Requirement mining for model-based product design

Romain Pinquié* and 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
Email: romain.pinquie@ensam.eu
Email: philippe.veron@ensam.eu
*Corresponding author

Frédéric Segonds

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

Nicolas Croué

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

Abstract: PLM software applications should enable engineers to develop and manage requirements throughout the product's lifecycle. However, PLM activities of the beginning-of-life and end-of-life of a product mainly deal with a fastidious document-based approach. Indeed, requirements are scattered in many different prescriptive documents (reports, specifications, standards, regulations, etc.) that make the feeding of a requirements management tool laborious. Our contribution is two-fold. First, we propose a natural language processing (NLP) pipeline to extract requirements from prescriptive documents. Second, we show how machine learning techniques can be used to develop a text classifier that will automatically classify requirements into disciplines. Both contributions support companies willing to feed a requirements management tool from prescriptive documents. The NLP experiment shows an average precision of 0.86 and an average recall of 0.95, whereas the SVM requirements classifier outperforms that of naive Bayes with a 76% accuracy rate.

Keywords: requirements; unstructured; extraction; classification; natural language processing; NLP; supervised learning; machine learning.

Reference to this paper should be made as follows: Pinquié, R., Véron, R., Segonds, F. and Croué, N. (2016) 'Requirement mining for model-based product design', *Int. J. Product Lifecycle Management*, Vol. 9, No. 4, pp.305–332.

Biographical notes: Romain Pinquié is a PhD candidate in the Information Sciences and Systems Laboratory (LSIS UMR CNRS 7296) at Arts et Métiers ParisTech. He received an MSc in Computational and Software Techniques in Engineering, specialising in Computer-Aided Engineering from Cranfield University. His research concentrates on systems engineering, product lifecycle management, and data science to support the design of complex systems.

Philippe Véron is a Full Professor at the Arts et Métiers 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 the 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.

Frédéric Segonds is an Assistant Professor at Arts et Metiers ParisTech and 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 Croué is the Solutions and 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 and Systems Engineering, he successfully worked for the Petroleum French Institute, SAMTECH and LMS and became an expert in systems engineering.

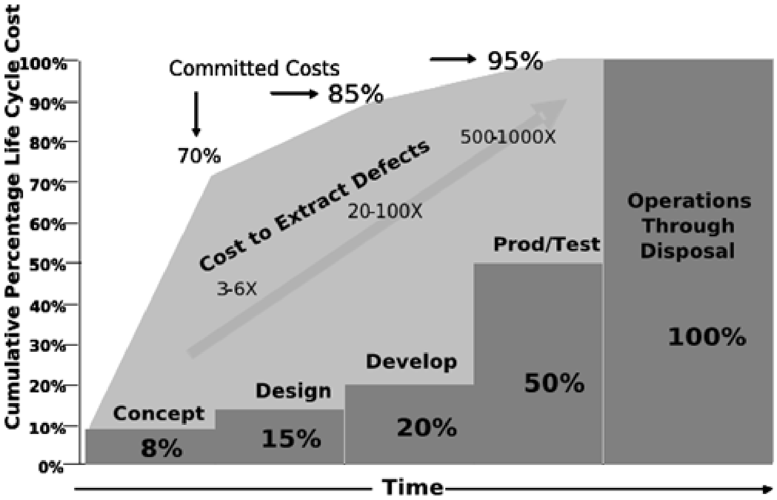
This paper is a revised and expanded version of a paper entitled 'Natural language processing of requirements for model-based product design with ENOVIA/CATIA V6' presented at PLM 15 International Conference, Doha, Qatar, 19–21 October 2015.

1 Introduction

1.1 Context

Requirements engineering is one of the most critical product lifecycle management (PLM) activities. Indeed, it is well-documented that the cost to fix errors increases as the project and product mature (INCOSE, 2015b; Stecklein et al., 2004). Figure 1 illustrates how the committed lifecycle's cost significantly increases with the time. The study of systems engineering effectiveness carried out by Elm (2008) showed that requirements-related factors had a moderately strong relationship to project performances. Moreover, in their analysis of the effectiveness of software defection detection using peer reviews, Selby and Selby (2007) discovered that 49.1% of defects were injected during the requirements engineering phase. The Standish Group (2005) already identified requirements engineering as a major cause of software project failure and success.

Figure 1 Committed lifecycle cost against time



Source: INCOSE (2015b)

Despite this observation, although a major difference between product data management (PDM) and PLM is the extension of the management activity from the design and manufacturing phases to the entire product's lifecycle, very few papers related to requirements engineering have either been presented at the PLM conferences or published in the *International Journal of Product Lifecycle Management* (*IJPLM*). However, requirements engineering has been extensively studied by systems engineering (Sommerville and Sawyer, 1997; Hull et al., 2011; INCOSE, 2015a) and software engineering (Leffingwell and Widrig, 2000; Aurum and Wohlin, 2005; Chemuturi, 2013; Wiegers and Beatty, 2013) practitioners.

Tools are also prone to inequalities. Indeed, there exist various very sophisticated technologies to support the engineering activities that belong to the middle-of-life of a product. 3D geometric modellers, virtual and augmented reality environment, dynamic simulation are standard technologies in industrial design offices. However, the downstream and upstream PLM activities such as requirements engineering still deal with elementary tools like text files, spreadsheets and, at best, relational databases.

1.2 Problem

Among the product's lifecycle 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 example, at Mercedes-Benz, the size of a building block specification varies from 60 to 2,000 pages and prescribes between 1,000 and 50,000 requirements (Houdek, 2010).

Langenfeld et al. (2016) report that, at Bosch, the project to develop a commercial DC-to-DC converter for a mild hybrid vehicle required more than 10,000 requirements. At Sony Ericsson, the total volume of market and platform system requirements exceeds 10,000 (Regnell et al., 2008). At Ericsson Microwave Systems, a product that is tailored to customer requests involves between 3,000 and 6,000 requirements, whereas the development of a subsystem involves between 700 and 1,300 requirements (Alenljung and Persson, 2006). The specification of the FBI Virtual Case file contained more than 800 pages of requirements (Goldstein, 2005). 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, requirements are scattered in unstructured prescriptive documents – e.g., Word, PDF, Excel – and a majority – 79% according to Mich et al. (2004) – are written in unrestricted natural language.

The mushrooming of requirements causes several requirements engineering problems such as the collection, analysis, and traceability of requirements. In this paper, we address two challenges:

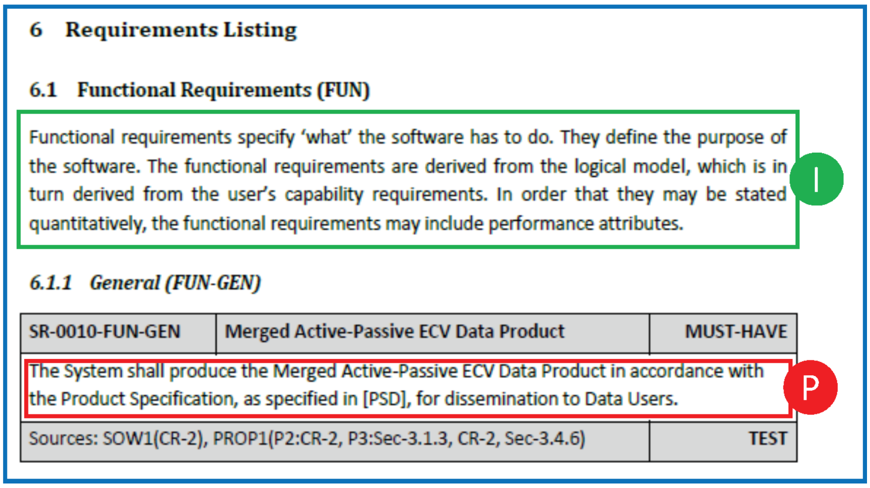
1 the extraction of requirements from prescriptive documents

2 the classification of requirements into disciplines.

1.2.1 Extraction of requirements from prescriptive documents

In this study, a prescriptive document is a document that prescribes requirements and which also usually contains informative statements providing contextual information to better interpret the requirements. For instance, Figure 2 is an extract of a prescriptive document, a specification, that is made up of informative statements (in green) introducing functional requirements (in red). Market surveys, reports of interviews with customers, specifications, concepts of operations, business cases or meeting notes can be prescriptive documents.

