UNIVERSITÀ
DEGLI STUDI
FIRENZE

Università degli Studi di Firenze
Dipartimento di Ingegneria dell'Informazione (DINFO)
Corso di Dottorato in Ingegneria dell'Informazione
Curriculum: Informatica

Detecting Quality Defects: Methods to Improve Product Lifecycle Cost-effectiveness in the Railway Domain

Candidate
Gloria Gori

Supervisor
Prof. Alessandro Fantechi

PhD Coordinator
Prof. Luigi Chisci

Ciclo XXX, 2014-2017

Università degli Studi di Firenze, Dipartimento di Ingegneria dell'Informazione (DINFO).

To my family
you make my days brighter

Acknowledgments

Alessandro Fantechi · Alessio Ferrari · Stefania Gnesi
Benedetta Rosadini, Jacopo Trotta and Stefano Bacherini
Daniel M. Berry · Fabiano Dalpiaz and Paola Spoletini · Laura Carnevali
Dario D'Amico and his wife Carolin · Tiberio Uricchio
Francesco Orsini and his wife Yasamin
Alba Pennisi, the (ex) ENS Team and all my ex-Delta Team colleagues
Mom and Dad · Martina
My husband Marco and my three little ones
Carlo and Gabriella · Stefano and Daria

This thesis would not be possible without the help and the support of many people. First of all I would like to acknowledge the efforts and inputs of my supervisor, Prof. Alessandro Fantechi, and of Alessio Ferrari of the ISTI-CNR who were of great help during my research.

I thank Alstom Ferroviaria S.p.A. especially Benedetta Rosadini (a friend beyond work experience), Jacopo Trotta, Stefano Bacherini for their support. I thank Stefania Gnesi of ISTI-CNR who collaborated on the main parts of my research work.

I would like to thank Daniel M. Berry for kindly providing the dictionaries used by SREE.

Tnanks to Fabiano Dalpiaz and Paola Spoletini for their revision work on this thesis.

Thanks to Laura Carnevali for her support.

A big thank for his friendship and support to my long-standing friend Dario and his wife Carolin.

A big thank for their support to my old friends found again Tiberio Uricchio (my "PhD Counselor"... thank you so much for your support and numerous hints) and Francesco Orsini (I am so happy that you're back to Italy).

Many many thanks to my ex-colleagues and friends from GETS especially Alba Pennisi who strongly encouraged me to begin, continue and finish this PhD course, Laura Padovani, Francesca Vezzosi, Gianluca Magnani, Simone Lunardo, Alessandro Felleca and all my ex-Delta Team colleagues. I miss you, guys!

Thanks from the deep of my heart to my family; without you no PhD would be possible to me. Thanks to my mom, my dad (you read this from Heaven), my sister Martina; thanks to my husband Marco and to my three wonderful-enjoyable-neverendingenergy children Gabriele, Massimiliano and Matteo Giacomo. We didn't get bored in the last three years; thanks to my father-in-law Carlo, Gabriella, Stefano and Daria for their support.

Every morning in the Academia savannah one PhD candidate wakes up and knows that she needs time to study, research, write down papers, review papers, scan papers, answer to useful and less useful emails; every morning in the Home savannah one mom-of-three wakes up and knows that she has to run faster than a PhD candidate, but the most she knows that time won't be enough for anything else.... you can be a PhD candidate or a mom, but you know that time is all you need. Things are much more complicated when you are both mom and PhD candidate.

Abstract

The increased complexity and ubiquity of cyber-physical systems in recent times demands for more efficient and cost effective techniques to analyze software and hardware correctness, as well as to assess their performance at a given time in the future. Two disciplines that deal with these aspects of system development are *verification* and *performance evaluation*. During this thesis work we focused in methods for improving quality in both of these areas in the context of railway safety-critical domain.

Verifying a system means to prove or disprove that the system is the correct implementation of a *specification*, often expressed as a collection of *properties* – the Requirements – written in a given *language*. In the railway safety-critical domain the requirements play a key role in the product lifecycle as the system is developed and verified according to them; they are often expressed in natural language – which is flexible, but inherently ambiguous – albeit the strong needs of clearness and precision of the context. The requirements have to abide to strict quality criteria and the requirement review is therefore a very important activity to indentify quality defects and it is traditionally performed manually. Rule-based natural language processing (NLP) techniques have been developed to automatically perform this task. However, the literature is lacking empirical studies on the application of these techniques in industrial settings. This thesis mainly focuses on investigating to which extent NLP can be practically applied to detect defects in the requirements documents of a railway signalling manufacturer. The contribution is in carrying out one of the first works in which NLP techniques for defect detection are applied on a large set of industrial requirements annotated by domain experts. We contribute with a comparison between traditional manual techniques used in industry for requirements analysis, and analysis performed with NLP. Our experience shows that several discrepancies can be observed between the two approaches. The analysis of the

discrepancies offers hints to improve the capabilities of NLP techniques with company specific solutions, and suggests that also company practices need to be modified to effectively exploit NLP tools.

