AUTOMATED QUALITY ASSURANCE OF NON-FUNCTIONAL REQUIREMENTS FOR TESTABILITY

Abderahman Rashwan

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science in
Software Engineering
Concordia University
Montréal, Québec, Canada

April 2015

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By:	**Abderahman Rashwan**
Entitled:	**Automated Quality Assurance of Non-Functional Requirements for Testability**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in
Software Engineering

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________ Chair
Dr. Emad Shihab

______________________________ Examiner
Dr. Leila Kosseim

______________________________ Examiner
Dr. Nikolaos Tsantalis

______________________________ Supervisor
Dr. René Witte

______________________________ Supervisor
Dr. Olga Ormandjieva

Approved ______________________________
Chair of Department or Graduate Program Director

__________ 20 ______ ______________________________
Dr. Amir Asif, Dean
Faculty of Engineering and Computer Science

Abstract

Automated Quality Assurance of
Non-Functional Requirements for Testability

Abderahman Rashwan

A Software Requirements Specification (SRS) document contains all the requirements to describe a software system to be developed. These requirements are typically separated into Functional Requirements (FRs), which describe the features of the system under development and Non-Functional Requirements (NFRs), which include quality attributes and design constraints, among others. NFRs can have a significant impact on the time of a system's development process and its total cost, as they frequently describe cross-cutting concerns. NFRs that are not testable are typically ignored in system development, as there is no way to verify them. Thus, NFRs must be checked for *testability*. However, for natural language requirements, this so far had to be done manually, which is time-consuming and therefore costly.

In order to improve software development support, we propose a semantic quality assurance method that automatically detects non-testable NFRs in natural language specifications. Our work contains four significant contributions towards this goal: (1) building a generic ontology which represents the main concepts in requirements statements and their relations; (2) Based on this generic ontology, two corpora are developed: The first one is a new gold standard corpus containing annotations for different NFR types. The second one is for requirements thematic roles and testability; (3) A Support Vector Machine (SVM) classifier to automatically categorize requirements sentences into the different ontology classes is introduced; (4) Finally, a rule-based text mining system is used to analyze requirement thematic roles and to flag non-testable NFRs. Based on the SRS corpus, our results demonstrate that the proposed approach is feasible and effective, with an F-measure of 80% for non-testability detection.

Acknowledgments

أَنِ اشْكُرْ لِى وَلِوَالِدَيْكَ

Thank Me and your parents

I pay my sincere gratitude to all the people who made this thesis possible. Much of my appreciations go to my supervisors, Dr. René Witte, and Dr. Olga Ormandjieva, for their continuous guidance and support.

Many thanks to the members of the Semantic Software Lab for their timely suggestions, including Nona Naderi, Elian Angius, and Bahar Sateli.

Rolan Abdukalykov, Olga Ormandjieva, Ishrar Hussain, Mohamad Kassab, and Zakaria Siddiqui are acknowledged for annotating the corpus. I also would like to thank Matthew Smith for managing the manual annotation process.

On a personal note, I would like to convey my thanks to my parents, and my wife for their inspirations and encouragements to complete this task.

Table of Contents

List of Figures

List of Tables

List of Acronyms

API Application Programming Interface

CREOLE A Collection of REusable Objects for Language Engineering

DT Decision Tree

EBNF Extended Backus–Naur Form

EM Expectation-Maximization

FR Functional Requirement

GATE General Architecture for Text Engineering

GUI Graphical User Interface

IR Information Retrieval

JAPE Java Annotation Patterns Engine

KNN K-Nearest Neighbor

LR Language Resource

ML Machine Learning

NFR Non-Functional Requirement

NLP Natural Language Processing

NN Neural Network

NR Non Requirement

OWL Web Ontology Language

PAUM Perceptron Algorithm with Uneven Margins

POS Part-of-speech

PR	Processing Resources
QA	Quality Assurance
SBVR	Semantic of Business Vocabulary and Rules
SDLC	Software Development Life-Cycle
SQWRL	Semantic Query-Enhanced Web Rule Language
SRS	Software Requirement Specification
SVM	Support Vector Machine
TF-IDF	Term Frequency - Inverse Document Frequency
UML	Unified Modeling Language
XML	Extensible Markup Language

Chapter 1

Introduction

If you can't measure a requirement, it is not really a requirement.

Suzanne Robertson

This thesis is concerned with the development of an automatic quality assurance system, focused on providing confidence that non-functional requirements (NFRs) can be fulfilled. The goal of our approach is to ensure the testability of NFRs written in a Software Requirements Specifications (SRS) document. The application of this work is to ensure high quality of the NFRs and thereby improve effectiveness for the consequent testing of these NFRs. We propose a domain-independent quality assurance framework that extracts the different types of NFRs from requirements text, analyzing their main thematic roles, in order to use it in the quality assurance process.

1.1 Motivation

When an initial set of requirements has been elicited and evaluated, it can be captured in a requirements document. Natural language requirements specifications are the most commonly used form (as opposed to formal models, based on a logical framework), accounting for up to 90% of all specifications [MFI04]. However, natural language specifications are prone to a number of errors and flaws, in particular due to the ambiguity inherent in natural language. Moreover, there is a lack of available methods and tools that aid software engineers in managing textual requirements. As the requirements are written in informal natural language, they cannot be easily analyzed for defects. Our approach to overcome these challenges is based on natural language processing (NLP), machine learning techniques, and ontologies.

Recent studies show that designers and developers often focus more on the behaviour of a system (i.e., functional requirements (FRs)) and underestimate the cost and the time of the NFRs [Kas09]. This can lead to cost and time overruns, and ultimately to project failures. Hence, the detection and classification of NFRs has become more important in Requirement Engineering (RE), and is therefore the goal of the work described here.

Most of the terms and concepts in use for describing NFRs have been loosely defined and often there is no commonly accepted term for a general concept [Gli07]. In [CNYM00], the authors present a decomposition and operationalization of NFRs into types and managing them by refining and inter-relating NFRs, justifying decisions, and determining their impact, elaborated in the NFR framework [CNYM00]. Decomposition refines NFRs into more detailed NFRs. For instance, performance can be decomposed into response time and throughput; while operationalization results in strategies for achieving the NFRs, such as prototyping for a usability NFR. Another NFR decomposition operationalization technique is classification, e.g., as provided by the ISO/IEC 25010 international standard [ISO10]. NFR refinement is often enhanced with domain-specific knowledge, as in [JKCW08], where the authors introduce knowledge and rules provided by a domain ontology to induce non-functional requirements in specified domains. Al Balushi and Dabhi [ABSDL07] also use an ontology-based approach to requirements elicitation, aimed at empowering requirements analysts with a knowledge repository that helps in the process of capturing precise non-functional requirements during elicitation interviews. The approach is based on the application of functional and non-functional domain ontologies (quality ontologies) to underpin the elicitation activities. In contrast, our work aims at providing a more generic solution to all types of NFRs, independent from any context.

NFRs that are not testable are typically ignored in system development, as there is no way to verify them. Thus, NFRs must be checked for testability. However, for natural language requirements, this so far had to be done manually, which is time-consuming and therefore costly. We propose a semantic quality assurance method that automatically detects non-testable NFRs in natural language specifications, in order to improve software development support.

1.2 Problem Statement

NFRs represent the borders or the constraints for a software system. They are hard to model, as they are stated informally, and it is difficult to measure them, due to their subjective nature.

There are requirements artifacts and documents written in natural language, describing the system-to-be within the requirement gathering phase. Requirements are generally categorized into FRs and NFRs. Usually, NFRs receive less attention than the FRs, and this may lead to project failure, huge budget increases and/or delays for project delivery [Kas09]. So the problem has many dimensions, requirement statements written in natural language that can be vague and interpreted in different ways. When NFRs are not testable or quantifiable, they are likely to be ambiguous, incomplete, or incorrect [PA09]. The following examples illustrate this issue [RR06]:

1. *"The application shall be user-friendly."*
   This requirement is vague and non-measurable. A possible re-stated requirement could be:
   A new administrator shall be able to add a student, change a student's data, and delete a student within 30 minutes of their first attempt at using the application.

2. *"The system shall be intuitive."*
   The word "intuitive" here is not clear and has different meanings. In

addition, we also do not know for what user group it should be intuitive. A re-phrased requirement can be:
The student shall be able to apply for the course within ten minutes of encountering the application for the first time without reference to any out-of-application help.

3. *"The response shall be fast enough."*
   The concept "fast enough" is not measurable. A modified requirement can be:
   The response time shall be no more than 2 seconds for 90 percent of responses, and no more than 5 seconds for the remainder.

In Table 1, we provide examples for both non-testable and testable types of NFRs.

1.3 Research Goals and Objectives

The main goal of this work is to provide a quality assurance assessment framework of NFR using an automated system. We aim to turn unclear requirements into testable shape, by highlighting all non-testable requirements to the stakeholders, in order to encourage them to improve the requirement. This also makes the system maintainable after the end of a project, and gives the ability to measure progress during project development, through clear objectives and measures. The long-term vision of this work is to create quality assurance applications for different types of defects and errors, in order to decrease the probability of software project failures. This main goal is further decomposed into the following four subgoals: (1) Building a generic ontology that represents the main concepts in the requirements domain, as well as their relations; (2) Based on this generic ontology, two corpora are developed: The first one is a new gold standard corpus containing annotations for different NFR types, the second one is for requirements thematic roles and testability; (3) A Support Vector Machine (SVM) classifier to automatically categorize requirements sentences into the different ontology classes is introduced; (4) Finally, a

Table 1: Examples for Testable and Non-Testable NFRs

NFR	**Non-Testable**	**Testable**
Availability (A)	The product shall be available most of the time.	The product shall achieve at least 98% uptime.
Look-and-Feel (LF)	The intranet pages should display appropriately in all resolutions.	The intranet pages should display appropriately in all resolutions from 800x600 and higher.
Legal (L)	All actions that modify an existing dispute case must be recorded.	All actions that modify an existing dispute case must be recorded for 7 years.
Maintainability (M)	The product shall be updated on a regular basis.	Maintenance releases will be offered to customers once a year.
Operational (O)	The System shall allow many users to work at the same time.	The System shall allow work for a minimum of 6 users to work at the same time.
Performance (P)	The product shall be fast to respond to the queries.	On a 56k connection the system response time must be no more than 6 seconds 90% of the time.
Scalability (SC)	The concurrency capacity must be able to handle peak scheduling times.	The product shall be able to process 10000 transactions per hour within two years of its launch.
Security (SE)	The system shall store messages for tracking purposes.	The product shall store messages for a minimum of one year for audit and transaction tracking purposes.
Usability (US)	The product shall be intuitive and self-explanatory.	At least 90% of untrained realtors shall be able to install the product without printed instructions.

rule-based text mining system is used to analyze requirement thematic roles and to flag non-testable NFRs. We can break down these goals in the following more detailed research objectives:

1. Design a gold standard corpus for six different requirement documents from different backgrounds, to annotate the different types of NFRs and FRs.

2. Design a classifier to classify the requirements into FRs, different types of NFRs, and non requirements (NRs). The classifier allows to convert the requirements artifacts into a machine processable form. This also help the stakeholders by highlighting the NFRs to the analysts and designers.

3. Design a gold standard corpus for the main thematic roles of the requirements statements, including Agent, Modality, Action, Theme, Condition, Goal, and Instrument.

4. Design a rule-based application to automatically extract these main thematic roles and evaluate it by comparing its output with the gold standard corpus.

5. Develop a rule-based technique to highlight non-testable NFRs. This may encourage the stakeholders to enhance these requirements. This is developed to measure the progress during project development through clear objectives and measures.

1.4 Outline

In this chapter, we explained the motivation for applying automatic quality assurance to requirements documents and briefly described our research goals and objectives towards this goal. The remainder of this thesis is structured as follows:

In Chapter 2, we cover the foundations for our work and describe the software engineering concepts that relate to this research. In addition, we also provide an overview of semantic computing concepts that we used in our work, including ontology representation, natural language processing, and support vector machines.

Chapter 3 contains our literature survey, where we describe the related work for NFR classification, requirements conceptualization, and SRS document quality.

The system design is introduced in Chapter 4. The design includes three layers: (1) The description of the ontology and corpus as a data layer; (2) The quality assurance layer, including the NFR classifier and thematic roles extractor; and (3) The non-testability detector layer.

Chapter 5 provides details for the implementation of each layer.

The detailed specification for the corpus and the evaluation of our system is covered in Chapter 6.

Finally, a summary of this research work and possible future developments are discussed in Chapter 7.

Chapter 2

Background

Informal textual descriptions written in natural language are a common means for specifying requirements in early phases of software projects.

Luisa Mich

In this chapter, a number of basic concepts underlying this thesis are introduced. In particular, software engineering, semantic computing, and machine learning concepts are presented.

2.1 Software Engineering Concepts

In this section, we will briefly define the main concepts for the software engineering domain involved in our thesis, in particular requirements engineering, SRS documents, NFRs, and Testability.

2.1.1 Requirements Engineering

Requirements Engineering (RE) is one of the most important phases of a software project. The success or failure of software projects is highly dependent on successful requirements engineering. Industry statistics show that insufficient RE is the root cause in more than 50% of all unsuccessful software projects [van09].

The requirement engineering activities include requirements elicitation, requirements identification, requirements analysis and negotiation, requirements specification, requirements validation, and requirements management [Som06]. Our goal in this work is to focus on quality assurance for requirements specifications, which is introduced in the following subsection.

2.1.2 Software Requirement Specifications

The software requirements specification (SRS) document is the main artifact in software requirements engineering. The SRS document is designed to foster communication between the technical stakeholders, such as analysts, developers, and testers on one side, and non-technical people, such as the clients and the product managers, on the other side. It may be considered as a contract between the service provider and the client to ensure the software will meet their needs. It is typically written in informal natural language [MFI04], which impedes its automated analysis. The SRS includes all system requirements, including the functional requirements (FRs) and the non-functional requirements (NFRs). The FRs describe the system functions, while the NFRs represent quality requirements or constraints in the design and the implementation. In our work, we concentrate on the quality assurance of the NFRs, which are addressed below.

2.1.3 Non-Functional Requirements

Non-functional requirements (NFRs) define the system qualities or attributes for a software system [van09]. The different types of NFRs interact both with each other and with the FRs. In this thesis, the ISO 25010 standard [ISO10] is used to define the different types of NFRs, such as testability, maintainability, extensibility, security, and scalability. Testability for NFRs is the main goal for this work. Table 2 defines the NFR types that we use in our ontology, together with examples.

Table 2: NFR Definitions

Class	**Definition**	**Example**
Constraint	Constraints are defined in [LW03] as restrictions on the design of the system, or the process by which a system is developed, that do not affect the external behavior of the system but that must be fulfilled to meet technical, business, or contractual obligations.	"The system's design will rely heavily on existing patterns and models for the organization of system components."
Utility	The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component (External and internal quality, Utility [Fir03]).	"The GUI under the rehearsal session should be designed to help students to prepare exams."
Security	A measure of the system's ability to resist unauthorized attempts at usage and denial of service while still providing its services to legitimate users (Functionality quality requirement [ISO10]).	"The system shall allow system administrators to manage the users by creating, editing, or deleting users."
Efficiency	The performance relative to the amount of resources used under stated conditions [ISO10].	"The system should be able to handle the concurrent access of the maximum capacity of an exam room during an exam session."
Reliability	The ability of a system or component to perform its required functions under stated conditions for a specified period of time [ISO10].	"The system shall provide the capability to back-up the Data."
Maintainability	The degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers [ISO10].	"The system shall keep a log of all the errors."
Functional Suitability	The degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions [ISO10]. It is only concerned with whether the functions meet stated and implied needs, not the functional specification	"The system's intrinsic characteristic of being designed for change will allow it to easily integrate with any required third-party components."

2.2 Requirements Quality Assurance

Based on a list of defects contained in [van09], Figure 1 shows a requirements defects map. Out of this list, our focus in this thesis is non-testability defect detection. A complete list of requirements defects and their definitions are provided in Table 3.

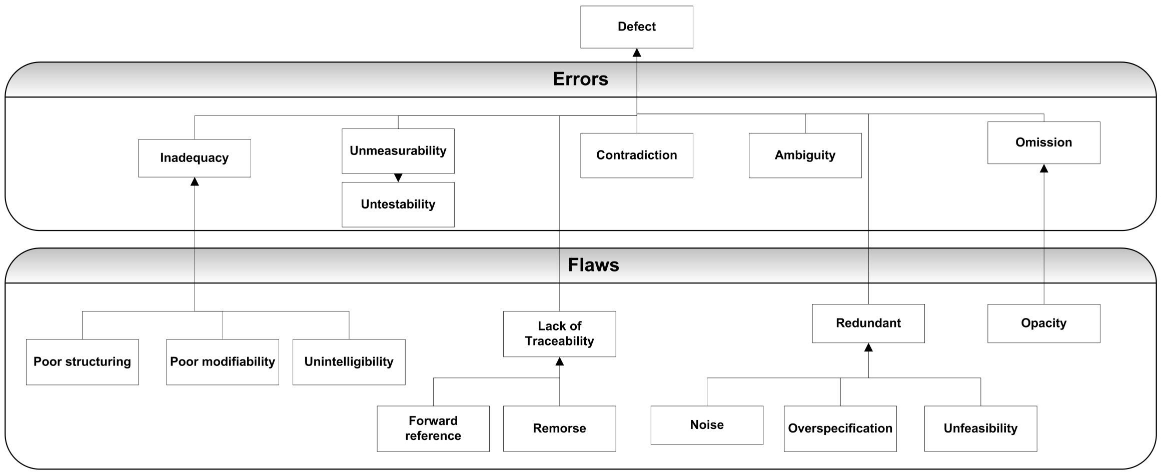


Figure 1: Defects Map

ISO/IEC 25010:2011

The Systems and software Quality Requirements and Evaluation (SQuaRE) standard [ISO10] provides system and software quality models, including:

1. *Quality in use model*, which defines five characteristics that relate to the outcome of an interaction when a product is used in a particular context. The quality in use model includes effectiveness, efficiency, satisfaction, freedom from risk, and context coverage.

2. *Product quality model*, which defines eight characteristics related to the static properties of software and dynamic properties of the computer system. The product quality model includes functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability. Each characteristic has subcharacteristics, for example, maintainability includes modularity, reusability, and testability.

Since testability is part of the product quality model, our definitions and concepts are based on that model.

Testability

Testability is defined according to ISO25010 [ISO10] as a degree of effectiveness and efficiency with which test criteria can be established for a

Table 3: Requirements Defects Definitions [van09]

Defect	**Definition**
Omission	A problem world feature not stated by any requirement
Contradiction	A requirement defining a feature in an incompatible way
Inadequacy	A requirement not adequately stating a problem world feature
Ambiguity	A requirement allowing a feature to be interpreted in different ways
Unmeasurability	A requirement stating a feature in a way that cannot be precisely compared with alternative options
Noise	A requirement has no information related to any problem world feature
Overspecification	A requirement has information related to the solution world not the problem world
Unfeasibility	A requirement that cannot be realistically implemented within assigned budget, schedule or development platform
Unintelligibility	A requirement stated in an incomprehensive way for those who need to use it
Poor structuring	Requirement not organized according to any sensible and visible structure rule
Forward reference	A requirement stating a feature that is not defined yet
Remorse	A requirement stating a feature too late or incidentally
Poor modifiability	A requirement whose modification may propagate to other requirements
Opacity	A requirement whose rationale, source or dependencies are invisible

system, product or component and tests can be performed to determine whether those criteria have been met. Testability is part of maintainability, which is the efficiency with which a product or system can be modified by the intended maintainers. A testable requirement is a requirement that has been broken down to a level where it is precise, unambiguous, and not divisible into lower level requirements.

2.3 Semantic Computing Concepts

In this section, we give a brief introduction to knowledge representation using ontologies, natural language processing (NLP), machine learning (ML), supervised learning, and support vector machines (SVMs), which are used in this thesis.

2.3.1 Knowledge Representation using Ontologies

Ontologies have the ability to model a domain through a formal and explicit representation. An ontology contains concepts and relations to represent a domain semantically. Recently, researchers have increasingly adapted ontologies to conceptualize large amounts of information [CBCG10].

Ontologies have been used in the requirements engineering field for a number of years [CBCG10] for describing SRS documents. Dragoni et al. [DDCPT10] introduced an ontological representation approach for SRS documents. The system queries the concepts using a vector space model. Ontologies also used for formally representing requirements. Dobson and Sawyer [DS06] introduce an ontology for requirements dependability representation. It includes several NFRs, such as: availability, reliability, safety, integrity, maintainability, and confidentiality. Kassab [Kas09] proposed an approach for using ontologies for representing NFRs knowledge. His approach provides NFRs definitions for ontology concepts, without reference to any specific domain. Ontologies are also used for formally representing application domain knowledge. Ontologies should be designed for a specific task [Dev02]. They can be used at development time or at run time within software development [Fon07]. Breitman and Sampaio do Prado

Leite [BSdPL03] proposed an application ontology building process, based on the Language Extended Lexicon (LEL). The lexicon provides elicitation, model and analysis systematization of ontology terms.

Web Ontology Language (OWL). The World Wide Web Consortium (W3C) published the OWL1 standard in 2004, with three levels of expressiveness, including OWL Lite, OWL DL (Description Logic), and OWL full. Powerful software tools for ontologies are reasoners, which have the capability to infer logical consequences from the existing ontology concepts and relations, such as the Racer [HM01] and FaCT++ [TH06] systems.

2.3.2 Natural Language Processing

Natural Language processing (NLP) is defined according to [Lid01] as *"a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications."*

Natural Language processing is considered part of the Artificial Intelligence (AI) discipline. In our work, NLP is used to implement tasks, such as software quality assurance for documents and artifacts, according to specific guidelines in the problem domain.
Natural Language Processing is often divided into [Lid01]:

1. *Phonology level* deals with sounds.

2. *Morphology level* identifies units, such as root and affixes, within words.