Figure 2 Example of a prescriptive document with a set of informative (I) and prescriptive (P) statements (see online version for colours)



6 Requirements Listing

6.1 Functional Requirements (FUN)

Functional requirements specify 'what' the software has to do. They define the purpose of the software. The functional requirements are derived from the logical model, which is in turn derived from the user's capability requirements. In order that they may be stated quantitatively, the functional requirements may include performance attributes.

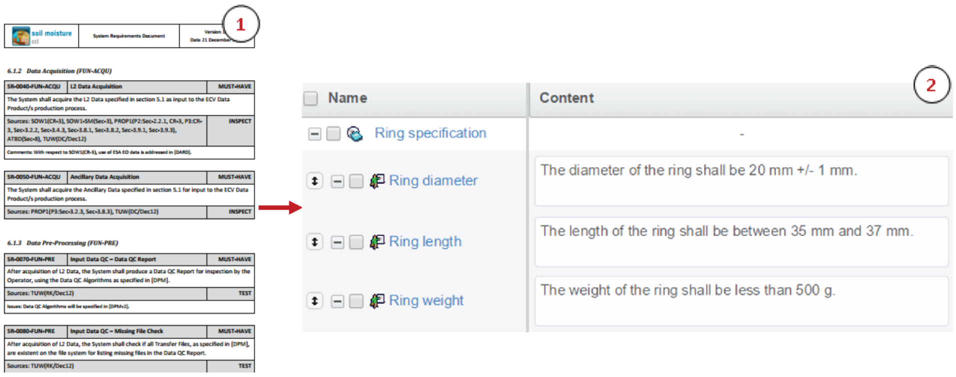
6.1.1 General (FUN-GEN)

SR-0010-FUN-GEN	Merged Active-Passive ECV Data Product	MUST-HAVE
The System shall produce the Merged Active-Passive ECV Data Product in accordance with the Product Specification, as specified in [PSD], for dissemination to Data Users.		
Sources: SOW1(CR-2), PROP1(P2:CR-2, P3:Sec-3.1.3, CR-2, Sec-3.4.6)		TEST

There is no universal definition of what a requirement is. According to the International Council on Systems Engineering (INCOSE, 2015a), a requirement is a formal transformation of one or more needs into an agreed-to obligation for an entity to perform some function or possess some quality within specified constraints. The INCOSE distinguishes a requirement from a need. A need is an agreed-to expectation for an entity to perform some function or possess some quality within specified constraints (INCOSE, 2015a). To put it simply, a need is an ill-defined (incomplete, unverifiable, etc.) prescriptive statement issued by a (sub-)system acquirer, whereas a requirement is a well-defined (complete, verifiable, etc.) prescriptive statement issued by a (sub-)system supplier. By prescriptive statement, we mean a piece of text – a sentence generally – that prescribes a structural or behavioural property (Micouin, 2014) that an object shall own under some potential conditions. There is a derivation phase to transform needs into requirements. For instance, the need ‘the user interface shall be user friendly’ can be derived into the requirement ‘any information shall be accessible within three clicks from each other’. Additionally, there is a third kind of prescriptive statement: constraints. Whereas needs and requirements make certain designs inappropriate for their intended use, constraints make certain design solutions not allowed. The former are usually prescribed by a (sub-)system acquirer or supplier, whereas the latter are prescribed by third-party stakeholders such as regulation authorities or legislations. This segmentation of prescriptive statements into needs, requirements, and constraints is subjective. Labels and underlying concepts vary from one source to another. For instance, needs are sometimes called stakeholder requirements, whereas requirements and constraints are sometimes called system requirements and legal requirements, respectively. Thus, any prescriptive statement can be identified as a requirement with a different qualifying adjective.

When a (sub-)system supplier receives a set of large prescriptive documents, he has no other alternative than to go through the documents to identify the applicable requirements, manually enter them in a requirements management tool, design and verify candidate solutions. Figure 3 illustrates the problem corresponding to the rework task that consists in moving the existing requirements from large prescriptive documents to a requirements management tool.

Figure 3 Inefficient process that consists in (1) identifying the requirements in large prescriptive documents and (2) rewriting them in a requirements management tool (see online version for colours)



soil moisture cci	System Requirements Document	Version Date 21 December

6.1.2 Data Acquisition (FUN-ACQU)

SR-0040-FUN-ACQU	L2 Data Acquisition	MUST-HAVE
The System shall acquire the L2 Data specified in section 5.1 as input to the ECV Data Product/s production process.		
Sources: SOW1(CR-3), SOW1-SM(Sec-3), PROP1(P2:Sec-2.2.1, CR-3, P3:CR-3, Sec-3.2.2, Sec-3.4.3, Sec-3.8.1, Sec-3.8.2, Sec-3.9.1, Sec-3.9.3), ATBD(Sec-8), TUW(DC/Dec12)		INSPECT
Comments: With respect to SOW1(CR-3), use of ESA EO data is addressed in [DARD].		

SR-0050-FUN-ACQU	Ancillary Data Acquisition	MUST-HAVE
The System shall acquire the Ancillary Data specified in section 5.1 for input to the ECV Data Product/s production process.		
Sources: PROP1(P3:Sec-3.2.3, Sec-3.8.3), TUW(DC/Dec12)		INSPECT

6.1.3 Data Pre-Processing (FUN-PRE)

SR-0070-FUN-PRE	Input Data QC – Data QC Report	MUST-HAVE
After acquisition of L2 Data, the System shall produce a Data QC Report for inspection by the Operator, using the Data QC Algorithms as specified in [DPM].		
Sources: TUW(RK/Dec12)		TEST
Issues: Data QC Algorithms will be specified in [DPMv2].		

SR-0080-FUN-PRE	Input Data QC – Missing File Check	MUST-HAVE
After acquisition of L2 Data, the System shall check if all Transfer Files, as specified in [DPM], are existent on the file system for listing missing files in the Data QC Report.		
Sources: TUW(RK/Dec12)		TEST

□ Name	Content
⊟ □ Ring specification	-
↕ ⊟ □ Ring diameter	The diameter of the ring shall be 20 mm +/- 1 mm.
↕ ⊟ □ Ring length	The length of the ring shall be between 35 mm and 37 mm.
↕ ⊟ □ Ring weight	The weight of the ring shall be less than 500 g.

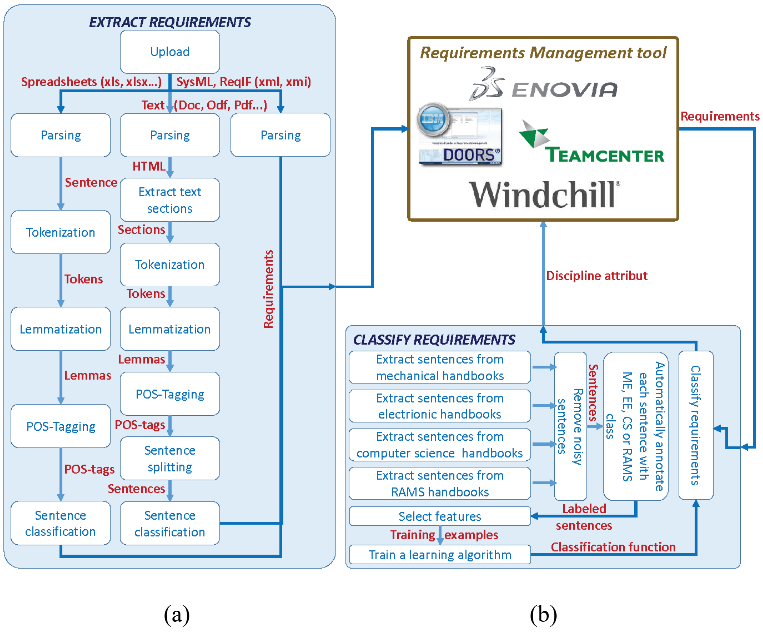
The solution we look for shall be flexible enough to enable (sub-)system suppliers to extract any kind of textual requirements. It shall help to extract and import stakeholder requirements in a database within which their maturity is controlled while deriving them into system requirements. The solution shall also enable suppliers to extract system requirements when reusing in whole or in part a specification from a previous version or a similar existing product. Finally, the solution shall automatically extract legal requirements issued by a third-party.

Being able to automatically extract and import several hundreds or thousands of requirements helps to feed a requirements management tool, but it does not ease their exploitation.

