
Requirement Mining for Model-Based Product Design

Romain Pinquié

Laboratoire des Sciences de l'Information et des Systèmes,
UMR CNRS 7296,
Arts et Métiers ParisTech,
2, cours des Arts et Métiers, 13617 Aix-en-Provence, France,
E-mail: romain.pinquie@ensam.eu
*Corresponding author

Philippe Véron

Laboratoire des Sciences de l'Information et des Systèmes,
UMR CNRS 7296,
Arts et Métiers ParisTech,
2, cours des Arts et Métiers, 13617 Aix-en-Provence, France,
E-mail: philippe.veron@ensam.eu

Frédéric Segonds

Laboratoire Conception de Produits et Innovation,
Arts et Métiers ParisTech,
151, Boulevard de l'Hôpital, 75013 Paris, France,
E-mail: frederic.segonds@ensam.eu

Nicolas Croué

Keonys,
5, avenue de l'Escadrille Normandie Niemen, 31700 Blagnac, France,
E-mail: nicolas.croue@keonys.com

Abstract: The enterprise level software application that supports the PLM business approach should enable engineers to develop and manage requirements. The ENOVIA/CATIA V6 RFLP environment makes it possible to use parametric modelling to map requirements to design artefacts. Simulation can therefore be used to verify that the design complies with the requirements. Nevertheless, when dealing with large document-based specifications, the definition of the knowledge parameters for each requirement is a labour-intensive task. We propose to use Natural Language Processing techniques to automatically generate Parametric Property-Based Requirements from prescriptive documents. Finally, we will be showing how a company can use machine learning to develop a text classifier that automatically assigns a topic to a requirement so as to enable designers to retrieve the requirements that belong to their discipline. The NLP experiment shows a precision of 0.73 and an expected recall of 1, whereas the SVM requirements classifier outperforms that of Naive Bayes with an accuracy of 74%.

Keywords: Requirements; Natural Language Processing; Text classification; Parametric Modelling; Functional Digital Mock-Up; ENOVIA V6; CATIA V6

Reference to this paper should be made as follows: Pinqu  , R., V  ron, P., Segonds, F. and Crou  , N. (2016) ‘Requirement Mining for Model-Based Product Design’, *International Journal of Product Lifecycle Management*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Romain Pinqu   is a Ph.D candidate at the Arts et M  tiers ParisTech engineering school in Aix-en-Provence, France. He received an MSc in Computational and Software Techniques in Engineering, specialising in Computer Aided Engineering from Cranfield University. His research concentrates on requirements engineering, combining aspects of systems theory and data science to support the design of complex systems.

Philippe V  ron is full Professor at the Arts et M  tiers ParisTech engineering school in Aix-en-Provence, France and a member of the Information and Systems Science Laboratory (LSIS, UMR CNRS 7296). Currently, he is also head of the Research and Training Department of Design and Production Engineering, Risk Management and Decision Making. His main research interests are in the development of geometric modelling approaches in the context of a multi-view and integrated design environment. He also has a particular interest in multi-site collaborative design product approaches.

Fr  d  ric Segonds is an assistant professor at Arts et M  tiers ParisTech and a member of the Product Design and Innovation Laboratory (LCPI). His research interests focus on the early stages of design collaboration, optimisation and collaborative design. This area includes the integration of stakeholders’ core competences into the early stages of design, and providing methodologies and tools to support early product design.

Nicolas Crou   is the Solutions & Consulting Director for the company Keonys and has responsibility for developing business and technical activities in a range of domains including simulation, systems engineering, search-based applications, business intelligence and manufacturing so as to develop solutions for the factory of the future. After graduating from the ESILV in Mechanical & Systems Engineering, he successfully worked for the Petroleum French Institute, SAMTECH and LMS and became an expert in Systems Engineering.

1 Introduction

1.1 ENOVIA/CATIA V6 RFLP for integrated, model-based product design

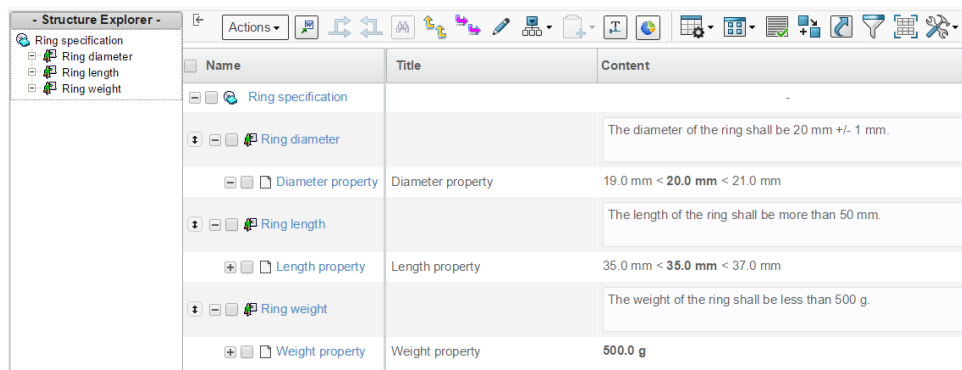
In 1990, Gero [1990] proposed the FBS ontology where F stands for the set of functions, B for the set of expected behaviours (Be) and the set of actual behaviours (Bs), and S for the structure. Twenty years later, Christophe et al. [2010] extended the FBS ontology to RFBS by including the R for requirements. Back in the nineties, in his theory of axiomatic design, Suh [3] defined four domains of activities: the customer domain, the functional domain, the physical domain and the process domain. Stepping back and looking at these product design models, which could also be assimilated to the systems engineering process

[ISO/IEC/IEEE 15288, 2015], we notice that product design relies on an iterative process that involves requirements, functions, behaviours and structures.

Dassault Systèmes' 3D Experience platform – ENOVIA/CATIA V6 – proposes a similar integrated product design model called RFLP [Kleiner and Kramer, 2013]. The R is for ENOVIA V6 Requirements, a requirements management workbench. The F, L and P layers are used to recursively break down the complexity of the design problem according to the Functional, Logical and Physical perspectives of the product. This design approach follows from Descartes' reductionism method that consists in understanding a complicated problem by investigating simple parts and then reassembling each part to recreate the whole. In RFLP, the functional layer (F) relies on a Functional Flow Block Diagram to design functional architectures in which functions transform material, energy or information input flows into output flows. The logical layer (L) is the behavioural viewpoint of the product and is materialised by a logical architecture within which, each logical unit's behaviour is equation-based modelled with the Modelica language. Modelica models are executable thanks to the Dynamic Behaviour Modelling workbench that is the integration of Dymola within CATIA V6. Finally, the physical layer (P) is very similar to the CATIA V5 CAD modeller.

To make sure that the product complies with the requirements, engineers have the choice among four standard verification methods: inspection, simulation, demonstration, and test [ISO/IEC/IEEE 29148, 2011]. The integrated, model-based RFLP environment product design environment enables designers to define implementation links between a pair of requirements, functions, logical units or physical organs so as to trace implementation relationships thanks to a traceability matrix. In addition to the traceability capability, the tight integration of ENOVIA V6 Requirements and CATIA V6 offers parametric modelling functionalities that can be used to make sure that the properties of a design artefact comply with the requirements.

Figure 1 shows that Enovia V6 Requirements also gives analysts the opportunity to define a range of attributes (e.g. version, priority, author, statement, source, rationale, etc.) among which we can define a specific one to classify requirements into disciplines.



Name	Title	Content
Ring specification		-
Ring diameter		The diameter of the ring shall be 20 mm +/- 1 mm.
Diameter property	Diameter property	19.0 mm < 20.0 mm < 21.0 mm
Ring length		The length of the ring shall be more than 50 mm.
Length property	Length property	35.0 mm < 35.0 mm < 37.0 mm
Ring weight		The weight of the ring shall be less than 500 g.
Weight property	Weight property	500.0 g

Figure 1: A specification with 3 requirements in the workbench Requirements of ENOVIA V6.

1.2 "Parametric" Property-Based Requirement

Micouin [2008] introduced the concept of Property-Based Requirement – PBR. A PBR is an unambiguous formal definition of a requirement as a predicate and is defined as follows:

$$PBR : \text{When } \mathcal{C} \longrightarrow \text{val}(\mathcal{O}.\mathcal{P}) \in \mathcal{D} \quad (1)$$

This formal statement means: "When condition \mathcal{C} is true, property \mathcal{P} of object type \mathcal{O} is actual and its value shall belong to domain \mathcal{D} ". Below is an example of a statement that meets the PBR definition:

- *When the temperature of water in the Boiler is less than 85 °C, the Control_Subsystem shall open the Inlet_Valve in less than 3 seconds.*
 - **Condition:** When the temperature of water in the Boiler is less than 85 °C
 - **Object:** Inlet_Valve
 - **Property:** seconds
 - **Domain:** less than 3

A relevant characteristic of the concept of PBR is that it is grammar-free, that is, a PBR does not have any particular syntactic structure. So far, the only required function that a simulation language must own to implement a PBR is a conditional assertion. Consequently, most modelling and simulation languages like VHDL-AMS [Micouin, 2014], Modelica [Pinqu   et al., 2016] and Simulink can be used to implement PBRs.