For what concerns the *performance evaluation* area we had the opportunity to focus on the system availability in the context of a different project of the laboratory. With the increased city population, the integration of public and private transport flows introduces new challenges, especially in urban transport. As it is often the case in scientific and engineering problems, the object of study is a *model* of the system, rather than the system itself. We provide one modeling and analysis method using stochastic Time Petri Nets for those city intersections where public and private transport flows integration is often cause of traffic congestion leading to train delays and even run deletion. The use of the STPN instead of simulation techniques provides a more effective way to set timing for traffic lights and train timetables in order to improve system availability.

Contents

Introduction

The objective

The increased complexity and ubiquity of cyber-physical systems in recent times demand more efficient and cost effective techniques to analyze software and hardware correctness, as well as to assess their performance at a given time in the future. Two disciplines that deal with these aspects of system development are *verification* and *performance evaluation*. During this thesis work we focused in methods for improving quality in both of these areas in the context of railway safety-critical domain.

Verification

For what concerns the *verification*, the author cooperated with Alstom Ferroviaria S.p.A., a major manufacturer in the railway domain, and with the Formal Methods and Tools Lab of the ISTI institute of CNR.

Context and motivation

Verifying a system means to prove or disprove that the system is the correct implementation of a *specification*, often expressed as a collection of *properties* – the Requirements – written in a given *language*. In the railway safety-critical domain the requirements play a key role in the product lifecycle as the system is developed and verified according to them. Albeit the strong needs of clearness and precision of the context, they are often expressed in natural language [37, 87] – which is flexible, yet inherently ambiguous – and they are progressively refined along the development process. All the requirement documents have to abide to strict quality criteria and the requirement review is therefore a crucial activity to identify quality defects and it is traditionally

performed manually, thus it is time consuming and error prone. Rule-based Natural Language Processing (NLP) techniques [117, 12, 62, 61, 111, 6, 46] have been developed to automatically perform this task. However, the literature is lacking empirical studies on the application of these techniques in industrial settings.

Goal and contribution

Our goal consisted in investigating to which extent NLP can be practically applied to detect defects in the requirements documents of a railway signalling system.
To address it we first identified a set of typical defects classes and, for each class, an engineer of the company implemented a set of defect-detection patterns by means of the GATE tool for text processing [35]. After a preliminary analysis, we applied the patterns to a large set of 1866 requirements previously annotated for defects. The output of the patterns was further inspected by two domain experts to check the false positive cases. Additional discard-patterns were defined to automatically remove these cases. Finally, SREE [111], a tool that searches for typically ambiguous terms, was applied to the requirements. The experiments show that SREE and our patterns may play complementary roles in the detection of requirements defects. We applied NLP techniques for defect detection on a large set of industrial requirements annotated by domain experts. The contribution consists in a comparison between traditional manual techniques used in industry for requirements analysis, and analysis performed with NLP. Our experience tells that several discrepancies can be observed between the two approaches. The analysis of the discrepancies offers hints to improve the capabilities of NLP techniques with company specific solutions, and suggests that also company practices need to be modified to actively exploit NLP tools.

Performance evaluation

For what concerns the *performance evaluation*, the author had the opportunity to focus on a research application of the system availability attribute in the context of a different project (funded by the Fondazione Cassa di Risparmio di Firenze) of the laboratory.

Context and motivation

With the increased city population, the integration of public and private transport flows introduces new challenges; Intelligent Transportation Systems (ITS) for urban mobility aim at the grand objective of reducing environmental impact and minimize urban congestion, also integrating different mobility modes and solutions [41, 1]. However, the different transportation modalities may end in a conflict due to physical constraints concerned with the urban structure itself: an example is the case of intersection between a public road and a tramway right-of-way, where traffic lights priority given to trams may trigger road congestion, while an intense car traffic can impact on trams' performance. These situations can be anticipated and avoided by accurately modeling and analyzing the possible congestion events. Typically, modeling tools provide simulation facilities, by which various scenarios can be played to understand the response of the intersection to different traffic loads. While supporting early verification of design choices, simulation encounters difficulties in the evaluation of rare events. Only modeling techniques and tools that support the analysis of the complete space of possible scenarios are able to find out such rare events [20, 14].