3. *Lexical level* converts a sequence of characters into a sequence of tokens.

4. *Syntactic level* deals with the word within the sentence, in order to determine its linguistic structure and meaning.

5. *Semantic level* deals with meaning. Our system is considered at the semantic level, as it enhances the requirements sentences quality. It tries to detect non-testable sentences.

6. *Discourse level* deals with many sentences together as we cannot understand the sentence without their context.

7. *Pragmatic level* deals with the general context understanding of the text.

The main NLP applications can be summarized as [Lid01]:

1. *Information Retrieval (IR)* to search for documents in a large document collection.

2. *Information Extraction (IE)* for extracting structured information from unstructured text. Our system tries to extract the main requirement phrases from a SRS.

3. *Question-Answering* provides a list of answers relative to a user question.

4. *Summarization* provides a shorter version of a larger text.

5. *Dialogue systems* are usually focused in a particular domain. Dialogue systems can support a lot of business applications, such as responding to customers' questions in the domains like flight booking or hotel reservations.

The main NLP tasks that we are using in our system are:

1. *Tokenization*, which deals with chopping a stream of text into pieces such as words, phrases, symbols, or other meaningful elements called tokens [MRS08].

2. *Sentence splitting* task combines tokens into sentences.

3. *Part-Of-Speech tagging* task reads a stream of text in some language and assigns parts of speech to each token, such as noun, verb, adjective, etc. [TM00].

4. *Morphological Analysis* identifies units, such as root and affixes, within tokens.

5. *Text pattern extraction* provides for building custom rules to extract text patterns, e.g., by regular expression matching. The Common Pattern Specification Language (CPSL) is another example technique. CPSL is designed to specify information extraction rules by specifying finite-state grammars [AO98]. We use the Java Annotation Patterns Engine (JAPE), a variant of CPSL, in our rule-based requirement thematic roles extractor system.

Shallow Semantic Parsing

Semantic role labeling or shallow semantic parsing is an NLP task to label the semantic arguments in a sentence. The labels are the thematic roles or relations that were introduced in the generative grammar by [Gru65, Fil68, Jac72]. For example, in the sentence *"The system shall refresh the display"*, the task would be to recognize *"refresh"* as the verb, *"The system"* as representing the AGENT, and *"The display"* as representing the THEME. A list of the most commonly used thematic roles is provided in Table 4.

2.3.3 Machine Learning

Machine Learning (ML) is defined according to [Sim13] as *"the field of study that gives computers the ability to learn without being explicitly programmed"*. It is also defined by [Mit97] as *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"*. Machine learning is categorized into three main fields, including: supervised learning, unsupervised learning, and reinforcement learning. We discuss supervised learning here, as it is the technique applied in this thesis.

Table 4: Thematic Roles [JM09]

Class	**Definition**	**Example**
Agent	The volitional causer of an event	"The *waiter* spilled the soup"
Experiencer	The experiencer of an event	"*John* has a headache"
Force	The non-volitional causer of the event	"*The wind* blows debris from the mall into our yard"
Theme	The participant most directly affected by an event	"Only after John broke *the ice*"
Result	The end product of an event	"The government has built a *regulation-size baseball diamond*"
Content	The proposition or content of a propositional event	"John asked *'You met Mary at a supermarket'*"
Instrument	An instrument used by an event	"He gave John a *shocking device*"
Beneficiary	The beneficiary of an event	"She makes hotel reservations *for her boss*"
Source	The origin of the object of a transfer event	"I flew in *from Boston*"
Goal	The destination of an object of a transfer event	"I drove *to Portland*"

Supervised Learning

Supervised learning is one of the machine learning branches that requires having an output associated with each input data point. To be able to map inputs to correct outputs, parameters for these algorithms are trained mainly to minimize an objective function that quantifies the discrepancy between the predicted outputs produced by the algorithm and the true outputs provided to the trainer. If the desired output exhibits continuous values, the task is referred to as a regression problem, which are normally evaluated by using the squared error between the true and predicted output values (e.g., estimating an apartment rent given inputs related to its location, area, and date of construction). Classification problems, on the other hand, have categorical outputs like predicting the sentiment of social media post concerning certain topics of interest. Classification tasks are evaluated by using either the classification error rate (CER), which is the percentage of predicting the wrong class, or the cross entropy (CE), which measures the distance between the correct output vector and the predicted output.

Examples algorithms in this category include Support Vector Machine (SVM), Neural Network (NN), Decision Tree (DT), K-Nearest Neighbor (KNN), and Gaussian Process (GP). The range of successful applications for supervised learning cover a wide variety of domains, including text classification, machine translation, Automatic Speech Recognition (ASR), Optical Character Recognition (OCR), visual object detection and recognition, among others.

Support Vector Machines (SVMs)

SVMs were theoretically developed from Statistical Learning Theory in the 60s and introduced in COLT-92 [BGV92] by Boser, Guyon & Vapnik.

A Support Vector Machine or Support Vector Network is a supervised learning model used for classification and regression analysis. The training phase in the binary classification problem assigns each example into one category or class. Then, the SVM algorithm assigns new examples to the given two categories. The SVM model is a representation of the examples in space as points and tries to find the widest gap (margin) to act as a separator to the categories, as shown in Figure 2.

In addition to linear classification, kernels are used to perform non-linear classification by projecting the points into a higher dimensional space to improve the categories separation. Linear, Gaussian, and polynomial are examples of kernels. We use a third order polynomial kernel in our NFR classification system.

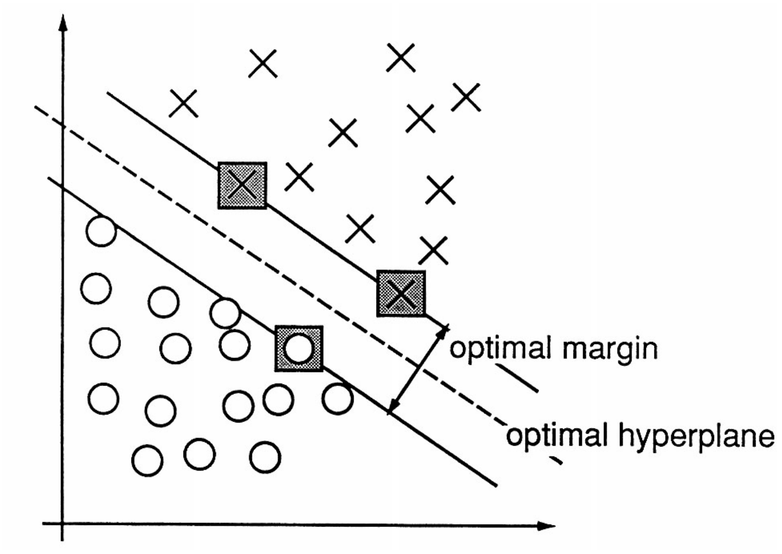


Figure 2: An Example [CV95] of a Separable Problem in a Two-dimensional Space.

2.4 Summary

In this chapter, we detailed the foundations related to our work. In Chapter 3, we compare similar efforts related to the different parts of this thesis, namely NFR classifiers, requirement thematic roles extraction, and requirements quality.

Chapter 3

Literature Review

The idea is to quantify the extent to which each requirement must be met.

Joanne Atlee

In this chapter, we survey existing efforts that are similar to our research work. In Section 3.1, general reviews and studies on requirements engineering related to our work are presented. Section 3.2 addresses related work for NFR classification. Then, Section 3.3 introduces a discussion of similar work for requirements quality assurance. Finally, Section 3.4 discusses work related to requirements thematic roles extraction.

3.1 NLP-based Requirements Engineering

The foundational requirement engineering concepts have been defined in Section 2.1. In this section, we will discuss five market research studies, surveys, and reviews related to our research work.

An interesting market research survey [LMP04] studied market needs for linguistic tools in requirements analysis. This survey, done on 142 American and European software companies, shows that about 70% of the companies need requirements identification automation. It also shows that only one-third of the cases use tools to support requirements analysis. The study found that the market needs requirements analysis tools, because most of the requirements artifacts are in unrestricted natural language or controlled natural language.

Betty [CA07] presents requirements engineering state of the art and challenges facing the domain. The study states that linguistic analysis and ontologies are one of the important state of the art techniques for the analysis, validation, verification of requirements.

Anthony Finkelstein [Fin94] created a review and research agenda for requirements engineering. The study lists a number of important points, such as research direction preconditions, including organizational settings, groundwork, acquisition, modeling, analysis, measurement, communication, and documentation. In the analysis section, the author mentioned several points, such as using automation tools for requirements inspection. In addition, he mentioned using automation with formal reasoning for requirement verification.

Requirements documents, use cases, detailed design documents, use case maps, source code, and comments are all artifacts that use text throughout the software life cycle, as stated in the review [CGC12] by Agustin Casamayor. The study lists a number of areas that use NLP and IR techniques in the RE field. This includes detection and classification of requirements, detection of potentially ambiguous requirements, clustering requirement specifications by functionality, NFR classification, and mapping of concerns in the problem domain to solution domain components.

The use of ontologies in RE was introduced in [CBCG10]. The paper presents three areas where a researcher can apply ontologies, such as requirements specification documents description, the application domain knowledge formal representation, and the formal representation of requirements.

3.2 NFR Classification

We previously defined the different NFR types in Section 2.1.3. Several attempts have been made to develop automated tools for detecting and classifying NFRs from SRS. Cleland-Huang et al. [CHSZS06] developed the

PROMISE corpus, which contains 15 SRS documents annotated by master students at DePaul University. They use the corpus to design an NFRs classifier that consists of two stages: The first stage is identifying indicator items for each NFR class and then calculates a probabilistic weight for each indicator. The weight represents the level of an indicator's importance to a specific NFR class. The second stage is calculating the probability of the classified sentence, based on these indicators. Then, thresholds are applied to decide which class a requirements statement belongs to. All unclassified sentences belong to the functional requirement class. A normalization step is performed before classification, including removing stop words and stemming all remaining words.

Examples of indicators for the security class are “authen” and “access”. Due to the limited amount of training data, the leave-one-out strategy is used to evaluate the algorithm on the PROMISE corpus. The indicators are extracted and weighted based on two different methods: The first method is selecting the highest 5, 10, or 15 indicators repeated for each NFR class. The second method includes all terms as indicators. The classification is done by picking the top score or doing a multi-classification. The multi-classification was finally adopted, as it achieves better results. The work performed three different experiments: The first one using the fixed indicators, giving the poorest results, the second one is using dynamic indicators during the training phase, and the third one is based on test data collected from an industrial domain. Using all indicator terms, the system achieves 59% recall and 29% precision [FBY92]. Using indicator terms mined from 30% of the industrial test data results in 79% recall and 43% precision, which is their best result.

Hussain et al. [HKO08] have built their classifier on the assumption that an NFR sentence's characteristic is that it contains numeric values. The PROMISE corpus is used for classifier training, but with only two classes, FR and NFR. The Stanford parser is used to morphologically stem the words in order to extract the features that train the classifier. A large set of features is then extracted to train the classifier: The authors found that three syntactic features and eight sets of keywords features are dominant in the classification process. A Java application was built to select the features parsed from the sentences. In addition, the Weka [HFH+09] decision tree C4.5 tool kit is used for classification. A ten cross-fold validation method was performed for the evaluation, resulting in 100% recall and 97.8% precision.

Casamayor et al. [CGC09] proposed a recommender system using a semi-supervised learning technique. At the beginning, a sentence is classified into FR or NFR. If the sentence is a NFR, the system suggests the type of the NFR for the analyst to choose. The analyst's feedback is used to enhance the system for the next iteration. A Java tool is built for the recommendation system interface. A Naive Bayes classifier is implemented using the EM (Expectation-Maximization) strategy. For their evaluation, the authors compare the results with supervised classifiers, such as TF-IDF, Naive Bayes, and KNN. Ten cross-fold validation is used to evaluate the algorithms, trained on the PROMISE corpus. For EM training, 75% of the data corpus are used, and 25% are used for testing. The advantage of this approach is a reduction of the labeling effort, compared to supervised methods. It can help to improve a system during the analysis phase. In contrast, it can be hard to build more layers on top of the classification layer, such as the automatic quality assurance layer, using this method, as the tool needs input from a user.

3.3 Requirement Quality Assurance

Requirements quality assurance (QA) is an active area of research, where numerous automated tools have been proposed.

Hussain et al. [HOK07, OHK07] developed a decision tree C4.5 classifier to detect ambiguity in SRS documents. The process is semi-automated, due to client interaction. The results using 10-fold cross-validation demonstrate 86.67% accuracy. This work concentrates on detecting surface understanding ambiguities, rather than conceptual understanding. Ambiguous keywords, syntactic and discourse-level features are used to build the classifier. The problem descriptions are collected from a corpus[1], which is annotated to train the classifier.

Ferrari et al. [FdSG14] proposed a quality assurance tool, named *Completeness Assistant for Requirements* (CAR), which helps a requirements engineer in discovering relevant concepts and interactions in a requirements document. The development steps are Part of Speech (POS) tagging, selecting some sequences of POS as linguistic filters, calculating the C-NC [BDMV10] value that indicates how much a word or a multi-word is likely to be conceptually independent from the context in which it appears. Two terms are related when they frequently appear together. The authors performed a pilot project to evaluate their CAR tool, by writing requirements both with and without tool support. The work also presents two different metrics to measure the requirements completeness, called degree of concept completeness and degree of interaction completeness. The usage of the tool helped in improving the completeness of the requirements specification. The authors argue that the proposed tool can play a complementary role during requirements definition. Backward functional completeness is higher when the tool is employed with 8.6% in average. Forward functional completeness is higher when the tool is employed with 14.3% on one subject, lower with 10% one the other subject.

Park et al. [PKKS00] proposed a requirements analysis support system, where they identify possible redundancies and inconsistencies. In addition, they extract possibly ambiguous requirements, by measuring the similarity between requirement sentences. The authors use an indexing scheme, then combine a sliding window method with a syntactic parser. The system uses z-scores [MBK91] and Salton's cosine coefficients [SM86] to measure similarity between sentences.

Fabbrini et al. [FFGL01] propose a tool called QuARS (Quality Analyzer of Requirements Specification) for natural language software requirements analysis. First, SRS documents are analyzed by a lexical analyzer to verify the English grammar. A syntactical analyzer is used to build the derivation trees of each sentence. Finally, a quality evaluator, which depends

[1]ACM's OOPSLA designfest available online at http://designfest.acm.org/

on the rules of a quality model and dictionaries to perform the sentences evaluation, is applied. The tool aims to provide its users with warning messages about potential defects. Their quality model contains several high-level properties, including completeness, understandability, and consistency. The authors applied QuARS on four different domains: business, space software, telecommunication, and security applications. The tool detects about 50% of the defects on each document. Multiplicity, vagueness and under-specification indicators are the most common defects detected in the test set. In one example of the security domain, the requirement mentions the word “key”, but it is vague because it could be public key, private key, or secret key.

Castaneda et al. [CBC12] propose a tool called OntoSRS for SRS documents to improve the quality using an ontology. This work attempts to address requirements quality attributes, such as ambiguity, insufficiency, and incompleteness. OntoSRS is based on the organized SRS defined in the IEEE 830 standard [Pre98]. Their main idea is to populate the SRS into an ontology and then apply a query language, such as SQWRL [OD08], to extract the defects. The authors did not evaluate the impact of their tool on a specification.

ReqWiki [SAW13] is a novel open source web-based approach for software requirements engineering. It is based on a semantic wiki that includes natural language processing (NLP) assistants, which work collaboratively with humans on the requirements specification documents. It is the first Requirements Engineering tool that combines wiki technology for collaborative use and semantic knowledge representation for formal queries and reasoning with natural language processing assistants within a single, cohesive interface. ReqWiki provide a number of services to help the analysts to write a better requirement such as, writing quality assessment, readability assessment, information extractor, requirement quality assurance, and document indexer. Requirement quality assurance is a service based on the NASA requirements quality metrics [Pow07]. It detects issues like incompletes, Options and Weak Phrases within specifications. To measure the effectiveness of the NLP services, the authors compared outstanding defects in revised SRS documents with and without NLP support. They found that using NLP services for SRS quality assessment purposes significantly reduced the number of remaining issues throughout all defects.

3.4 Semantic Analysis of RE Statements

Several attempts have been made to develop automated tools for extracting requirements thematic roles.

Farfeleder et al. [FMK$^+$11] propose a semantic guidance system (i.e., a boilerplate requirements elicitation tool) to assist requirements engineers. In their approach, capturing requirements is based on a semi-formal representation. The relations and axioms of the domain ontology are used to suggest concept names and thematic roles. The guidance system provides good suggestions for more than 85% of the cases. However, the authors mention that their tool needs to be evaluated on a larger data set.

Umber et al. [UBN11] developed a prototype tool based on the semantics of a business vocabulary and corresponding rules (SBVR). SBVR business vocabulary consists of terms and concepts used by a business organization or community. Their tool can be used by software engineers to record and automatically transform natural language software requirements to a (SBVR) software requirements specification. However, this work does not address the analysis of NFRs for testability. The analysis steps of their tool are tokenization, sentence splitting, Part-of-Speech (POS) Tagging, morphological analysis, semantic interpretation, extracting object types, extracting individual and verb concepts, extracting quantifications, constructing fact types, applying semantic formulation, and finally generating SBVR requirements. This tool achieves a recall of 91.66% and a precision of 93.61% on a small case study, containing seven sentences from the domain of an online ordering system.

OntRep [MWHB11] is designed to keep a set of requirements consistent. The OntRep tool creates predefined concepts, providing the requirements to be categorized, removing stop words, stemming, finding synonyms, and hyponyms using WordNet [Mil95], assigning requirements to categories, saving the elements in the ontology. Finally, semantic requirement conflict analysis is performed by parsing requirements using a EBNF [RSH09] grammar templates, linking requirements components to semantic concepts, and applying ontology-based reasoning to extract logical inconsistencies between facts, as well as numerical inconsistencies. The authors evaluated the effectiveness of the OntRep conflict analysis approach in a case study with 6 project managers in 2 teams. A requirements expert and an OntRep user performed the same tasks to enable comparing the quality of results. OntRep found all conflicts in the requirements, while manual conflict analysis identified 30 to 80% of the conflicts.

3.5 Discussion

In Section 3.1, the use of ontologies in RE study is presented. In this thesis, an ontology is used to represent the different types of NFRs, such as security, usability, and maintainability. We also employ an ontology model for requirements phrase constituents, such as agent, modality, action, theme, condition, goal, instrument.

In Section 3.2, we presented Gokhan et al.'s work [GCSY08]. Their approach is very close to our work [ROW13], where machine learning is used for NFR classification. However, the target of their research is quite different from our work, where quality assurance is the main task to be automated.

In this thesis, we developed a new NFR corpus that contains richer annotations, based on a formal ontology. Our SVM-based classifier also significantly improves on previously described works based on the PROMISE corpus. The ontological foundation of our work allows to automatically transform software requirements documents into a semantic representation, which can then be further processed in order to *(i)* estimate the cost of the software system and *(ii)* measure the quality of the written requirements.

Now, we will discuss the research gap we detected in existing work.

1. While evaluating the PROMISE corpus that most of the NFR classifiers systems described above use [CHSZS06, HKO08, CGC09], we realized that:

   (a) It does not cover all requirements artifact types, such as vision documents, use case descriptions, supplementary specifications, as well as information included in e-mails or minutes of meetings.

   (b) Requirement sentences in this corpus contain only a single requirements type. This is an artificial assumption, as a single sentence can contain multiple requirements.

   (c) The documents in the corpus were written by master students. However, real-world SRS and related requirements documents are written using different writing styles and at levels of abstraction.

   In this thesis, we develop a new corpus that handles the limitations of the PROMISE corpus above.

2. The NFR classifiers discussed before [CHSZS06, HKO08, CGC09] are not based on any NFR semantic ontology. This means, they do not provide for populating an ontology and run any queries using reasoners. In our work, we first propose an ontology for the different types of NFRs, which forms the formal basis for constructing an NFR corpus and training a classifier based on it.

3. The requirement quality assurance tools presented in Section 3.3 address quality attributes for requirements, such as, ambiguity detection, completeness assistance, inconsistencies, understandability, insufficient, and consistency identification. However, testability has not been addressed so far. In addition, existing research [HOK07, OHK07, FdSG14, PKKS00, FFGL01, CBC12, OD08] covers requirements specification in a general way. In our work, we deal with NFRs specifically, as it is one of the important failure points in software projects.

In Chapter 4, we discuss our system's requirements, based on the research gap presented above. Then, we will address its design according to these requirements.

Chapter 4

System Design

Elaborating a good requirement
document is difficult. We need to cater
for multiple diverse quality factors.
Each of them may be hard to reach.

Axel van Lamsweerde

The goal of our work is to assess the requirement quality attribute *Testability*.

In this chapter, we present our system design. In particular, the system requirements are derived based on the research gap analysis presented in the last chapter. Our system design contains three layers, which are (1) corpus and ontology, (2) SVM classifier and thematic roles extractor, and (3) the non-testability detector.

4.1 Methodology

In this section, we analyze the requirements for our system, based on what we discussed and summarized in the literature review in Chapter 3. In particular, the non-testability quality assurance methodology phases are presented in Figure 3.