1.2.2 Classification of requirements into disciplines

In the requirements management tool, 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. In most requirements management tools, analysts can manually specify the value of an attribute standing for the discipline a given requirement belongs to. In this scenario, each time an analyst specifies a new requirement, he also has to input the associated topic. The main problem 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.

Figure 4 Flowchart that depicts both contributions, (a) the NLP pipeline (b) the classifier (right) (see online version for colours)



1.3 Contribution

Figure 4 illustrates our twofold contribution. First, we propose a natural language processing (NLP) pipeline to extract textual requirements from unstructured and semi-structured prescriptive documents. Second, we show how to implement supervised machine-learning techniques so as to automatically classify requirements into disciplines.

1.3.1 An NLP pipeline to extract requirements

As we will be showing in Section 2.1, although modal verbs are 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, systems engineering principles defined in the INCOSE guidelines or IEEE standards, but instead, prefer 'to colour outside the line'. It is therefore usual for them to 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. To follow the principle of parsimony, we propose to enhance the existing lists of prescriptive verbs and to stick to a rule-based classifier that does not sacrifice recall for the sake of precision.

The NLP task of sentence splitting is challenged by the authors' lack of rigour when using lists, bullets, numbering, tables, etc. To limit the side effects of incomplete sentence, we combine the NLP capabilities of the Stanford Parser with the parsing ones of Apache Tika to consider the structure (chapters, sections, footers, headers, lists, tables, etc.) of a prescriptive document.

1.3.2 A machine learning-based classifier to classify requirements

As previously explained, the labelling of requirements with a predefined set of disciplines is a text classification problem. Most supervised 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 too time-and-resource consuming. In addition, previous contributions focused on German written requirements only. Thus, we propose to build a supervised classifier for English written requirements. However, instead of asking a supervisor to manually write and classify examples, 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:

- mechanical engineering (ME)
- electrical engineering (EE)
- computer science (CS)
- reliability, availability, maintenance and safety (RAMS).

The rest of the paper is organised as follows. Section 2 is a literature review of the existing techniques to extract (Section 2.1) and classify (Section 2.2) requirements. Section 3 describes the NLP pipeline to extract requirements from prescriptive documents (Section 3.1) and the machine learning-based classifier to classify requirements into disciplines (Section 3.2). Section 4 provides experimental results for the NLP pipeline (Section 4.1) and the machine learning-based classifier (Section 4.2). Finally, Section 5 summarises our contributions and gives perspectives for improvements.

2 Literature review

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, analysed and managed in a PLM software application. Relevant research studies, including, but not limited to, Madhusudanan et al. (2016), Jones et al. (2016) and Zhang et al. (2016) – presented at the last PLM 2015 international conference – or Feldhusen et al. (2012) have demonstrated the benefits of applying such bodies of knowledge to solve PLM problems. This section gives a broader literature review of the current existing solutions to not only extract textual requirements from prescriptive documents, but also classify them into disciplines.

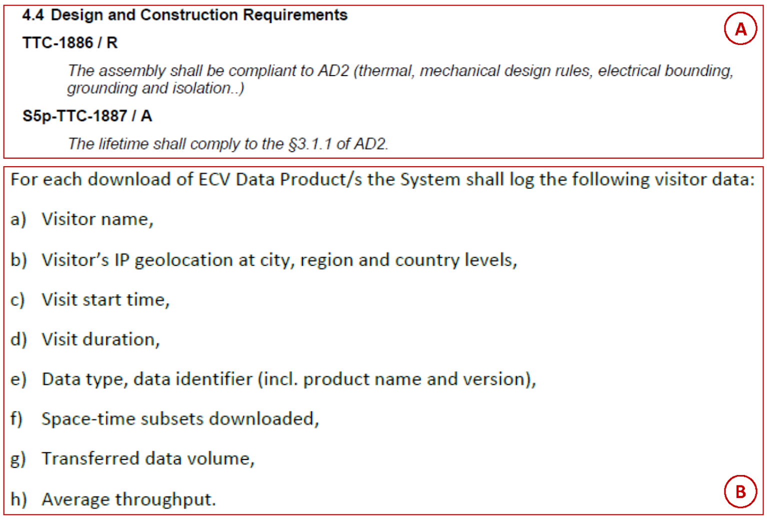
2.1 How can we extract requirements?

A specification $\mathcal{S}$ is a couple $(\mathcal{R}, \mathcal{I})$, where $\mathcal{R}$ is a set of requirements and $\mathcal{I}$ is a set of informative statements. We notice that very few research studies attempt to automatically distinguish both kinds of statements in unstructured prescriptive documents.

Commercial requirements management tools: commercial requirements management tools do not embed any NLP capabilities. When importing a Word document into IBM Rational DOORS, users choose the decomposition level – none, paragraph, sentence or keyword – he prefers to create similar doors objects. The DOORS and Word documents are twins. Outline level 1–9 are imported as rational DOORS headings objects, whereas tables and bulleted lists become rational DOORS tables and lists, respectively. PLM integrated solutions such as ENOVIA requirements central require manual selections to capture the structure of the document before importing it. Requirements also have to be selected manually. Most requirements management tools propose to distinguish informative sentences from requirements based on a list of keywords that is manually predefined by the user. Users shall therefore know and specify all the inflectional forms of each keyword. Such a solution does not help to maximise recall. Morphological analysis like lemmatisation should be used to remove inflectional endings. Finally, some companies, small and medium-sized business specially, do not want to buy, or cannot afford, such heavy database environments, but prefer a flexible standalone processing tool that can extract requirements at a glance.

Modal verbs: modal verbs such as *shall*, *must* and *should* are key lexical features for identifying requirements (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 tokenisation 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 and bulleted lists 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. Figure 5 gives an example where the extraction of complete sentences is not an easy task.

Figure 5 Examples of unstructured text for which basic rules based on capital characters and full stops cannot extract complete sentences (see online version for colours)



4.4 Design and Construction Requirements

TTC-1886 / R

The assembly shall be compliant to AD2 (thermal, mechanical design rules, electrical bounding, grounding and isolation..)

S5p-TTC-1887 / A

The lifetime shall comply to the §3.1.1 of AD2.

For each download of ECV Data Product/s the System shall log the following visitor data:

a) Visitor name,

b) Visitor's IP geolocation at city, region and country levels,

c) Visit start time,

d) Visit duration,

e) Data type, data identifier (incl. product name and version),

f) Space-time subsets downloaded,

g) Transferred data volume,

h) Average throughput.

In case A, the heading would be concatenated with both ids and statements in a single sentence. In case B, the sentence would correspond to the whole enumeration. Kof (2004) discussed these sources of incomplete sentences that require a time-consuming manual rephrasing phase: 1.5 working days for an 80-pages specification. 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. Bernard et al. (2014) reused this solution without stating the potential problems that Kof (2004) and us faced to extract complete sentences from unstructured documents. Although systems engineering best practices recommend to use modal verbs in general and the modal shall in particular, to specify requirements, such a rule is not followed by all companies. Moreover, as explained in Section 1.3.1, in this work, we not only want to collect well-defined requirements, but also ill-defined ones which are often implicitly stated by using prescriptive verbs such as (want, desire, expect, etc.) and derived forms.

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. (2015) 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: knowledge engineering and statistical learning are the two main text classification approaches (Feldman and Sanger, 2006). 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, 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. A rule-based approach usually returns a higher recall than a machine learning-based one.

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. There are two approaches for classifying things: *knowledge engineering* and *machine learning*.

A *knowledge engineering* approach consists in encoding, either declaratively or in the form of procedural classification rules, experts' knowledge about the categories into the system (Feldman and Sanger, 2006). For instance, one can implement a rule that labels all sentences containing the term *computer* with the category *computer science*. Rules can be much more sophisticated regular expressions.

Alternatively, a *supervised learning* approach consists in using a learning algorithm to estimate a classification function from a set of manually pre-classified examples. An example is made up of a vector of features and a label. The vector of features is a set of key characteristics that describes the things to classify. For instance, if one wants to distinguish interrogative sentences from exclamatory ones, question and exclamation marks, interrogative and emotion words are key features. In addition to the vector of features, each training example is defined by a label that represents the category it belongs to. In the previous example, each sentence has to be manually labelled with the category *interrogative* or *exclamatory*. Derived from a learning algorithm, the classification function is used to automatically 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 rule-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, 2007a; Sannier and Baudry, 2012), requirements prioritisation and triage (Laurent et al., 2007; Duan et al., 2009), detecting cross-cutting concerns (Duan and Cleland-Huang, 2007b), or discovering and improving a prescriptive document's structure (Ferrari et al., 2013).