By combining the PBR theory with parametric CAD modelling, we coin the concept of Parametric PBR – PPBR. A PPBR is a PBR that is implemented with a parametric modeller thanks to knowledge parameters and knowledge verification rules. To the best of our knowledge, ENOVIA/CATIA V6 RFLP is the only platform that can be used to model and simulate PPBRs today.

Figure 2 shows a PPBR in CATIA V6. Object \mathcal{O} is the ring. Property \mathcal{P} is the length that stands for the external diameter property \mathcal{P} . Minimum and maximum values are the bounds of domain \mathcal{D} . To verify whether the actual value of the external diameter property \mathcal{P} belongs to domain \mathcal{D} , a designer must specify a knowledge verification rule. This rule is a conjunction of two partial order relations "<". The first relation constrains the external diameter property \mathcal{P} with the minimum value of domain \mathcal{D} , whereas the second one refers to the maximum value of domain \mathcal{D} . This example does not include the modelling of a condition \mathcal{C} that can be implemented thanks to a knowledge rule too.

One major advantage of using ENOVIA/CATIA V6 RFLP environment to implement PPBRs is that it addresses both perspectives of a product: the behavioural viewpoint thanks to the integration of Dymola within the Dynamic Behaviour Modelling workbench, as well as the structural viewpoint thanks to the CAD modeller.

1.3 Problematic

Among the product life cycle phases defined by Terzi et al. [2010], our research study focuses on the requirements analysis phase.

Nowadays, a set of requirements is tremendously challenging to develop and manage because of its very large size. For instance, at Mercedes-Benz, the size of a building block specification varies from 60 to 2000 pages and prescribes between 1000 and 50 000

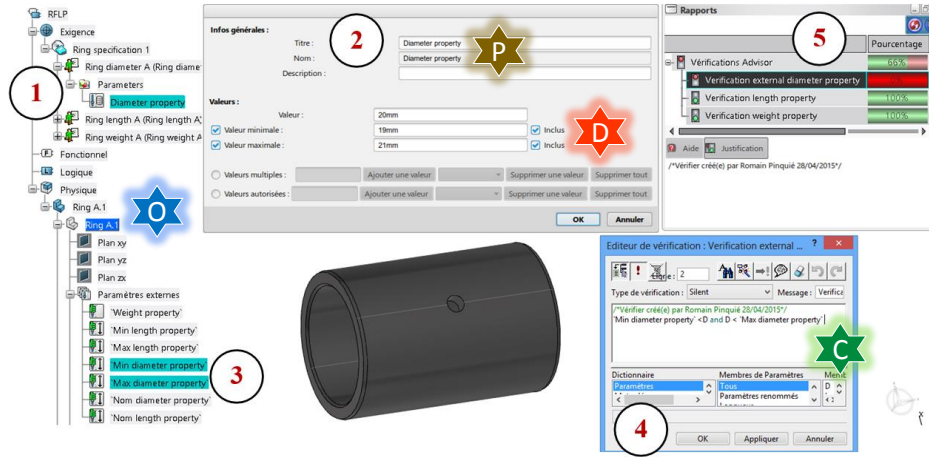


Figure 2: (1) 3 PPBRs originally defined in ENOVIA V6 Requirements, (2) external diameter PPBR that constrains property \mathcal{P} of type "length" whose value shall belong to domain \mathcal{D} , (3) knowledge parameters, (4) knowledge verification rule, and (5) quantitative level of compliance.

requirements [Houdek, 2010]. Among the leading causes of such a staggering increase in the number of requirements we find: the ever-increasing complexity of products and their relentless customisation; the mushrooming accumulation of legal documents; and the geographically dispersed teams through whom products are developed. In addition to the massive volume of requirements, most specifications are unstructured documents – e.g. Word, PDF – and 79% of requirements are written in unrestricted natural language [Mich et al., 2004].

With such large document-based specifications, OEMs struggle to deliver products that comply with the legal and contractor's requirements. Indeed, when an OEM collects the specifications and the applicable documents the specification refers to, he has no other alternative than to go through the documents to identify the applicable requirements, design solutions and verify that the designs comply with the requirements. This paper addresses two problems:

- **Problem A - Specification of PPBR:** Although parametric modelling-based verification is a powerful approach compared to existing software solutions, it can be a very time consuming activity.

Indeed, in a "buy approach" of a "make vs. buy" decision, analysts start to read all prescriptive documents. In this study, a prescriptive document is a document that prescribes needs or requirements. Reports of interviews with customers, specifications, concepts of operations, business cases or meeting notes can be prescriptive documents. While reading, analysts must identify performance requirements and design constraints which then are manually modelled as requirements' knowledge parameters in ENOVIA V6. Finally, designers not only design the behavioural and structural artefacts using parametric modelling in CATIA V6, but also define the knowledge verification rules that map requirements' knowledge parameters with design artefacts' knowledge parameters so as to finally ensure a minimum level of compliance.

In a "make approach" there is no exchange of document-based specifications. Therefore, before designing, analysts simultaneously prescribe the product's

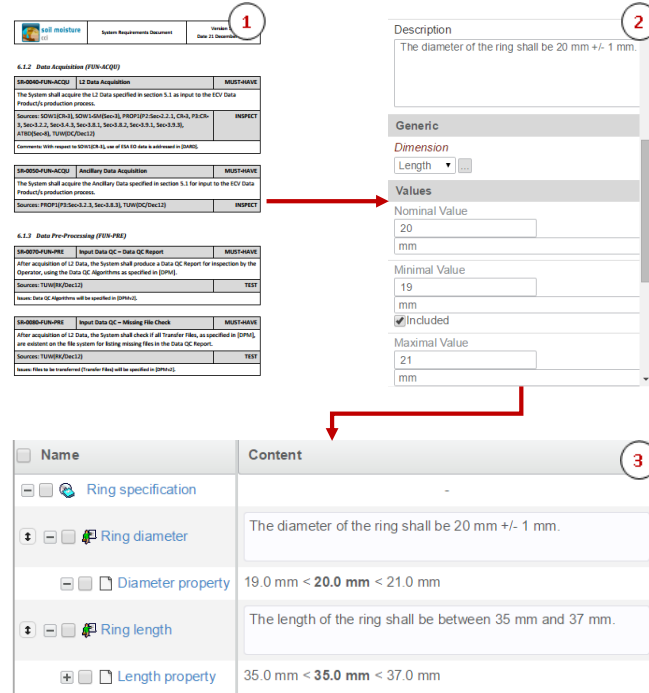


Figure 3: Process that consists in: (1) identifying requirements in large prescriptive documents, (2) defining each requirement as a PPBR in ENOVIA V6 Requirements so as to finally have a new specification in ENOVIA V6 Requirements (3).

requirements and the requirements' knowledge parameters in ENOVIA V6 Requirements.

⇒ **The main problem here is the rework task that consists in moving the existing requirements from large prescriptive documents into the ENOVIA V6 Requirements database and manually build requirements' knowledge parameters. Figure 3 illustrates the rework process.**

- **Problem B - Classification of requirements:** Each domain expert should be able to filter the requirements according to his domain of expertise. For instance, marketers should be able to filter marketing requirements, whereas mechanical engineers should quickly retrieve mechanical ones. Analysts who are in charge of prescribing the requirements in ENOVIA V6 can manually specify the value of an attribute *discipline*. In this scenario, each time an analyst specifies a new requirement he also has to put it in a discipline.

⇒ **The main problem here is the time required to fill one more attribute in addition to the author, version, statement, priority, source, etc. Moreover, if analysts are not domain experts, the categorisation task can also be rather fastidious for them.**

The rest of the paper is organised as follows. Section 2 is a literature review. Section 3 describes the natural language processing pipeline that generates PBRs from unstructured and semi-structured prescriptive documents. In section 4, we propose a machine learning-based classifier that categorises a set of requirement statements into classes corresponding

to disciplines. Finally, section 5 summarises our contribution and gives perspectives for improvements.

2 Related work & Contributions

Recently, information retrieval, NLP, web semantics and text mining techniques have been attracting more attention from academics and industrialists who are challenged by the increasing size of textual data that needs to be collected, stored and managed in a PLM software application. Relevant research studies, including, but not limited to, Madhusudanan et al. [2015], Jones et al. [2015] and Zhang et al. [2015] - presented at the last PLM 2015 international conference - or Feldhusen et al. [2012] have demonstrated the benefits of applying such knowledge to solve PLM problems. The rest of this section gives a broader literature review of the current existing solutions to the problem A and B.