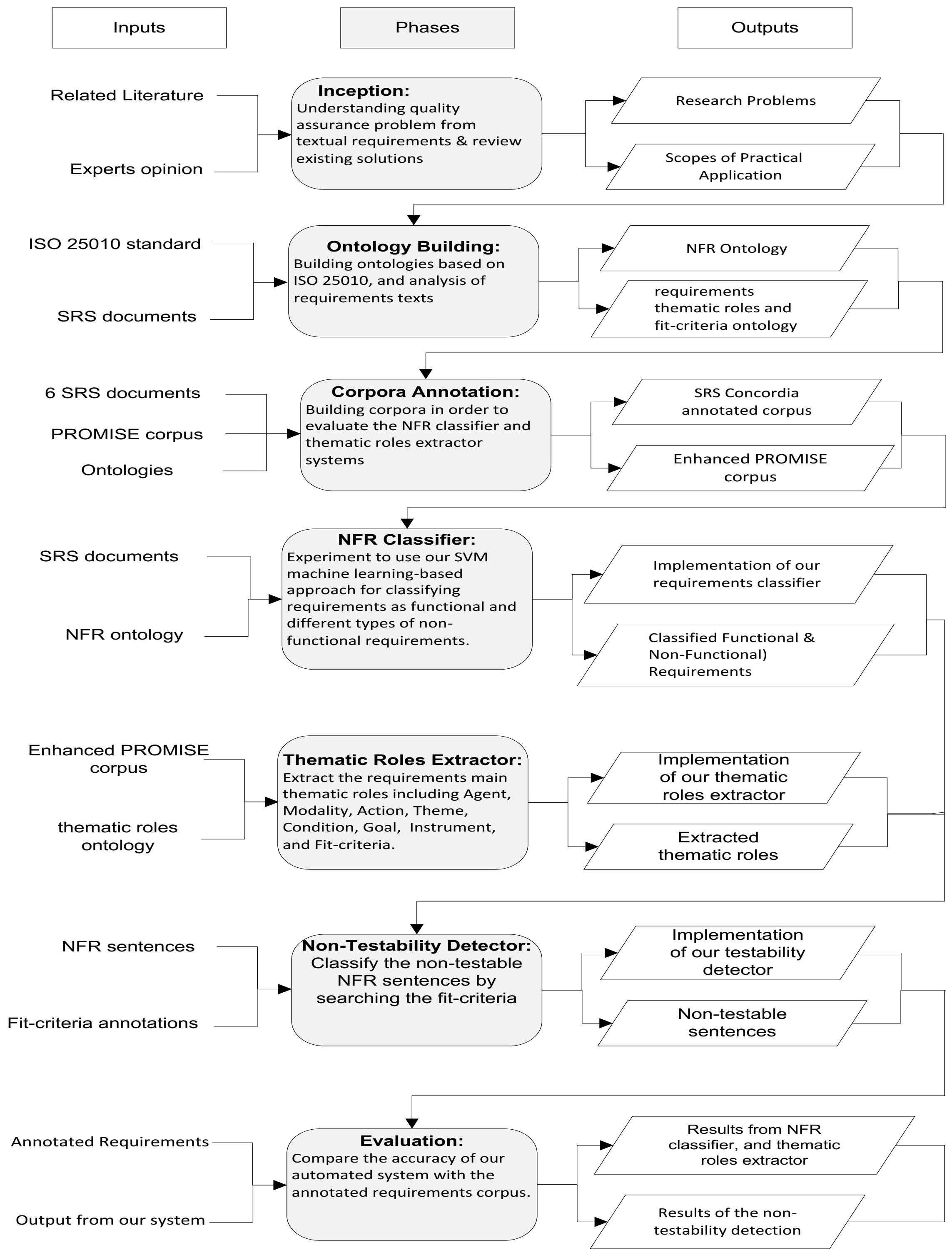

Inputs
Phases
Outputs
Related Literature
Experts opinion
Inception:
Understanding quality assurance problem from textual requirements & review existing solutions
Research Problems
Scopes of Practical Application
ISO 25010 standard
SRS documents
Ontology Building:
Building ontologies based on ISO 25010, and analysis of requirements texts
NFR Ontology
requirements thematic roles and fit-criteria ontology
6 SRS documents
PROMISE corpus
Ontologies
Corpora Annotation:
Building corpora in order to evaluate the NFR classifier and thematic roles extractor systems
SRS Concordia annotated corpus
Enhanced PROMISE corpus
SRS documents
NFR ontology
NFR Classifier:
Experiment to use our SVM machine learning-based approach for classifying requirements as functional and different types of non-functional requirements.
Implementation of our requirements classifier
Classified Functional & Non-Functional) Requirements
Enhanced PROMISE corpus
thematic roles ontology
Thematic Roles Extractor:
Extract the requirements main thematic roles including Agent, Modality, Action, Theme, Condition, Goal, Instrument, and Fit-criteria.
Implementation of our thematic roles extractor
Extracted thematic roles
NFR sentences
Fit-criteria annotations
Non-Testability Detector:
Classify the non-testable NFR sentences by searching the fit-criteria
Implementation of our testability detector
Non-testable sentences
Annotated Requirements
Output from our system
Evaluation:
Compare the accuracy of our automated system with the annotated requirements corpus.
Results from NFR classifier, and thematic roles extractor
Results of the non-testability detection

Figure 3: Phases of our Methodology

4.1.1 Ontology Building Phase

Research question #Q1: Can the NFR types and the requirements main thematic roles be modeled through an ontology, in order to use it in the quality assurance process?

Requirement #1.0: Building an NFR and Requirements thematic roles Ontology: To build corpora with annotated NFRs and requirement thematic roles, an ontology has to be created. We can then use the annotations of the developed corpora to populate this ontology with instances (individuals). We base our NFR ontology on the work by Kassab [Kas09]; However, due to sparseness in our corpus we create an adapted version of the ontology. In addition, to support automatic detection of non-testable requirements, it is necessary to design a fit-criteria and requirement phrase ontology and link it to the NFR one.

4.1.2 Corpus Annotation Phase

Research question #Q2: Can we create corpora for the NFR types and the requirements main thematic roles in order to use them in a quality assurance text mining application?

Requirement #2.1: NFR Corpus. In order to be able to develop and evaluate automated requirements analysis tools, we need a gold standard corpus. This gold standard corpus will provide fine-grained annotations of the requirements. In addition, an ontological classification of different NFR types, such as constraint, security, usability, maintainability is needed. Since no such corpus existed, we developed a new corpus, based on the NFR ontology.

Requirement #2.2: Requirements thematic roles Corpus. To build a quality assurance framework, we need to have an annotated corpus with the main requirement thematic roles, such as *Agent, Modality, Action, Theme, Condition, Goal,* and *Instrument*.

4.1.3 NFR Classification Phase

Research question #Q3.1: Can we build an application to classify the different types of NFRs?

Requirement #3.1: NFR Classification. The NFR quality assurance application has to classify requirements according to the different NFR types defined in the ontology. To be able to ignore sentences that do not contain any requirement, the classifier additionally has a special class called 'non-requirement' (NR). The classifier's output is then consumed in the subsequent quality assurance phases.

4.1.4 Requirements Thematic Roles Extraction Phase

Research question #Q4.1: Can we build an application to extract the main requirements thematic roles from a SRS document, based on the developed ontology?

Requirement #4.1: Requirements Thematic Roles Extraction. Our application has to extract the possible thematic roles that exist in requirement sentences, such as, agent, modality, action, theme, condition, goal, and instrument. Additionally, it has to identify the fit-criteria, including numbers, as well as units of time, distance, display, and connection speed.

4.1.5 Non-Testability Detection Phase

Research question #Q5.1: Can we build an application to extract non-testable NFR sentences?

Requirement #5.1: Non-Testability Detection in NFR. The tool has to identify non-testable NFR sentences, based on the existence of fit-criteria.

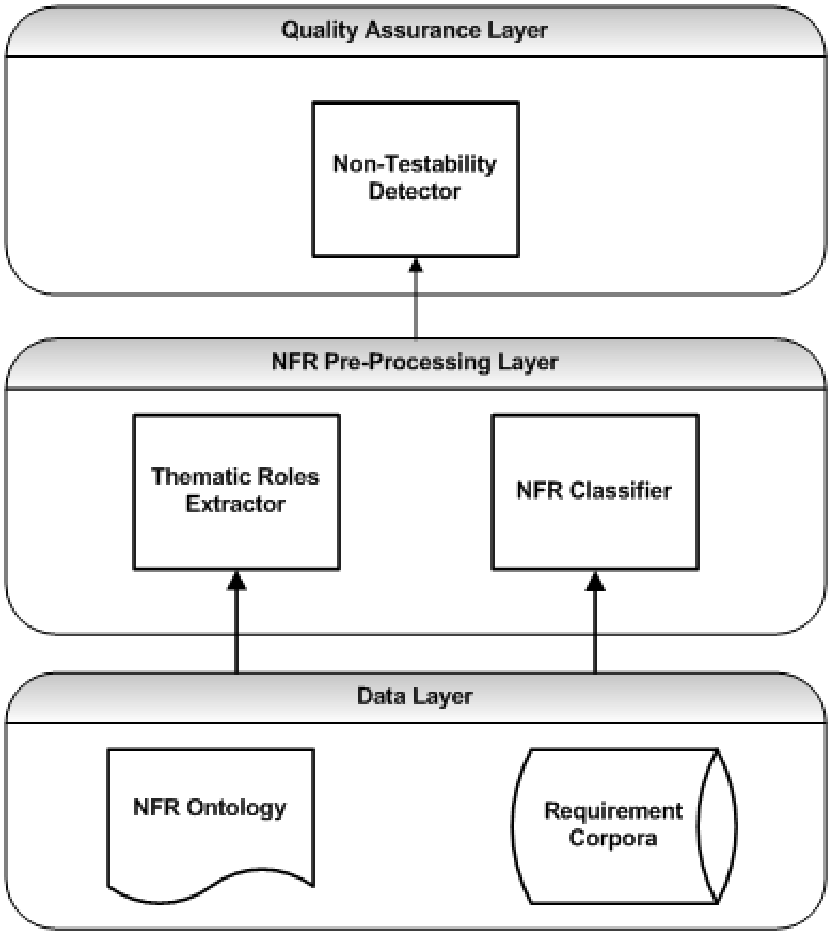


Figure 4: High-Level System Design

4.2 System Overview

Based on the above requirements analysis, we can now develop the system's design. Our approach contains three layers, as presented in Figure 4:

1. The **Data Layer**, which contains:

    - The conceptualization of the NFRs, modeled using the Web Ontology Language (OWL) [Mv04]. We designed an adapted version of the ontology presented in [Kas09], containing two extensions, requirements thematic roles and fit-criteria, as well as their relations to the NFR types ontology [Kas09].
    - Two manually annotated gold standard corpora:

    - A 'SRS Concordia' corpus, which is used for the NFR classifier. It contains six SRS documents, including 234 NFRs and 787 FRs.
    - New annotations added to the PROMISE corpus such as agent, action, theme, condition, and instrument. The corpus is used for the thematic roles extractor. It includes 15 SRS documents, 326 NFRs and 358 FRs. This new annotated corpus name forms our"Enhanced PROMISE corpus".

2. The **NFR Pre-processing Layer**, which contains:
    - An automatic requirements classification system, based on support vector machines (SVMs) [ROW13], which can automatically categorize requirements sentences into different NFR ontology classes.
    - A rule-based text mining system that automatically identifies requirements thematic roles in sentences.

3. The **Quality Assurance Layer**, which contains:
    - The automatic non-testability detection, which is based on a generic approach for building a quality assurance model. In this thesis, we used a rule-based approach for non-testability detection. Statistical methods can also be used to do the same task.

In the following sections, the system layers will be described in more detail.

4.3 Data Layer

In this section, the requirements ontology, which is the basic component of the data layer, is introduced. The corpora that are dependent on the designed requirements ontology are detailed in Chapter 6.

In our approach, NFRs are classified based on a requirements ontology, which is modeled using the Web Ontology Language (OWL) [Mv04]. This allow us to populate the sentences to the ontology, and query it using

SPARQL [SPA08] (SPARQL Protocol and RDF Query Language). Our conceptualization is an adapted version of the one developed in [Kas09], as shown in Figure 5. This is done by limiting the major classes and concepts to those that frequently appear in requirements documents. Most of the concept definitions are based on the ISO25010 [ISO10] standard; with some additional sources for further refinements, as indicated in the background Section 2.1.3. This ontology contains several views:

1. The NFR view, which is concerned with the different types of NFRs and divided into sub-categories;
2. The thematic role view, which represents the relations between the requirements thematic roles;
3. The fit-criteria view, which represents a measurement model for NFR fit-criteria.

4.3.1 The NFRs View

Figure 5 illustrates the semantic structure of the different types of NFRs. Table 2 shows the class definitions, together with examples from our SRS Concordia corpus. This corpus is used for automatic ontology classification.

4.3.2 Thematic Roles View

Figure 6 illustrates the semantic structure of requirement thematic roles. Definitions of the main concepts are detailed in Table 5.

4.3.3 Fit-Criteria View

This ontology is designed to connect the different NFRs, such as *Performance*, with concrete measurement units suitable within a fit-criterion. Figure 7 illustrates the main concepts of different NFRs and their relations with fit-criteria and units. The main concepts are:

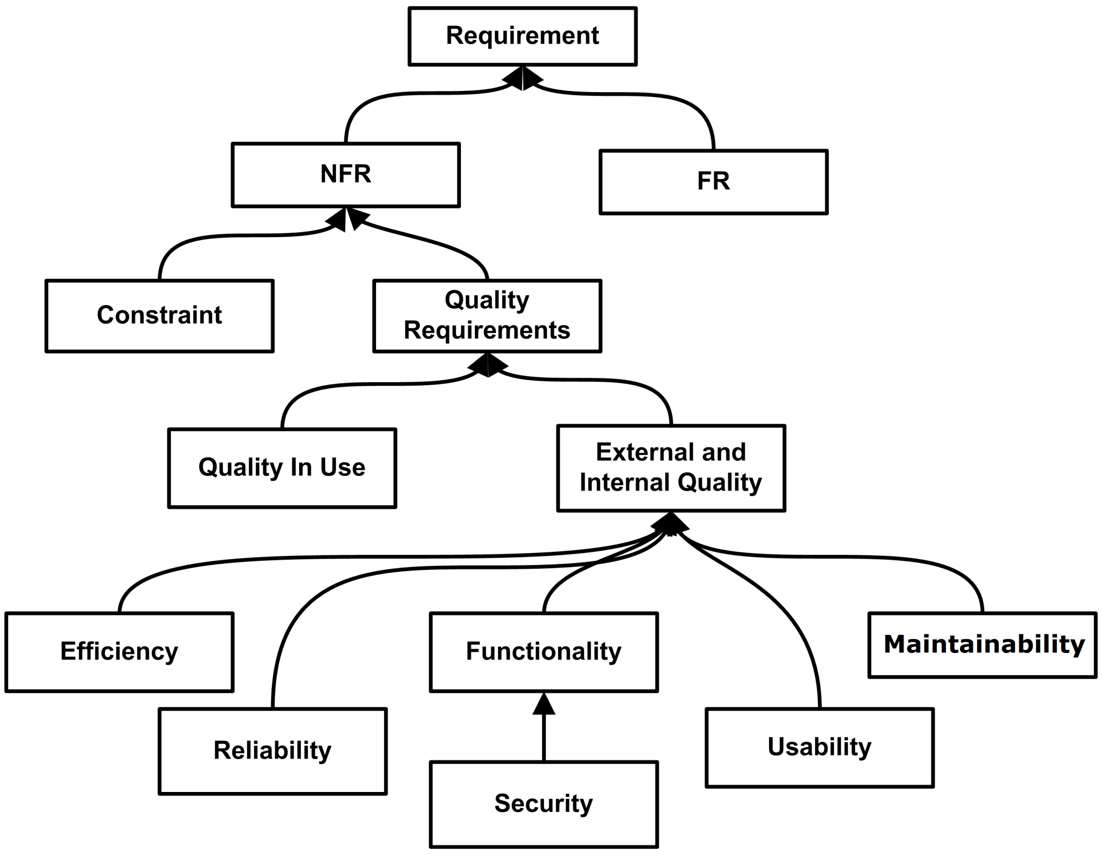


Figure 5: Requirements Ontology (excerpt)

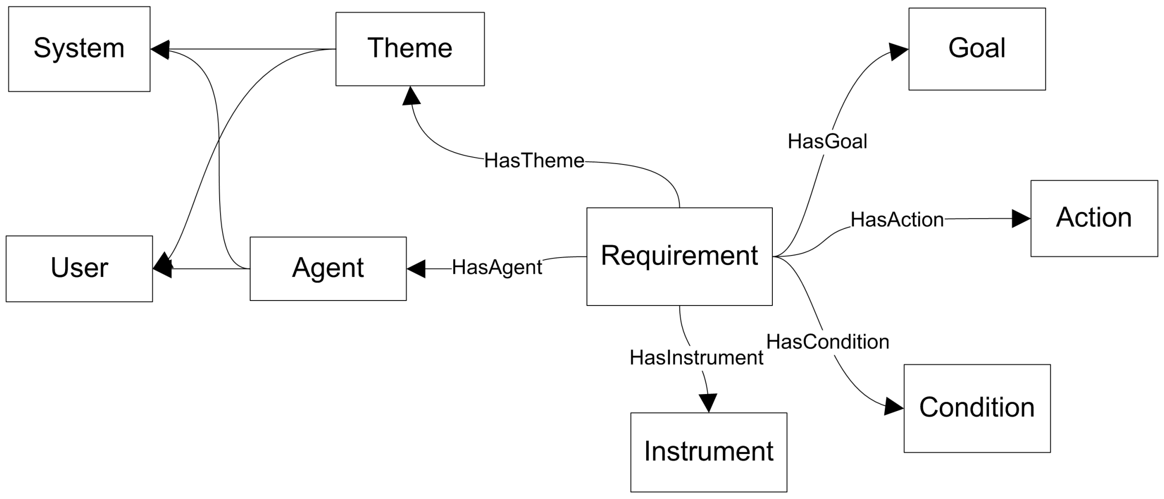


Figure 6: RE Ontology (Thematic Role View)

Table 5: Thematic Roles in SRS Documents

Concept	Description	Example
Agent	This concept represents the system, part of the system, or stakeholders	The system, The website
Modality	This concept is the auxiliary verbs; including "must" for mandatory, "shall" and "will" for required and "may" for optional	shall, must
Action	This concept is a verb or VG that represents the action from the agent	display, extract
Theme	This concept contains the description of the thing that the agent acts on; it can be part of the system or external, such as a stakeholders	the screen
Condition	This concept represents the condition on the action of the agent. Most of the fit-criteria are located in the condition phrase.	every 60 seconds
Goal	This concept represents the reason behind an action	in order to load the data
Instrument	This concept represents the instrument used to perform an action	using IE6

NFR: This class has nine subclasses of the different types of NFRs, as indicated in Section 4.3.1.

Fit-criteria: This concept has two subclasses, *Unit* and *Quantity*.

Quantity: This subclass contains the quantity appearing before a unit.

Unit: This subclass has nine subclasses with the different types of unit categories, including *Time, Percentage, Limit, Connection speed, Frequency, Distance, Currency,* and *Display*, as shown in Table 6. Each fit criterion can include units.

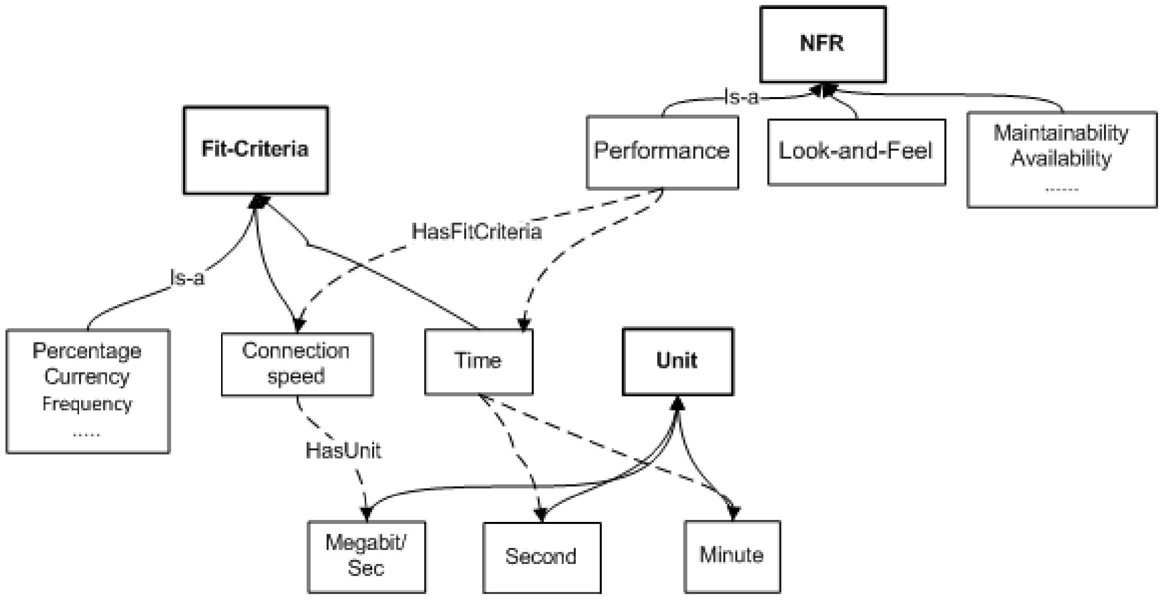


Figure 7: RE Ontology (NFR-Fit Criteria View)

Table 7 presents the statistics of the fit criteria occurrence for each NFR in the Enhanced PROMISE corpus. This was used to build the relationships between the measurements and the NFRs in the ontology, as illustrated in same table. It can be seen that specific NFR subclasses are correlated with concrete fit-criteria classes.

4.4 NFR Preprocessing Layer

In this section, the NFR classifier and SRS thematic roles extractor components are presented.