Although unsupervised learning techniques such as clustering and topic modelling are of interest to facilitate the exploitation and management of a large set of requirements, they are not directly suited to our problem as we have predefined categories corresponding to the functional areas of a company.

3 Extraction and classification of requirements

This section introduces the proposed method to extract and classify requirements. First, we introduce a NLP pipeline that extracts text-based requirements from unstructured and semi-structured prescriptive documents. Second, we show how a machine learning-based classifier can automatically map requirements to disciplines.

3.1 Requirements extraction from prescriptive documents

Before introducing the NLP pipeline, we shall clearly distinguish *unstructured* from *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, 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. Requirements management tools such as IBM Rational Doors or Enovia V6 requirements store structured specifications in relational databases.

As the name implies, unstructured data 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 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 format – or as a graph – e.g., XMI format. For instance, a specification that complies with the requirements interchange format (ReqIF) (Object Modeling Group, 2016) is semi-structured.

In this study, we concentrate on the extraction of requirements from unstructured and semi-structured prescriptive documents, with an emphasis on the challenging unstructured ones. To do so, we rely on 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 requirements diagram edited with Papyrus as it is a commonly used open-source tool. Once saved in XMI format, semi-structured SysML requirements diagram can be uploaded in our prototype. Our solution also supports semi-structured ReqIF specifications exported from a requirements management tool. In this study, we use the ProR open-source editor.

Step 2 *<Parsing>:* we trigger a specific parser according to the document format (MIME) identified with 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 are not only useful to extract complete sentences, but also 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 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 rules. The second scenario leads to much less accurate results as we lose the document's structure. 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, 2015a); therefore, it is a single cell in the spreadsheet. Each sentences in a cell being a potential requirement statement.

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

Step 3 *<Extract text sections>:* we use various regular expressions and analyse HTML tags to clean the HTML document generated in Step 2. For instance, we rebuild complete 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, acronyms, preamble, terminology, appendix, etc. to skip the informative sections that may generate false positive results. Finally, by parsing the HTML document, we can extract the text sections with the associated headings. If the document does not contain any section, then the document itself stands for a single section.

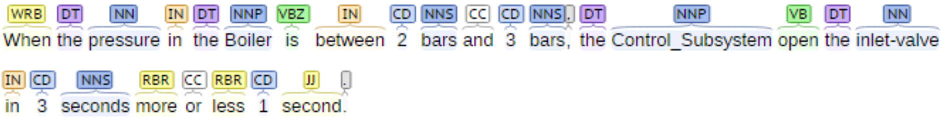
The rest of the process benefits from the NLP capabilities offered by the Java-based Stanford CoreNLP library. Among the various existing NLP libraries (Gate, OpenNLP, etc.), Standford CoreNLP was preferred for several reasons. First, the contributions presented in this paper are part of a larger Java web application. Java as a programming language was therefore mandatory. Second, CoreNLP is one of the most complete NLP library that is actively improved by the Stanford NLP group. Third, the CoreNLP is very well-documented. Finally, the CoreNLP functionalities are also available for Chinese, French, German and Spanish, which will enable us to extend our proposition.

Step 4 *<Tokenisation>:* the Stanford CoreNLP (Manning et al., 2014) tokeniser API iteratively tokenises each specification content, that is, it chops the text up into pieces of a sequence of characters that are grouped together as a useful semantic unit for processing, the tokens. For instance, the tokenisation of the statement “The customer requires that the system shall comply with the ISO-15288 standard.” returns “The, customer, requires, that, the, system, shall, comply, with, the, ISO-15288, standard, .”.

Step 5 *<Lemmatisation>:* the Stanford CoreNLP lemmatiser API iteratively normalises each token by removing the inflectional ending and returns the dictionary form, the lemma. For instance, lemmatisation reduces the tokens ‘requires’, ‘required’ and ‘require’ to their canonical form ‘require’. This not only enables us to increase the recall of the rule-based sentence classifier in Step 8 <sentence classification>, but also to reduce the vocabulary size that can be of very high dimension.

Step 6 *<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 6 shows an example of a labelled sentence.

Figure 6 A sentence whose tokens are annotated with their pos tags (see online version for colours)


WRB DT NN IN DT NNP VBZ IN CD NNS CC CD NNS , DT NNP VB DT NN
When the pressure in the Boiler is between 2 bars and 3 bars, the Control_Subsystem open the inlet-valve
IN CD NNS RBR CC RBR CD JJ .
in 3 seconds more or less 1 second.

Step 7 *<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 8 *<Sentence classification>:* a binary knowledge engineering – a.k.a. rule-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 ∈ {*shall*, *must*, *should*, *may*, *will*, *have to*, *require*, *need*, *obligate*, *restrict*, *provide*, *permit*, *mandate*, *want*, *expect*, *wish*, *desire*, *crave*, *demand* or *recommend*}” is true, then the current sentence$_j$ is classified as a requirement.

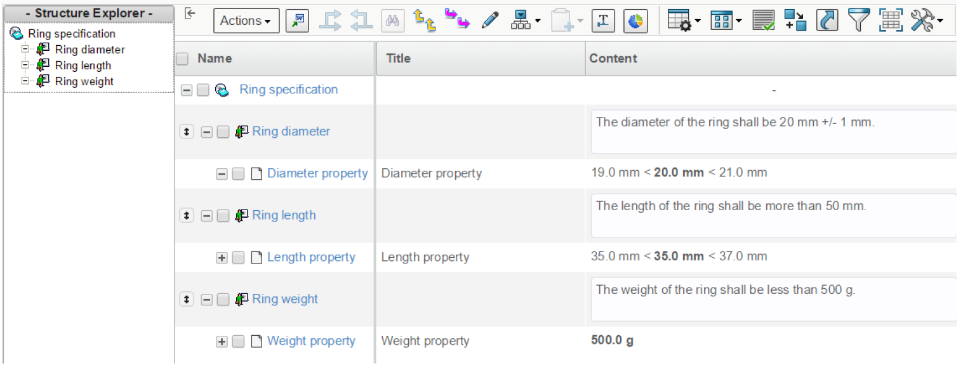
The NLP tasks presented in this section enable users to extract requirements from unstructured and semi-structured prescriptive documents. To date, this capability is part of a standalone JAVA web application, but it could be integrated as a new module or plug-in in commercial requirements management tools.

Once requirements have been extracted, imported and matured in a requirements management tool, designers start the functional, behavioural and physical design synthesis. 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). Nonetheless, when designers deal with a large number of requirements, model-based product design brings about the need to quickly retrieve requirements from the database according to the user’s domain of expertise.

3.2 Classification of requirements into disciplines

Figure 7 shows that requirements management tools also gives analysts the opportunity to define a range of attributes (e.g., name, title, content, author, source, priority, rationale, cost, etc.) among which one can define a specific attribute to classify requirements into disciplines.

Figure 7 A specification with three requirements in the workbench requirements of ENOVIA V6 (see online version for colours)



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**

This section proposes a machine learning-based classifier that automatically predicts a new attribute corresponding to the discipline a given requirement belongs to. As an example, the research study concentrates on four disciplines:

1 ME

2 EE

3 CS

4 RAMS.

The development of the proposed machine learning-based classifier is a three-step process that includes:

1 the creation of three data sets for training, development and testing

2 the selection of a subset of features and a learning algorithm by evaluating several configurations on the development set

3 the evaluation of the best classifier on the test set.

Before detailing each step, we shall clearly define our classification problem.

3.2.1 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* $\mathbb{S} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\}$ where *n* is the number of examples.

3 A fixed *set of classes* $\mathbb{C} = \{c_1, c_2, c_3, c_4\} = \{\text{ME, EE, CS, RAMS}\}$ where each class corresponds to a discipline.

4 A *training example* $(\vec{x}_i, c_i)$.

5 A training set $\mathbb{X} = \{(\vec{x}_1, c_1), (\vec{x}_2, c_2), \ldots, (\vec{x}_n, c_n)\}$ where c_i is the *class*, that is, the discipline, to which the i^{th} requirement belongs to.