2.1 How can we extract requirements?

A specification S is a couple $(\mathcal{P}, \mathcal{I})$, where \mathcal{P} is a set of *prescriptive* statements and \mathcal{I} is a set of *informative* statements. Prescriptive statements include *need statements* and *requirement statements* [INCOSE, 2015]. Regarding the problem A, we notice that very few research studies attempt to automatically distinguish both kinds of statements in unstructured prescriptive documents.

Modal verbs. Modal verbs such as *shall*, *must* and *should* are key lexical features for identifying prescriptive statements [Lash, 2013]. Coatanéa et al. [2013] propose a three-step process to extract requirements from unstructured specifications. The first step is to extract the sentences from a set of documents using a syntactic rule that identifies sentences by looking at capital characters and full stops. The summary report does not give further details; consequently, we can only conclude that the proposition seems sub-optimal. Indeed, although Riley [1989] has shown as far back as 1989 that 90% of full stops are sentence boundary indicators, heuristic algorithms suffer from the tokenization process that requires a lot of hand-coding and domain knowledge [Manning and Schütze, 1999]. In prescriptive documents, we observe that there are many instances of frequent omissions of full stops. For instance, when tables are used to collect requirements. Additionally, headings do not usually finish with a full stop and are therefore concatenated with the consecutive sentence. Full stops can also be used in abbreviations or numbers. These pitfalls would not be avoided by a mere rule that only look at capital characters and full stops. Once the identification of the sentence has been done, Coatanéa et al. [2013] use the Stanford Parser to identify modal verbs whose POS-tag is "MD". Finally, a binary rule-based classifier applies a syntactic rule that labels a sentence as a requirement based on the presence or absence of a modal verb.

Legal terms. Legal terms help to identify legal requirements. Breaux and Antón [2005] developed three lists of syntactic patterns that are commonly used to encode rights, obligations and constraints. More recently, Zeni et al. [2013] have proposed GaiuST, a framework that extracts legal requirements for ensuring regulatory compliance. In their study, they propose to identify rights, anti-rights, obligations and anti-obligations thanks to lexical features too. Their four lists extend the usual list of modal verbs by adding verbs such as *permit* and *require*, as well as derived forms like *is not required*. However, these features exclusively concentrate on legal text. Requirements are not only specified in legal documents

– e.g. standards, policies or laws –, but also in reports of interviews, specifications, concepts of operations, business cases or meeting notes whose jargon differs from the legal one.

Syntactic rules. Kang and Saint-Dizier [2013, 2014] implemented 12 syntactic rules for identifying requirements without requiring domain knowledge. In their list, requirements fall into two categories: *lexically induced requirements* and *requirements composed of modal*. The former enhances the latter as it adds the verbs *require*, *need* and their derived forms such as *requires*, *required*, *need* or *needed* to the list of modal verbs. Their solution gives promising results as they experienced a precision of 0.97 and a recall of 0.96 on a corpus of 64 pages prescribing 215 requirements.

Rules VS. Statistical learning. We did not find any study that proposes to solve the problem of requirements extraction by adopting a machine learning approach. This is not surprising since the main constraint is to obtain a high recall with an acceptable precision rate, rather than the opposite. Indeed, finding true requirements takes at least an order of magnitude more time than rejecting a false positive. The former requires a tedious manual search, whereas the latter can be done usually in a split second by looking at the claimed requirement.

- **Limit A:** Although modal verbs are undeniably the most reliable lexical features to identify requirements, the literature review shows that legal terms, as well as the verbs *require*, *need* and their derived forms, must be considered too. However, we contend that the solution is still sub-optimal. Indeed, most companies are not aware of, or do not want to follow, the INCOSE's guidelines or IEEE standards, but instead, prefer "to colour outside the lines". It is therefore usual that they use verbs that not only express a requirement or a need, but also an expectation, a wish, a hope, a desire, etc. Then, verbs including, but not limited to, *expect*, *need*, *wish*, *want* or *desire* also need to be considered.

In addition, as there are diverse writing styles, the use of syntactic rules to identify requirements is very likely to be too restrictive in some cases.

⇒ **Contribution A:** To follow the principle of parsimony, we propose to enhance the existing lists of prescriptive verbs and to stick to a rules-based classifier that does not sacrifice recall in favour of precision.

- **Limit B:** The NLP task of sentence splitting is challenged by the authors' lack of rigour when using lists, bullets, numbering, tables, etc.

⇒ **Contribution B:** We combine the NLP capabilities of the Stanford Parser with the parsing functionalities of Apache Tika that enable us to consider the structure of a prescriptive document.

- **Limit C:** To the best of our knowledge, there are no research studies that have attempted to extract performance requirements and design constraints to automatically build PBRs.

⇒ **Contribution C:** To avoid the very time-consuming requirements' knowledge parameters definition process, we propose a NLP pipeline. The NLP pipeline extracts text-based requirements (TBRs) from unstructured and semi-structured specifications and models them in an XML file that can be imported in ENOVIA V6 to automatically generate PPBRs.

2.2 How can we automatically classify requirements into disciplines?

There are two main approaches to regrouping pieces of text – i.e. document, paragraph, sentence, chunk, etc. – into topics: *classification* or *clustering* [Manning and Schütze, 1999, Manning et al., 2008, Feldman and Sanger, 2006].

Classification. Given a finite set of categories, the problem of text classification consists in determining which category a given piece of text belongs to. A text classifier can be developed by encoding, either declaratively or in the form of procedural classification rules, experts' knowledge about the categories into the system [Feldman and Sanger, 2006]. Such an approach is called *knowledge engineering*.

Alternatively, a *supervised learning* approach consists in using a learning algorithm to learn a classification function from a set of preclassified examples. The classification function is used to map an unlabelled text to one or several categories. Sebastiani [2002] gives a detailed introduction to text classification.

Ott [2013] evaluates several classifiers that classify requirements into topics. Each topic is manually defined as a set of keywords. The best classifier will be used to enable inspectors to clean requirements' defects in parallel. The study states that decision trees and rules based learning algorithms returned poor results. Therefore, the authors focused on the Multinomial Naive Bayes (MNB) and the Support Vector Machine (SVM) algorithms. The preprocessing pipeline is not relevant for our problem because the classifier is tuned for German written specifications, whereas our work focuses on English text. One major problem that the research study points out is the difficulty of getting sufficient training examples so as to improve the current recall of 0.8 and precision of 0.6.

Knauss and Ott [2014] enhanced the previous proposition by comparing three classification approaches: manual, semi-automatic and automatic. The first approach is a manual classification where the user has to assign one or several categories to each requirement. The semi-automatic method automatically classifies a given requirement but requires a confirmation or modification from the user. Finally, the automatic classifier does not require a manual confirmation or modification. The semi-automatic approach is an interesting way of overcoming the cold start problem that is due to a small training set. Indeed, once the semi-automatic classification of a requirement is confirmed or modified by the user, the system adds the requirement to the training set. Although the main ideas are of interest, we cannot directly transpose their method as they also focus on German written specifications.

Clustering. In contrast, clustering is a form of *unsupervised learning* since a given example is not labelled with a class label. In text mining, clustering algorithms aim at creating clusters of documents where documents within a cluster should be highly similar, and documents in a given cluster should be highly dissimilar from documents in other clusters. Aggarwal and Zhai [2012] give a survey of text clustering.

There have been many attempts to cluster requirements [Duan, 2008] to automate various tasks like keywords list development [Ko et al., 2007], requirements tracing [Duan and Cleland-Huang, 2007, Sannier and Baudry, 2012, Sannier, 2013], requirements prioritisation and triage [Laurent et al., 2007, Duan et al., 2009] or discovering and improving a prescriptive document's structure [Ferrari et al., 2013].

Unsupervised learning techniques such as clustering and topic modelling are not directly suited to our problem as we have predefined categories corresponding to the functional areas of a company.

- **Limit A:** As previously explained, the labelling of requirement statements with a predefined set of disciplines is a text classification problem. Most supervised learning techniques that have been proposed to classify requirements face the need for a large enough training set. So far, there are no training sets that we can reuse. PLM being a blend of disciplines, the development of several hundreds or thousands of examples for each discipline would be time- and resource-consuming. In addition, previous contributions focused on German written specifications only.

⇒ **Contribution A:** We propose to build a supervised classifier for English written prescriptive documents. However, instead of asking a supervisor to manually write and classify examples of requirements, we propose to automatically generate a training set. The training set is a large set of sentences extracted from handbooks corresponding to disciplines. In our experiment, we have chosen four disciplines: mechanics, electronics, computer science and RAMS.

