak verb | *may, can, could, …* where *may* is more likely a variability than *can, could*, and requirements containing "the user can" do not have to be considered |
| **Variability** terms and constructs that explicitly refer to variability | terms like: *configuration, feature, range* or *dependent* clauses containing both a conjunction *if, where, when, whether, …* and a verb like *available, provided, implemented* |
| **Inter-feature constraints** A requirement may indicate some constraints expressing implication or mutual exclusion relations between features | *imply, require, entail, implicate, demand, exclude, rule out, mutually exclusive, need, …* |

Observe that we do not have to consider ambiguities here. The precision measures are reported in the last raw of each block: *p_var* and *p_const*, respectively.

When using the *variability dictionary*, we have found a limited amount of indicators in the documents, however the precision value of 0.44 is fairly good. We must point out that the manual revision here took longer than expected due to a limitation of the syntactic analysis capabilities of QuARS, which cannot decompose sentences into clauses and recognize subordinate clauses. Manual screening was done by looking at requirements that contained both a term between *if, where, when, whether, …* and a term between *available, provided, implemented* in the same subordinate clause.

As far as the *constraint* dictionary is used, the results, with an overall precision of 0.063, tell us that this indicator is not as interesting as we expected. We believe that this indicator should be however included in the analysis since we want to avoid false negatives.

Summing up, in Table 16 we collect the ambiguity and ad hoc indicators that we have shown to most likely indicate variability.

## 7. From detected variabilities to SPL variation points

According to Fig. 1, the VIBE process consists of three steps. So far we have described the first two: (i) detecting candidate variabilities in a requirements document with an NLP tool; (ii) distinguishing variabilities from ambiguities and false positives with a manual review to obtain a list of identified variabilities.

The third and last step of the process is to associate variation points to these identified variabilities.

A well known language to graphically model variation points in the context of SPL is given by the *feature diagrams* (Kang et al., 1990) and we discuss how to map the identified variabilities into the variation points in this graphic language.

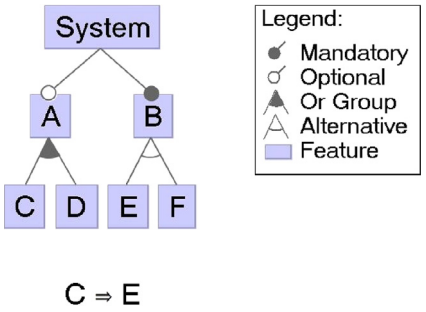



**Fig. 5.** Example of feature diagram.

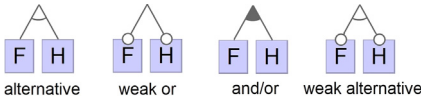



**Fig. 6.** The four interpretation of disjunction in FD notation.

### 7.1. Feature diagrams

Feature diagrams (FD) have been proposed by Kang et al. (1990) and are a graphical formalism to describe the different possible variants of a system in a reader friendly, compact, and uniform way. In the graphical formalism features are represented as nodes in a tree, a necessary condition for a feature to be present in a variant is that the parent feature is present too, and arc labelling is used to specify sufficient conditions, that represent the variation points of the product line. In Fig. 5, we show an exemplary feature diagram: Feature A is optional, and if it is present then at least one among C and D must be present too. Feature B is mandatory, and one and only one of its children features E and F must be present in each product.

Requires and Excludes are *cross-tree constraints*: they can relate non sibling features and there is no graphic symbol to represent these variation points, propositional logic formulae are used instead.[3] In the example, the presence of C in a product implies the presence on E in the same product.

### 7.2. The mapping to variation points

We present here how the identified variabilities can be mapped to variation points using the feature diagrams modelling capabilities. We discuss these mappings referring to the running examples, that all together cover all the ambiguity classes relevant for variability.

#### 7.2.1. Disjunction

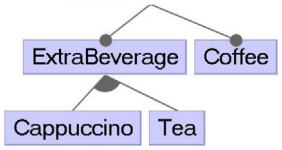
A disjunction can have four possible interpretations: *exclusive or*, *weak or*, *and/or*, and *weak alternative*. The first interpretation corresponds to an *alternative*, one and only one feature is present. The second admits zero, one or more features and corresponds to as many *optional* features as there are disjuncts. In the third case at least one of the features must be present in each product, i.e. there is an *or* variation point. Finally, in the fourth interpretation all the features are optional, but no more than one is present. In terms of feature diagrams the four interpretations are represented, respectively, as depicted in Fig. 6.