4.4.1 Automatic Classification of Requirements

We describe our automatic sentence-based classifier for requirements documents. A custom text mining application detects candidate sentences and classifies them using a machine learning algorithm. We trained our system both on the PROMISE and Concordia corpora. The application is divided into four steps, as shown in Figure 8. Documents are pre-processed for the classification, using existing tokenization, sentence splitting, and

Table 6: Fit-Criteria Concepts Description

Concept	**Description**	**Example**
Time	The time units	5 second, one day, 8 AM
Percentage	The percentage of the measured object	90% of the users
Limit	The measured entities that do not belong to the other fit-criteria	5 movies
Connection speed	The data transfer connection speed	15 mbps
Frequency	The occurrence number of events per time interval	2 times per day
Distance	The distance between two objects	1.4 miles
Display	The display measures	32 inch screen

Table 7: Analysis of the Fit-Criteria on the Enhanced PROMISE Corpus

Class	F	PE	US	A	SE	LF	SC	L	O	FT	MN	PO	Total
Time	3	49	20	13	3	1	6		5		8		108
Percentage	1	7	17	11	7	3	1		4	2	3		56
Limit	4	2	3	1		1	12		2				25
Cn. speed		2			10				1				13
Frequency							2		1				3
Distance	2		2										4
Currency	1												1
Display	3					2							5
Total	12	60	42	25	20	7	21	0	13	2	11	0	215
Total Req.	271	76	99	29	97	51	29	15	86	13	29	1	796
Percentage	5	79	42	86	21	14	72	0	15	15	38	0	27

token stemming components (cf. Section 2.3.2). For the last step, we designed a machine learning-based FR/NFR classifier for SRS documents.

The goal of this machine learning module is to classify input sentences

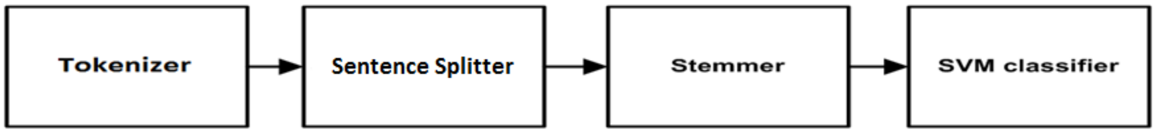


Figure 8: NFR Classifier Design

into four major categories, with eight classes: FR (Functional Requirements), Design Constraints, NR (Not a Requirement) and several types of NFRs (security, efficiency, reliability, functionality, usability and maintainability).

Example input sentences for each type are shown in Table 8.

Table 8: NFR Classifier Example Input Sentences

Class	Example
FR	The ASPERA-3 data set shall be stored on a local SwRI archive
NFR	The APAF ground data system shall have built-in error handling
NR	Section 4 contains general information to aid in the understanding of this specification

In our experiments, Support Vector Machines (SVM) with a third order polynomial kernel provided the best performance. We tried other machine learning algorithms including K-Nearest Neighbour (KNN), and Perceptron Algorithm with Uneven Margins (PAUM) [LZH+02] beside the SVM. The features used for training are the unigram of the sentences' tokens, using their stem. Instead of a multi-classification, we perform a binary classification for each type of FR/NFR. This is because some sentences contain two or more types of requirements. E.g., *"Web-based displays of the most current ASPERA-3 data shall be provided for public view"* is annotated as both a FR and design constraint.

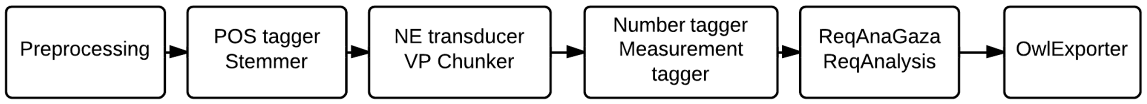


Figure 9: Text Mining System Design for Analyzing NL Requirements Statements

4.4.2 Thematic Roles Extractor

In this section, our goal is to define a method for labeling requirements statements into thematic roles. The main thematic roles are *Agent, Action, Condition, Instrument* and *Theme*. We defined the requirement thematic roles according to the semantic roles labeling that described before in 2.3.2. We designed a generic component that can then be used in an automatic quality assurance (QA) process. One of the important aspects for requirements QA is to ensure whether NFRs are testable or not.

Figure 9 shows our high-level system design. Sentences are preprocessed, using standard tokenization, sentence splitting, POS tagging, and stemming. Then, sentences are chunked for noun and verb phrases in order to identify the agents, instruments and actions. Measurements and their quantities are extracted through dictionary-based lookups, in order to extract the fit-criteria and their measures. Finally, transducer-based rules analyze the thematic roles based on the ontology classes and report non-testable NFRs, i.e., statements lacking a (compatible) fit-criterion.

The number of dictionary lists, which are used in the thematic roles extractor, are listed below:

Shall: lists 51 terms with all possible combinations, as shown in Figure 10, such as, *shall, must be able to,* or *should be able to easily*;

Shall allow: lists 15 terms of the *shall allow* combinations presented in Figure 10, such as, *shall allow, must prevent,* or *must provide the ability to*;

Condition: lists conditional phrases at the beginning of a requirement, such as, *if, when, once,* or *as long as*;

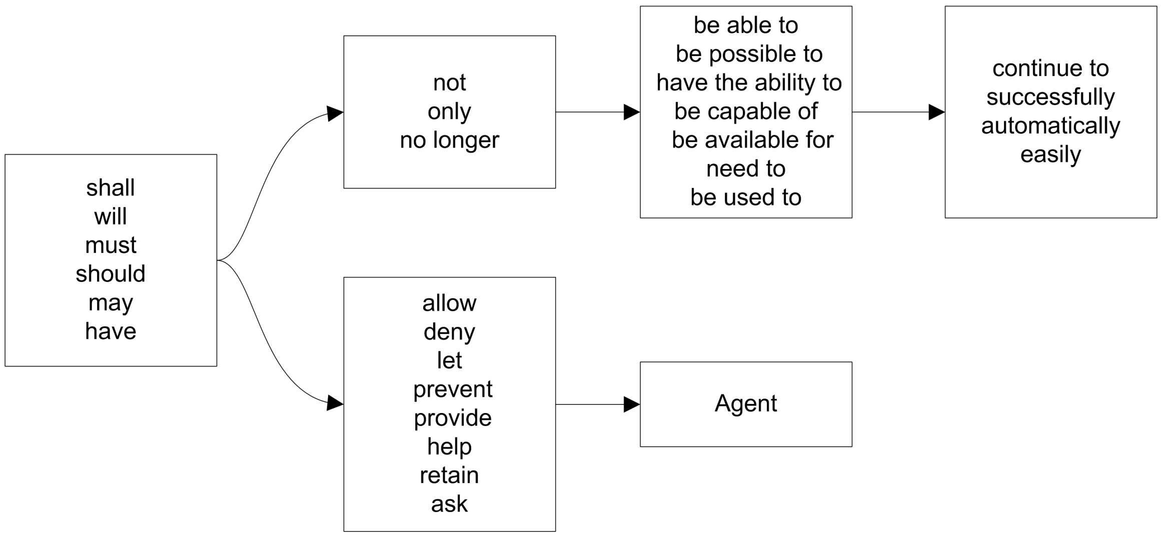


Figure 10: Auxiliary Verbs Structure

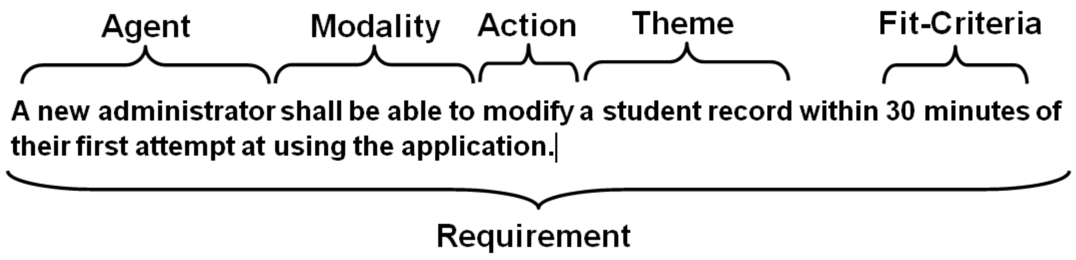

Agent
Modality
Action
Theme
Fit-Criteria
A new administrator shall be able to modify a student record within 30 minutes of their first attempt at using the application.
Requirement

Figure 11: Thematic Role Output Example

Limit: lists 28 terms conditions can start with, such as, *a maximum of, more than, either,* or *without.*

Detailed descriptions of the dictionary lists are provided in Appendix B. The transducer-based rules that are used to extract the thematic roles can be described as follows:

Thematic Role Extraction Rules

Based on the syntactic information and the semantic dictionary labels, we designed a number of rules to extract the different types of thematic roles, as listed in Table 9.

Figure 11 shows example of thematic roles extractor output.

Table 9: Patterns for Detecting the Different Thematic Roles in the Requirements

Class	**Pattern**
Agent	Match noun phrases (NPs), using the MuNPEx chunker, that come before a "Modality" phrase, as well as NPs that come after a "shall allow" phrase
Action	Match a verb group, using the ANNIE [ea11] VP Chunker, that comes after a "Modality" phrase
Theme	Match NPs that come after the action phrase
Condition	Match a condition statement before the agent in the beginning of the sentence or within the sentence
Goal	Match a verb group coming after phrases *in order to, to,* or *for*
Instrument	Match NPs coming after phrases *using* or *via*

Fit-Criteria Extraction Rules

The rules for detecting the different types of fit-criteria, based on patterns developed after a corpus analysis, are shown in Table 10.

4.5 Quality Assurance Layer

Our non-testability detector is introduced as an example of a requirements quality assurance application. Other quality assurance components could be build based on the data and preprocessing layers described in Sections 4.3 and 4.4.

Non-Testability Detector In this Section, the design of the non-testability detector is introduced. In Figure 12, an example of the system's output annotation is shown.

The sentence "The application shall be user-friendly." is not testable as it does not contain fit-criteria. On the other hand, the sentence "A new administrator shall be able to modify a student record within 30 minutes

Table 10: Patterns of Different Types of Fit-criteria

Class	**Pattern**
Time	(Number + Adjective + Time Unit) or (Number + Time Unit)
Percentage	number + %
Limit	(Number + Adjective + Noun) or (Number + Noun)
Connection speed	(Number + Adjective + Connection Unit) or (Number + Connection Unit)
Frequency	Number + times per time
Distance	(Number + Adjective + Distance Unit) or (Number + Distance Unit)
Display	(Number + X + Number + X + Number) or (Number + X + Number)

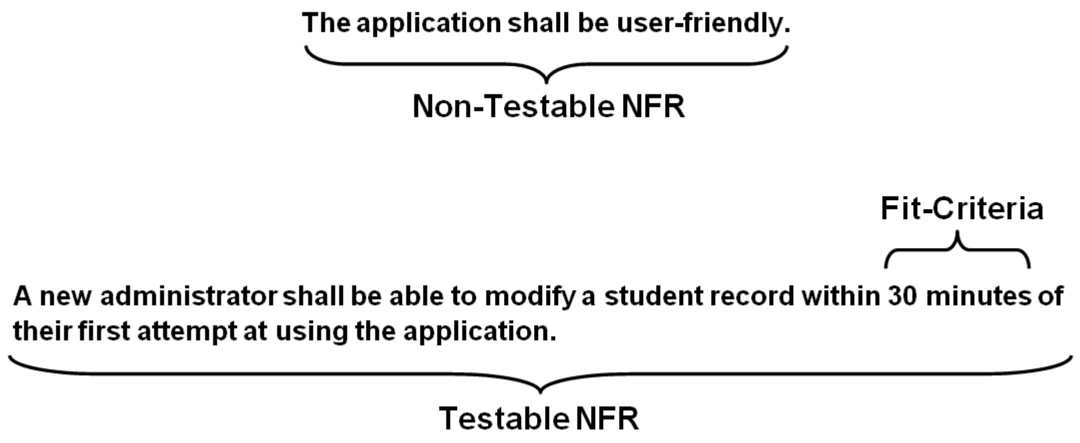

The application shall be user-friendly.
Non-Testable NFR
Fit-Criteria
A new administrator shall be able to modify a student record within 30 minutes of their first attempt at using the application.
Testable NFR

Figure 12: Non-Testability Detector Example

of their first attempt at using the application." is testable, as it contains a fit-criteria.

Non-testability rule-based system

The Non-testability rule-based system decides the non-testability of a requirement, based on the fit-criteria's existence in the NFRs. For the non-testability rule-based system, we are using the fit-criteria thematic role only, but we could use the thematic roles in more quality assurance applications. For example, we can analyze the action role in the sentences for passive voice defects, such as "be validated". We can also analyze the agent roles in the system and report inconsistencies, such as "student management system", and "user application". We can populate the ontology with the thematic roles individuals, and apply a set of rules using SPARQL query language to highlight potential contradictions between two sentences.

Non-testability Statistical System

In addition, we developed a statistical model using a SVM with a polynomial kernel. We trained our system on the Enhanced PROMISE corpus that contains 797 sentences. The Enhanced PROMISE corpus will be described in Section 6.1.1. Documents are pre-processed to be prepared for the classification, using tokenization, sentence splitting and token stemmer. We built a machine learning-based non-testability classifier for SRS documents. The goal of this module is to classify input sentences into two categories: testable sentences, such as *"The system shall refresh the display every 60 seconds"*, and non-testable sentences, such as *"The system shall allow a user to define the time segments"*. In our experiments, a SVM with a first order polynomial kernel is used. The features that are used for training are the unigram of the sentences' tokens, using its stem.

4.6 Summary

In this chapter, we have presented the design decisions for the main components to develop a non-testability quality assurance system. Each system

Table 11: System Requirements vs. Design

System Requirements	**Design**
Requirement #1.0: Building an NFR and Requirements thematic roles Ontology	Section 4.3.1 (The NFRs View), Section 4.3.2 (Thematic Roles View), and Section 4.3.3 (Fit-Criteria View)
Requirement #2.1: NFR Corpus	Section 6.1.2 SRS Concordia Corpus
Requirement #2.2: Requirements thematic roles Corpus	Section 6.1.1 Enhanced PROMISE Corpus
Requirement #3.1: NFR Classification	Section 4.4.1 (Automatic Classification of Requirements)
Requirement #4.1: Requirements Thematic Roles Extraction	Section 4.4.2 (Thematic Roles Extractor)
Requirement #5.1: Non-Testability Detection in NFR	Section 4.5 (Quality Assurance Layer)

requirement presented in Section 4.1 is mapped to a component in our design. We have presented the data layer that includes the ontology design in Section 4.3, which meets the Requirement #1.1. The second layer consists of a machine learning NFR classifier, and the rule based thematic roles extractor, which meet the Requirement #3.1 and Requirement #4.1, respectively. Finally, we presented the non-testability detector, which meets the Requirement #5.1. Table 11 list all the system requirements presented in Section 4.1, and link it to the design sections.

In Chapter 5, we will discuss the implementation tools and the implementation details for the proposed design.

Chapter 5

Implementation

Be the measure great or small. . .
let it be honest in every part.

John Bright

This chapter details the steps taken during the implementation process of the solution described in Chapter 4. The main implementation tools, GATE and Protégé, will be described in the first section. In the second section, our implementation for the system design is presented.

5.1 Implementation Tools

In this section, we will introduce the implementation tools GATE for text engineering, and Protégé for ontology building:

5.1.1 GATE

GATE [ea11] is a *"General Architecture for Text Engineering"*. It is an open source framework, developed since 1995 at Sheffield university with the goal to help developers, students, users, educators, and scientists to solve text processing problems. GATE is a Java based software, providing a user interface tool, called GATE Developer, and a set of libraries exposed by an Application Programming Interface (API), called GATE Embedded, as shown in Figure 13.

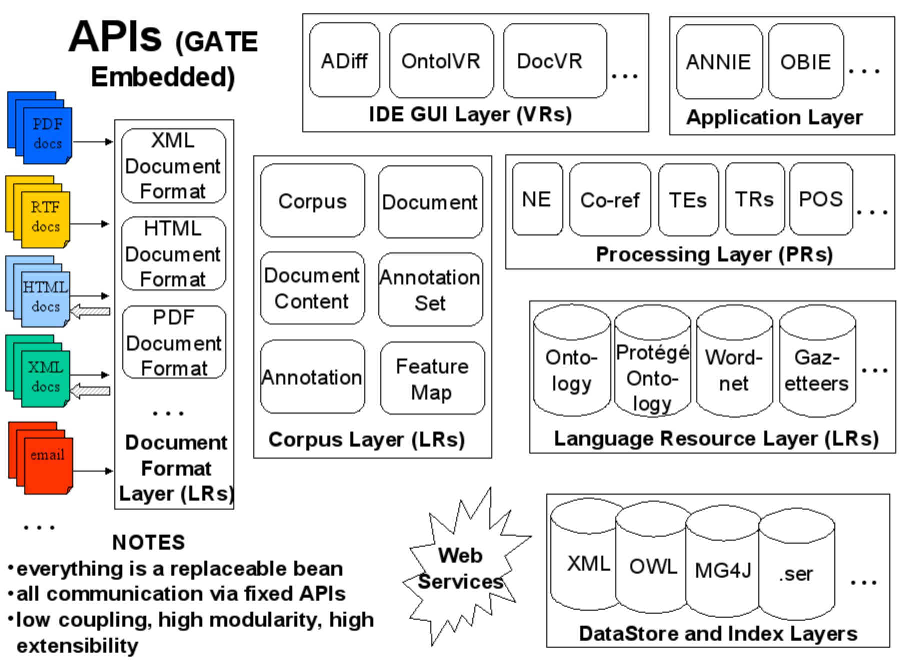


Figure 13: GATE Architecture Overview [ea11]

GATE Main Characteristics

1. GATE is a component-based architecture, as shown in Figure 13, where data and application are separated. It has a large set of plugins and a capability to develop customized plugins in a standard interface. GATE has four main types of components:

    - Language Resources (LR): a set of entities to be processed, such as documents, corpora, annotation schemas, and ontologies.
    - Processing Resources (PR): a set of tools and plugins that run a certain text analysis function, such as parsers and tokenizers.
    - Applications: consist of a pipeline of PRs, to be executed on the LRs.
    - Data Store: a place to store the processed LRs.

2. The components, or plugins, are called CREOLE, which stands for *"A Collection of REusable Objects for Language Engineering"*. CREOLEs consist of Java JAR files, plus configuration files, and are managed by

the CREOLE Plugin Manager. It is the base for developing customized PRs.

3. GATE LRs can contain documents of different format types, such as, xml, pdf, rtf, or html.

4. ANNIE: One of the main and essential GATE plugins, stands for *"A Nearly-New Information Extraction system"*. It implements many tasks such as tokenization, POS tagging, verb phrase chunking, and so on.

5. GATE has a rule engine called JAPE. JAPE stands for *"Java Annotation Patterns Engine"*. It is a finite-state transducer, but executed over GATE annotated documents. It provides the user with ability to create grammar rules, where each rule contains a set of patterns.

6. A machine learning plugin provides the capability to perform three tasks, namely text classification, chunk recognition, and relation extraction. It supports many algorithms, such as SVM, which we use for our NFR classifier. In addition, the Perceptron Algorithm with Uneven Margins (PAUM), Naive Bayes, KNN and the C4.5 decision tree are supported. XML configuration files are used to define an algorithm and its parameters.

7. GATE facilitates the manual annotation process using a web-based platform, called GATE Teamware. GATE can also generate corpus statistics, and support evaluation using standard metrics, such as cross-fold validation.

5.1.2 Protégé

Protégé is a free, open source ontology editor for intelligent applications. It supports different formats, such as RDF/XML, Turtle, OWL/XML, OBO, and others. Protégé has been developed by Stanford University in collaboration with the University of Manchester.

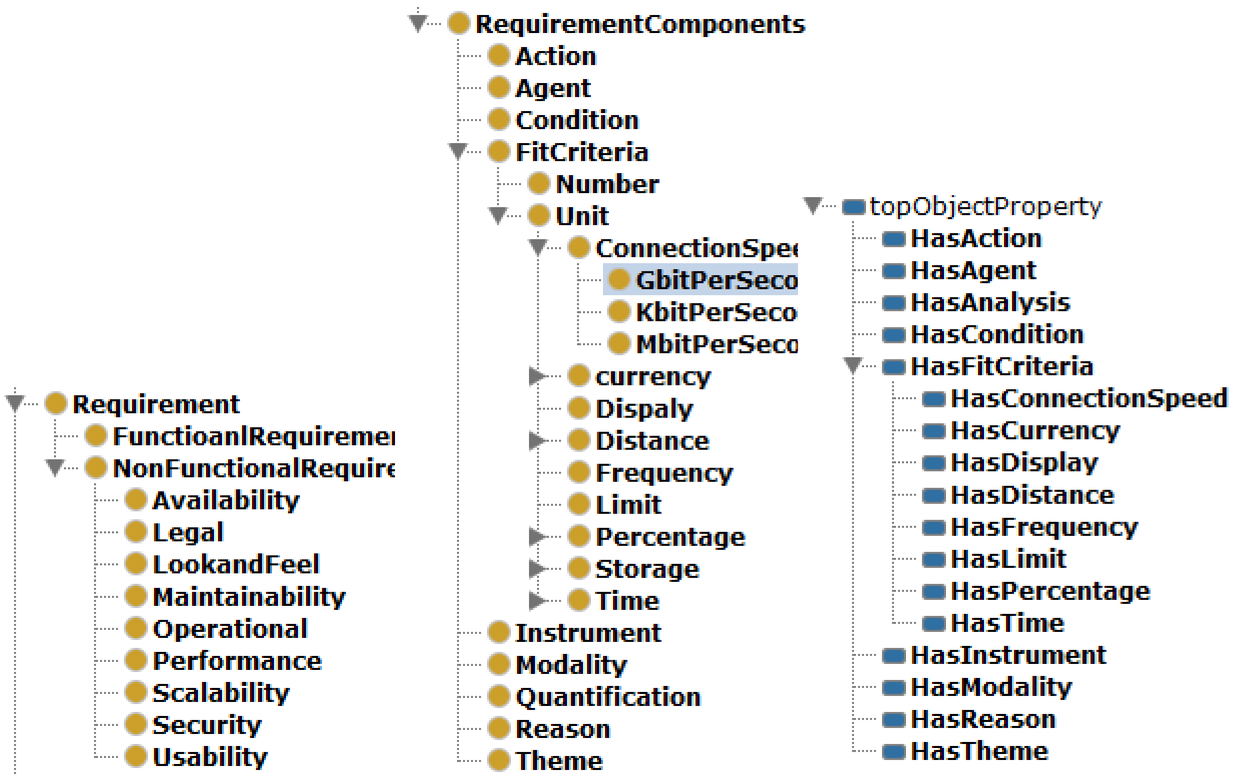


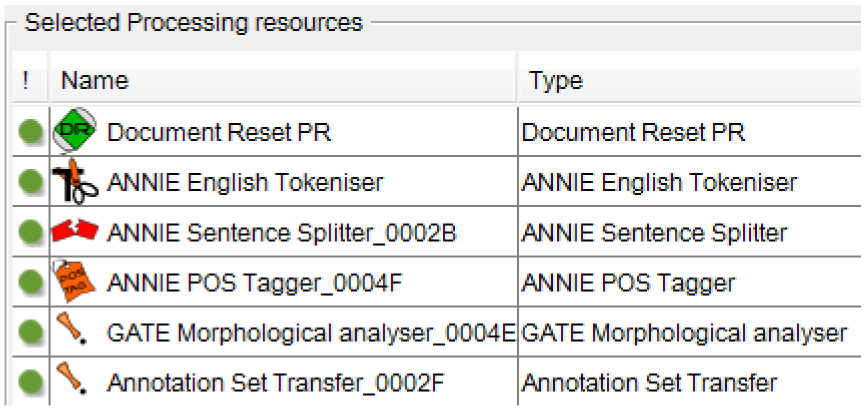
Figure 14: NFR Ontology

5.2 System Implementation

The NFR ontology, NFR classifier, ontology population, thematic roles extractor, and non-testability labeler implementation details are described in this section.