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

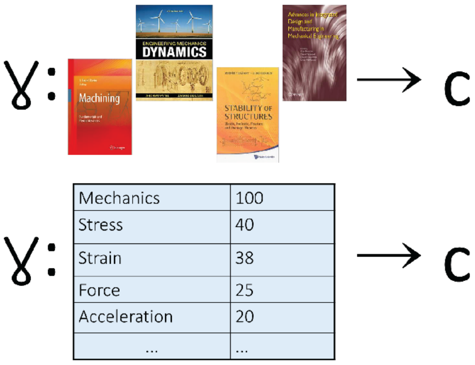
$$\gamma = \mathbb{X} \rightarrow \mathbb{C} \tag{1}$$

3.2.2 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) and 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 Over-simplified illustration that gives the gist of the idea of using handbooks to extract a distribution of keywords that finally serves as features (see online version for colours)



Mechanics	100
Stress	40
Strain	38
Force	25
Acceleration	20
...	...

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 automatically 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 noisy sentences corresponding to equations or bibliographic references thanks to regular expressions, 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 *tokenisation*, *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 lemmatisation of tokens was harmful to performance. Therefore, we did not add the lemmatisation task to the pre-processing pipeline. Stemming being even more aggressive than lemmatisation, we have discarded it too. Thanks to the pre-processing pipeline the initial vocabulary size was reduced from 40,729 features, that is, a 77,481 × 40,729 sentence-feature matrix, to 32,885 features.

3.2.3 Features selection and learning algorithms evaluation

Although the pre-processing tasks reduce the vocabulary size, the sentence-feature matrix is still very large and sparse. The larger the matrix, the longer it takes to train the model. Therefore, it is of interest to reduce the size of the matrix to not only compensate the data sparseness, but also reduce the computing cost. Moreover, 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. In summary, feature selection helps to reduce the training time while preserving, or improving, the accuracy.

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.

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 (2) 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 $\hat{y}$. $I(y_i \neq \hat{y}_i) = 1$ 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.

$$\text{Training misclassification rate} = \frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \hat{y}_i\right) \tag{2}$$

To evaluate the learning algorithms we look at the *misclassification rate* given in equation (3) on the development set and try to minimise it. The best classifier is the one for which the *development misclassification rate* is the smallest.

$$\text{Development misclassification rate} = Ave\left(I\left(y_i \neq \hat{y}_i\right)\right) \tag{3}$$

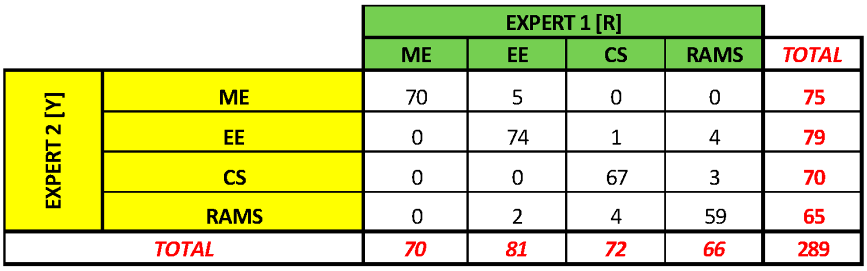
In our experiment, we are evaluating two learning algorithms: naive Bayes (NB) and SVM. NB and SVM have been preferred because the former is general enough to be applied to almost any classification problem, whereas the latter usually outperforms other learning algorithms in text classification. The choice of NB and SVM was also motivated by two common trade-offs (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 $\hat{y}$ 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 $\hat{y}$. 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 $\hat{y}$ can fit a wide range of training data closely. A linear function is for instance less flexible than a polynomial function of degree 2. 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 y. NB exhibits less flexibility than SVM and is therefore less accurate but more interpretable.

In a perfect statistical text classification experiment, we should not look at the test set while developing the classifier, but set aside a development set for testing while our purpose is to find a good value for a parameter, for instance, the number of selected features, the type of kernel function (polynomial or radial), the degree of the kernel function, etc. Then, ideally, at the very end, when all parameters have been set, we should run one final experiment on the test set. In this case, because no information about the test set would be used in developing the classifier, the results of this experiment should be indicative of actual performance in practice and prevent overfitting.

Thus, in addition to the training set used to learn a classification function, the main author of this article manually built a development set by collecting 289 requirements from 15 different industrial specifications and concurrently labelled them manually. The development set is made up of 70 ME, 81 EE, 72 CS and 66 RAMS examples.

Figure 9 4 × 4 contingency table that depicts the annotations of both experts (see online version for colours)



		EXPERT 1 [R]				
		ME	EE	CS	RAMS	*TOTAL*
EXPERT 2 [Y]	ME	70	5	0	0	**75**
	EE	0	74	1	4	**79**
	CS	0	0	67	3	**70**
	RAMS	0	2	4	59	**65**
TOTAL		*70*	*81*	*72*	*66*	**289**

To validate the development set, a second researcher, which was not part of this research study, was asked to manually annotate the development set. Figure 9 shows the contingency table that collects the results of both annotators. The contingency table serves to measure the agreement between both annotators by calculating the inter-rater agreement Cohen’s kappa score (κ). The observed agreement (P_{obs}) is equal to 0.934 [equation (4)], whereas the expected agreement (P_{exp}) is equal to 0.251 [equation (5)]. Therefore, the kappa score is equal to 0.912, which represents an almost perfect inter-agreement between both annotators [equation (6)]. Consequently, there is a shared, objective understanding of the disciplines between both judges.

$$P_{obs} = \sum_{i=1}^{r} P_{ii} = \frac{1}{N}\sum_{i=1}^{r} n_{ii} \approx 0.934 \tag{4}$$

$$P_{exp} = \sum_{i=1}^{r} P_{i.}P_{.i} = \frac{1}{N^2}\sum_{i=1}^{4} n_{i.}n_{.i} \approx 0.251 \tag{5}$$

$$\kappa = \frac{P_{obs} - P_{exp}}{1 - P_{exp}} = \frac{0.934 - 0.251}{1 - 0.251} = 0.911 \tag{6}$$

Table 1 gives 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.

Table 1 Results of NB and SVM classifiers according to the number of feature selected

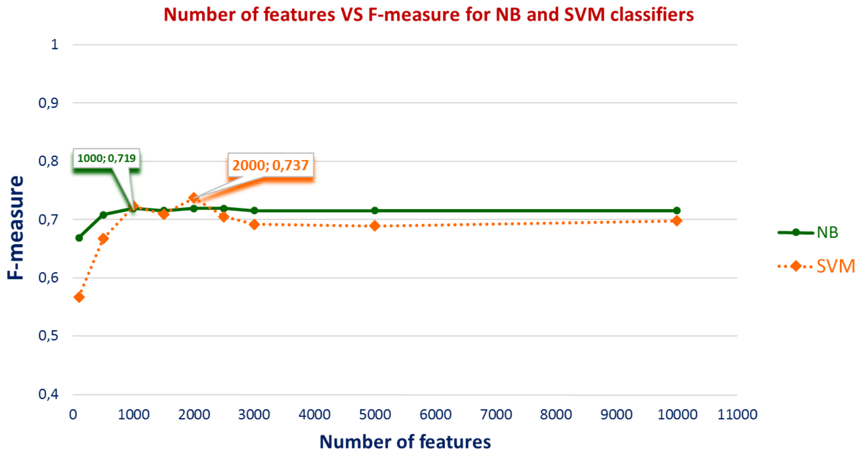
# features	*Accuracy*		*Precision*		*Recall*		*F-measure*	
	NB	*SVM*	*NB*	*SVM*	*NB*	*SVM*	*NB*	*SVM*
100	0.664	0.567	0.719	0.666	0.664	0.567	0.669	0.557
500	0.705	0.667	0.726	0.690	0.706	0.668	0.708	0.667
1,000	0.716	0.723	0.734	0.740	0.716	0.723	0.719	0.725
1,500	0.712	0.709	0.729	0.725	0.713	0.709	0.715	0.710
2,000	0.716	0.737	0.734	0.753	0.716	0.737	0.719	0.738
2,500	0.716	0.705	0.734	0.730	0.716	0.706	0.719	0.707
3,000	0.712	0.692	0.729	0.720	0.713	0.692	0.715	0.695
5,000	0.712	0.689	0.729	0.717	0.713	0.699	0.715	0.703
10,000	0.712	0.698	0.727	0.711	0.713	0.699	0.715	0.701

In our experiment, with a weighted average accuracy of 74%, the SVM learning algorithm that was trained with a subset of 2,000 features is the best classifier. Table 2 shows 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.