Goal and contribution

Our goal consisted in the implementation and evaluation of an analytical approach to model and evaluate a critical intersection for the Florence tramway, where frequent traffic blocks used to happen. Specifically, we exploited the ORIS tool to evaluate the probability of a traffic block, leveraging regenerative transient analysis based on the method of stochastic state classes to analyze a model of the intersection specified through Stochastic Time Petri Nets (STPNs). This experience shows that the frequency of tram rides impacts on the road congestion, and hence compensating measures (such as sychronizing the passage of trams in opposite directions on the road crossing) should be considered.

Thesis organization

The present thesis is organized as follows:

- Chapter 1 describes the context in which this thesis sits with an introduction on safety-critical systems and the description of norms and

product life cycle in the railway domain;

- Chapter 2 introduces the problem of ambiguity in natural language providing the evaluation measures used in the following chapters;
- Chapter 3 is dedicated to the defect detection approaches and after the literature review describes the rule-based approach used in our reported case-study to detect ambiguity in requirements expressed in natural language;
- Chapter 4 explains the research methodology and the case-study design, presenting the Research Questions and the adopted procedures;
- Chapter 5 describes the execution of the case study, focusing on the involved subjects, the used dataset, and the iterations performed in order to answer the Research Questions;
- Chapter 6 presents the results obtained through the case study execution and provides the answers to the Research Questions;
- Chapter 7 highlights the lessons learned and the return of experience, and presents the implication for practice and future work directions;
- Chapter 8 describes the problem of performance evaluation of public urban transport. The model of an intersection between public transport and private traffic is provided and analyzed by using Stochastic Time Petri Nets (STPNs);
- Chapter 9 concludes the thesis.

Chapter 1

Safety-Critical Systems and their Development Process

1.1 Introduction

Part of this thesis work has been carried out with the collaboration of Alstom, a leading railway signalling systems manufacturer. The author was involved in the development product life cycle of hardware and software products for real railway systems. This experience oriented the research work for the thesis creating a basis for the author's interest in the area of quality and cost-effectiveness improvement in the development life cycle of such systems.

The exercise of Safety-Critical systems involves a critical level of risk of exposure for people, environment and material assets to dangerous situations with the possibility of accidents due to malfunctions caused by errors or failures [25, 26, 24]. No system can be defined "absolutely safe", thus *safety* is *the absence of unacceptable levels of risk* [25], or even *the property of a system to not cause harm to human life or to the environment.* [104]. The choice of an appropriate risk management approach defines a reasonable probability for risks considered *acceptable* (THR, Tolerable Hazard Rate) in the operational conditions in which the system works. It is therefore important to ensure before commissioning that the probability of risky events caused by the system is lower than THR. The assessment that the system meets all the required conditions is carried out according to standards that, in the European railway domain, are provided by CENELEC (Comité Européen de Normalization en Électronique et en Électrotechnique). These standards

have been also prescribed since 2002 by RFI (Rete Ferroviaria Italiana) as a reference for the safety certification of products and electronic systems in railway signalling. Some of these standards are listed below:

- **EN-50126 [25]**: Railway applications - The specification and demonstration of dependability, reliability, availability, maintainability and safety (RAMS);
- **EN-50129 [24]**: Railway applications - Safety related electronic systems;
- **EN-50128 [26]**: Railway applications - Software for railway control and protection systems;
- **EN-50159 [27]**: Railway applications - Communication, signalling and processing systems - Part 1: Safety related communication in closed transmission systems.

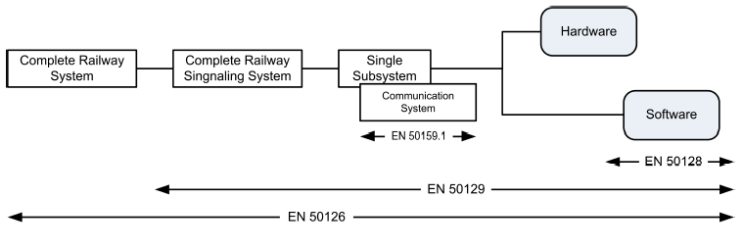

Figure 1.1: Scope of CENELEC norms.

Figure 1.1 shows the relations among the listed standards. The horizontal axis represents the progression from the most general to the most specific one, while the vertical axis represents the scope each standard has.

1.2 Dependability attributes

Dependability is defined as the property of a system to be usable by an human being, or a community, without the danger of unacceptable risks [80]. The development of safety-critical systems and their verification is oriented to fulfill the dependability requirements.

Dependability includes the following attributes:

- **Reliability:** the ability of a system to perform a required function under certain conditions and for a specified period of time;
- **Availability:** the ability of a system to perform a required function at a certain time or in a specified time interval, given the necessary resources;
- **Maintainability:** probability that for a given system unit during a certain time interval is carried out a given active maintenance activity, implemented through procedures and required means;
- **Safety:** absence of unacceptable levels of risk of harm.