3 From prescriptive documents to design synthesis

This section introduces a NLP pipeline that (1) extracts TBRs from both unstructured and semi-structured prescriptive documents, and (2) transforms TBRs into PBRs. PBRs are finally stored in a semi-structured XML file that can be imported in ENOVIA V6 to generate PPBRs.

Before introducing the NLP pipeline, we shall clearly distinguish *unstructured* and *semi-structured* documents. Digital documents store data that falls into three categories: *structured*, *semi-structured* and *unstructured* [Li et al., 2008, Sint et al., 2009, Abiteboul et al., 2011].

In structured data – e.g. relational databases –, one can distinguish the data structure (the schema) from the data itself (an instance). We say that "*an instance conforms to the schema*". Structured data is synonymous with tabular data organised in a matrix where rows and columns correspond to data and attributes, respectively. Tools such as IBM Doors or Enovia V6 Requirements store structured specifications in relational databases.

As the name implies, unstructured data – e.g. Word or PDF documents – do not have any underlying structure. They are sequences of characters. Unstructured data cannot be stored in tables. Specifications written with software like Word or OpenOffice, or encoded in PDF are unstructured specifications.

Lastly, semi-structured data – e.g. XML or XMI files – are blends of structured and unstructured data. Indeed, it is more structured than a raw sequence of characters but less than a table. Usually, semi-structured data can be represented as a tree – e.g. XML – or as a graph – e.g. XMI. A specification that complies with the Requirements Interchange Format (ReqIF) is semi-structured.

To derive PBRs from TBRs prescribed in unstructured and semi-structured prescriptive documents we propose the following NLP pipeline:

Step 1 <Uploading>: The user uploads one or more prescriptive documents. The prototype can process unstructured documents whose format is .doc(x) (Word), .odf (OpenOffice), .pdf (Portable Document Format) and .xls(x) (Excel). Regarding semi-structured documents, there are various SysML authoring tools and their XML Schema Definition is very likely to change from one to another. Our experiment focuses on

the Papyrus open-source tool. Once exported in XMI format, semi-structured SysML requirements diagram can be uploaded in our prototype.

Step 2 <Parsing>: We trigger a specific parser according to the document format identified by the Apache Tika API.

If it is an unstructured .doc(x) or .odf document, the parser uses the Apache Tika API to extract the textual content and transform it into .html semi-structured data. We transform the content into HTML because it enables us to get the document's structure by seeking specific HTML tags: header, footer, headings, sections, tables, bullets, numbering, etc. Headings help to identify the sections which may be used to run NLP tasks in parallel (multi-threading).

If it is an unstructured PDF document, in batch mode, the native capability of Word converts the document from .pdf into .doc. Then, as for .doc, the Apache Tika API converts from .doc into .html. The structure of PDF document is usually lost except for the ones whose native format is Word and partially OpenOffice. Thus, by parsing the new .html semi-structured document, we verify whether the .pdf was generated with Word or OpenOffice. HTML tags such as header, footer or table HTML are of interest for this task. If we find that the .pdf was generated with Word or OpenOffice, then we call the .doc parser; otherwise, we use the .pdf parser that relies on the Apache Tika API and various regular expressions. The second scenario leads to much less accurate results as we lose the document's structures. Fortunately, today, Word and OpenOffice are leading the text processor market.

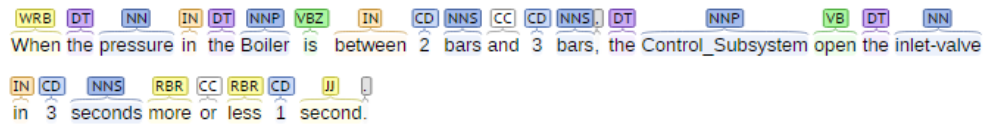
The .xls(x) parser uses the Apache POI API to parse the textual content of each cell. We make the assumption that analysts follow the requirements writing rules advising that a requirement statement is a single sentence [INCOSE, 2015]; therefore, it is a single cell in the spreadsheet. Each sentence in a cell being a potential requirement statement.

Finally, the .xml parser uses the JDOM Parser to parse semi-structured SysML Requirements diagrams by extracting the content of XML elements whose tags stand for a requirement statement.

Step 3 <Tokenization>: The Stanford CoreNLP [Manning et al., 2014] Tokenizer API iteratively tokenizes each specification content, that is, it chops the textual content up into pieces of a sequence of characters that are grouped together as a useful semantic unit for processing, the *tokens*.

Step 4 <Lemmatization>: The Stanford CoreNLP Lemmatizer API iteratively normalises each token by removing the inflectional ending and returns the dictionary form, the *lemma*. For instance, lemmatization reduces the tokens "requires", "required" and "require" to their canonical form "require". This not only enables us to increase the recall of the rules-based sentence classifier in step 8 <Classification>, but also to reduce the vocabulary size that can be of very high dimension.

Step 5 <POS-tagging>: The Stanford CoreNLP POS-tagger API iteratively annotates each token with its grammatical category (noun, verb, adjective, adverb, etc.), the *Part Of Speech* (POS). Figure 4 shows an example of a labelled sentence.



When the pressure in the Boiler is between 2 bars and 3 bars, the Control_Subsystem open the inlet-valve in 3 seconds more or less 1 second.

Figure 4: A sentence whose tokens are annotated with their POS tags.

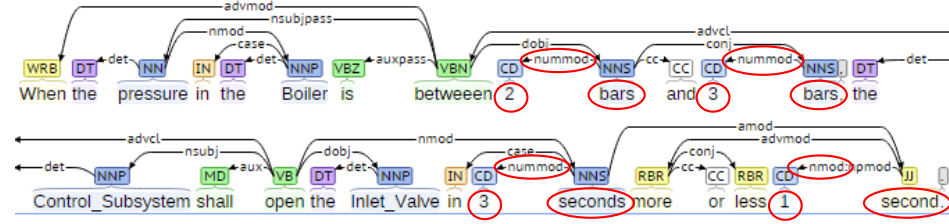


Figure 5: A requirement that includes 4 numerical dependencies $\langle Unit \rightarrow nummod \rightarrow Value \rangle$.

Step 6 <Sentence splitting>: The Stanford CoreNLP Parser API iteratively splits the textual content of each document into sentences. A sentence-lemma matrix stores the sentences in rows and the lemmas in columns.

Step 7 <Sentence cleaning>: We use various regular expressions and analyse HTML tags to clean the sentences. For instance, we rebuild sentences from bullets and numberings, get rid of the headers, footers, and extract the textual content from tables. We also use a list of keywords such as *introduction*, *scope*, *table of content*, *glossary*, *list of acronyms*, etc. to skip the informative sections that may generate false positive requirements.

Step 8 <Sentence classification>: A binary knowledge engineering – a.k.a rules-based – text classifier classifies each sentence into "requirement" vs "information". The matrix of sentence-lemma is traversed, and when the condition "if lemma_i of sentence_j is a prescriptive verb $\in \{\text{shall, must, should, have to, require, need, want, expect, wish or desire}\}$ " is true, the current sentence_j is classified as a requirement.

Step 9 <Numerical dependency analysis>: The Stanford CoreNLP Dependencies Analyzer API is a typed dependency parser [Chen and Manning, 2014] that implements a neural network to not only represent dependencies between individual words, but also label dependencies with grammatical relations. In the NLP pipeline, the dependency parser

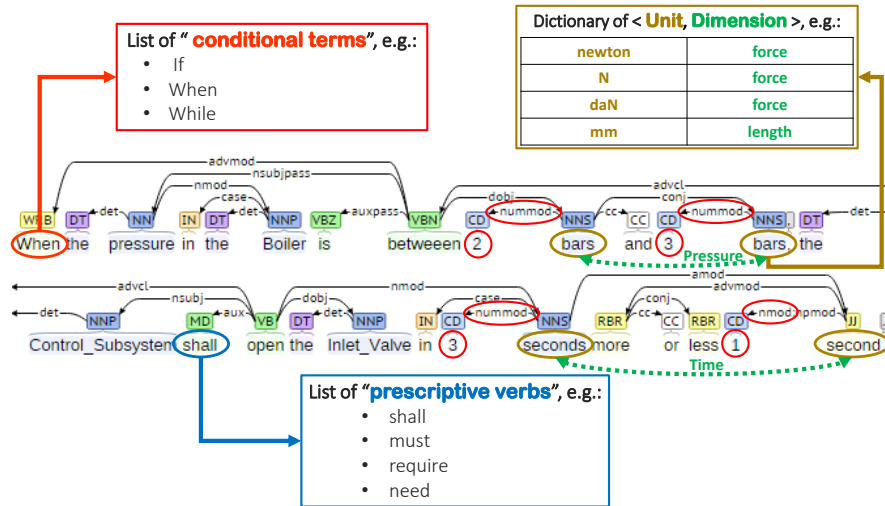


Figure 6: Resources used to find the (1) conditional term, (2) prescriptive verb, and (3) performances and/or constraints values within the semantic graph of a requirement.

converts each requirement into a semantic graph within which the *nummod* dependency type stands for a dependency that links a numerical value and a modifier. For instance, Figure 5 shows the semantic graph of a requirement that contains 4 numerical dependencies whose source nodes are numerical token labelled with the POS tag *CD* and target nodes are modifiers.