5.2.1 NFR Ontology

Our NFR ontology is designed with Protégé. It contains two main classes: requirements and requirement thematic roles. The requirement class contains the different types of NFRs, as described in Section 4.3.1, Figure 5. The requirement thematic roles ontology contains the main roles of a requirements sentence, including the fit-criteria as described in Figure 6. The ontology also contains the relation between NFR and their fit-criteria, as shown in Figure 14.


Selected Processing resources

!	Name	Type
●	Document Reset PR	Document Reset PR
●	ANNIE English Tokeniser	ANNIE English Tokeniser
●	ANNIE Sentence Splitter_0002B	ANNIE Sentence Splitter
●	ANNIE POS Tagger_0004F	ANNIE POS Tagger
●	GATE Morphological analyser_0004E	GATE Morphological analyser
●	Annotation Set Transfer_0002F	Annotation Set Transfer

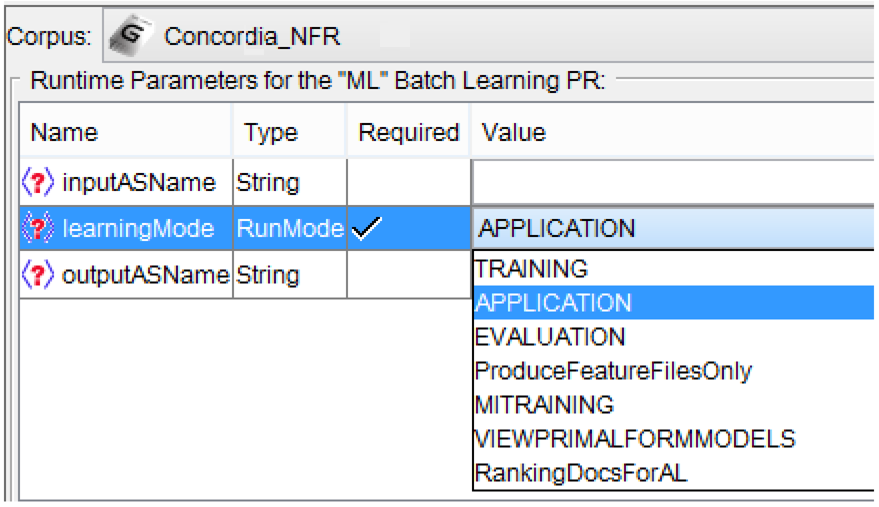
Figure 15: NFR classifier pipeline

5.2.2 NFR Classifier

Our NFR Classifier design is described in Section 4.4.1. In this section, we will introduce the implementation details. Preprocessing is performed by a pipeline implemented using GATE [ea11]. It extracts features from the documents, which are then fed into a machine learning component. This pipeline contains PRs, in particular the ANNIE components [ea11] and a machine learning PR. To obtain the features for machine learning, documents are pre-processed by using the ANNIE English Tokenizer PR, the ANNIE Sentence Splitter, and the Snowball stemmer, as shown in Figure 15.

Support Vector Machine Classifier

To perform a machine learning task in GATE, we use the batch learning Processing Resource (PR). The configuration parameters for the batch learning PR are specified in an external XML file, which contains the configuration parameters of the PR and the linguistic data parameters. The directory that contains the XML configuration file has a subdirectory called 'savedFiles'. This subdirectory contains the resultant NLP model files and a log file with the evaluation results. Only a few parameters are set as initialization outside the XML configuration file, as shown in Figure 16. The batch learning PR supports different modes, such as:


Corpus: Concordia_NFR

Runtime Parameters for the "ML" Batch Learning PR:

Name	Type	Required	Value
inputASName	String		
learningMode	RunMode	✓	APPLICATION
outputASName	String		

TRAINING
APPLICATION
EVALUATION
ProduceFeatureFilesOnly
MITRAINING
VIEWPRIMALFORMMODELS
RankingDocsForAL

Figure 16: Batch Learning PR

1. Training mode, which aims to create training data from a provided corpus;

2. Application mode, which aims to apply the trained models on unseen data; and

3. Evaluation mode, which aims to evaluate the algorithm on the provided corpus, using the configuration file to specify the evaluation type, such as k-fold or hold-out test.

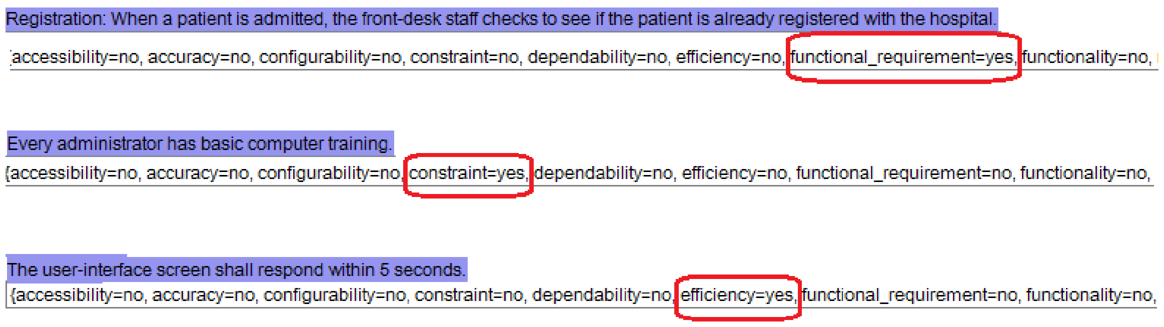
Here, a support vector machine (SVM) is used with a third-order polynomial kernel, as defined in the parameter d shown in the configuration file:

```
<ENGINE nickname="SVM" implementationName="SVMLibSvmJava"
        options=" -c 1 -t 0 -d 3 -m 40 -tau 0.3  "/>
```

Moreover, the configuration file contains the evaluation method: Six-fold cross validation is used on the NFR classifier, as we have six documents in our corpus:

```
<!-- Evaluation : how to split the corpus into test and learn? -->
 <EVALUATION method="kfold" runs="6"/>
```

In addition, the configuration file contains the features used to train the model, which in our case is a unigram of the sentences' tokens, using their stem:


Registration: When a patient is admitted, the front-desk staff checks to see if the patient is already registered with the hospital.

{accessibility=no, accuracy=no, configurability=no, constraint=no, dependability=no, efficiency=no, functional_requirement=yes, functionality=no,

Every administrator has basic computer training.

{accessibility=no, accuracy=no, configurability=no, constraint=yes, dependability=no, efficiency=no, functional_requirement=no, functionality=no,

The user-interface screen shall respond within 5 seconds.

{accessibility=no, accuracy=no, configurability=no, constraint=no, dependability=no, efficiency=yes, functional_requirement=no, functionality=no,

Figure 17: NFR Classifier Output Annotations

```
    <NGRAM>
        <NAME>Sent1gram</NAME>
        <NUMBER>1</NUMBER>
        <CONSNUM>1</CONSNUM>
        <CONS-1>
            <TYPE>Token</TYPE>
                        <FEATURE>stem</FEATURE>
        </CONS-1>
     </NGRAM>
```

Finally, the configuration file contains the target class, such as a functional requirement in this example:

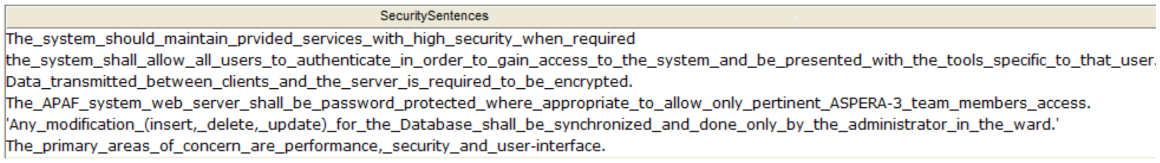
```
<ATTRIBUTE>
      <NAME>Class</NAME>
      <SEMTYPE>NOMINAL</SEMTYPE>
      <TYPE>Sentence</TYPE>
               <FEATURE>functional_requirement</FEATURE>
      <POSITION>0</POSITION>
      <CLASS/>
    </ATTRIBUTE>
```

The complete configuration file is contained in Appendix A.

Examples for the NFR classifier output are shown in Figure 17.

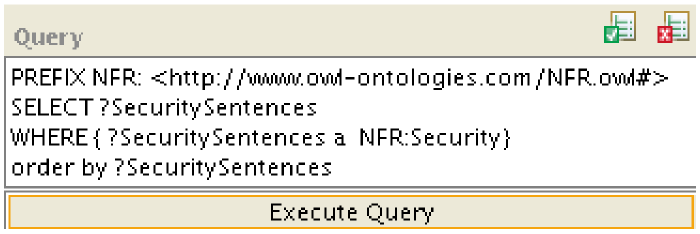
5.2.3 Ontology Population

We use the OwlExporter [WKR10] component to populate the extracted functional and non-functional requirements into our requirements ontology. OwlExporter is a GATE plug-in used to export the document annotations to individuals in a Web Ontology Language (OWL) model as shown



SecuritySentences
The_system_should_maintain_prvided_services_with_high_security_when_required
the_system_shall_allow_all_users_to_authenticate_in_order_to_gain_access_to_the_system_and_be_presented_with_the_tools_specific_to_that_user.
Data_transmitted_between_clients_and_the_server_is_required_to_be_encrypted.
The_APAF_system_web_server_shall_be_password_protected_where_appropriate_to_allow_only_pertinent_ASPERA-3_team_members_access.
'Any_modification_(insert,_delete,_update)_for_the_Database_shall_be_synchronized_and_done_only_by_the_administrator_in_the_ward.'
The_primary_areas_of_concern_are_performance,_security_and_user-interface.

Figure 18: Individuals Populated into the Ontology for Security NFR using OwlExporter



```
PREFIX NFR: <http://www.owl-ontologies.com/NFR.owl#>
SELECT ?SecuritySentences
WHERE { ?SecuritySentences a NFR:Security}
order by ?SecuritySentences
```

Figure 19: SPARQL Query for all Security NFR Sentences in the Ontology using Protégé

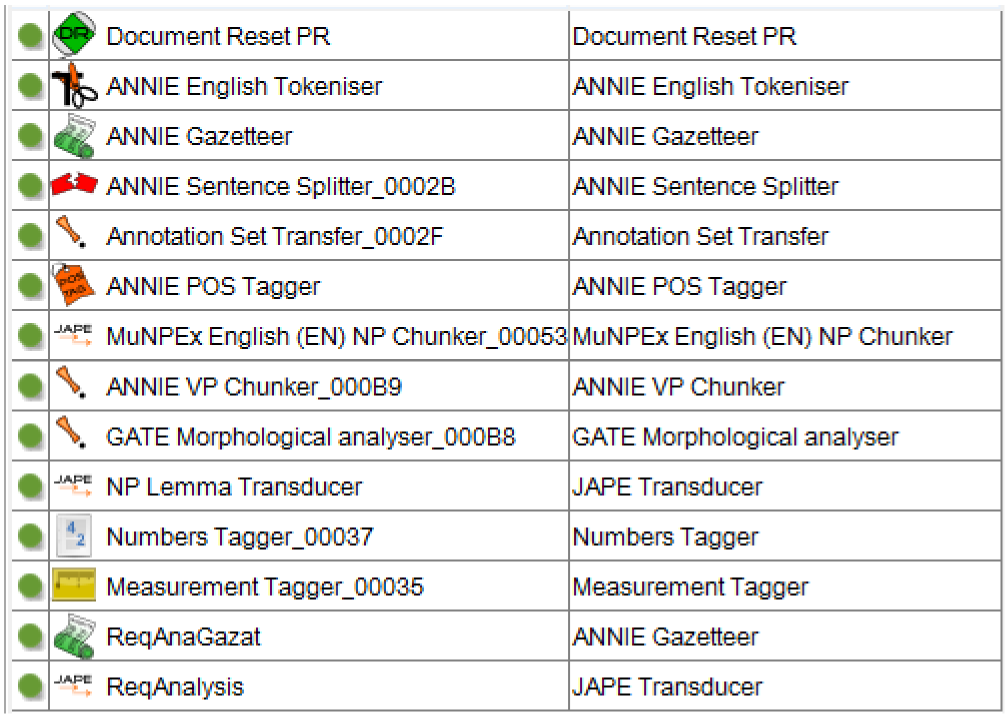
in Figure 18 We convert the NFR classifier output annotations format to the OwlExporter input annotations format using JAPE rules. The input annotations for OwlExporter are then mapped to the ontology classes. The implementation for these JAPE rules is provided in Appendix D

SPARQL [SPA08] (SPARQL Protocol and RDF Query Language) is a powerful standard query language for ontologies and RDF databases. We use the SPARQL query module in Protégé 3.4.8 to query the populated ontology. An example of a simple retrieval of all security sentences from the ontology is shown in Figure 19.

5.2.4 Requirement Analysis ReqAnalysis

Sections 4.4.2, and 4.5 shown the design for the thematic roles extraction, fit criteria detection, and non-testability detection phases. In this section, we will describe the implementation for the three of them.

In Figure 20, the ReqAnalysis pipeline that is used for requirements thematic roles extraction, fit criteria detection, and non-testability detection is shown. The thematic roles classes and their definitions are detailed in Table 5. We use a mix of ANNIE, Concordia Semantic Software Lab, and



Document Reset PR	Document Reset PR
ANNIE English Tokeniser	ANNIE English Tokeniser
ANNIE Gazetteer	ANNIE Gazetteer
ANNIE Sentence Splitter_0002B	ANNIE Sentence Splitter
Annotation Set Transfer_0002F	Annotation Set Transfer
ANNIE POS Tagger	ANNIE POS Tagger
MuNPEx English (EN) NP Chunker_00053	MuNPEx English (EN) NP Chunker
ANNIE VP Chunker_000B9	ANNIE VP Chunker
GATE Morphological analyser_000B8	GATE Morphological analyser
NP Lemma Transducer	JAPE Transducer
Numbers Tagger_00037	Numbers Tagger
Measurement Tagger_00035	Measurement Tagger
ReqAnaGazat	ANNIE Gazetteer
ReqAnalysis	JAPE Transducer

Figure 20: Requirement Thematic Roles Extractor Pipeline

our own custom components to build this application. The following steps are performed in the pipeline:

1. ANNIE Document Reset is used to reset a corpus to its original state.

2. ANNIE Tokeniser is used to split a text into very basic units, such as numbers and words.

3. ANNIE Gazetteer is a name entity recognizer, which is used to annotate words or sequences of words, based on predefined lists, such as abbreviations, countries, and days.

4. ANNIE Sentence Splitter is used to split the text into sentences, which in our case are the requirements statements. The Sentence Splitter is using a Gazetteer to distinguish between the full stop at the end of the sentence and an abbreviation.

5. ANNIE POS tagger is a modified version of the Brill tagger [Hep00].

6. Multilingual Noun Phrase Extractor (MuNPEx) is a Processing Resource (PR) developed by Concordia's Semantic Software Lab. It is a noun phrase chunker. This component and ANNIE NE transducer are used to extract the theme class in our application.

7. ANNIE VP Chunker is used to extract verb groups, such as *is investigating, to investigate, investigated, is going to investigate.* It is used to extract the action class in our application.

8. GATE Morphological analyser is used to extract the root and affix for each token.

9. NP Lemma Transducer is part of the Multilingual Noun Phrase Extractor (MuNPEx) PR. It depends on the GATE morphological analyser and it is used to extract lemmas.

10. Number Tagger is used to annotate numbers. It has two features: the first feature "Type", which identifies whether a number is a word or a numeral. The second feature, "Value", stores the exact value as a double variable of the annotated number in a text.

11. Measurements Tagger is a parser used to recognize and annotate spans of text as being a measurement. It is also used to normalize the measurement value units. Number and measurement taggers are used to extract the fit-criteria.

12. ReqAnaGazat is a set of Gazetteers used to extract the modality and to help the main JAPE rules to extract requirement thematic roles. The complete Gazetteers are in Appendix B.

13. ReqAnalysis is a set of JAPE rules used to extract the main requirement thematic roles listed in Table 5. The rules are listed in Table 9. Examples for the thematic roles rules is shown in Figure 21. The full implementation code is listed in Appendix C. It also contains the non-testability detection application.

For example, the non-testability detection JAPE file contains two rules. The first rule, as shown below, means that if the NFR sentence contains

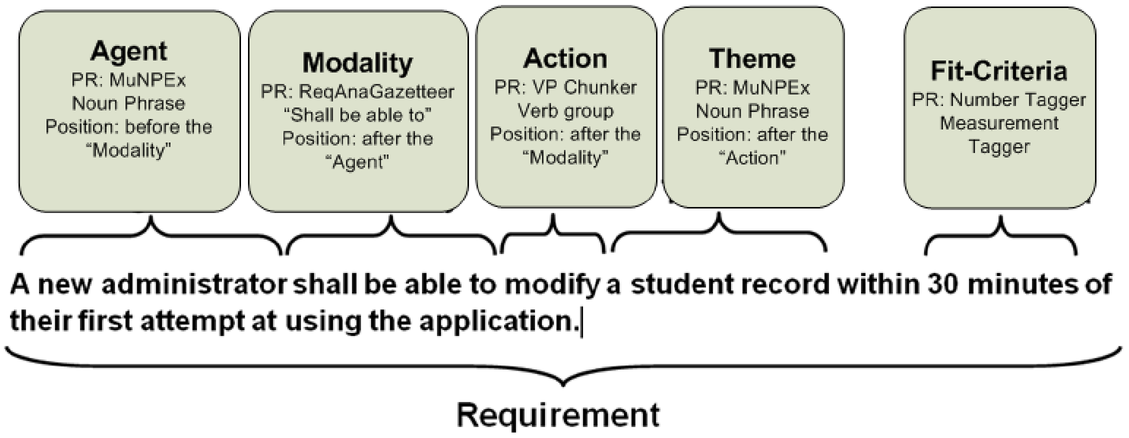

Agent
PR: MuNPEx
Noun Phrase
Position: before the "Modality"
Modality
PR: ReqAnaGazetteer
"Shall be able to"
Position: after the "Agent"
Action
PR: VP Chunker
Verb group
Position: after the "Modality"
Theme
PR: MuNPEx
Noun Phrase
Position: after the "Action"
Fit-Criteria
PR: Number Tagger
Measurement
Tagger
A new administrator shall be able to modify a student record within 30 minutes of their first attempt at using the application.
Requirement

Figure 21: Thematic Roles Rules Example

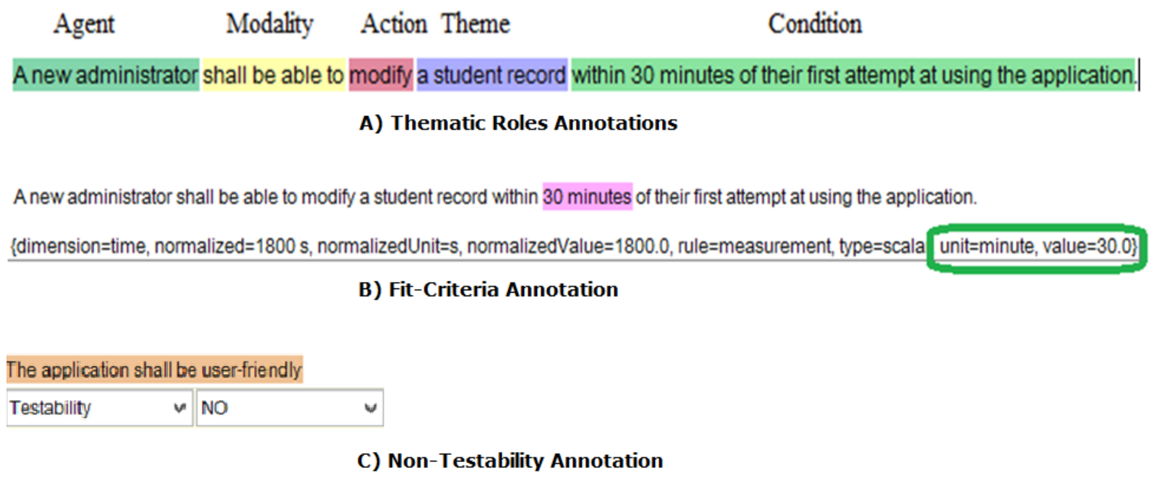

Agent
Modality
Action
Theme
Condition
A new administrator shall be able to modify a student record within 30 minutes of their first attempt at using the application.
A) Thematic Roles Annotations
A new administrator shall be able to modify a student record within 30 minutes of their first attempt at using the application.
{dimension=time, normalized=1800 s, normalizedUnit=s, normalizedValue=1800.0, rule=measurement, type=scala unit=minute, value=30.0}
B) Fit-Criteria Annotation
The application shall be user-friendly
Testability
NO
C) Non-Testability Annotation

Figure 22: Thematic Roles, Fit-Criteria, and Non-Testability Annotations

a fit-criteria, then mark this sentence as a testable NFR. Another rule to mark the sentence as a non-testable NFR if there is no fit-criteria, as shown in Appendix C.

```
Rule : rule1
(
({NFRSentence contains FitCriteria }) : testYES
)
:ann
testYES.Requirement = {Testable = "YES"}
```

Examples for the thematic roles, fit-criteria, and non-testability annotations are shown in Figure 22.