We show the application of this pattern taking the requirements from the case studies containing disjunctions.

[3] The diagrams in the paper have been drawn using FeatureIDE (Thüm et al., 2014). Other tools use a graphical notation for cross-tree constraints as well.

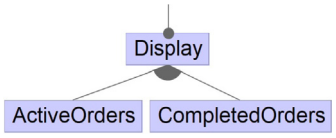
**Coffee machine Rc2:** The machine shall offer, as beverages, coffee and cappuccino or tea.

We remind that the coffee shall be offered in each machine and hence this should be represented by means of a mandatory feature. The interpretation of this requirement hence led to foresee the realization of three possible products, offering, besides coffee, respectively: tea, cappuccino, tea + cappuccino. We hence model the requirement using an *and/or* construct:



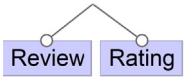

**E-shop Re6:** The system shall display the list of active and/or the list of completed orders in the customer profile.

This requirement can be naturally interpreted and modelled as an *and/or*:




**E-shop Re16:** After delivery, the system may enable the users to enter their reviews or ratings.

This requirement admits four interpretations, review, rating, both, and no feedback, that we model with a *weak or*:

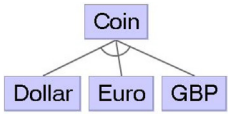



*7.2.2. Vagueness*

In general, a vague term abstracts from a set of instances, and the process of requirement review will make these instances explicit. A vague term can thus correspond to a feature having as many child nodes as the considered instances, and these are in disjunction.

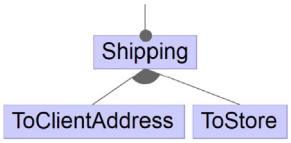
**Coffee machine Rc1:** After inserting a suitable coin [...]

Suitable coin is a vague expression, which has been clarified with the customer by indicating that each machine accepts only one of dollars, pounds or euros. The vague term suitable corresponds in this case to an *alternative* variation point. We can represent this choice with the following diagram:




**E-shop Re10:** The system shall display various shipping methods.

The analysis of this vagueness, also using the information contained in Re11, led to the conclusion that an e-shop offers one or both (*and/or*) of the following shipping methods: ToClientAddress and ToStore.




*7.2.3. Escape clauses*

A requirement containing an escape clause, naturally leads to an *optional* feature.

**Coffee machine Rc4:** A ringtone possibly has to be played after beverage delivery.

The requirement says that a ringer is an optional feature of the coffee machine:


**E-shop Re3:** The system possibly notifies with a pop-up the user when no matching product is found on the search.


*7.2.4. Weakness*

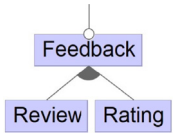
Similarly to escape clauses, an *optional* feature is introduced when a requirement includes a variability because of a weak verb such as *may, can*.

**E-shop Re13:** The system may display the current tracking information about the order.

The requirement says that providing tracking information is an optional feature.


**E-shop Re16:** After delivery, the system may enable the users to enter their reviews or ratings.

The requirement specifies that both review and rating insertion are optional (as shown above in the *disjunction* paragraph). This requirement can also be modelled adding an auxiliary feature, a kind of *pure fabrication*, that models the fact that a product may or may not offer a feedback feature and, if so, it allows for a review, rating, or both.




*7.2.5. Variability*

This class of indicators contains terms and constructs that explicitly refer to variability. These are dependent clauses containing both a conjunction *if, where, when, whether, ...* and a verb like *available, provided, implemented*.

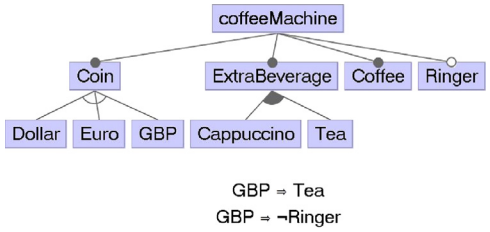
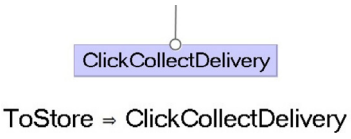



**Fig. 7.** Feature diagram for the coffee vending machine.

**E-shop Re11:** The order shall be shipped to the client address or, if the click&collect service is available, to an associated store.

The interpretation is that click&collect is an optional service and it is a necessary condition for shipping to store.




*7.2.6. Inter-feature constraints*

A requirement that contains an inter-feature constraint expresses necessity or mutual exclusion relations between features. If the features involved are *close relatives* (father and son or siblings), then cross tree constraints are modelled with the graphical notation, otherwise propositional logic is used.