Table 2 Confusion matrix for the classification function approximated with SVM and 2,000 features evaluated on the development set

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

Figure 10 Effect of feature set size on the accuracy of the NB and SVM classifiers (see online version for colours)



Finally, both curves in Figure 10 illustrate that the number of selected features has a low impact on the accuracy as long as there are at least 1,000 features selected.

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 2,000 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 (Manning et al., 2008). Consequently, we did not investigate further classifiers including stemming and stuck with the SVM classifier.

4 Results

4.1 Extraction of prescriptive statements

To evaluate the proposed NLP pipeline, we have processed two industrial, unconstrained document-based system specifications.

The first specification specifies an information system to produce global soil moisture data record based on active and passive microwave sensors. This specification is a 106-page document that is freely available on the website of the European Space Agency. It is a Word native PDF document whose structure can be retrieved from its HTML representation. Figure 11 shows that, in this specification, requirements are specified in tables.

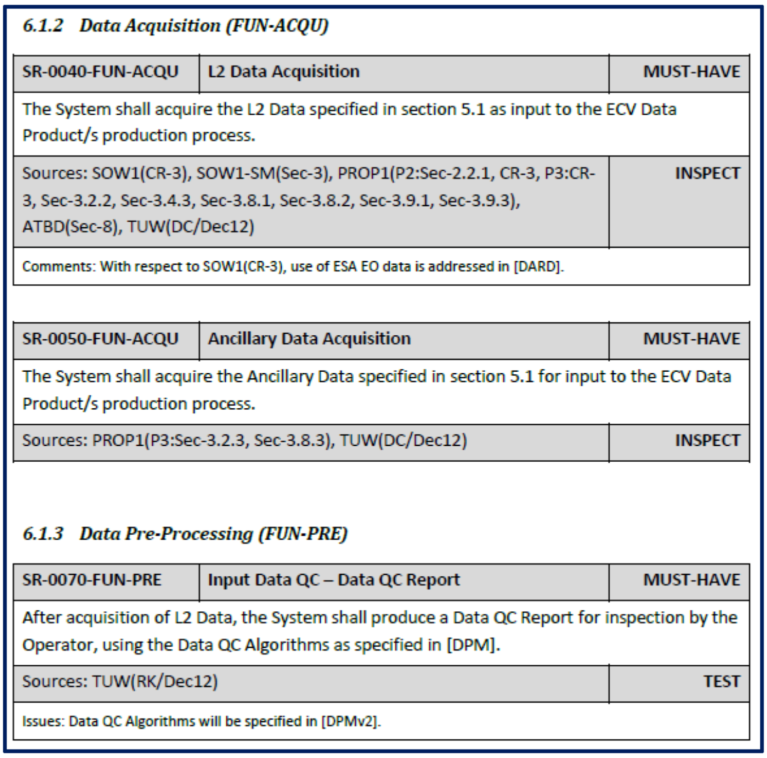
The main author of this article, who has a deep understanding of requirements engineering, has manually reviewed the specification and found 194 requirements. Then, we sent the specification down the NLP pipeline that found 194 true-positive requirements leading to the expected recall of 1. However, 72 informative statements have been mistakenly classified as requirements yielding to a precision of 0.73. This first experimental evaluation is satisfactory as we prefer a high recall with an acceptable precision rather than the opposite.

The second specification is a Word document stored on the website of the South African Government's Department of Public Works. It covers the general technical requirements for the equipment, materials, installation, testing, commissioning and maintenance of electrical installations. Figure 12 shows that, in this specification, requirements are specified in raw text.

As with the first specification, the main author has manually found 1,174 requirements. The NLP pipeline found 1,069 requirements among which three were false-positive ones. The number of true-positive statements is equal to 1,066, which leads to a very high precision of 0.99. However, 108 requirements were missed by the NLP pipeline. This results in a recall of 0.90. This 10% rate of false-negative requirements is due to the use of bulleted lists that cannot be retrieved based on the detection of specific HTML tags. Indeed, these lists were manually defined without using the bullets or numbering feature in Word, but with line breaks, parentheses and whitespaces.

The average precision and recall over both documents are equal to 0.86 and 0.95, respectively. Such performances are typical of classifiers developed with a rule-based approach.

Figure 11 Extract of the specification 1 (see online version for colours)



6.1.2 Data Acquisition (FUN-ACQU)

SR-0040-FUN-ACQU	L2 Data Acquisition	MUST-HAVE
The System shall acquire the L2 Data specified in section 5.1 as input to the ECV Data Product/s production process.		
Sources: SOW1(CR-3), SOW1-SM(Sec-3), PROP1(P2:Sec-2.2.1, CR-3, P3:CR-3, Sec-3.2.2, Sec-3.4.3, Sec-3.8.1, Sec-3.8.2, Sec-3.9.1, Sec-3.9.3), ATBD(Sec-8), TUW(DC/Dec12)		INSPECT
Comments: With respect to SOW1(CR-3), use of ESA EO data is addressed in [DARD].		

SR-0050-FUN-ACQU	Ancillary Data Acquisition	MUST-HAVE
The System shall acquire the Ancillary Data specified in section 5.1 for input to the ECV Data Product/s production process.		
Sources: PROP1(P3:Sec-3.2.3, Sec-3.8.3), TUW(DC/Dec12)		INSPECT

6.1.3 Data Pre-Processing (FUN-PRE)

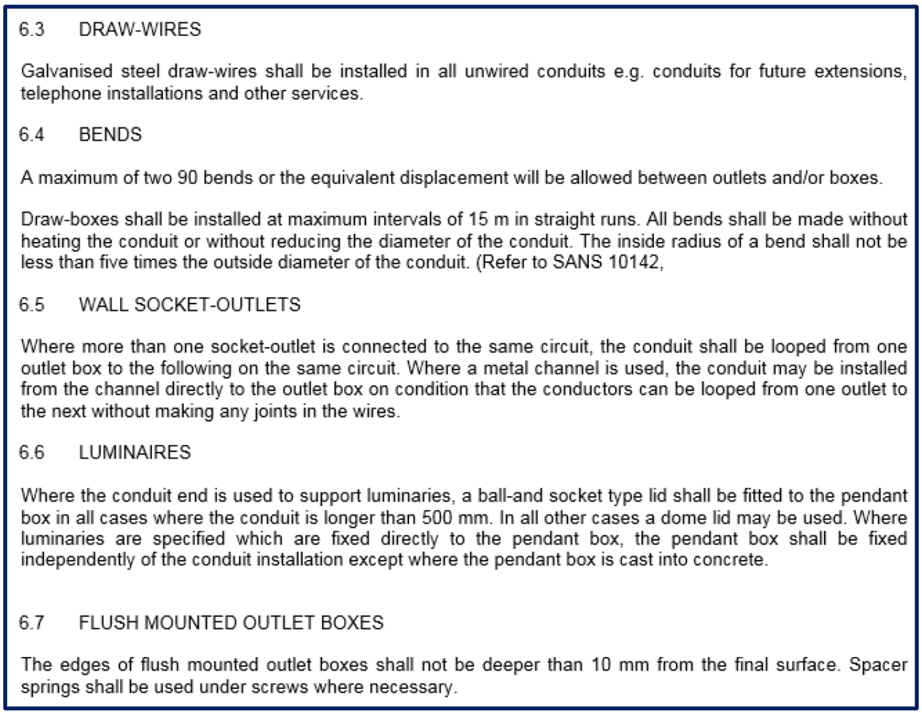
SR-0070-FUN-PRE	Input Data QC – Data QC Report	MUST-HAVE
After acquisition of L2 Data, the System shall produce a Data QC Report for inspection by the Operator, using the Data QC Algorithms as specified in [DPM].		
Sources: TUW(RK/Dec12)		TEST
Issues: Data QC Algorithms will be specified in [DPMv2].		