Step 10 <Performance requirements identification>: While going through the dependencies list of a given requirement, we check whether the term stored in the target node of each dependency is a physical unit such as N, °C, kg, Pa, etc. by querying a dictionary that stores the abbreviated and expanded form of physical units – e.g. "N" and "Newton". Each time a given numerical dependency is classified as a physical numerical dependency, we add a third attribute from our resource file that is the dimension of the physical unit – e.g. "Force" for the unit "N" or "Newton". Figure 6 shows four physical numerical dependencies where the first two modifiers are units whose magnitude is *pressure*, whereas the magnitude of the last two modifiers is *time*.

Step 11 <Pattern analysis>: A well-written TBR prescribing a functional level of performance or a design constraint usually follows three distinct syntactic patterns (Pattern 1, 2 and 3) [INCOSE, 2015]. Note that the condition is not always specified; consequently, there is one more syntactic pattern (Pattern 4). The 4 syntactic patterns are listed in Table 1.

Pattern 1: <Prescriptive> <Domain> <Condition> – \mathcal{PDC}
The Control_Subsystem shall open the Inlet_Valve in less than 3 seconds when the temperature of water in the Boiler is less than 85 °C.
Pattern 2: <Prescriptive> <Condition> <Domain> – \mathcal{PCD}
The Control_Subsystem shall, when the temperature of water in the Boiler is less than 85 °C, open the Inlet_Valve in less than 3 seconds.
Pattern 3: <Condition> <Prescriptive> <Domain> – \mathcal{CPD}
When the temperature of water in the Boiler is less than 85 °C the Control_Subsystem shall open the Inlet_Valve in less than 3 seconds.
Pattern 4: <Prescriptive> <Domain> – \mathcal{PD}
The Control_Subsystem shall open the Inlet_Valve in less than 3 seconds.

Table 1 4 syntactic patterns to identify a PBR.

Given a list of **conditional terms** such as *when*, *if* or *while* and a list of **prescriptive verbs** like *shall*, *must*, *should*, *will*, *have to*, *require*, *need*, *want*, *expect*, *wish*, *desire*, *crave* or *demand*, we can find the index of the conditional term – C_i – and the index of the prescriptive verb – P_i – by iterating through the tokens of a given requirement.

Table 2 lists the set of rules to identify the patterns of Table 1. These rules compare the index of the physical numerical dependencies – PND_i –, index P_i , and index C_i .

RULE 1:	IF $P_i < PND_i < C_i$ THEN PATTERN 1
RULE 2:	IF $P_i < C_i < PND_i$ THEN PATTERN 2
RULE 3:	IF $C_i < P_i < PND_i$ THEN PATTERN 3
RULE 4:	IF $C_i = \emptyset$ AND $P_i < PND_i$ THEN PATTERN 4

Table 2 Rules to identify the 4 patterns of Table 1.

A physical numerical value is sometimes followed by a tolerance. Thus, patterns 1, 2 and 3 give rise to four more patterns where domain \mathcal{D} and condition \mathcal{C} are split into a nominal domain $n\mathcal{D}$, a tolerance domain $t\mathcal{D}$, a nominal condition $n\mathcal{C}$ and a tolerance condition $t\mathcal{C}$ – e.g. pattern 5 in Table 3 follows from pattern 1 in Table 1.

Pattern 5: <Prescriptive> <Nominal Domain> <Tolerance Domain> <Nominal Condition> <Tolerance Condition> – $P_n D_t D_n C_t C$
The Control_Subsystem shall open the Inlet_Valve in 3 seconds +/- 1 second, when the temperature of water in the Boiler is between 70 °C and 85 °C.

Table 3 Pattern that follows from pattern 1 in Table 1 where the domain is divided into a nominal and tolerance value.

To make sure that two consecutive physical numerical dependencies form the so called <nominal, tolerance> pair of a domain D or a condition C , we check whether their units belong to the same physical dimension. For instance, in the requirement: "When the temperature is less than 40 °C, the pressure shall be less than 30 Pa", the consecutive physical numerical dependencies <40 °C> and <30 Pa> do not belong to the same physical dimension since the former is a temperature (°C), whereas the latter is a pressure (Pa). However, in the requirement "The system shall control a pressure of 30 Mpa +/- 5 Pa", the physical numerical dependencies <30 Mpa> and <5 Pa> belong to the same physical dimension – a pressure.

Finally, there are six more syntactic patterns according to whether there is a tolerance associated to the domain but not to the condition and *vice-versa* – e.g. pattern 6 in Table 4 follows from pattern 5 in Table 3.

Pattern 6: <Prescriptive>> <Domain> <Nominal Condition> <Tolerance Condition> – $P D_n C_t C$
The Control_Subsystem shall open the Inlet_Valve in less than 3 seconds, when the temperature of water in the Boiler is between 70 °C and 85 °C.

Table 4 Pattern that follows from pattern 5 in Table 3 where the domain is a single nominal value and the condition is divided into a nominal and tolerance value.

Step 12 <Tolerance calculation>: The calculation of the minimum, maximum and nominal values defining the tolerance of condition C or domain D is a challenging problem. Hand-written rules can be implemented to identify patterns such as those in Table 5. However, at present, there is a limit when the units differ. For instance, in the chunk "1.5 daN +/- 10 N" that one could rewrite as "15 N +/- 1 daN" we cannot directly compute the tolerance without using a physical unit convertor. This is the main limitation of the proposed NLP pipeline that will be addressed in future work.

Pattern A:	X +/- Y where X > Y
	<i>The Control_Subsystem shall open the Inlet_Valve in 3 seconds +/- 1 second.</i>
Pattern B:	X more or less Y where X > Y
	<i>The Control_Subsystem shall open the Inlet_Valve in 3 seconds more or less 1 second.</i>
Pattern C:	from X to Y where X < Y
	<i>The speed of the Vehicle shall vary from 90 km/h to 130 km/h.</i>
Pattern D:	between X and Y where X < Y
	<i>The speed of the Vehicle shall vary between 90 km/h and 130 km/h.</i>

Table 5 Example of patterns that help to identify tolerances and compute minimum, maximum and nominal values.

Step 13 <PBRs modelling>: The NLP pipeline ends up with an XML within which PBRs' structure complies with the PPBRs data model in ENOVIA V6 Requirements. Thus,

each PBR element is a scheme of attributes that includes: *statement*, *dimension*, *nominal value*, *nominal unit*, *minimum value*, *minimum unit*, *maximum value*, *maximum unit*.

The XML file can finally be imported into ENOVIA V6 Requirements so as to automatically generate the PPBRs. This pure software development part of our proposal has not been implemented yet. Figure 7 summarises the general process to generate PPBRs from prescriptive documents where step (1) is the proposed NLP pipeline.

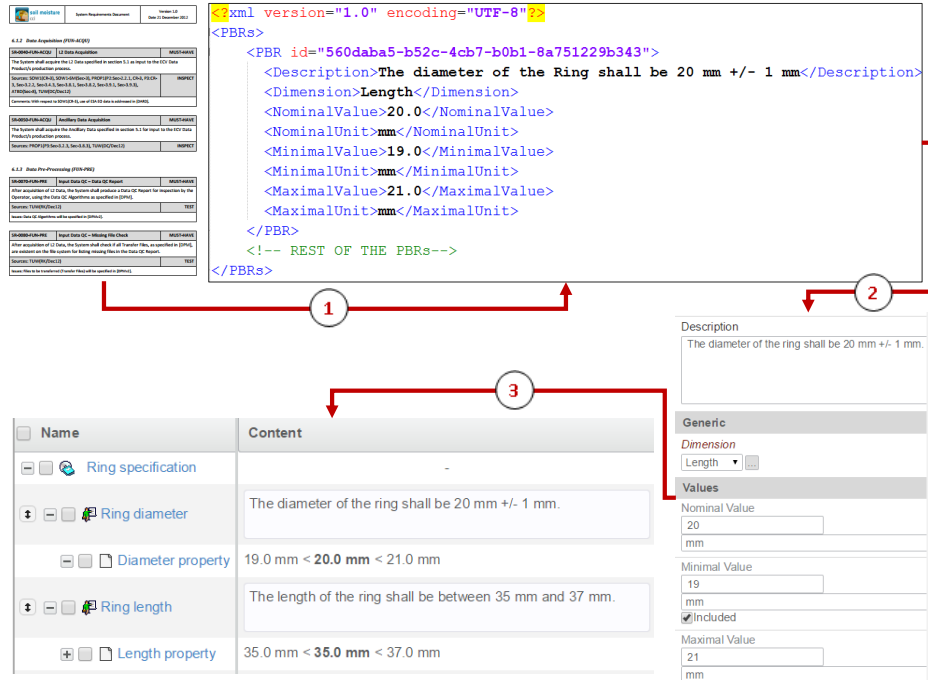


Figure 7: Process that consists in: (1) extracting PBRs from prescriptive documents, (2) generating PPBR in ENOVIA V6, and (3) storing PPBRs in a new specification in ENOVIA V6.