**Coffee machine Rc6:** The British market requires tea and excludes any ring tone.

The constraints can be represented with the following propositional logic formulas:

$$GBP \Rightarrow Tea$$
$$GBP \Rightarrow \neg Ringer$$

*7.3. Building a feature diagram*

The mapping patterns in the previous section make it possible to systematically construct fragments of a feature diagram. These can then be joined together to build the model of the product line under consideration. This step requires the presence of a domain expert and is not automated. However, a first feature diagram draft can be built mechanically by assembling (as siblings) the fragments built so far, and defining them as children of a feature named after the product line.

This is how we have built the feature diagrams of the coffee machine (Fig. 7) and of the e-shop (Fig. 8).

When the root of a fragment A is the leave of another fragment, B, then fragment A is attached as subtree of B.

## 8. Threats to validity

*Construct Validity.* An objective and widely used metric, i.e., precision, was used to assess the appropriateness of the examined indicators in identifying variabilities. We limited the experimental analysis to a subset of the ambiguity classes listed in Table 1: as discussed in Section 3 some classes are clearly not relevant for variability.

The data used to calculate the precision are based on subjective evaluations; to mitigate subjectivity threats, in Fantechi et al. (2018b) we assessed the agreement among annotators by means of the Fleiss Kappa measure, which indicated a moderate agreement. Similarly to other works (Femmer et al., 2017), we arguably consider this value acceptable in the inherently ambiguous context of NL requirements. Furthermore, the final data set used for analysis is the result of an assessment of the independent review outcome.


**Fig. 8.** Feature diagram for the eshop.

It is worth noting that we did not use the recall measure to assess our results. Therefore, some variability indicators might exist in the requirements that are not discovered with our analysis. To calculate recall we would have needed ground truth feature diagrams associated to the case studies: this way, we could have counted the false negatives, i.e. variabilities not discovered by VIBE. Unfortunately, we did not succeed in finding large case studies including NL requirements and a corresponding feature diagram. Nevertheless, in an effort to mitigate this threat, we have applied VIBE to the example in Sree-Kumar et al. (2018), finding the same variabilities. In addition, we have asked an independent SPL expert to draw the feature diagrams starting form the requirements of coffee machine and e-shop examples.[4] Also in this case VIBE was able to detect all the variabilities. Unsurprisingly, by its very nature VIBE missed to detect some mandatory features since they are specified in requirements in unambiguous and assertive propositions.

*Internal Validity.* The main threat to the internal validity of the study is the involvement of the authors of this work in the Review and Assessment phase of the data collection procedure. We agree that the researcher bias might have played a role in these phases. However, this is mitigated by the agreement measure. Furthermore, other researchers can replicate our process using the publicly available QuARS tool, and using the documents employed in our evaluation.

*External Validity.* Our results are limited to six requirements documents. However, the documents are very different from each other in terms of length, domain, characteristics of the systems, elaboration and validation stages and authors' background, academic and industrial. Despite all this diversity, we observed that several variability-related terms are common among the documents. Therefore, we argue that our study has the potential to be generalized to other domains, and other requirements documents.

## 9. Related work

We classify the work related to VIBE according to different dimensions, namely ambiguity detection in NL requirements; variability detection and feature identification from NL documents.

[4] Hand-drawn diagrams can be found at https://github.com/Vibe-NLP/RequirementsForValidation.

*Ambiguity detection in NL requirements.* is a lively research field, with several contributions published already in the nineties (e.g. the ARM tool Wilson et al., 1997, LOLITA Mich, 1996 and CIRCE Ambriola and Gervasi, 1997).

The research in this field had continued in the following decade with the classification of the typically defective terms and constructions in the ambiguity handbook of Berry et al. (2003b) and with new rule-based NLP tools such as QuARS (Gnesi et al., 2005), SREE (Tjong and Berry, 2013) and the tool of Gleich et al. (2010).

A more recent tool that employs rule-based approaches to detect typical ambiguous terms and constructions is RETA (REquirements Template Analyzer) (Arora et al., 2015), which, in addition, checks the conformance of the requirements to a given template. Industrial applications of these approaches were studied by Femmer et al. (2017) and by Rosadini et al. (2017).