5.3 Summary

In this chapter, we introduced briefly the implementation tools. We discussed the implementation of our system. Our pipeline is assembled using existing ANNIE PRs, as well as PRs developed by us, such as the ReqAnaGazat and ReqAnalysis. In Figures 15 and 20, we show the NFR classifier and the non-testability quality assurance system pipelines loaded within GATE Developer, respectively.

In Chapter 6, we discuss the Enhanced PROMISE and Concordia corpora. In addition, we present the system evaluation.

Chapter 6

Corpora and Evaluation

Never promise more than you can perform.

Publilius Syrus

In this chapter, we describe two SRS corpora. These corpora are used to classify the different types of NFR, to extract the requirements thematic roles, and to automatically detect non-testable NFR sentences. Afterwards, we detail the evaluation of our proposed system based on our corpora.

6.1 NFR Corpora

In this section, we will introduce the corpora we use in our evaluation. The Enhanced PROMISE, and SRS Concordia corpora will be discussed in detail. Details on the annotation process, statistics, and a discussion are provided for both corpora.

6.1.1 Enhanced PROMISE Corpus

The PROMISE corpus [PRO] consists of 15 SRS documents. It was developed based on term projects by Master students at DePaul University. The corpus' specifications contain 326 NFRs and 358 FRs. The NFRs types

Table 12: NFR Classes within the Enhanced PROMISE Corpus

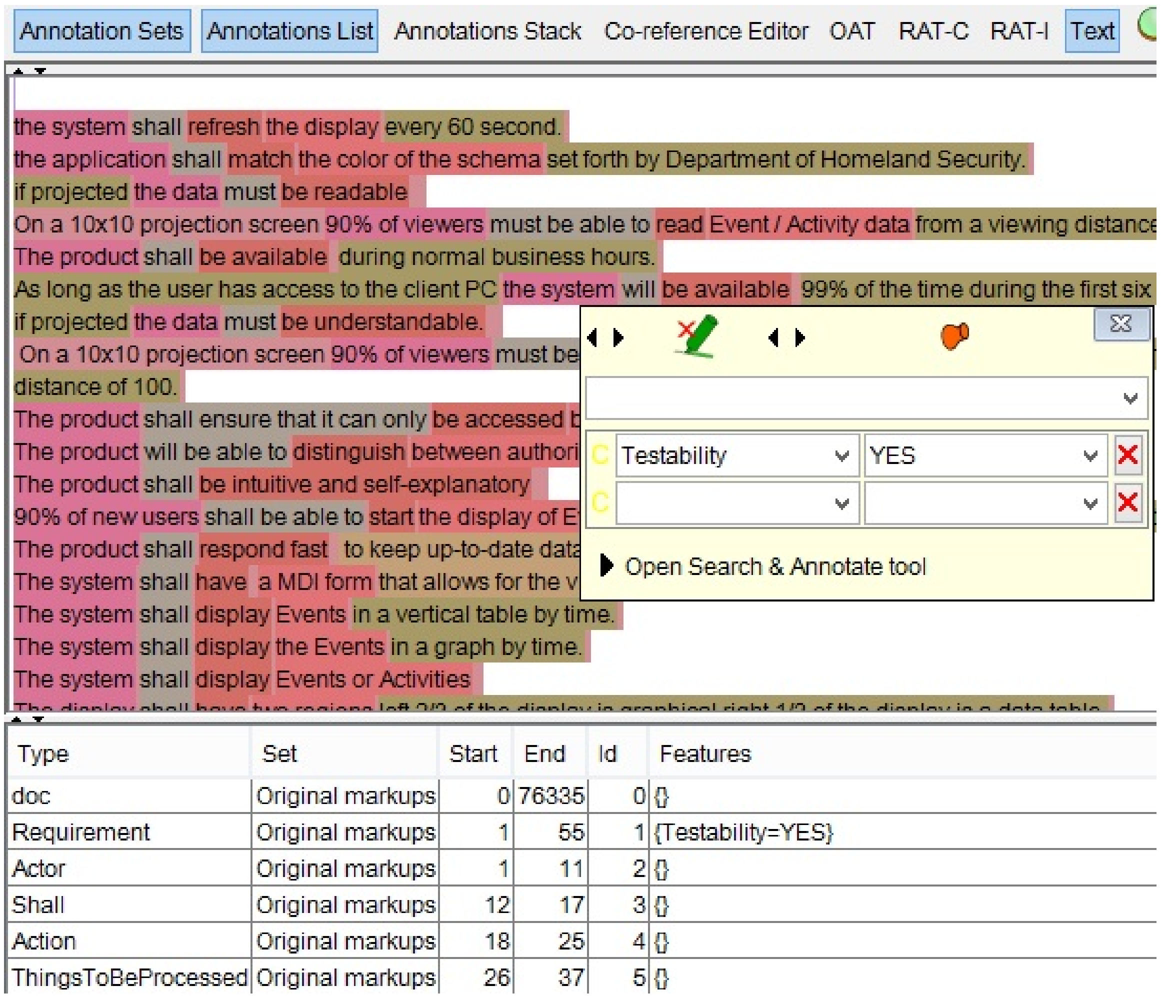
Doc.	A	LF	L	M	O	P	SC	SE	US	NFR	FR	Total
# 1	1	1	0	0	0	2	0	1	3	8	20	28
# 2	1	2	0	0	0	3	1	3	5	15	11	26
# 3	1	0	0	0	6	1	3	6	4	21	47	68
# 4	0	1	3	0	6	2	0	6	4	21	25	47
# 5	2	3	3	0	10	4	3	7	5	37	36	73
# 6	1	2	0	3	15	1	4	5	13	44	26	70
# 7	0	0	1	0	3	2	0	2	0	8	15	23
# 8	5	6	3	2	9	17	4	15	10	71	20	91
# 9	1	0	0	1	2	4	0	0	0	8	16	24
# 10	1	7	0	0	0	4	0	1	2	15	38	53
# 11	1	2	0	1	0	1	0	3	2	10	22	32
# 12	1	2	0	3	2	5	1	3	3	20	13	33
# 13	1	4	0	2	2	0	2	2	6	19	3	22
# 14	1	3	0	2	3	1	0	2	4	16	51	67
# 15	1	2	0	2	3	1	0	2	1	12	15	27
# sent.	18	35	10	16	61	48	18	58	62	326	358	684

include availability (A), look-and-feel (LF), legal (L), maintainability (M), operational (O), performance (P), scalability (SC), security (SE), and usability (US). Table 12 presents the number of sentences in each NFR class.

Thematic Roles Annotation

In order to develop and evaluate automated requirements analysis tools, annotating a requirement corpus, with an ontological representation of different requirement thematic roles is required. Since no such corpus existed, we annotated the PROMISE [PRO] corpus based on the developed requirement thematic roles ontology presented in Section 4.3.2.

Figure 24 shows examples for these different patterns. An index represents the different types, such as pattern 1111000, where each bit represents whether the class is present in the sentence or not: in this example,



the system shall refresh the display every 60 second.
the application shall match the color of the schema set forth by Department of Homeland Security.
if projected the data must be readable
On a 10x10 projection screen 90% of viewers must be able to read Event / Activity data from a viewing distance
The product shall be available during normal business hours.
As long as the user has access to the client PC the system will be available 99% of the time during the first six
if projected the data must be understandable.
On a 10x10 projection screen 90% of viewers must be
distance of 100.
The product shall ensure that it can only be accessed b
The product will be able to distinguish between authori
The product shall be intuitive and self-explanatory
90% of new users shall be able to start the display of E
The product shall respond fast to keep up-to-date data
The system shall have a MDI form that allows for the v
The system shall display Events in a vertical table by time.
The system shall display the Events in a graph by time.
The system shall display Events or Activities
The display shall have two regions left 2/3 of the display is graphical right 1/3 of the display is a data table

Testability	YES

Type	Set	Start	End	Id	Features
doc	Original markups	0	76335	0	{}
Requirement	Original markups	1	55	1	{Testability=YES}
Actor	Original markups	1	11	2	{}
Shall	Original markups	12	17	3	{}
Action	Original markups	18	25	4	{}
ThingsToBeProcessed	Original markups	26	37	5	{}

Figure 23: Manual Annotation Process Example for the Enhanced PROMISE Corpus

the statement contains *Agent*, *Modality*, *Action*, and *Theme*, but not *Condition*, *Goal* or *Instrument*.

The corpus was annotated using the GATE Developer GUI as shown in Figure 23.

The annotation was done by one annotator. Table 13 shows the total number of annotations for different classes such as agent, modality (mod.), action, theme, condition (con.), goal, and instrument (ins.). Moreover, it represents the numbers of all the requirements sentences patterns that exist in the corpus.

During a manual corpus analysis, we determined the following details

Table 13: Corpus Patterns Statistics of the Enhanced PROMISE Corpus.

S	Pattern	Agent	Mod.	Action	Theme	Cond.	Goal	Inst.	Total
1	1111100	✓	✓	✓	✓	✓			268
2	1110100	✓	✓	✓		✓			139
3	1111000	✓	✓	✓	✓				265
4	1110000	✓	✓	✓					13
5	1110010	✓	✓	✓			✓		14
6	1111001	✓	✓	✓	✓			✓	19
7	1111010	✓	✓	✓	✓		✓		34
8	1111101	✓	✓	✓	✓	✓		✓	6
9	1011000	✓		✓	✓				5
10	1011100	✓		✓	✓	✓			2
11	1110101	✓	✓	✓		✓		✓	4
12	1110001	✓	✓	✓				✓	7
13	1111110	✓	✓	✓	✓	✓	✓		7
14	1100000	✓	✓						1
15	1110110	✓	✓	✓		✓	✓		9
16	1111011	✓	✓	✓	✓		✓	✓	1
Tot.		798	787	792	606	435	65	38	798

about the description of NFRs:

1. The different types of *Agents* usually depend on the type of a requirement:

   (a) Requirements describing a system usually start with an agent, like*the system, the product, the name of the system*, or a subsystem, such as *report, data, file, interface, server*. Example: *"The system shall refresh the display every 60 second."*

   (b) Requirements describing how stakeholders interact with a system usually start with an agent, like *the user, realtor, customer, administrator*. Example: *"The user shall select to view the preferred repair facility ratings."*

   (c) Requirements describing a process or scenario usually contain conditions, either in the beginning or in the last phrase of the sentence. Example: *"When a ship is sunk, the product shall simulate the sound of a sinking ship."*
   (d) Requirements describing the user interface and the application screens of a system usually start with an agent, like *The user interface, the look and feel of the system, The table side of the display*. Example: *"The user interface shall have standard menus buttons for navigation."*
   (e) Quality requirements start with a statement, such as *The response time*. Example: *"The response time of schedule generation shall take no longer than 30 seconds."*
   (f) Requirements describing technical specifications usually start with an agent, like *database, media players*. Example: *"The Statement Database provides the transaction details to the Disputes System."*

2. The different patterns for actions are:
   (a) verb: such as *"display"*.
   (b) Be + passive verb: such as *"be restored"*.
   (c) Be + Adjective: such as *"be available", "be consistent"*.
   (d) Verb + ing: *"handling", "processing"*. Here, quality assurance can be applied on the action concept for ambiguity by detecting passive verbs.

3. Conditions in SRS statements appear in two forms::
   (a) *Quantity conditions,* which usually start with, e.g., *"a maximum of", "more than", "up to"*, followed by the quantity, such as time or performance. Quality assurance can be applied on the conditions for testability;
   (b) *If/when condition,* which sometimes appears in the beginning of a sentence before the agent, such as: *"If projected, the data must be readable."*

4. *Goals* usually start with *"for"*, *"in order to"*, or *"to"*, such as in *"to maintain the flow of the game"* or *"for navigation"*.

5. The *Instrument* class usually starts with *"by"*, *"using"*, *"via"*, or *"with"*, such as in *"via the Administration section"* or *"used by POS terminals"*.

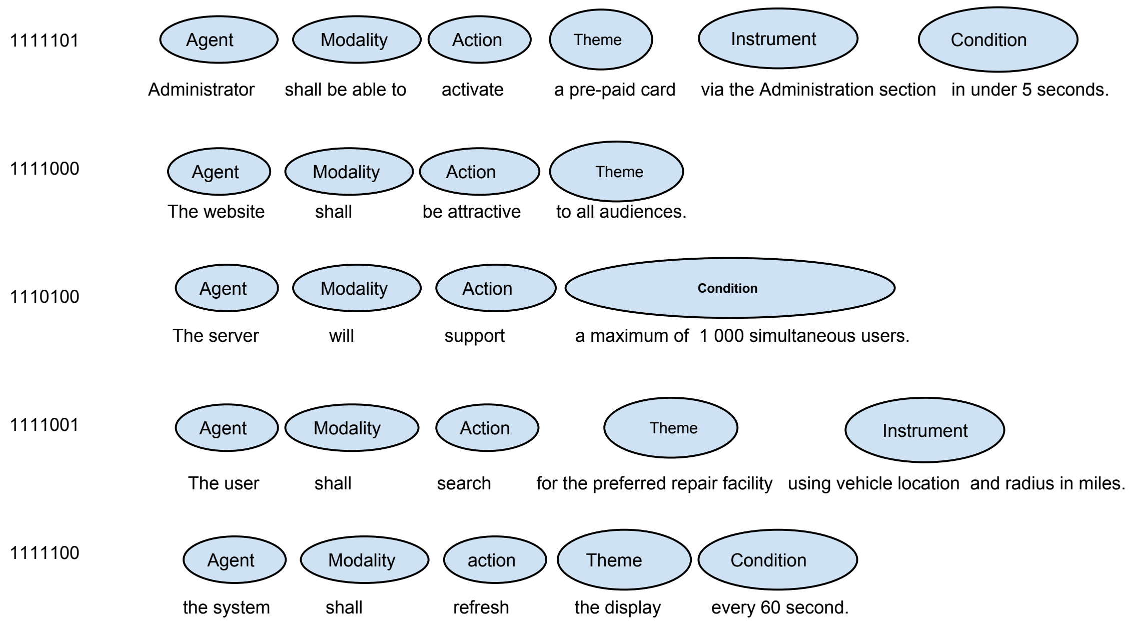


Figure 24: Examples of Different Types of Syntactic Forms Present in the Enhanced PROMISE Corpus

6.1.2 SRS Concordia Corpus

In order to develop and evaluate automated requirements analysis tools, a gold standard requirement corpus, with an ontological representation of different NFR types is annotated. The corpus represents software projects from different problem domains. The documents are written by students and software professionals. The documents were selected for the following reasons:

1. Availability of requirements in three different formats suitable to our experiment (SRS, supplementary specifications, use case model).

Table 14: SRS Concordia Corpus: SRS Documents and their Source for the SRS Concordia Corpus

Doc. #	Doc. Name	Type	Year	Owner	# Sent.
1	Online shopping centre	SRS	2009	Indian Institute of Information Technology	107
2	Student Management System	SRS	2005	University of Portsmouth	174
3	Hospital Patient	SRS	2002	University of Calgary	259
4	Mars Express Processing and Archiving Facility	SRS	2001	Swedish Institute of Space Physics	237
5	Electronic Examination Management	Suppl. spec.	2009	Concordia University	255
6	Student Management System	Use case	2007	Concordia University	2032
Total					3064

2. High quality of the requirement documents.

3. Project domains differ considerably.

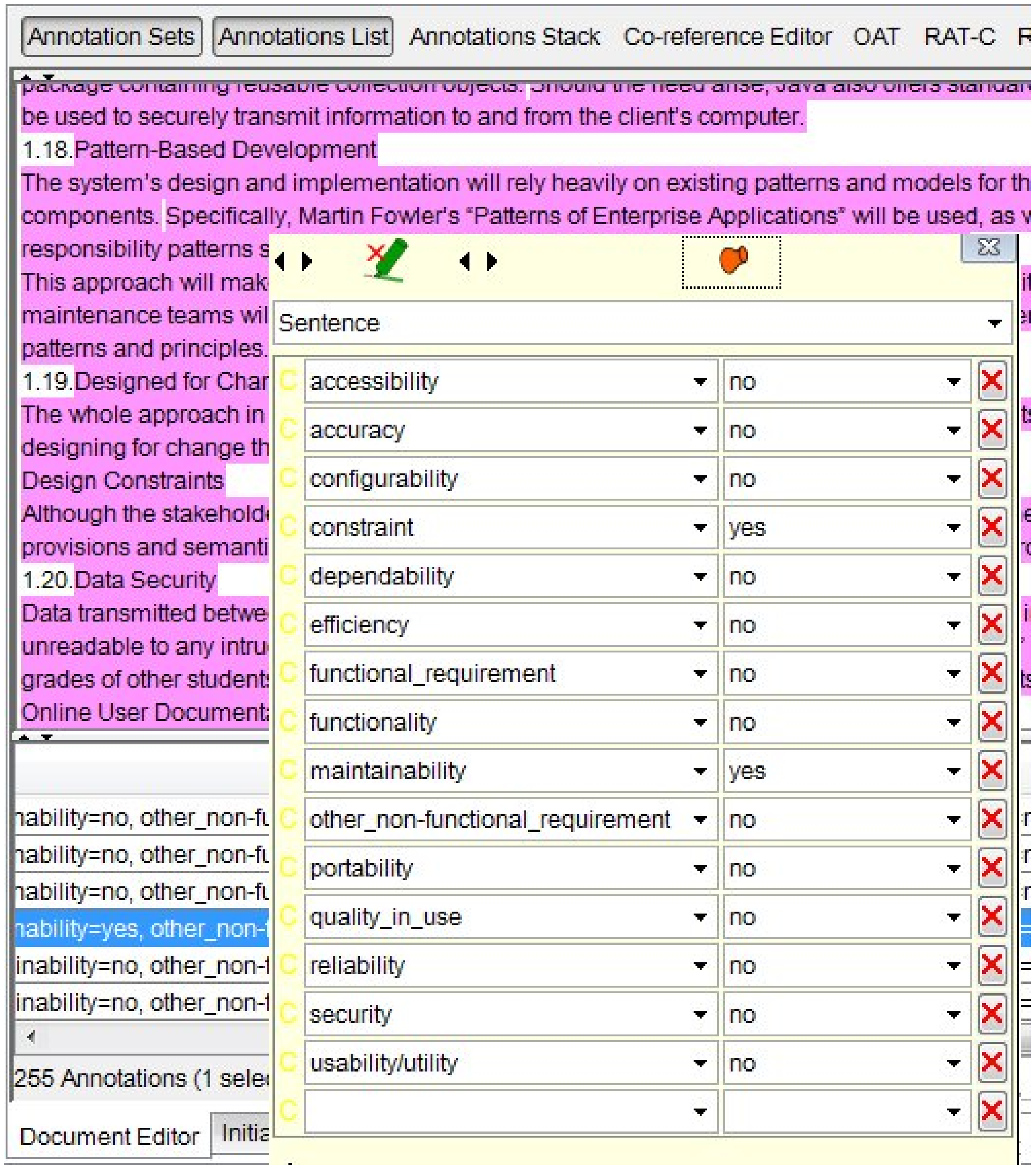
The source documents of our corpus are listed in Table 14.

Manual Annotation

Our manual SRS corpus annotation process was implemented based on GATE Teamware.[1] This is a web-based platform for managing collaborative annotation. The annotation task was carried out by four annotators in order to guarantee the reliability of the annotations. The first step was

[1]GATE Teamware, http://gate.ac.uk/teamware/

to pre-process each document, by automatically splitting it into sentences. Each document is assigned to several annotators, who then examined each sentence and selected the corresponding type of software requirement. Annotators are free to choose any number of requirements for each sentence, including zero. Figure 25 shows an example of this annotation process for one sentence.


Annotation Sets | Annotations List | Annotations Stack | Co-reference Editor | OAT | RAT-C | R

package containing reusable collection objects. Should the need arise, Java also offers standar
be used to securely transmit information to and from the client's computer.
1.18.Pattern-Based Development
The system's design and implementation will rely heavily on existing patterns and models for th
components. Specifically, Martin Fowler's "Patterns of Enterprise Applications" will be used, as v
responsibility patterns s
This approach will mak
maintenance teams wil
patterns and principles.
1.19.Designed for Char
The whole approach in
designing for change th
Design Constraints
Although the stakehold
provisions and semanti
1.20.Data Security
Data transmitted betwe
unreadable to any intru
grades of other student
Online User Document

Sentence

accessibility	no
accuracy	no
configurability	no
constraint	yes
dependability	no
efficiency	no
functional_requirement	no
functionality	no
maintainability	yes
other_non-functional_requirement	no
portability	no
quality_in_use	no
reliability	no
security	no
usability/utility	no

nability=no, other_non-fu
nability=no, other_non-fu
nability=no, other_non-fu
nability=yes, other_non-
inability=no, other_non-
inability=no, other_non-

255 Annotations (1 sele

Document Editor | Initia

Figure 25: Manual Annotation Process Example for SRS Concordia Corpus

Table 15: Numbers of Annotation Classes Sentences per each Document

(NR: Not Requirement, FR: Functional Requirement, CO: Constraint, US: Usability/Utility, SE: Security, EF: Efficiency, FU: Functionality, RE: Reliability)

Doc.	NR	FR	CO	US	SE	EF	FU	RE	Total
1	59	17	26	7	1	1	0	1	112
2	114	32	20	7	10	1	1	2	187
3	180	54	14	6	8	4	1	1	268
4	191	19	21	0	2	3	13	5	254
5	213	23	13	8	2	5	1	0	265
6	1365	642	16	0	34	0	0	0	2057
Total	2122	787	110	28	57	14	16	9	3140

Corpus Statistics

The annotation classes of the SRS Concordia corpus are divided into four main categories: (i) Functional Requirements (FR); (ii) External and Internal Quality: (Accessibility, Accuracy, Configurability, Dependability, Efficiency, Functionality, Maintainability, Portability, Reliability, Security and Usability/Utility); (iii) Constraints; and (iv) other NFR as shown in the ontology design in Section 4.3.1.