The recall on specification 1 (1.0) was greater than on specification 2 (0.9) because in specification 1, the modal *shall* is used to prescribe the requirements, whereas in specification 2, some requirements do not contain any prescriptive term – e.g., “PVC or pitch fibre sleeves are not acceptable – refer to par. 3.10 of the department’s standard specification for ‘INSTALLATION OF CABLES’, Section B6” – and were therefore missed by the prototype. The fact that bulleted lists in specification 2 are defined as raw sequences of characters also has an adverse effect on the recall. The main cause of false-positive results in specification 1 is the use of prescriptive words in informative statements which account for more than a third of the document. Conversely, in specification 2, there are very few informative statements and therefore less chance to mistakenly classify informative statements as requirements. This is the reason why the precision is greater in specification 2 (0.99) than in specification 1 (0.73). Though the style and content of both prescriptive documents differ, the effectiveness of the solution is promising. Nevertheless, the variance, that is the amount by which results would vary if we experience the proposed NLP pipeline using different data sets, needs to be more extensively studied by testing the proposition on numerous prescriptive documents.

4.2 Automatic classification of requirements

As explained in the previous section, to prevent overfitting we set aside a development set for testing while our purpose was to tune the parameters of the classifier. We found that the classification function that performs the best on the development set is the one trained with the SVM learning algorithm and 2,000 features. In this section, we run one final experiment on the test set that corresponds to 200 unseen requirements including 50 ME, 50 EE, 50 CS, and 50 RAMS. Because no information about the test set has been used while developing the classifier; the results of this experiment should be indicative of actual performance. Table 3 is the confusion matrix that sums up the results obtained on the 200 examples of the test set with the preferred classifier.

Figure 12 Extract of the specification 2 (see online version for colours)



6.3 DRAW-WIRES

Galvanised steel draw-wires shall be installed in all unwired conduits e.g. conduits for future extensions, telephone installations and other services.

6.4 BENDS

A maximum of two 90 bends or the equivalent displacement will be allowed between outlets and/or boxes.

Draw-boxes shall be installed at maximum intervals of 15 m in straight runs. All bends shall be made without heating the conduit or without reducing the diameter of the conduit. The inside radius of a bend shall not be less than five times the outside diameter of the conduit. (Refer to SANS 10142,

6.5 WALL SOCKET-OUTLETS

Where more than one socket-outlet is connected to the same circuit, the conduit shall be looped from one outlet box to the following on the same circuit. Where a metal channel is used, the conduit may be installed from the channel directly to the outlet box on condition that the conductors can be looped from one outlet to the next without making any joints in the wires.

6.6 LUMINAIRES

Where the conduit end is used to support luminaries, a ball-and socket type lid shall be fitted to the pendant box in all cases where the conduit is longer than 500 mm. In all other cases a dome lid may be used. Where luminaries are specified which are fixed directly to the pendant box, the pendant box shall be fixed independently of the conduit installation except where the pendant box is cast into concrete.

6.7 FLUSH MOUNTED OUTLET BOXES

The edges of flush mounted outlet boxes shall not be deeper than 10 mm from the final surface. Spacer springs shall be used under screws where necessary.

Table 3 Confusion matrix for the classification function approximated with SVM and 2,000 features evaluated on the test set

a	*b*	*c*	*d*	← *classified as*
30	10	5	5	a = ME
3	39	0	8	b = EE
0	2	47	1	c = CS
0	11	3	36	d = RAMS

The results are slightly better than the ones obtained on the development set. The accuracy is equal to 76% (74% on the development set). The precision (0.78), the recall (0.76) and the F-measure (0.76) are also slightly improved. Since there is no significant difference between the results obtained with the development set and the test set, we can conclude that the classification function is not prone to generalisation error. The only threats to validity are the impartiality of the manual classifications and the definition of finer grained categories. To evaluate the first threat, one could ask to experts coming from different backgrounds (functional area, company, country, etc.) to manually classify the test set so as to finally calculate the inter-rater agreement Cohen's kappa score. Regarding the second threat, one could study the quality of the solution with further categories (marketing, quality, etc.) and sub-categories (solid mechanics, fluid mechanics, etc.).

5 Conclusions and further work

This paper proposes:

1 an NLP pipeline to automatically extract requirements from prescriptive documents

2 a machine learning-based text classifier to automatically assign a discipline to a requirement.

To evaluate both contributions we carried out two experiments. In the first one, the NLP pipeline analysed two unstructured specification documents. For the first specification, the NLP pipeline extracted 194 prescriptive statements with a recall of 1 and a precision of 0.73. For the second specification, the NLP pipeline extracted 1,069 prescriptive statements with a recall of 0.9 and a precision of 0.99. The average precision and recall over both documents are 0.86 and 0.95, respectively. In the second experiment, we evaluated the NB and SVM learning algorithms with different pre-processing pipelines and subsets of features. After applying the case-folding, stop-word removal and tokenisation pre-processing tasks and selecting 2,000 features with the information gain utility measure, SVM outperformed other learning algorithms with an accuracy of 0.76.

The NLP pipeline has to be tested on further prescriptive documents issued by different stakeholders. Image processing techniques may also be of interest to analyse unstructured documents within which structural elements such as tables, bulleted lists, etc. cannot be retrieved. Regarding the proposed machine learning-based classifier, the first enhancement will be to increase the size of the test set. Testing of further algorithms such as logistic regression, random forest, tree and neural network is also worth investigating. We have 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. Since requirements that belong to the same topic are often grouped together in a section of the document, the context, that is, the previous and following requirements may improve current results. Another way to overcome this shortcoming is to work out a multi-label classifier that can annotate a requirement statement with, one or several disciplines, or none. 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. Last but not least, the approach must also be tested in various industrial operational contexts so as to not only identify deviations, but also collect feedback from users.

References

Abiteboul, S., Manolescu, I., Rigaux, P., Rousset, M-C. and Senellart, P. (2011) *Web Data Management*, Cambridge University Press, New York, NY, USA.

Aggarwal, C.C. and Zhai, C. (2012) 'A survey of text clustering algorithms', pp.77–128, Springer US, Boston, MA.

Alenljung, B. and Persson, A. (2006) *Decision-Making Activities in Requirements Engineering Decision Processes: A Case Study*, pp.707–718, Springer US, Boston, MA.

Aurum, A. and Wohlin, C. (2005) *Engineering and Managing Software Requirements*, Springer, Berlin Heidelberg.

Bernard, A., Coatanea, E., Christophe, F. and Laroche, F. (2014) 'Design: a key stage of product lifecycle', *Procedia CIRP, 24th CIRP Design Conference*, Vol. 21, pp.3–9.

Breaux, T.D. and Antón, A.I. (2005) 'Mining rule semantics to understand legislative compliance', *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES '05*, pp.51–54.

Chemuturi, M. (2013) *Requirements Engineering and Management for Software Development Projects*, Springer, New York.

Coatanéa, E., Mokammel, F. and Christophe, F. (2013) *Requirements Model for Engineering, Procurement and Interoperability: A Graph and Power Laws Vision of Requirements Engineering*, Technical report, Aalto University.

Duan, and Cleland-Huang, J. (2007b) 'A clustering technique for early detection of dominant and recessive cross-cutting concerns', *Proceedings of the Early Aspects at ICSE: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design, EARLYASPECTS '07*, pp.1–7, IEEE Computer Society, Washington, DC, USA.

Duan, C. (2008) *Clustering and Its Application in Requirements Engineering*, Technical report, DePaul University, College of Computing and Digital Media.

Duan, C. and Cleland-Huang, J. (2007a) 'Clustering support for automated tracing', *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, ASE '07*, New York, NY, USA, pp.244–253.

Duan, C., Laurent, P., Cleland-Huang, J. and Kwiatkowski, J. (2009) 'Towards automated requirements prioritization and triage', *Requirements Engineering*, Vol. 14, No. 2, pp.73–89.

Elm, J.P. (2008) 'A study of systems engineering effectiveness – initial results', *Systems Conference, 2008 2nd Annual IEEE*, April, pp.1–7.

Feldhusen, J., Milonia, E., Nagarajah, A. and Schubert, A. (2012) 'Enhancement of adaptable product development by computerised comparison of requirement lists', *International Journal of Product Lifecycle Management*, Vol. 6, No. 1, pp.20–32.

Feldman, R. and Sanger, J. (2006) *Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*, Cambridge University Press, New York, NY, USA.

Ferrari, A., Gnesi, A. and Tolomei, A. (2013) *Using Clustering to Improve the Structure of Natural Language Requirements Documents*, pp.34–49, Springer Berlin Heidelberg, Berlin; Heidelberg.

Forman, G. (2003) 'An extensive empirical study of feature selection metrics for text classification', *J. Mach. Learn. Res.*, March, Vol. 3, pp.1289–1305.