Rule-based approaches tend to produce a high number of false positive cases. Hence, *statistical* approaches, accompanied by prototypical tools, were proposed by Chantree et al. (2006) and by Yang et al. (2010) to reduce the number of false positive cases, referred as *innocuous ambiguities*. Statistical NLP approaches are also used in Ferrari et al. (2017a), to identify domain-dependent ambiguities, i.e., pragmatic ambiguities that depend on the domain background of the reader of the requirements.

However, statistical approaches are built on the expertise gained over the past decades in the detection of ambiguities in requirements, expertise that is not comparable to that gained in the younger field of variability detection. For the purposes of this work, which is experimental and pioneering in adapting tools created to detect ambiguity to variability search, we preferred to use rule-based approaches, which are more easily customizable.

*Variability detection and feature identification.* A pioneering work, presented by Acher et al. (2012), is based on searching for variability patterns within tables in which the description of the products are stored in a semi-structured manner. Instead, we are dealing with the extraction of variability from unstructured NL requirements documents.

Another line of research focuses on variability extraction from a set of independent requirements documents (or product brochures/flyers) that belong to a common domain. They perform a semantic analysis based on the construction of a similarity matrix or on a contrastive analysis with the aim of detecting, in the considered documents, commonalities and variabilities of the systems under examination. Ferrari et al. (2013, 2015) extract domain-specific terms from product descriptions belonging to different vendors, to identify common and variant domain terms, which can be used as pointers for product commonalities and variabilities. Nasr et al. (2017) leverage on an analogous approach and derive comparison matrices for different products. Similar techniques for synthesizing a feature models using comparison matrices and requirements similarity identification and clustering are in Nasr et al. (2014). A recent reverse engineering based approach, that applies clustering techniques, is used to compute the relations between features by Li et al. (2018), having as input the requirement specifications from different variants and a mapping between requirements and variants. We are interested instead in extracting the variability hidden in a single requirements document and to explore the use of lightweight lexical and syntactic NL techniques.

Other approaches have been defined to identify requirements similarities: requirements are then grouped together according to their similarity, letting each group of requirements represent a feature. These techniques analyse term frequency and exploit clustering techniques for feature identification, aided with syntactic and semantic analysis. This is the research line taken by Weston et al. (2009), Chen et al. (2005), Alves et al. (2008) and Niu and Easterbrook (2008a,b). Itzik et al. (2016) present instead the SOVA approach, which leverages semantic ontologies to extract features from requirements and derive a feature model. Our work differs from all of these contributions in that it has a focus on the variability identification and applies a completely different technique, based on ambiguity detection simply using lexical and syntactic analysis.

In a less related line of research Bécan et al. (2016) define a generic, ontologic-aware synthesis procedure that computes the likely siblings or parent candidates for a given feature. They develop six heuristics for clustering and weighting the logical, syntactical and semantical relationships between feature names. Vacchi et al. in Vacchi et al. (2014) present a clustering procedure to derive a variability model from a set of already implemented language components, in the context of (domain-specific) programming languages. The latter research papers are marginally related to ours in that they consider an input that is not a NL requirements document.

More recently Sree-Kumar et al. (2018) propose a framework, FeatureX, for identifying features and extracting feature relationships from NL requirements and in Sree-Kumar et al. (2021) present a method for the validation of a feature model with respect to the textual specifications from which it has been generated, using NLP techniques and supervised learning. The scope of their work is close to ours, but the solution shows some differences: FeatureX first identifies candidate features with lexical analysis and machine learning techniques, then looks for the conjunctions between the identified features and to weak and modal verbs as indicators of optionalities, whereas we use a technique that operates in an orthogonal way, looking directly for variation points. Moreover, VIBE integrates research on ambiguity detection with research on identifying points of variation. Indeed, the two approaches are complementary, and it would be of interest to combine them to exploit FetureX ability to extract features and synthesize feature models with VIBE's ability to extract variability information.

Finally, some interesting, though not very recent, literature reviews on feature identification and variability extraction from NL documents have been published by Li et al. (2017) and by Bakar et al. (2015).

## 10. Discussion and conclusions

Ambiguity defects in requirements can give an indication of variability, related to possible design or implementation choices, or configurability aspects. In fact the ambiguity defects that are found in a requirement document may be due to intentional or unintentional references to issues that can be solved in different ways, possibly envisioning a family of different products rather than a single product.

In this paper we have defined VIBE, a tool-supported process for variability identification. The first step of the process is the application of a NLP tool for ambiguity detection, customized to identify variability. We have compared a set of academic and commercial NLP tools to understand which classes of ambiguity are detected by the different tools and if they offer customization (selection and modification of indicators) features. Then, to validate the VIBE process, we have used one of these tools, namely QuARS.