Once the PPBRs have been generated in ENOVIA V6 Requirements, designers start the design synthesis thanks to the F, L and P layers of CATIA V6. The "zigzagging" iterative and recursive process from requirements to design solutions described in the Axiomatic design theory of Suh [2001] should be adopted so as to get the expected results and rise opportunities for innovation [Suh, 2005].

Model-based product design in CATIA V6 brings about the need to quickly retrieve requirements from the database according to the user's domain of expertise and this will be further discussed in Section 4.

3.1 Evaluation of the NLP pipeline

To evaluate the proposed NLP pipeline, we have processed a 106-page unconstrained document-based system specification that is freely available on the website of the European Space. It is a Word native PDF document whose structure can be retrieved from its HTML representation.

We beforehand manually reviewed the document and found 194 requirements. Then, we sent the specification down the NLP pipeline that found 194 true-positive requirements leading to the expected recall of 1. However, 72 informative statements have been mistakenly classified as requirements yielding to a precision of 0.73. This first experimental evaluation is acceptable as we previously explained that we prefer a high recall with an acceptable precision rather than the opposite.

Regarding the automatic generation of PPBRs from TBRs, we observed that most analysts do not follow the general rules of requirements writing. Indeed, most requirements do not prescribe a quantitative domain to which the value of a property shall belong to and are consequently not verifiable.

4 Machine learning-based classification of requirements into disciplines

This section proposes a machine-learning based classifier that infers a new attribute corresponding to the discipline a given requirement belongs to. As an example, the research study concentrates on 4 disciplines: mechanical engineering (ME), electrical engineering (EE), computer science (CS) and reliability, availability, maintenance and safety (RAMS).

The development of the proposed machine-learning based classifier is a three-step process that includes: (1) the creation of a training set, (2) the selection of a subset of features, (3) the training and evaluation of various learning algorithms among which we select the one that gives the best results. Before detailing each step, we shall clearly define the classification problem.

4.1 Definition of the classification problem

To formally define our machine-learning based classification problem, we shall consider five elements:

1. A **feature vector** \vec{x}_i . Terms is the default form of features in text classification, but it could also be the POS tag of tokens, the results of more sophisticated regular expressions or dimensionality reduction techniques like singular value decomposition, latent semantic indexing or principal component analysis.
2. A **feature space** $\mathbb{S} = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n\}$ where n is the number of examples
3. A fixed **set of classes** $\mathbb{C} = \{\text{ME}, \text{EE}, \text{CS}, \text{RAMS}\}$ where each class corresponds to a discipline.
4. A **training example** (\vec{x}_i, c_i)
5. A **training set** $\mathbb{X} = \{(\vec{x}_1, c_1), (\vec{x}_2, c_2), \dots, (\vec{x}_n, c_n)\}$ where c_i is the *class*, that is, the discipline, to which the i^{th} requirement belongs to.

The statical learning classification problem consists in estimating a classification function γ that maps requirements to disciplines:

$$\gamma : \mathbb{X} \rightarrow \mathbb{C} \quad (2)$$

4.2 Automatically generating a training set

According to Manning et al. [2008]: "the biggest practical challenge in fielding a machine learning classifier in real application is creating training data". Ott [2013] as well as Knauss and Ott [2014] also mention the difficulty of getting a large amount of labelled requirements.

Rather than hand writing and labelling the hundreds or thousands of requirements needed to produce a high-performance classifier, we propose to automatically generate a large amount of training examples.

Figure 8 depicts a naive solution to the text classification problem, where the solution consists in labelling a piece of text according to the keywords occurrences. For instance, a classifier would label a document with the class *mechanics* if the term *strain* occurred more often than the term *voltage*. This naive definition prompted us to build the training set by collecting several handbooks for each discipline and to extract the sentences from each handbook. Once extracted, the sentences are stored in a text file and labelled with the class that corresponds to the discipline the handbook belongs to. For instance, the sentences extracted from a mechanical engineering handbook are labelled with the *ME* class. We use Apache Tika to parse the handbooks' textual content and the Stanford Parser to extract the sentences. After removing the sentences corresponding to equations or bibliographic references we ended up with a large set of 77 481 training examples – 18 135, 19 464, 20 183 and 19 699 examples for the ME, EE, CS and RAMS class, respectively. These sentences will finally be transformed into a vector space model. For the sake of simplicity, one can imagine the feature space \mathbb{S} as a high dimensional $\mathbb{N} \times \mathbb{M}$ sentence-feature matrix, where \mathbb{N} is the number of sentences and \mathbb{M} is the number of features.

To transform a given sentence into a vector of features we need to perform a range of natural language pre-processing tasks. In this experiment, we apply *tokenization*, *case-folding* and *stop-word removal*. *Stemming* does not usually deliver an additional value but can help to compensate the data sparseness and reduce the computing cost by reducing the vocabulary size [Manning et al., 2008]. Nevertheless, in their experiments of classification at the sentence level, Khoo et al. [2006] demonstrated that the lemmatization of tokens was harmful to performance. Stemming being even more aggressive than lemmatization, we have decided not to include it into the pre-processing pipeline. Our initial vocabulary size before stemming is large, 40 729 features yielding to a 77 481 x 40 729 sentence-feature matrix. After applying the prep-processing pipeline, the size of the vocabulary was reduced to 32 885 tokens.

4.3 Feature selection

Although the pre-processing tasks reduce the vocabulary size, the sentence-feature matrix is still very large and sparse. Feature selection consists in selecting a subset of features by avoiding *noise features* occurring in the training set and using this subset to train a learning algorithm. A noise feature is a feature that increases the misclassification error rate. For instance, if the term *temperature* has no information about the *ME* class, but all sentences that contain the term *temperature* happen to occur in the *ME* class, then the learning algorithm may produce a classifier that misclassifies new unlabelled requirement statements. When such an incorrect generalisation happens, we say that the classifier *overfits* the data.

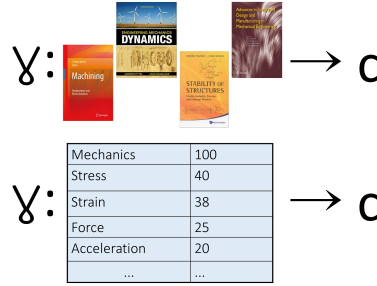


Figure 8: Over-simplified illustration that gives the gist of the idea of using handbooks to extract a distribution of keywords that finally serves as features.

There are three main paradigms of feature selection: *filter*, *wrapper* and *embedded* methods [Forman, 2007]. Although wrapper methods perform best, they are impracticable in our case because of the large size of the initial set of features. Conversely, filtering methods are the most scalable. With filtering methods, the selection of features requires to compute a utility measure $A(t_i, c_i)$ for each term t_i of the vocabulary against each class c_i . The number of features to select depends on the number of features the user wants to keep or a cut-off value of the utility measure. There exist diverse utility measures [Forman, 2003], but the main ones are: *mutual information* (MI) that is similar to *information gain* (IG), X^2 and *term-frequency* (TF) [Manning et al., 2008].

TF-based feature selection techniques generally provide less accurate results than X^2 or IG. Moreover, Manning et al. [2008] point out that no matter which X^2 or IG utility measure is preferred, accuracy does not usually varies significantly. Therefore, we have decided to use IG in our experiment. Lastly, to define the number of features to select, we iteratively increase the number of features as long as the F1 measure of the classifier increases. The optimal number of features to select can be determined by looking at the point where the F1 measure starts converging. Figure 9 illustrates the evolution of the F-Measure according to the number of features.