Cohen's Kappa

The agreement between the annotators in our corpus is measured for each pair as shown in Table 16. The overall Cohen's Kappa average is 60%, which indicates that the quality of corpus is high and not ambiguous among all annotators.

Gold Standard

Once the annotators had completed their task, their results were used to create a gold standard for each document. The results of all the annotators for all sentences are compared where they agreed on the annotation and retain their choice for the gold standard. For all sentences where

Table 16: Cohen's Kappa between each Pair of Annotators

Doc	1,2	1,3	1,4	2,3	2,4	3,4	Avg.
1	0.75	0.79	0.9	0.7	0.72	0.79	0.78
2	0.74	0.73	0.88	0.68	0.78	0.82	0.77
3	0.64	0.52	0.62	0.67	0.67	0.56	0.61
4	0.73	0.7	N/A	0.74	N/A	N/A	0.72
5	0.27	0.36	N/A	0.36	N/A	N/A	0.33
6	0.7	0.27	N/A	0.21	N/A	N/A	0.39
Average							0.60

they disagreed, the gold standard annotation was obtained through group discussions.

Concordia Test Set

We used the Enhanced PROMISE corpus for developing the rules of our system. We annotated another small set to perform the testing. This test set is extracted from the SRS Concordia Corpus described in our previous work [ROW13]. We have chosen 87 NFRs sentences from the SRS Concordia corpus and annotated it by two different annotators. A session was held to discuss the differences and propose the final annotation.

The NFR sentences in the SRS Concordia corpus are 224 sentences. The duplicate sentences are excluded, so we have 197 sentences. We removed the incomplete sentences such as "works for medium size information databases", "Mysql for database", and "multiple user interface". Some sentences are incomplete as they are part of items or lists. The final test set is 87 NFRs.

6.2 System Evaluation

Using the corpora described in Section 6.1, we now present the evaluation of the NFR classifier, thematic roles extractor, and the non-testability detector.

6.2.1 NFR Classifier

The ML classifier is evaluated using 6-fold cross validation with the metrics precision, recall and F-measure [FBY92], defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \; \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

$$\text{F1-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}},$$

where TP (true positive) is the number of correctly classified requirements, FP (false positive) the number of requirements incorrectly classified and FN (false negative) the number of requirements incorrectly not classified.

Why SVM?

As mentioned in Section 4.4.1, Support Vector Machines (SVMs) provided the best performance in our experiments. We tested other machine learning algorithms, including K-Nearest Neighbour (KNN), Naive Bayes, and Perceptron Algorithm with Uneven Margins (PAUM) [LZH+02], besides SVM. In the system development initial stages, we performed an experiment on three documents from the SRS Concordia corpus, with a total of 703 FRs, 53 NFRs, and 1769 NRs. The results show an average F1 measure of 76% for SVM, compared to 74% for PAUM, and 71% for KNN, as shown in Table 17.

Results on the SRS Concordia Corpus

The results obtained on the SRS Concordia corpus are summarized in Table 18. The overall weighted average F1 measure is 84.96%.

Analysis and Discussion

To analyze the classification results, we computed the confusion matrices [Faw04], which presents the false positives and false negatives of the classifier outcome. Table 19 illustrates the confusion matrices for the seven classifiers. For example, the confusion matrix of the Functional Requirement (FR) Classifier shows that 76 sentences are classified as false positive,

Table 17: SVM Results Compared to other Machine Learning Algorithms

Classifier	Class	Precision	Recall	F-Measure
SVM	FR	0.74	0.77	0.75
	NFR	0.58	0.69	0.61
	NR	0.95	0.92	0.94
	Avg	0.76	0.80	0.76
PAUM	FR	0.78	0.78	0.77
	NFR	0.50	0.59	0.52
	NR	0.92	0.94	0.93
	Avg	0.73	0.77	0.74
KNN	FR	0.74	0.65	0.69
	NFR	0.53	0.59	0.53
	NR	0.89	0.96	0.92
	Avg	0.72	0.73	0.71
Naive Bayes	FR	0	0	0
	NFR	0	0	0
	NR	0.80	0.68	0.72
	Avg	0.80	0.52	0.62

and 4 sentences as false negative. The true positive and true negative results are located in the diagonal.

By analyzing the confusion matrices and the evaluation results, the results should be improved for the FR classifier in false positives, and constraint classifier in false negatives. For improving some of the categories, it will be necessary to increase the amount of training data, especially in the reliability, efficiency and functionality classes.

Comparison between our work and other published work based on the PROMISE corpus

We applied our NFR classifier algorithm on the PROMISE corpus using Weka [HFH+09], in order to evaluate its improvement over previously published results. Three related work are described in Section 3.2. Hussain

Table 18: Results for the SVM Classifiers on the SRS Concordia Corpus

Class	#	Precision	Recall	F-Measure
FR	787	0.82	0.82	0.82
Constraint	110	0.91	0.91	0.91
Security	57	0.97	0.97	0.97
Usability/Utility	28	0.97	0.97	0.97
Efficiency	14	0.98	0.98	0.98
Functionality	16	0.98	0.98	0.98
Reliability	9	0.99	0.99	0.99
Weighted Average	1021	0.84	0.84	0.84

et al. [HKO08] designed their algorithm to classify for only two classes (FR, and NFR). Otherwise, our NFR algorithm classify for different types of NFRs. Casamayor et al. [CGC09] proposed a recommender system using a semi-supervised learning technique. Otherwise, our NFR classifier is a supervised learning technique. Table 20 compares the performance of our SVM classifier (column SVM) with the approach described by Cleland-Huang et al. in [CHSZS07] (column 'Weighted Indicator'). As can be seen from the table, the precision is roughly comparable, but our approach has significantly higher recall.

6.2.2 Thematic Roles Extractor

The system is evaluated with the metrics precision (P), recall (R), and F-measure [ea11] as defined above. The results in Table 21 summarize the evaluation on the training corpus and the test corpus described in Section 6.1.2.

The overall weighted average F1 measure from the testing set is 72%. The system produces good results when it is near to the *Modality* class, which produce the best result, 95% F-Measure. *Agent* and *Action* classes are annotated with 75% and 73%, respectively, as they depend on the *Modality* class. *Goal* is annotated with 13% F-Measure, as it is typically located in the last part of a sentence, where analysis errors can compound.

Table 19: Confusion Matrices

FR		
	Yes	No
Yes	774	4
No	76	2210

Constraint		
	Yes	No
Yes	97	13
No	1	2954

Security		
	Yes	No
Yes	56	2
No	9	2998

Usability		
	Yes	No
Yes	21	7
No	0	3037

Efficiency		
	Yes	No
Yes	9	5
No	0	3051

Functionality		
	Yes	No
Yes	5	11
No	0	3049

Reliability		
	Yes	No
Yes	9	0
No	0	3056

6.2.3 Non-Testability Detector

The non-testability detector is evaluated with the metrics precision, recall and F-measure. The system produces excellent results, which are 86% for the training set and 80% for testing set for the rule-based system. We also compared it with a statistical model described in Section 4.5 using SVM that produces 80% after preprocessing, containing tokenization and Snowball-based stemming, as presented in Table 22,

The automatic sentence splitter has some limitations. For example, it generates 26 sentences more than the manual annotation. The confusion matrices summarize the quality of the non-testability detector, as shown in Table 23.

The training Enhanced PROMISE corpus was developed by master students in one university, which may be too limited to represent a global

Table 20: Comparison between SVM and Indicator Classifiers on the PROMISE Corpus

	SVM			Weighted Indicator [CHSZS07]		
Class	Prec.	Recall	F-Meas.	Prec.	Recall	F-Meas.
AV	0.93	0.66	0.77	0.88	0.11	0.19
LE	0.80	0.61	0.69	0.70	0.16	0.26
LF	0.64	0.63	0.64	0.51	0.11	0.19
MA	0.77	0.41	0.53	0.88	0.10	0.19
OP	0.64	0.66	0.65	0.72	0.11	0.19
PE	0.84	0.70	0.76	0.62	0.27	0.37
SC	0.66	0.38	0.48	0.72	0.11	0.19
SE	0.83	0.77	0.80	0.80	0.18	0.29
US	0.79	0.62	0.70	0.98	0.14	0.25
Avg.	0.77	0.60	0.67	0.76	0.14	0.23

Table 21: Thematic Role Evaluation Results

	Corpus							
	Enhanced PROMISE Training corpus				Concordia Test corpus			
Class	#	P	R	F-1	#	P	R	F-1
Modality	798	0.97	0.98	0.98	82	0.96	0.94	0.95
Action	809	0.97	0.83	0.89	89	0.88	0.63	0.73
Agent	799	0.93	0.90	0.92	87	0.81	0.69	0.75
Theme	622	0.70	0.76	0.73	79	0.80	0.56	0.66
Condition	458	0.82	0.45	0.58	34	0.50	0.15	0.23
Goal	61	0.51	0.30	0.37	9	0.17	0.11	0.13
Instrument	31	0.43	0.39	0.41	0	0	1	0
Average	797	0.88	0.81	0.84	87	0.83	0.64	0.72

NFRs non-testability detection. Therefore, our test corpus documents are completely drawn from different domains. In the future, further annotation is required to improve the recall of the system.

The initial results for the machine learning approach using SVM shows

Table 22: Evaluation of the Automatic Non-Testability Detector on the Enhanced PROMISE Corpus

Class	#	Precision	Recall	F-Measure
Rule-Based Training (Enhanced PROMISE)	196	0.86	0.85	0.86
Rule-Based Testing (SRS Concordia)	20	0.79	0.80	0.80
SVM	196	0.80	0.81	0.80

Table 23: Non-Testability Detector: Confusion Matrices

Rule-based		
	Yes	No
Yes	117	65
No	44	570

SVM		
	Yes	No
Yes	91	92
No	57	577

that the F1 measure is 80.6%.

6.3 Summary

In this chapter, we described the Concordia NFR gold standard corpus, its statistics, and its manual annotation process. We also presented the Enhanced PROMISE corpus, annotated for requirement thematic roles. The NFR classifier was evaluated on the SRS Concordia corpus, resulting in an F1 measure of 84.96%. The rule-based thematic roles extractor leads to an F1 measure of 82%. Finally, our rule-based non-testability detector system produces excellent F1 results of 86%. Chapter 7 will conclude the thesis and briefly discuss possible future work.

Chapter 7

Conclusions and Future Work

The best way to predict the future is to create it.

Peter Drucker

In this chapter, we provide a summary and a conclusion of our research work. We will also suggest some research directions to be undertaken in the near future.

We developed a novel, manually annotated (gold standard) corpus for sentence-based classification of requirements. In particular, it focuses on non-functional requirements (NFRs).

We developed a new classification algorithm for the automatic categorization of requirements in software specifications. In this work, we focused on NFRs classification. The results of this work will be of interest to researchers as well as practitioners from industry, who are interested in estimating the effort for building requirements in general and improving software quality in particular, and use measurement data in requirements engineering.

The ontological foundation of our work allows to automatically transform software requirements documents into a semantic representation, which can then be further processed in order to (i) estimate the cost of the software system and (ii) measure the quality of the written requirements.

We developed an automatic non-testability detector system to help and assist the analyst to write clear, testable, and measurable requirements, as testable requirements reduce the ambiguity and increase the understandability. This accordingly helps the testers to write the system test cases and also improve the requirements and testing traceability.

All developed resources described here, including the ontology, corpus annotations, and NLP pipeline, are available as open source software.[1]

In future work, our goal is to address existing mis-classifications by developing additional syntactic and semantic features for the classifiers. Additionally, we aim to apply quality assurance methods by applying further reasoning on the populated ontology.

A semantic framework was developed for NFR quality assurance, by presenting the conceptualization of requirements statements at the micro level. *Testability* is analyzed as one NFR quality attribute using an automated, rule-based system. In future work, we plan to add additional QA criteria, such as contradiction, redundant, and ambiguity.

The developed system will be available to project stakeholders as part of our ReqWiki[2] semantic collaborative requirements engineering platform. It is known that providing NLP support can significantly enhance the quality of a developed specification [SAW13]. The additional QA support developed here is highly relevant for both researchers and industry practitioners concerned with software measurement data, effort estimation, and overall project quality.

The long-term vision of this work is to create quality assurance applications for different types of defects and errors, in order to decrease the probability of software project failures.

[1] http://www.semanticsoftware.info/non-testable-nfr-detector
[2] ReqWiki, http://www.semanticsoftware.info/reqwiki

Bibliography

[ABSDL07] Taiseera Hazeem Al Balushi, Pedro R. Falcone Sampaio, Divyesh Dabhi, and Pericles Loucopoulos. ElicitO: a quality ontology-guided NFR elicitation tool. In *Proceedings of the 13th International Working Conference on Requirements Engineering: foundation for software quality*, REFSQ'07, pages 306–319, Berlin, Heidelberg, 2007. Springer-Verlag.

[AO98] Douglas E. Appelt and Boyan Onyshkevych. The Common Pattern Specification Language. In *Proceedings of a Workshop on Held at Baltimore, Maryland*, TIPSTER '98, pages 23–30, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics.

[BDMV10] Francesca Bonin, Felice Dell'Orletta, Simonetta Montemagni, and Giulia Venturi. A Contrastive Approach to Multi-word Extraction from Domain-specific Corpora. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May 2010. European Language Resources Association (ELRA).

[BGV92] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA,

1992. ACM.

[BSdPL03] K.K. Breitman and J.C. Sampaio do Prado Leite. Ontology as a requirements engineering product. In *Proceedings of the 11th IEEE International Requirements Engineering Conference*, pages 309–319, Sept 2003.

[CA07] B. H C Cheng and J.M. Atlee. Research Directions in Requirements Engineering. In *Future of Software Engineering, 2007. FOSE '07*, pages 285–303, May 2007.

[CBC12] Verónica Castaneda, Luciana C. Ballejos, and Maria Laura Caliusco. Improving the Quality of Software Requirements Specifications with Semantic Web Technologies. In *Proceedings of the Workshop em Engenharia de Requisitos (WER'12)*, 2012.

[CBCG10] Veronica Castañeda, Luciana Ballejos, Ma. Laura Caliusco, and Ma. Rosa Galli. The Use of Ontologies in Requirements Engineering. *Global Journal of Researches In Engineering*, 10(6), 2010.

[CGC09] Agustin Casamayor, Daniela Godoy, and Marcelo Campo. Semi-Supervised Classification of Non-Functional Requirements: An Empirical Analysis. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, pages 35–45, 2009.

[CGC12] Agustin Casamayor, Daniela Godoy, and Marcelo Campo. Mining Textual Requirements to Assist Architectural Software Design: A State of the Art Review. *Artificial Intelligence Review*, 38(3):173–191, October 2012.

[CHSZS06] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. The Detection and Classification of Non-Functional requirements with Application to Early Aspects. In *Proceedings of 14th IEEE International Conference on Requirements Engineering*, pages 39–48, Minneapolis/St. Paul, MN, 2006.

[CHSZS07] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. Automated Classification of Non-Functional Requirements. *Requirements Engineering*, 12:103–120, 2007.

[CNYM00] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, Boston/Dordrecht/London, 2000.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. In *Machine Learning*, pages 273–297, 1995.

[DDCPT10] Mauro Dragoni, Célia Da Costa Pereira, and Andrea G.B. Tettamanzi. An Ontological Representation of Documents and Queries for Information Retrieval Systems. In Nicolás Garcia-Pedrajas, Francisco Herrera, Colin Fyfe, José Manuel Benítez, and Moonis Ali, editors, *Trends in Applied Intelligent Systems*, volume 6097 of *Lecture Notes in Computer Science*, pages 555–564. Springer Berlin Heidelberg, 2010.

[Dev02] Vladan Devedzić. Understanding Ontological Engineering. *Commun. ACM*, 45(4):136–144, April 2002.

[DS06] Glen Dobson and Peter Sawyer. Revisiting Ontology-Based Requirements Engineering in the age of the Semantic Web. In: Dependable Requirements Engineering of Computerised Systems at NPPS, 2006.

[ea11] Hamish Cunningham et al. *Text Processing with GATE (Version 6)*. University of Sheffield, Department of Computer Science, 2011.

[Faw04] Tom Fawcett. ROC Graphs: Notes and Practical Considerations for Researchers. Technical report, HP Laboratories, 2004.

[FBY92] William B. Frakes and Ricardo Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, facsimile edition, 1992.

[FdSG14] Alessio Ferrari, Felice dell'Orletta, GiorgioOronzo Spagnolo, and Stefania Gnesi. Measuring and Improving the Completeness of Natural Language Requirements. In Camille Salinesi and Inge van de Weerd, editors, *Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 8396 of *Lecture Notes in Computer Science*, pages 23–38. Springer International Publishing, 2014.

[FFGL01] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. An Automatic Quality Evaluation for Natural Language Requirements. In *Proceedings of the Seventh International Workshop on RE: Foundation for Software Quality (REFSQ'2001)*, pages 4–5, 2001.

[Fil68] Charles J. Fillmore. The Case for Case. In Emmon Bach and Robert T. Harms, editors, *Universals in Linguistic Theory*. Holt, Rinehart and Winston, New York, 1968.

[Fin94] A. Finkelstein. Requirements engineering: a review and research agenda. In *Software Engineering Conference, First Asia-Pacific*, pages 10–19, Dec 1994.

[Fir03] D.G. Firesmith. *Common Concepts Underlying Safety, Security, and Survivability Engineering*. Technical note. Carnegie Mellon University, Software Engineering Institute, 2003.

[FMK$^+$11] Stefan Farfeleder, Thomas Moser, Andreas Krall, Tor Stålhane, Inah Omoronyia, and Herbert Zojer. Ontology-Driven Guidance for Requirements Elicitation. In Grigoris Antoniou, Marko Grobelnik, Elena Paslaru Bontas Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Z. Pan, editors, *ESWC (2)*, volume 6644 of *Lecture Notes in Computer Science*, pages 212–226. Springer, 2011.

[Fon07] Frederico Fonseca. The double role of ontologies in information science research. *Journal of the American Society for Information Science and Technology*, 58(6):786–793, 2007.

[GCSY08] G. Gokyer, S. Cetin, C. Sener, and M.T. Yondem. Non-functional Requirements to Architectural Concerns: ML and NLP at Crossroads. In *Software Engineering Advances, 2008. ICSEA '08. The Third International Conference on*, pages 400–406, Oct 2008.

[Gli07] M. Glinz. On Non-Functional Requirements. *Requirements Engineering, IEEE International Conference on*, pages 21–26, October 2007.

[Gru65] Jeffrey S. Gruber. *Studies in Lexical Relations*. PhD thesis, MIT, Cambridge, MA, 1965.

[Hep00] Mark Hepple. Independence and Commitment: Assumptions for Rapid Training and Execution of Rule-based POS Taggers. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 278–277, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[HFH$^+$09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. pages 10–18, New York, NY, USA, November 2009. ACM.

[HKO08] Ishrar Hussain, Leila Kosseim, and Olga Ormandjieva. Using Linguistic Knowledge to Classify Non-functional Requirements in SRS documents. In Epaminondas Kapetanios, Vijayan Sugumaran, and Myra Spiliopoulou, editors, *Natural Language and Information Systems*, volume 5039 of *Lecture Notes in Computer Science*, pages 287–298. Springer Berlin / Heidelberg, 2008.

[HM01] Volker Haarslev and Ralf Möller. RACER System Description. In *Proceedings of the First International Joint Conference on Automated Reasoning*, IJCAR '01, pages 701–706, London, UK, 2001. Springer-Verlag.

[HOK07] I. Hussain, O. Ormandjieva, and L. Kosseim. Automatic Quality Assessment of SRS Text by Means of a Decision-Tree-Based Text Classifier. In *Proceedings of the Seventh International Conference on Quality Software (QSIC '07)*, pages 209–218, Oct 2007.

[ISO10] ISO/IEC. ISO/IEC 25010:2011 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Technical report, 2010.

[Jac72] Ray Jackendoff. *Semantic Interpretation in Generative Grammar.* MIT Press, Cambridge, MA, 1972.

[JKCW08] Tian Jingbai, He Keqing, Wang Chong, and Liu Wei. A Context Awareness Non-functional Requirements Metamodel Based on Domain Ontology. In *Proceedings of the IEEE International Workshop on Semantic Computing and Systems*, WSCS '08, pages 1–7, Washington, DC, USA, 2008. IEEE Computer Society.

[JM09] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[Kas09] Mohamad Kassab. *Non-Functional Requirements: Modeling and Assessment.* VDM Verlag, 2009.

[Lid01] E. D. Liddy. *Natural Language Processing. In Encyclopedia of Library and Information Science.* NY. Marcel Decker, Inc., 2001.

[LMP04] Mich Luisa, Franch Mariangela, and Inverardi Pierluigi. Market Research for Requirements Analysis Using Linguistic Tools. *Requir. Eng.*, 9(1):40–56, February 2004.

[LW03] Dean Leffingwell and Don Widrig. *Managing software requirements: a unified approach*. Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA, 2003.

[LZH$^+$02] Yaoyong Li, Hugo Zaragoza, Ralf Herbrich, John Shawe-Taylor, and Jaz S. Kandola. The Perceptron Algorithm with Uneven Margins. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 379–386, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[MBK91] Y.S. Maarek, D.M. Berry, and G.E Kaiser. An information retrieval approach for automatically constructing software libraries. *Transactions on Software Engineering*, 17(8):800–813, 1991.