On the basis of the experience made – the analysis of six third-party requirements documents – we can assert that some linguistic defects that are recognized as source of ambiguity are also relevant to reveal a possible variation point. These are: disjunctions, weakness, and vagueness. With regard to escape clauses, we found no related indicators in the requirements documents considered. This is not surprising since they were established

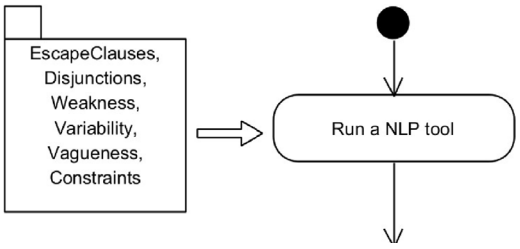



**Fig. 9.** Dictionaries to be used in the first VIBE step.

and validated documents. Actually, escape clauses are a natural indicator of variability, as we have seen with the vending machine and e-shop examples. We certainly recommend including escape clauses among the analyses to be done in search of indicators of variability. In the light of this experience we indicate in Fig. 9 which classes of indicators to use during the first step of VIBE to maximize precision while minimally affecting completeness.

A further consideration concerns the NLP tools used for the validation phase: we have limited ourselves to the use of QuARS because we had observed that the classes of defects analysed by the different tools were quite uniform; moreover only some tools had an academic licence and therefore the analysis would have been partial anyway. We recall that the purpose of the validation was not to assess the performance of QuARS in detecting variability, but to show which, if any, ambiguity indicators are useful to detect variation points.

From the comparison work between the tools, we learned another lesson, that allows us to define the requirements for a tool tailored for variability detection:

- have good built in dictionaries, especially for the classes of ambiguity relevant for variability;
- allow the user to select only the analyses of interest;
- allow the user to edit the dictionaries;
- allow the user to define new syntactic controls.

Based on these requirements we are currently re-engineering QuARS and implementing Toody, a new NLP tool to explicitly address variability mining, Fantechi et al. (2021), and we plan in the near future to use it in VIBE. Here we have used the established and available tool to make the experiments repeatable.

As a conclusion, the VIBE process proves useful to identify the variability of a SPL by detecting the ambiguities of a NL requirement document. To complete a product line specification, we also need to detect the mandatory features that occur in the requirements in clear and assertive propositions, i.e. the commonalities. To this purpose VIBE can be conveniently coupled with existing tools for feature extraction which exploit semantic analysis. A possible candidate is the Requirements Glossary term Identification and Clustering (REGICE) (Arora et al., 2017) tool: we have successfully followed this combined approach in Arganese et al. (2020), where we take advantage of the capabilities of REGICE to extract and cluster the glossary terms from the requirements documents, accordingly to the intuition that glossary terms correspond to features, with enough precision.

Another fruitful future research direction would be to extend VIBE to extract cardinality indicators or feature priority information, and exploit the more expressive variants of feature diagrams, e.g. cardinality-based feature modelling (Czarnecki et al., 2005) or attributed feature models (Bak et al., 2016). These enriched models support the decision-making process on which products in an SPL are most cost-effective to implement and distribute.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

All the data are available at https://github.com/Vibe-NLP/RequirementsForValidation.

## Acknowledgements

We gratefully thank Alessio Ferrari for the collaboration in previous work on this topic, Eleonora Arganese and Monica Arrabito for their work during the graduation project.

We gratefully thank the developers of the tools QVscribe and Requirements Scout that provided us with the licence for academic purposes.

The research has been partially supported by the MIUR, Italy project PRIN 2017 FTXR7S "IT-MaTTerS" (Methods and Tools for Trustworthy Smart Systems).

## References

Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., Lahire, P., 2012. On extracting feature models from product descriptions. In: Proc. of VaMoS '12. pp. 45–54.

Alves, V., Schwanninger, C., Barbosa, L., Rashid, A., Sawyer, P., Rayson, P., Pohl, C., Rummler, A., 2008. An exploratory study of information retrieval techniques in domain analysis. In: Proc. of SPLC '08. pp. 67–76.

Ambriola, V., Gervasi, V., 1997. Processing natural language requirements. In: 1997 International Conference on Automated Software Engineering, ASE 1997, Lake Tahoe, CA, USA, November 2-5, 1997. IEEE Computer Society, pp. 36–45.

Apel, S., Batory, D.S., Kästner, C., Saake, G., 2013. Feature-Oriented Software Product Lines - Concepts and Implementation. Springer.