4.4 Training and evaluation of learning algorithms

The training procedure aims at fitting a classification function that maps requirements to disciplines. In other words, the learning method searches for a good set of parameter values by optimising the *misclassification rate* criterion. Equation 3 shows the definition of the *training misclassification rate* where \hat{y}_i is the predicted discipline for the i^{th} example in the training set using the approximated classification function $\hat{\gamma}$. $I(y_i \neq \hat{y}_i) = 1$ if $y_i \neq \hat{y}_i$ and 0 otherwise. If $I(y_i \neq \hat{y}_i) = 0$, then the i^{th} training example was classified correctly; otherwise it was misclassified. The *training misclassification rate* is nothing more than the fraction of incorrect classifications. The most common approach for approximating the parameters consists in minimising the misclassification rate of the training examples.

$$\text{Training misclassification rate} = \frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i) \quad (3)$$

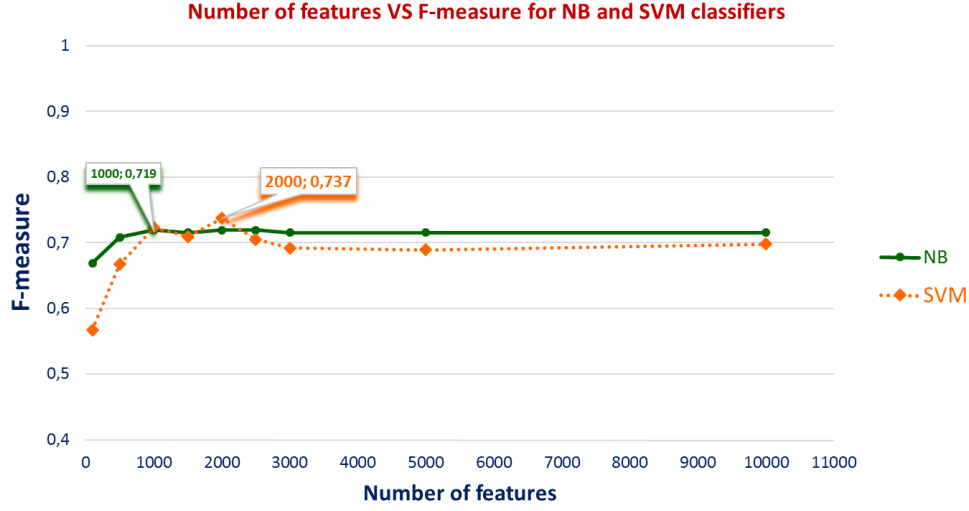


Figure 9: Effect of feature set size on the accuracy of the NB and SVM classifiers.

To evaluate the learning algorithms we look at the *misclassification rate* given in Equation 4 on the testing set and try to minimise it. The best classifier is the one for which the *testing misclassification rate* is the smallest.

$$\text{Testing misclassification rate} = \text{Ave}(I(y_i \neq \hat{y}_i)) \quad (4)$$

In our experiment, we are trying two learning algorithms: Naive Bayes (NB) and Support Vector Machine (SVM). NB and SVM have been preferred because the former is general enough to be applied to almost any classification problem, whereas the latter usually outperforms other learning algorithms in text classification. The choice of NB and SVM was also motivated by two common tradeoffs [James et al., 2013]:

- **Bias VS. Variance.** On the one hand, one can see *bias* as the error that is introduced by approximating a real life problem to a much simple one.

On the other hand, *variance* is the amount by which the approximated classification function $\hat{\gamma}$ would change if we estimate it using a different training set. If a classifier has a high variance, then small changes in the training data can result in a large change in $\hat{\gamma}$. Therefore, a classifier that has a high variance can perform well on the training set, but its performance on a new set of previously unseen data can be unacceptable. This is the problem of overfitting.

The underlying concept behind bias and variance is the *flexibility* of a learning algorithm. A model is flexible if the approximated classification function $\hat{\gamma}$ can fit a wide range of training data closely.

Generally, learning algorithms with low flexibility like NB have high bias but low variance, whereas learning algorithms with high flexibility like SVM have low bias but high variance.

- **Prediction accuracy VS. Model interpretability.** Flexible learning algorithms are more interpretable than restrictive ones, because they can produce a smaller range of

shapes to estimate the classification function γ . NB exhibits less flexibility than SVM and is therefore less accurate but more interpretable.

To carry out our experiment we manually built a testing set by collecting 289 requirements from industrial specifications and labelling them manually. The testing set is made up of 70 ME, 81 EE, 72 CS and 66 RAMS testing examples.

# Features	Accuracy	Precision	Recall	F-measure	ROC Area
100	0.664	0.719	0.664	0.669	0.853
500	0.705	0.726	0.706	0.708	0.879
1000	0.716	0.734	0.716	0.719	0.887
1500	0.712	0.729	0.713	0.715	0.887
2000	0.716	0.734	0.716	0.719	0.889
2500	0.716	0.734	0.716	0.719	0.89
3000	0.712	0.729	0.713	0.715	0.89
5000	0.712	0.729	0.713	0.715	0.892
10000	0.712	0.727	0.713	0.715	0.892

Table 6 Results of NB classifiers according to the number of feature selected.

# Features	Accuracy	Precision	Recall	F-measure	ROC Area
100	0.567	0.666	0.567	0.557	0.756
500	0.667	0.69	0.668	0.667	0.83
1000	0.723	0.74	0.723	0.725	0.859
1500	0.709	0.725	0.709	0.71	0.865
2000	0.737	0.753	0.737	0.738	0.875
2500	0.705	0.73	0.706	0.707	0.86
3000	0.692	0.72	0.692	0.695	0.86
5000	0.689	0.717	0.699	0.703	0.86
10000	0.698	0.711	0.699	0.701	0.856

Table 7 Results of SVM classifiers according to the number of features selected.

Table 6 and Table 7 give the weighted average statistics of our experiments with the NB and SVM algorithms, respectively. Because the SMO implementation of SVM is a single-class algorithm, Weka carries out pairwise classification (ME VS. EE, ME VS. CS, ME VS. RAMS, EE VS. CS, etc.) to solve our one-of multi-class classification problem. In our experiment, with a weighted average accuracy of 74%, the SVM learning algorithms that was trained with a subset of 2000 features is the best classifier. Table 8 shows the detailed statistics of the 4 one-vs-all classifiers, whereas Table 9 is the confusion matrix from which, one could attempt to introduce features that distinguish the mechanical requirements from the electrical requirements, as well as the mechanical requirements from the RAMS requirements. Finally, both curves in Figure 9 illustrate that the number of selected features has a low impact on the accuracy as long as there are at least 1000 features.

We were also curious to evaluate the impact of stemming on the accuracy of our best classifier. Therefore, we added the stemming task to the pre-processing pipeline. Although the resulting vector space was different from our previous experiment, we also selected 2000 features based on the IG utility measure and trained the SVM learning algorithm. The classification function was slightly less accurate as the accuracy was equal to 68.5% confirming the low impact of stemming on classification. Consequently, we did not investigate further classifiers including stemming.

TP rate	FP rate	Precision	Recall	F-Measure	ROC Area	Class
0.643	0.023	0.9	0.643	0.75	0.914	ME
0.79	0.144	0.681	0.79	0.731	0.828	EE
0.792	0.069	0.792	0.792	0.792	0.916	CS
0.712	0.117	0.644	0.712	0.676	0.848	RAMS
0.737	0.09	0.753	0.737	0.738	0.875	

Table 8 Measures by class where last row is the weighted average over one-vs-all binary classifiers.

a	b	c	d	← classified as
45	10	3	12	a = ME
4	64	5	8	b = EE
1	8	57	6	c = CS
0	12	7	47	d = RAMS

Table 9 Confusion matrix of the best classifier: SVM with 2000 features.

5 Conclusion and further work

This paper proposes (1) a natural language processing pipeline to automatically generate Parametric Property-Based Requirements in ENOVIA/CATIA V6 from prescriptive documents, and (2) a machine learning-based text classifier to automatically assign a discipline to a requirement.

The first experiment has shown that the NLP pipeline extracted 194 requirements from an unconstrained document-based system specification with a recall of 1 and a precision rate of 0.73. The extraction of PPBRs was not only challenged by the lack of well-specified verifiable requirements, but also by the need to add a physical unit-converter to calculate the prescribed maximum, minimum and nominal values. In our second experiment, we evaluated the Naive Bayes and Support Vector Machine (SVM) statistical learning algorithms with different pre-processing pipelines and subsets of selected features. After applying the case-folding, stop-word removal and tokenization pre-processing tasks and selecting 2000 features with the Information Gain utility measure, SVM outperformed other classifiers with an accuracy rate of 0.737.

The NLP pipeline can be improved by adding a physical unit converter and can be integrated in the proprietary ENOVIA V6 platform. Regarding the proposed machine learning-based classifier, the first enhancement will be to increase the size of the testing set. We also observed that the classification of a requirement statement is arduous because it contains very few words. For instance, in a manual classification task, one could classify a given requirement statement that contains the words *stress* and *failure* in the discipline *ME* or *RAMS*. One way to overcome this shortcoming is to work out a multi-label classifier that can annotate a requirement statement with none, one or several disciplines. For instance, in the previous example, the requirement would be labelled with both *ME* and *RAMS* classes. Another way to reduce the misclassification rate of the classifier would be to use a hybrid automatic/manual approach. Once classified, all high-confidence decisions are approved, but all low-confidence decisions are put in a queue for manual revision. After having been revised by an expert, the low-confidence decisions can be used to update the training set and the classifier for obtaining a better approximated classification function. Finally, nowadays, within the data mining discipline, there is a tremendous interest for the *deep learning*

methods. Deep learning is a range of learning algorithms that automatically learn feature representations from raw input. Therefore, instead of hand crafting a training set, one could implement a deep learning algorithm that learns feature representations from an extensive large set of domain-specific documents owned by a company. For instance, Lai et al. [2015] studied the use of recurrent convolutional neural networks for text classification.