[MFI04] Luisa Mich, Mariangela Franch, and Pier Luigi Novi Inverardi. Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9:40–56, 2004.

[Mil95] George A. Miller. WordNet: A Lexical Database for English. *Communications of The ACM*, 38:39–41, 1995.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[Mv04] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/, February 2004.

[MWHB11] Thomas Moser, Dietmar Winkler, Matthias Heindl, and Stefan Biffl. Automating the Detection of Complex Semantic Conflicts between Software Requirements(An empirical study on requirements conflict analysis with semantic technology). In *SEKE*, pages 729–735. Knowledge Systems Institute Graduate School, 2011.

[OD08] Martin J. O'Connor and Amar K. Das. SQWRL: A Query Language for OWL. In Rinke Hoekstra and Peter F. Patel-Schneider, editors, *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[OHK07] Olga Ormandjieva, Ishrar Hussain, and Leila Kosseim. Toward a Text Classification System for the Quality Assessment of Software Requirements Written in Natural Language. In *Fourth International Workshop on Software Quality Assurance: In Conjunction with the 6th ESEC/FSE Joint Meeting*, SOQUA '07, pages 39–45, New York, NY, USA, 2007. ACM.

[PA09] Shari Lawrence Pfleeger and Joanne M. Atlee. *Student Study Guide for Software Engineering: Theory and Practice*. Prentice Hall Press, Upper Saddle River, NJ, USA, 4th edition, 2009.

[PKKS00] S Park, H Kim, Y Ko, and J Seo. Implementation of an efficient requirements-analysis supporting system using similarity measure techniques. *Information and Software Technology*, 42(6):429 – 438, 2000.

[Pow07] David M. W. Powers. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. Technical Report SIE-07-001, School of Informatics and Engineering, Flinders University, Adelaide, Australia, 2007.

[Pre98] IEEE Computer Society Press. IEEE guide to software requirement specification, standard 830-1998. 1998.

[PRO] PROMISE Software Engineering Repository. https://code.google.com/p/promisedata/wiki/nfr.

[ROW13] Abderahman Rashwan, Olga Ormandjieva, and René Witte. Ontology-Based Classification of Non-Functional Requirements in Software Specifications: A new Corpus and SVM-Based Classifier. In *The 37th Annual International Computer Software & Applications Conference (COMPSAC 2013)*, page 381–386, Kyoto, Japan, July 2013. IEEE.

[RR06] Suzanne Robertson and James Robertson. *Mastering the Requirements Process (2nd Edition)*. Addison-Wesley Professional, 2006.

[RSH09] Chris Rupp, Matthias Simon, and Florian Hocker. Requirements Engineering und Management. *HMD Praxis der Wirtschaftsinformatik*, 46(3):94–103, 2009.

[SAW13] Bahar Sateli, Elian Angius, and René Witte. The ReqWiki Approach for Collaborative Software Requirements Engineering with Integrated Text Analysis Support. In *The 37th Annual International Computer Software & Applications Conference (COMPSAC 2013)*, page 405–414, Kyoto, Japan, July 2013. IEEE.

[Sim13] Phil Simon. *Too big to ignore: the business case for big data*. Wiley and SAS Business Series. Wiley, New Delhi, 2013.

[SM86] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[Som06] Ian Sommerville. *Software Engineering: (Update) (8th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[SPA08] SPARQL query language for RDF. Technical report, World Wide Web Consortium http://www.w3.org/TR/rdf-sparql-query/, January 2008.

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description, booktitle = Proceedings of the Third International Joint Conference on Automated Reasoning. IJCAR'06, pages 292–297, Berlin, Heidelberg, 2006. Springer-Verlag.

[TM00] Kristina Toutanova and Christopher D. Manning. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-speech Tagger. In *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora: Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, EMNLP '00, pages 63–70, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[UBN11] Ashfa Umber, Imran Sarwar Bajwa, and M. Asif Naeem. NL-Based Automated Software Requirements Elicitation and Specification. In Ajith Abraham, Jaime Lloret Mauri, John F. Buford, Junichi Suzuki, and Sabu M. Thampi, editors, *ACC (2)*, volume 191 of *Communications in Computer and Information Science*, pages 30–39. Springer, 2011.

[van09] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[WKR10] René Witte, Ninus Khamis, and Juergen Rilling. Flexible Ontology Population from Text: The OwlExporter. In *The Seventh International Conference on Language Resources and Evaluation (LREC 2010)*, pages 3845–3850, Valletta, Malta, May 19–21 2010. ELRA.

Appendix A

NFR Classifier Configuration

Listing A.1: SVM Configuration File

```
<?xml version="1.0"?>
<ML-CONFIG>
  <VERBOSITY level="0"/>
  <SURROUND value="false"/>

  <IS-LABEL-UPDATABLE value="true"/>
  <IS-NLPFEATURELIST-UPDATABLE value="true"/>

  <PARAMETER name="thresholdProbabilityEntity" value="0.2"/>
  <PARAMETER name="thresholdProbabilityBoundary" value="0.42"/>
  <PARAMETER name="thresholdProbabilityClassification" value="0.4"/>

  <multiClassification2Binary method="one-vs-others"/>

  <!-- Evaluation : how to split the corpus into test and learn? -->
  <EVALUATION method="kfold" runs="6"/>

  <FILTERING ratio="0.0" dis="near"/>

  <ENGINE nickname="SVM" implementationName="SVMLibSvmJava"
          options=" -c 1 -t 0 -d 3 -m 40 -tau 0.3  "/>
  <!--

        <ENGINE nickname="KNN" implementationName="KNNWeka" options = "-k 1"/>
     <ENGINE nickname="C45" implementationName="C4.5Weka"/>

          <ENGINE nickname="SVM" implementationName="PAUM" options=" -p 50 -n 5 -optB 0.0
                "/>
          <ENGINE nickname="NB" implementationName="NaiveBayesWeka"/>  -->

  <DATASET>
     <INSTANCE-TYPE>Sentence</INSTANCE-TYPE>
```

```
        <NGRAM>
            <NAME>Sent1gram</NAME>
            <NUMBER>1</NUMBER>
            <CONSNUM>1</CONSNUM>
            <CONS-1>
                 <TYPE>Token</TYPE>
                                <FEATURE>stem</FEATURE>
            </CONS-1>
        </NGRAM>

         <ATTRIBUTE>
          <NAME>Class</NAME>
          <SEMTYPE>NOMINAL</SEMTYPE>
          <TYPE>Sentence</TYPE>
                      <FEATURE>functional_requirement</FEATURE>
          <POSITION>0</POSITION>
          <CLASS/>
        </ATTRIBUTE>




    </DATASET>

</ML-CONFIG>

```

Appendix B

Gazetteer Lists

B.1 Modality

Listing B.1: Shall.lst

```
are able to
can
can only
cannot
have the ability to
is expected to
is not expected to
may
must
must be able to
must ensure that
shall
shall able to
shall accurately
shall allow to
shall ask the user to
shall automatically
shall be able
shall be able to
shall be able to continue to
shall be able to successfully
shall be available for
shall be capable of
shall be easy to
shall be expected to
shall be intuitive to
shall continue to
shall easily
shall ensure that
shall ensure that it can only
shall have the ability to
shall not
shall not be able to
shall only
shall successfully be able to use the system to
shell be able to
should
should be able to
should be able to easily
should be able to successfully use the system to
should be possible to
should not
should only
should only have to
should/should not
will
will be able to
will be able to successfully
will be used to
will need to
will no longer be able to
will not be able to
```

Listing B.2: Shallallow.lst

```
System must provide the ability to
does not allow
must allow
must further allow
must prevent
must prompt
must provide
shall allow
shall help
shall let
shall make
shall prevent
shall provide
shall retain
will allow
will provide
```

Listing B.3: Ability.lst

```
the ability to
with the ability to
```

B.2 Quantification

Listing B.4: Quantification.lst

```
after
after
after the first
after which
at least
between
during the first
every
every
less than
maximum of
maximum response time of
minimum of
minimum response time of
more than
no later
no longer than
no more than
not be more than
not exceed
over
under
up to
with
within
within the
```

B.3 Condition and Limit

Listing B.5: Condition.lst

```
As long as
If
Once
When
if
once
when
```

Listing B.6: Conjunction.lst

```
and
but
```

```
for
nor
not
or
so
yet
```

Listing B.7: Limit.lst

```
a maximum of
at most
based on
before
between
during
either
even
every
if
minimum of
more than
no later
no longer
no more than
once
that
that are
the minimum
up to
via
when
whenever
which
while
with
within
without
```

Listing B.8: NFR.lst

```
GUI
Integrity
Measure
Originality
Primary Actor is authenticated
Qualification
achieve
appropriate
as possible
available
benfits
```

12 concurrent
13 critical
14 design
15 designed
16 easily
17 effect
18 efficient
19 efficiently
20 enough
21 heavily
22 issue
23 maximum
24 minimizing
25 quickly
26 response time
27 reuse
28 simplest
29 user friendly
30 user interface
31 validate

Appendix C

JAPE Rules for Requirement Thematic Roles Extractor

C.1 Modality

Listing C.1: Modality.jape

```
Phase:Modality
Input: Lookup
Options: control = applet

Rule: rule2
(
  {Lookup.minorType == shall1}
)
:ann
-->
 :ann.Modality = {type = "Modality", string = :ann@string, minor = "Modality"}


 Rule: rule3
(
  {Lookup.minorType == shallallow}
)
:ann
-->
 :ann.Modality = {type = "Modality", string = :ann@string, minor = "ModalityAllow"}
```

C.2 Agent

Listing C.2: Agent.jape

```
Phase:Agent
Input: Modality NP Token
Options: control = applet


Rule: rule1
(
  (({NP})( {Token.string == and} | {Token.string == of})({NP})):Agent
  {Modality.minor == Modality}
)
:ann
-->
 :Agent.Agent = {type = "Agent", string = :Agent@string}



Rule: rule2
(
  ({NP}):Agent
  {Modality.minor == Modality}
)
:ann
-->
 :Agent.Agent = {type = "Agent", string = :Agent@string}




 Rule: rule3
(
  {Modality.minor == ModalityAllow}
  (({NP})( {Token.string == and} | {Token.string == of})({NP})):Agent

)
:ann
-->
 :Agent.Agent = {type = "Agent", string = :Agent@string}

 Rule: rule4
(
  {Modality.minor == ModalityAllow}
  ({NP}):Agent

)
:ann
-->
 :Agent.Agent = {type = "Agent", string = :Agent@string}

```

C.3 Action

Listing C.3: Action.jape

```
Phase:action
Input: Modality Token Someone Agent
Options: control = first

 Rule: rule2
(
 ({Modality.minor == ModalityAllow}{Agent})({Token.string == to})?
 ( {Token.category == VB} | {Token.category == VBP} | {Token.category == VBG} ):action
)
:ann
-->
 :action.Action = {type = "Action", string = :action@string}



 Rule: rule1
(
  {Modality}
  (({Token.string == allow}|{Token.string == prevent} | {Token.string == notify} | {Token.
      string == help} ) ({Someone}) ({Token.string == to}|{Token.string == from}|{Token.
      string == of}))?

  (
     {Token.string == be}{Token.category == VB}{Token.category == IN}
   | {Token.string == be}{Token.category == VBP}{Token.category == IN}
   | {Token.string == be}{Token.category == VBN}{Token.category == IN}
    | {Token.string == be}{Token.category == JJ}{Token.category == IN}
   | {Token.string == be}{Token.category == VB}{Token.category == TO}
   | {Token.string == be}{Token.category == VBP}{Token.category == TO}
   | {Token.string == be}{Token.category == VBN}{Token.category == TO}
    | {Token.string == be}{Token.category == JJ}{Token.category == TO}

   | {Token.string == be}{Token.category == JJ}
   | {Token.string == be}{Token.category == RB}
   | {Token.string == be}{Token.category == VBN}
   | {Token.string == be}{Token.category == VB}

   | {Token.category == VB, Token.string != be}{Token.category == IN}
   | {Token.category == VBP}{Token.category == IN}
   | {Token.category == VB, Token.string != be}{Token.category == TO}
   | {Token.category == VBP}{Token.category == TO}

   | {Token.category == RB}{Token.category == VB}
   | {Token.category == RB}{Token.category == VBP}
   | {Token.category == RB}{Token.category == VBG}

   | {Token.category == VB, Token.string != allow, Token.string != prevent , Token.string
       != notify , Token.string != help , Token.string != be}
   | {Token.category == VBP}
   | {Token.category == VBG}
   | {Token.category == VB}
   ):action

)
:ann
-->
 :action.Action = {type = "Action", string = :action@string}





```

C.4 Theme

Listing C.4: Theme.jape

```
Phase: Theme
Input: NP Action Token
Options: control = first

Rule: rule1
(
  {Action}
  ({Token})(0,3)
  ({NP} ):Theme
)
:ann
-->
 :Theme.Theme = {type = "Theme", string = :Theme@string}

```

C.5 Fit-Criteria

Listing C.5: Quantification.jape

```
Phase: Quantification
Input: NP Lookup Measurement Token Percent Number
Options: control = applet

```

```
Rule: rule1
(
        ({Lookup.majorType == Quantification})?
        (({Measurement}) |
        ({Token.kind == number}{Token.category == JJ}{Token.category == NNS})|
        ({Token.kind == number}{Token.category == NNS})|
        ({Number} {Token.category == NNS}) )
)
:Quan
-->
:Quan.Quantification = {type = "Quantification", string = :Quan@string, majorType="units"}

Rule: rule2
(
  ({Token.kind == number}{Token.string == x}{Token.kind == number}{Token.string == x}{
      Token.kind == number}) |
  ({Token.kind == number}{Token.string == x}{Token.kind == number})
)
:screen
-->
:screen.Quantification = {type = "Quantification", string = :screen@string, majorType="
    Screen"}


Rule: rule3
(
        {NP}({Token.string == "."}({Token.kind == number} | {Token.kind == x} | {Token.
            kind == X} ))(1,4)
)
:Product
-->
:Product.Quantification = {type = "Quantification", string = :Product@string ,majorType="
    Product"}

Rule: rule4
(
        ({Token.string == between}{Measurement}{Token.string == and}{Measurement}) |
        ({Token.string == between}{NP}{Token.string == and}{NP}) |
        ({Token.string == between}{Token.kind == number}{Token.string == and}{Token.kind
            == number})
)
:between
-->
:between.Quantification = {type = "Quantification", string = :between@string ,majorType="
    between"}

Rule: rule5
(
        {Percent}
)
:percent
```

```
-->
:percent.Quantification = {type = "Quantification", string = :percent@string ,majorType="
    Percent"}


```

C.6 Condition

Listing C.6: Condition.jape

```
Phase:Condition
Input: Lookup Agent NP Token Theme Action Quantification Modality
Options: control = first

Rule: rule1
(
  ({Lookup.majorType == condition}):left
  ({Token})(1,9)
  ({Agent}):right
)
:ann
-->
{
  Node start = ((AnnotationSet) bindings.get("left")).firstNode();
  Node end   = ((AnnotationSet) bindings.get("right")).firstNode();

  FeatureMap features = Factory.newFeatureMap();
  features.put("type", "Condition");
  outputAS.add(start, end, "Condition", features);
}

Rule: rule2
(
  ({Modality})
  ({Action})?
  ({Theme})?
  ({Token})(0,5)

  ({Quantification}):left
    ({Token})(1,9)
  ({Token.kind == punctuation}):right

)
:ann
-->
{
  Node start = ((AnnotationSet) bindings.get("left")).firstNode();
  Node end   = ((AnnotationSet) bindings.get("right")).firstNode();

  FeatureMap features = Factory.newFeatureMap();
  features.put("type", "Condition");
  outputAS.add(start, end, "Condition", features);
}


Rule: rule3
(
  ({Modality})
  ({Action})?
  ({Theme})?
  ({Token})[0,5]
  ({Lookup.majorType == limit}):left
    ({Token})[1,9]
  ({Token.kind == punctuation}):right
)
:ann
-->
{
  Node start = ((AnnotationSet) bindings.get("left")).firstNode();
  Node end   = ((AnnotationSet) bindings.get("right")).firstNode();

  FeatureMap features = Factory.newFeatureMap();
  features.put("type", "Condition");
  outputAS.add(start, end, "Condition", features);
}
```

C.7 Instrument

Listing C.7: Instrument.jape

```
Phase: Instrument
Input: NP ThingToBeProcessed Token Action
Options: control = all

Rule: rule4
(
  (( {Token.string == using} |{Token.string == via} )
  ({NP}{Lookup.majorType == conjunction}{NP} | {NP} )):Instrument

)
:ann
-->
 :Instrument.Instrument = {type = "Instrument", string = :Instrument@string}

```

Listing C.8: How.jape

```
Phase: how
Input: NP ThingToBeProcessed Token VG
Options: control = all

Rule: rule4
(
  {ThingToBeProcessed}
  {Token.string == by}

  {VG}({NP}):how

)
:ann
-->
 :how.How = {type = "How", string = :how@string}

```

C.8 Goal

Listing C.9: Goal.jape

```
Phase: Goal
Input: NP Theme Token VG Action Condition Sentence
Options: control = applet

Rule: rule4
(
  ({Action})
  ({Theme})?
  ({Condition})?
  ({Token})[0,3]
  (
  ((({Token.string == in}{Token.string == order}{Token.string == to})|({Token.string == to
      })({Token.category == VB} |({Token.string == for})| {VG} |{Token.category == VBP} |
      {Token.category == VBN} | {Token.category == JJ}))
  ({Token})[0,8]
  ) :Goal

  {Sentence}): sent
)
:ann
-->
 :Goal.Goal = {type = "Goal", string = :Goal@string}

```

C.9 Non-Testability Detection

Listing C.10: Testable.jape

```
Phase: Testable
Input: Sentence Quantification
Options: control = first

Rule: rule1
(
  ({Sentence contains Quantification}): testYES
)
:ann
-->
 :testYES.Requirement = {Testability = "YES"}



Rule: rule2
(
  ({Sentence}): testNO
)
:ann
-->
 :testNO.Requirement = {Testability = "NO"}
```

Appendix D

OwlExporter

```
/*
OwlExporter -- http://www.semanticsoftware.info/owlexporter

This file is part of the OwlExporter architecture.

Copyright (C) 2009, 2010 Semantic Software Lab, http://www.semanticsoftware.info
         Rene Witte
         Ninus Khamis

The OwlExporter architecture is free software: you can
redistribute and/or modify it under the terms of the GNU Affero General
Public License as published by the Free Software Foundation, either
version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

Phase: mention_map_domain_entities
Input: Document Sentence
Options: control = all  debug = true

Rule: mention_map_domain_entities
(
         {Sentence}
)
:ann
-->
{
        try     {
                AnnotationSet as = (gate.AnnotationSet)bindings.get("ann");
                Annotation ann = (gate.Annotation)as.iterator().next();
                FeatureMap features = ann.getFeatures();
                String in = doc.getContent().getContent(ann.getStartNode().getOffset(),
                    ann.getEndNode().getOffset()).toString();

                if(ann.getFeatures().get("functional_requirement").toString().
                    compareToIgnoreCase("yes") == 0) {
                //      features.put("className", "FunctionalRequirement");
                        features.put("className", "Person");
                        features.put("instanceName", in);
                        features.put("representationId", ann.getId());
                features.put("corefChain", null);
                        features.put("kind", "Class");
                //      outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain"
                    , features);
                }
                if(ann.getFeatures().get("constraint").toString().compareToIgnoreCase("yes
                    ") == 0) {
                        features.put("className", "OperatingConstraint");
                        features.put("instanceName", in);
                        features.put("representationId", ann.getId());
                features.put("corefChain", null);
                        features.put("kind", "Class");
                //      outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain"
                    , features);
                }
                if(ann.getFeatures().get("maintainability").toString().compareToIgnoreCase
                    ("yes") == 0) {
                        features.put("className", "Maintainability");
                        features.put("instanceName", in);
                        features.put("representationId", ann.getId());
                features.put("corefChain", null);
                        features.put("kind", "Class");
                //      outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain"
                    , features);
                }
                if(ann.getFeatures().get("reliability").toString().compareToIgnoreCase("
                    yes") == 0) {
                        features.put("className", "Reliability");
                        features.put("instanceName", in);
                        features.put("representationId", ann.getId());
                features.put("corefChain", null);
                        features.put("kind", "Class");
                //      outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain"
                    , features);
                }
                if(ann.getFeatures().get("security").toString().compareToIgnoreCase("yes")
                     == 0) {
                        features.put("className", "Security");
                        features.put("instanceName", in);
                        features.put("representationId", ann.getId());
                features.put("corefChain", null);
                        features.put("kind", "Class");
                //      outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain"
                    , features);
                }
                if(ann.getFeatures().get("usability/utility").toString().
                    compareToIgnoreCase("yes") == 0) {
                        features.put("className", "Usability");
                        features.put("instanceName", in);
                        features.put("representationId", ann.getId());
                features.put("corefChain", null);
                        features.put("kind", "Class");
                //      outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain"
                    , features);
                }
                if(ann.getFeatures().get("functionality").toString().compareToIgnoreCase("
                    yes") == 0) {
                        features.put("className", "Functionality");
                        features.put("instanceName", in);
                        features.put("representationId", ann.getId());
                features.put("corefChain", null);
                        features.put("kind", "Class");
                //      outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain"
                    , features);
                }
                if(ann.getFeatures().get("efficiency").toString().compareToIgnoreCase("yes
                    ") == 0) {
                        features.put("className", "Efficiency");
                        features.put("instanceName", in);
                        features.put("representationId", ann.getId());
                features.put("corefChain", null);
                        features.put("kind", "Class");
                //      outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain"
                    , features);
                }
        }
        catch(Exception e){
                e.printStackTrace();
        }
}
```