Apel, S., Kästner, C., 2009. An overview of feature-oriented software development. J. Object Technol. 8 (5), 49–84.

Arganese, E., Fantechi, A., Gnesi, S., Semini, L., 2020. Nuts and bolts of extracting variability models from natural language requirements documents. In: Integrating Research and Practice in Software Engineering. In: Studies in Computational Intelligence, vol. 851, Springer, pp. 125–143.

Arora, C., Sabetzadeh, M., Briand, L., Zimmber, F., 2017. Automated extraction and clustering of requirements glossary terms. IEEE Trans. Softw. Eng. 43 (10), 918–945.

Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., 2015. Automated checking of conformance to requirements templates using natural language processing. IEEE Trans. Softw. Eng. 41 (10), 944–968.

Arrabito, M., Fantechi, A., Gnesi, S., Semini, L., 2020. A comparison of NLP tools for RE to extract variation points. In: Proceeddings of NLP4RE 2022 Co-Located with the 26th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2020), Pisa, Italy, March 24, 2020. In: CEUR Workshop Proceedings, vol. 2584, CEUR-WS.org.

Bak, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., Wasowski, A., 2016. Clafer: unifying class and feature modeling. Softw. Syst. Model. 15 (3), 811–845.

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

Bécan, G., Acher, M., Baudry, B., Nasr, S.B., 2016. Breathing ontological knowledge into feature model synthesis: an empirical study. Empir. Softw. Eng. 21 (4), 1794–1841.

Berry, D., Kamsties, E., Krieger, M., 2003a. From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity - A Handbook Version 1.0.

Berry, D.M., Kamsties, E., Krieger, M.M., 2003b. From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity. Tech. Rep., University of Waterloo, Waterloo, ON, Canada.

Bucchiarone, A., Gnesi, S., Pierini, P., 2005. Quality analysis of NL requirements: An industrial case study. In: 13th IEEE International Conference on Requirements Engineering (RE 2005), 29 August - 2 September 2005, Paris. IEEE Computer Society, pp. 390–394.

Capozucca, A., Cheng, B., Georg, G., Guel, N., Istoan, P., Mussbacher, G., 2011. Requirements definition document for a software product line of car crash management systems. [Online]. Available: In ReMoDD repository, at http://www.cs.colostate.edu/remodd/v1/content/bcms-requirements-definition.

Chantree, F., Nuseibeh, B., De Roeck, A., Willis, A., 2006. Identifying nocuous ambiguities in natural language requirements. In: Proceedings of the 14th IEEE International Conference on Requirements Engineering. IEEE, Minneapolis/St.Paul, MN, USA, pp. 59–68.

Chen, K., Zhang, W., Zhao, H., Mei, H., 2005. An approach to constructing feature models based on requirements clustering. In: Proc. of RE'05. pp. 31–40.

Chowdhury, G.G., 2003. Natural language processing. Annu. Rev. Inf. Sci. Technol. 37 (1), 51–89.

Clements, P., Northrop, L.M., 2002. Software Product Lines–Practices and Patterns. In: SEI Sseries in Software Engineering, Addison-Wesley.

Czarnecki, K., Helsen, S., Eisenecker, U.W., 2005. Formalizing cardinality-based feature models and their specialization. Softw. Process. Improv. Pract. 10 (1), 7–29.

Dermeval, D., Vilela, J., Bittencourt, I.I., de Castro, J.B., Isotani, S., da S. Brito, P.H., Silva, A., 2015. Applications of ontologies in requirements engineering: a systematic review of the literature. Requir. Eng. 21, 405–437.

Fabbrini, F., Fusani, M., Gnesi, S., Lami, G., 2001. An automatic quality evaluation for natural language requirements. In: Proceedings of the 7th International Workshop on Requirements Engineering: Foundation for Software Quality, Vol. 1. REFSQ, pp. 4–5.

Fantechi, A., Ferrari, A., Gnesi, S., Semini, L., 2018a. Hacking an ambiguity detection tool to extract variation points: an experience report. In: Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems, VAMOS 2018, Madrid, February 7-9, 2018. ACM pp. 43–50.

Fantechi, A., Ferrari, A., Gnesi, S., Semini, L., 2018b. Requirement engineering of software product lines: Extracting variability using NLP. In: Proceedings of the 26th IEEE International Requirements Engineering Conference 2018, Banff, AB, Canada, August 20-24, 2018. IEEE, pp. 418–423.