Forman, G. (2007) 'Feature selection for text classification', in Liu, H. and Motoda, H. (Eds.): *Computational Method of Feature Selection*, pp.257–276, Chapman & Hall/CRC, Taylor & Francis Group, Boca Raton, FL, USA.

Goldstein, H. (2005) 'Who killed the virtual case file?', *IEEE Spectrum*.

Houdek, F. (2010) 'Challenges in automotive requirements management', *16th International Working Conference on Requirements Engineering: Foundation for Software Quality – Industrial Presentation*, REFSQ 2010, June, Essen, Germany.

Hull, E., Jackson, K. and Dick, J. (2011) *Requirements engineering*, 3rd ed., Springer, London.

INCOSE (2015a) *Guide for Writing Requirements*, Technical report, International Council on Systems Engineering (INCOSE) Requirements Working Group.

INCOSE (2015b) *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, 4th ed., John Wiley & Sons, Hoboken, New Jersey, USA.

James, G., Witten, D., Hastie, T. and Tibshirani, T. (2013) *An Introduction to Statistical Learning with Applications in R*, Springer-Verlag, New York.

Jones, D.E., Xie, Y., McMahon, C., Dotter, M., Chanchevrier, N. and Hicks, B. (2016) *Improving Enterprise Wide Search in Large Engineering Multinationals: A Linguistic Comparison of the Structures of Internet-Search and Enterprise-Search Queries*, pp.216–226, Springer International Publishing, Cham.

Kang, J. and Saint-Dizier, J. (2013) 'Discourse structure analysis for requirements mining', *International Journal of Knowledge Content Development & Technology*, Vol. 3, No. 2, pp.43–65.

Kang, J. and Saint-Dizier, P. (2014) 'Requirement compound mining and analysis;, *Proceedings of the 8th International Symposium, RuleML 2014*, Praque, Czech Republic, 18–20 August 2014, pp.186–200.

Khoo, A., Marom, Y. and Albrecht, Y. (2006) 'Experiments with sentence classification', in Cavedon, L. and Zukerman, I. (Eds.): *Proceedings of the 2006 Australasian Language Technology Workshop*, pp.18–25.

Knauss, E. and Ott, D. (2014) '(Semi-)automatic categorization of natural language requirements', *Proceedings of the 20th International Working Conference, REFSQ 2014*, Essen, Germany, 7–10 April 2014, pp.39–54.

Ko, Y., Park, S., Seo, J. and Choi, S. (2007) 'Using classification techniques for informal requirements in the requirements analysis-supporting system', *Information and Software Technology*, November, Vol. 49, No. 11–12, pp.1128–1140.

Kof, L. (2004) 'Natural language processing for requirements engineering: applicability to large requirements documents', in Russo, A., Garcez, A. and Menzies, T. (Eds.): *Workshop on Automated Software Engineering*, Linz, Austria.

Lai, S., Xu, S., Liu, K. and Zhao, K. (2015) 'Recurrent convolutional neural networks for text classification', *Proceedings of the 29th AAAI Conference on Artificial Intelligence, AAAI'15*, pp.2267–2273.

Langenfeld, V., Post, A. and Podelski, A. (2016) 'Requirements defects over a project lifetime: an empirical analysis of defect data from a 5-year automotive project at Bosch', *REFSQ 2016.*

Lash, A. (2013) *Computational Representation of Linguistic Semantics for Requirement Analysis in Engineering Design*, MSc thesis, Clemson University.

Laurent, P., Cleland-Huang, J. and Duan, C. (2007) 'Towards automated requirements triage', *Requirements Engineering Conference 2007, RE '07, 15th IEEE International*, pp.131–140.

Leffingwell, D. and Widrig, D. (2000) *Managing Software Requirements: A Unified Approach*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Li, G., Ooi, B.C., Feng, J., Wang, J. and Zhou, L. (2008) 'Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data', *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08*, pp.903–914.

Madhusudanan, N., Gurumoorthy, B. and Chakrabarti, A. (2016) 'Evaluation of methods to identify assembly issues in text', *12th IFIP WG 5.1 International Conference, PLM 2015*, Doha, Qatar, 19–21 October 2015, pp.494–504.

Manning, C.D. and Schütze, H. (1999) *Foundations of Statistical Natural Language Processing*, MIT Press, Cambridge, MA, USA.

Manning, C.D., Prabhakar, R. and Schütze, R. (2008) *An Introduction to Information Retrieval*, Cambridge University Press, Cambridge, England.

Manning, C.D., Surdeanu, C.D., Bauer, J., Finkel, J., Bethard, S.J. and McClosky, D. (2014) 'The Stanford CoreNLP natural language processing toolkit', *Association for Computational Linguistics (ACL) System Demonstrations*, pp.55–60, 2014.

Mich, L., Franch, M. and Novi Inverardi, P.L. (2004) 'Market research for requirements analysis using linguistic tools', *Requirements Engineering*, Vol. 9, No. 2, p.151.

Micouin, P. (2014) 'Property-model methodology: a model-based systems engineering approach using VHDL-AMS', *Systems Engineering*, Vol. 17, No. 3, pp.249–263.

Object Modeling Group (2016) *OMG Requirements Interchange Format V1.2 (OMG ReqIF).*

Ott, D. (2013) *Automatic Requirement Categorization of Large Natural Language Specifications at Mercedes-Benz for Review Improvements*, pp.50–64, Springer Berlin Heidelberg, Berlin; Heidelberg.

Regnell, B., Svensson, R.B. and Wnuk, R.B. (2008) *Can We Beat the Complexity of Very Large-Scale Requirements Engineering?*, pp.123–128, Springer Berlin Heidelberg, Berlin; Heidelberg.

Riley, M.D. (1989) 'Some applications of tree-based modelling to speech and language', *Proceedings of the Workshop on Speech and Natural Language, HLT '89*, pp.339–352.

Sannier, N. and Baudry, B. (2012) 'Defining and retrieving themes in nuclear regulations', *Fifth International Workshop on Requirements Engineering and Law (RELAW 2012)*, Chicago, USA.

Sebastiani, F. (2002) 'Machine learning in automated text categorization', *ACM Comput. Surv.*, Vol. 34 , No. 1, pp.1–47.

Selby, P.C. and Selby, R.W. (2007) '4.4.2 measurement-driven systems engineering using Six Sigma techniques to improve software defect detection', *INCOSE International Symposium*, Vol. 17, No. 1, pp.640–651.

Sint, R., Stroka, R., Schaffert, S. and Ferstl, S. (2009) 'Combining unstructured, fully structured and semi-structured information in semantic wikis', *Fourth Semantic Wiki Workshop (SemWiki 2009) at the 6th European Semantic Web Conference (ESWC 2009)*, Hersonissos, Greece, 1 June.

Sommerville, I. and Sawyer, P. (1997) *Requirements Engineering. A Good Practice Guide*, John Wiley & Sons, West Sussex, England.

Stecklein, J.M., Dabney, J., Dick, B., Haskins, B., Lovell, R. and Moroney, R. (2004) 'Error cost escalation through the project life cycle', *14th Annual International Symposium International Council on Systems Engineering (INCOSE)*, San Diego, CA.

Suh, N.P. (2001) *Axiomatic Design: Advances and Applications*, Oxford University Press, New York, USA.

Suh, N.P. (2005) *Complexity: Theory and Applications*, Oxford University Press, New York, USA.

Terzi, S., Bouras, A., Debashi, D., Garetti, M. and Kiritsis, D. (2010) 'Product lifecycle management – from its history to its new role', *International Journal of Product Lifecycle Management*, Vol. 4, No. 4, pp.360–389.

The Standish Group (2005) *Chaos*, Technical report, The Standish Group International, Inc.

Wiegers, K. and Beatty, J. (2013) *Software Requirements*, 3rd ed., Microsoft Press, Washington, USA.

Zeni, N., Kiyavitskaya, N., Mich, L., Cordy, J.R. and Mylopoulos, J. (2015) 'Gaiust: supporting the extraction of rights and obligations for regulatory compliance', *Requirements Engineering*, Vol. 20, No. 1, pp.1–22.

Zhang, H., Sekhari, A., Fourli-Kartsouni, A., Ouzrout, Y. and Bouras, A. (2016) *Customer Reviews Analysis Based on Information Extraction Approaches*, pp.227–237, Springer International Publishing, Cham.