References

- S. Abiteboul, I. Manolescu, P. Rigaux, M.-C. Rousset, and P. Senellart. *Web Data Management*. Cambridge University Press, New York, NY, USA, 2011.
- C. C. Aggarwal and C. Zhai. *Mining Text Data*, chapter A Survey of Text Clustering Algorithms, pages 77–128. Springer US, Boston, MA, 2012.
- T. D. Breaux and A. I. Antón. Mining rule semantics to understand legislative compliance. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES '05*, pages 51–54, New York, NY, USA, 2005.
- D. Chen and C. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar, October 2014. Association for Computational Linguistics.
- F. Christophe, A. Bernard, and E. Coatanéa. RFBS: A model for knowledge representation of conceptual design. *CIRP Annals - Manufacturing Technology*, 59(1):155 – 158, 2010.
- E. Coatanéa, F. Mokammel, and F. Christophe. Requirements model for engineering, procurement and interoperability: a graph and power laws vision of requirements engineering. Technical report, Aalto university, 2013.
- C. Duan. Clustering and its application in requirements engineering. Technical report, DePaul University, College of Computing and Digital Media, 2008.
- C. Duan and J. Cleland-Huang. Clustering support for automated tracing. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering, ASE '07*, pages 244–253, New York, NY, USA, 2007.
- C. Duan, P. Laurent, J. Cleland-Huang, and C. Kwiatkowski. Towards automated requirements prioritization and triage. *Requirements Engineering*, 14(2):73–89, 2009.
- J. Feldhusen, E. Milonia, A. Nagarajah, and S. Schubert. Enhancement of adaptable product development by computerised comparison of requirement lists. *International Journal of Product Lifecycle Management*, 6(1):20 – 32, 2012.
- R. Feldman and J. Sanger. *Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, New York, NY, USA, 2006.
- A. Ferrari, S. Gnesi, and G. Tolomei. *Requirements Engineering: Foundation for Software Quality: 19th International Working Conference, REFSQ 2013, Essen, Germany, April 8-11, 2013. Proceedings*, chapter Using Clustering to Improve the Structure of Natural Language Requirements Documents, pages 34–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- G. Forman. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, 3:1289–1305, 2003.
- G. Forman. *Computational methods of feature selection*, chapter Feature selection for text classification, pages 257–276. CRC Press Taylor & Francis Group, 2007.

- J. S. Gero. Design prototypes: A knowledge representation schema for design. *AI Mag.*, 11 (4):26–36, 1990.
- F. Houdek. Challenges in automotive requirements management. In *16th international working conference on requirements engineering: foundation for software quality – industrial presentation, REFSQ 2010*, Essen, Germany, June 30–July 2 2010.
- INCOSE. Guide for writing requirements. Report, International Council on Systems Engineering (INCOSE) Requirements Working Group, 2015.
- ISO/IEC/IEEE 15288. International Standard - Systems and software engineering – System life cycle processes, May 2015.
- ISO/IEC/IEEE 29148. International Standard - Systems and software engineering – Life cycle processes requirements engineering, 2011.
- G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning with applications in R*. Springer-Verlag New York, 2013.
- D. E. Jones, Y. Xie, C. McMahon, M. Dotter, N. Chanchevri, and B. Hicks. Improving enterprise wide search in large engineering multinationals: a linguistic comparison of the structures of internet-search and enterprise-search queries. In *12th IFIP WG 5.1 International Conference, PLM 2015, Doha, Qatar, October 19–21, 2015*, 2015.
- J. Kang and P. Saint-Dizier. Discourse structure analysis for requirements mining. *International journal of knowledge content development & technology*, 3(2):43–65, 2013.
- J. Kang and P. Saint-Dizier. *Requirement Compound Mining and Analysis*, chapter Rules on the Web. From Theory to Applications: 8th International Symposium, RuleML 2014, Co-located with the 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, August 18-20, 2014. Proceedings, pages 186–200. Springer International Publishing, Cham, 2014.
- A. Khoo, Y. Marom, and D. Albrecht. Experiments with sentence classification. In L. Cavendon and I. Zukerman, editors, *Proceedings of the 2006 Australasian language technology workshop*, pages 18–25, 2006.
- S. Kleiner and C. Kramer. *Smart Product Engineering: Proceedings of the 23rd CIRP Design Conference, Bochum, Germany, March 11th - 13th, 2013*, chapter Model Based Design with Systems Engineering Based on RFLP Using V6, pages 93–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- E. Knauss and D. Ott. *Requirements Engineering: Foundation for Software Quality: 20th International Working Conference, REFSQ 2014, Essen, Germany, April 7-10, 2014. Proceedings*, chapter (Semi-) automatic Categorization of Natural Language Requirements, pages 39–54. Springer International Publishing, Cham, 2014.
- Y. Ko, S. Park, J. Seo, and S. Choi. Using classification techniques for informal requirements in the requirements analysis-supporting system. *Information and software technology*, 49 (11-12):1128–1140, November 2007.
- S. Lai, L. Xu, K. Liu, and J. Zhao. Recurrent convolutional neural networks for text classification. 2015.

- A. Lash. *Computational representation of linguistic semantics for requirement analysis in engineering design*. Msc thesis, Clemson University, 2013.
- P. Laurent, J. Cleland-Huang, and C. Duan. Towards automated requirements triage. In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 131–140, 2007.
- G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08*, pages 903–914, New York, NY, USA, 2008. ACM.
- N. Madhusudanan, B. Gurumoorthy, and A. Chakrabarti. Evaluation of methods to identify assembly issues in text. In *12th IFIP WG 5.1 International Conference, PLM 2015, Doha, Qatar, October 19–21, 2015*, 2015.
- C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.
- C. D. Manning, R. Prabhakar, and H. Schütze. *An introduction to information retrieval*. Cambridge University Press, 2008.
- C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.
- L. Mich, M. Franch, and P. L. Novi Inverardi. Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(2):151–151, 2004.
- P. Micouin. Toward a property based requirements theory: system requirements structured as semilattice. *Systems engineering*, 11(3):235–245, 2008.
- P. Micouin. Property-model methodology: A model-based systems engineering approach using vhdl-ams. *Systems Engineering*, 17(3):249–263, 2014.
- D. Ott. *Requirements Engineering: Foundation for Software Quality: 19th International Working Conference, REFSQ 2013, Essen, Germany, April 8-11, 2013. Proceedings*, chapter Automatic Requirement Categorization of Large Natural Language Specifications at Mercedes-Benz for Review Improvements, pages 50–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- R. Pinquié, P. Micouin, P. Véron, and F. Segonds. Property model methodology: a case study with modelica. 2016.
- M. D. Riley. Some applications of tree-based modelling to speech and language. In *Proceedings of the Workshop on Speech and Natural Language, HLT '89*, pages 339–352, Stroudsburg, PA, USA, 1989. Association for Computational Linguistics.
- N. Sannier. *INCREMENT: une approche hybride pour modéliser et analyser dans le large les exigences réglementaires de sûreté*. PhD thesis, Université de Rennes 1, 2013.
- N. Sannier and B. Baudry. Defining and retrieving themes in nuclear regulations. In *Fifth International Workshop on Requirements Engineering and Law (RELAW 2012)*, Chicago, United States, 2012.

- F. Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.
- R. Sint, S. Schaffert, S. Stroka, and R. Ferstl. Combining unstructured, fully structured and semi-structured information in semantic wikis. 2009.
- N.-P. Suh. *Axiomatic design: advances and applications*. Oxford University Press, 2001.
- N.-P. Suh. *Complexity: theory and applications*. Oxford University Press, 2005.
- S. Terzi, A. Boura, D. Debashi, M. Garetti, and D. Kiritsis. Product lifecycle management – from its history to its new role. *International Journal of Product Lifecycle Management*, 4(4):360–389, 2010.
- N. Zeni, N. Kiyavitskaya, L. Mich, J. R. Cordy, and J. Mylopoulos. Gaiust: supporting the extraction of rights and obligations for regulatory compliance. *Requirements Engineering*, 20(1):1–22, 2013.
- H. Zhang, A. Sekhari, F. Fourli-Kartsouni, and Y. Ouzrout. Customer reviews analysis based on information extraction approaches. In *12th IFIP WG 5.1 International Conference, PLM 2015, Doha, Qatar, October 19–21, 2015*, 2015.