Fantechi, A., Gnesi, S., Livi, S., Semini, L., 2021. A spaCy-based tool for extracting variability from NL requirements. In: Mousavi, M., Schobbens, P. (Eds.), SPLC '21: 25th ACM International Systems and Software Product Line Conference, Leicester, United Kindom, September 6-11, 2021, Volume B. ACM, pp. 32–35.

Fantechi, A., Gnesi, S., Semini, L., 2017. Ambiguity defects as variation points in requirements. In: Proceedings of the Eleventh International Workshop on Variability Modelling of Software-Intensive Systems. VAMOS '17, ACM, Eindhoven, pp. 13–19.

Fantechi, A., Gnesi, S., Semini, L., 2019. Applying the QuARS tool to detect variability. In: Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume B, Paris, September 9-13, 2019. ACM, pp. 62:1–62:4.

Femmer, H., 2018. Requirements quality defect detection with the Qualicen requirements scout. In: Joint Proceedings of REFSQ-2018 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track Co-Located with the 23rd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2018), Utrecht, the Netherlands, March 19, 2018. In: CEUR Workshop Proceedings, vol. 2075, CEUR-WS.org.

Femmer, H., Fernández, D.M., Wagner, S., Eder, S., 2017. Rapid quality assurance with requirements smells. J. Syst. Softw. 123, 190–213.

Ferrari, A., Donati, B., Gnesi, S., 2017a. Detecting domain-specific ambiguities: an NLP approach based on wikipedia crawling and word embeddings. In: Proceedings of 4th International Workshop on Artificial Intelligence for Requirements (AIRE), IEEE 25th International Requirements Engineering Conference Workshops. IEEE, pp. 393–399.

Ferrari, A., Spagnolo, G.O., Dell'Orletta, F., 2013. Mining commonalities and variabilities from natural language documents. In: Proceedings of the 17th International Software Product Line Conference, SPLC 2013, Tokyo - August 26 - 30, 2013. pp. 116–120.

Ferrari, A., Spagnolo, G.O., Gnesi, S., 2017b. Towards a dataset for natural language requirements processing. In: Joint Proceedings of REFSQ-2017 Workshops, Co-Located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2017). In: CEUR Workshop Proceedings, vol. 1796, CEUR-WS.org.

Ferrari, A., Spagnolo, G.O., Gnesi, S., Dell'Orletta, F., 2015. CMT and FDE: tools to bridge the gap between natural language documents and feature diagrams. In: Schmidt, D.C. (Ed.), Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015. ACM, pp. 402–410.

Gervasi, V., Ferrari, A., Zowghi, D., Spoletini, P., 2019. Ambiguity in requirements engineering: Towards a unifying framework. In: From Software Engineering to Formal Methods and Tools, and Back - Essays Dedicated to Stefania Gnesi on the Occasion of her 65th Birthday. In: Lecture Notes in Computer Science, vol. 11865, Springer, pp. 191–210.

Gleich, B., Creighton, O., Kof, L., 2010. Ambiguity detection: Towards a tool explaining ambiguity sources. In: Requirements Engineering: Foundation for Software Quality, 16th International Working Conference, REFSQ 2010, Essen, Germany, June 30 - July 2. In: LNCS, vol. 6182, Springer, pp. 218–232.

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

IBM, Engineering Requirements Quality Assistant (RQA). [Online]. Available: www.ibm.com/products/requirements-quality-assistant.

INCOSE, 2019. Guide for Writing Requirements, TechGuideWR2019Soft V3.

Itzik, N., Reinhartz-Berger, I., Wand, Y., 2016. Variability analysis of requirements: Considering behavioral differences and reflecting stakeholders' perspectives. IEEE Trans. Softw. Eng. 42 (7), 687–706.

Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S., 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. Rep., Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

Kasser, J., 2022. TIGER-PRO. [Online]. Available: www.therightrequirement.com.

Kenney, O., Cooper, M., 2020. Automating requirement quality standards with QVscribe. In: Joint Proceedings of REFSQ-2020 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track Co-Located with the 26th International Conference on Requirements Engineering: Foundation for Software Quality. In: CEUR Workshop Proceedings, vol. 2584, CEUR-WS.org.

Lami, G., Fusani, M., Trentanni, G., 2019. QuARS: A pioneer tool for NL requirement analysis. In: From Software Engineering to Formal Methods and Tools, and Back - Essays Dedicated to Stefania Gnesi on the Occasion of her 65th Birthday. In: Lecture Notes in Computer Science, vol. 11865, Springer pp. 211–219.

Li, Y., Schulze, S., Saake, G., 2017. Reverse engineering variability from natural language documents: A systematic literature review. In: Proceedings of the 21st International Systems and Software Product Line Conference-Volume A. ACM, pp. 133–142.

Li, Y., Schulze, S., Saake, G., 2018. Reverse engineering variability from requirement documents based on probabilistic relevance and word embedding. In: Proceedings of the 22nd International Systems and Software Product Line Conference, SPLC'18. ACM, pp. 121–131.

Mich, L., 1996. NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA. Nat. Lang. Eng. 2 (2), 161–187.

Nasr, S.B., Bécan, G., Acher, M., Filho, J.B.F., Sannier, N., Baudry, B., Davril, J., 2017. Automated extraction of product comparison matrices from informal product descriptions. J. Syst. Softw. 124, 82–103.

Nasr, S.B., Sannier, N., Acher, M., Baudry, B., 2014. Moving toward product line engineering in a nuclear industry consortium. In: Gnesi, S., Fantechi, A., Heymans, P., Rubin, J., Czarnecki, K., Dhungana, D. (Eds.), 18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15-19, 2014. ACM, pp. 294–303.

Niu, N., Easterbrook, S.M., 2008a. Extracting and modeling product line functional requirements. In: Proc. of RE'08. pp. 155–164.

Niu, N., Easterbrook, S.M., 2008b. On-demand cluster analysis for product line functional requirements. In: Proc. of SPLC'08. pp. 87–96.

Northrop, L.M., Jones, L.G., 2010. Introduction to software product lines adoption. In: Software Product Lines: Going beyond - 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13-17, 2010. Proceedings. In: Lecture Notes in Computer Science, vol. 6287, Springer, pp. 519–520.

Pohl, K., Böckle, G., van der Linden, F., 2005. Software Product Line Engineering - Foundations, Principles, and Techniques. Springer.

Reuse Company, RAT. [Online]. Available: www.reusecompany.com/rat-authoring-tools.

Rosadini, B., Ferrari, A., Gori, G., Fantechi, A., Gnesi, S., Trotta, I., Bacherini, S., 2017. Using NLP to detect requirements defects: An industrial experience in the railway domain. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, pp. 344–360.

Shahin, R., Chechik, M., Salay, R., 2019. Lifting datalog-based analyses to software product lines. In: Dumas, M., Pfahl, D., Apel, S., Russo, A. (Eds.), Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019. ACM, pp. 39–49.

Sree-Kumar, A., Planas, E., Clarisó, R., 2018. Extracting software product line feature models from natural language specifications. In: Proceeedings of the 22nd International Systems and Software Product Line Conference - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018. ACM, pp. 43–53.

Sree-Kumar, A., Planas, E., Clarisó, R., 2021. Validating feature models with respect to textual product line specifications. In: VaMoS'21: 15th International Working Conference on Variability Modelling of Software-Intensive Systems. ACM, pp. 15:1–15:10.

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T., 2014. FeatureIDE: An extensible framework for feature-oriented software development. Sci. Comput. Program. 79, 70–85.

Tjong, S.F., Berry, D.M., 2013. The design of SREE - a prototype potential ambiguity finder for requirements specifications and lessons learned. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. In: LNCS, vol. 7830, Springer, Essen, Germany pp. 80–95.

Vacchi, E., Cazzola, W., Combemale, B., Acher, M., 2014. Automating variability model inference for component-based language implementations. In: Gnesi, S., Fantechi, A., Heymans, P., Rubin, J., Czarnecki, K., Dhungana, D. (Eds.), 18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15-19, 2014. ACM, pp. 167–176.

Weston, N., Chitchyan, R., Rashid, A., 2009. A framework for constructing semantically composable feature models from natural language requirements. In: Muthig, D., McGregor, J.D. (Eds.), 13th International Software Product Lines Conference, SPLC 2009, San Francisco, California, USA, August 24-28. ACM, pp. 211–220.

Wilson, W.M., Rosenberg, L.H., Hyatt, L.E., 1997. Automated analysis of requirement specifications. In: Proceedings of the 19th International Conference on Software Engineering. ICSE, ACM, Boston, MA, USA, pp. 161–171.

Yang, H., De Roeck, A., Gervasi, V., Willis, A., Nuseibeh, B., 2010. Extending nocuous ambiguity analysis for anaphora in natural language requirements. In: Proceedings of the 18 Proceedings of the 26th. RE, IEEE, Sydney, Australia, pp. 25–34.