These attributes are often referred with the acronym of RAMS (Reliability, Availability, Maintainability and Safety); they guide the design and implementation of the system and are used as a reference when evaluating the model itself.

The development process that aims at fulfilling each attribute at the requested level is composed by several phases shown in Figure 1.2 [91, 25].

Figure 1.3 shows the involved actors and their role in the process.

Reliability, Availability and Maintainability can be quantifiable by direct measurements while Safety is a subjective assessment that requires judgmental informations to be applied to give a level of confidence. In order to discuss the Safety attribute, we first need to specify what we mean for *risk*. A *risk* is strictly related to an *hazard* (i.e., an event that can lead to an *accident*). It is defined as the combination of the frequency of occurrence of the hazard and its severity. The frequency of occurrence, based on the event probability, is classified by levels, ranging from *Incredible* to *Frequent*. The hazard severity, based on the consequences for people and environment, is classified by levels, ranging from *Insignificant* to *Catastrophic*. The combination of frequency and severity levels generates a set of *risk classes*. To each risk class is associated an index, the Risk Class Index (RCI), ranging from *Negligible* to *Intolerable*. Table 1.1 shows an example of risk acceptance evaluation.

On the basis of the previous analysis, each part of the system is then classified on the basis of its criticality level. The parameter used is called Safety Integrity Level (SIL) which varies from 0 (for systems or subsystems

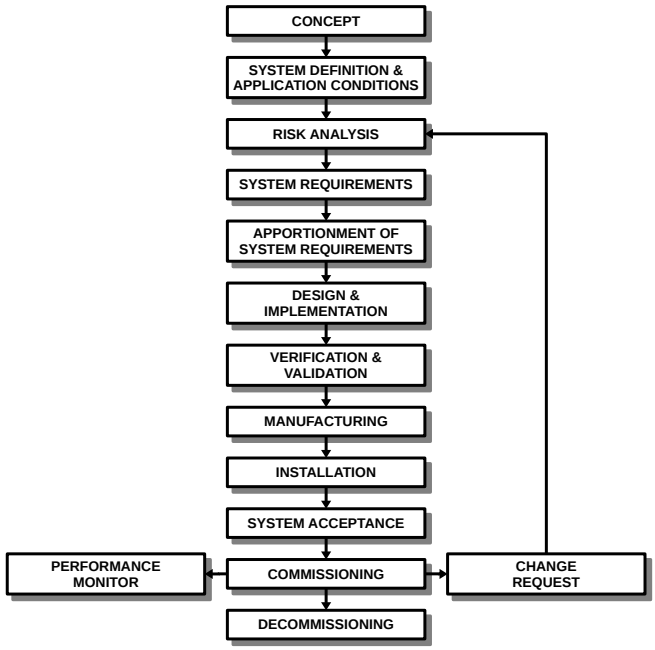

Figure 1.2: Development phases.

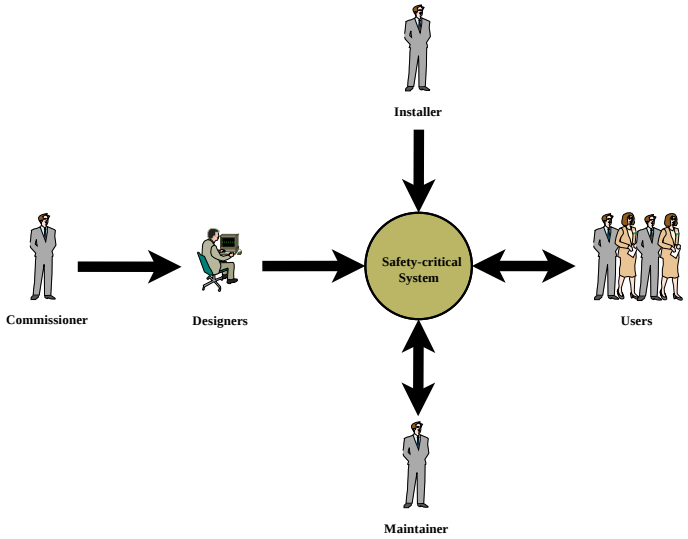

Figure 1.3: Safety actors.

with no critical features) to 4 (for systems or subsystems with high risk level). Each SIL class is associated with two factors:

- a range of values for the THR (see Table 1.2);
- a set of actions to be performed during the life cycle process.

1.3 The software development life cycle

Among the norms that are part of the CENELEC standards, EN-50128 is the one that specifies the procedures and technical requirements for the development of programmable electronic systems for the usage in railway control and protection applications. This norm applies within the scope of the software (e.g., firmware, operating systems, applications) and its interaction with the system. EN-50128 describes the software lifecycle from the specification, to the development and finally the verification and validation phases.

It also specifies some basic principles in the development of safe software as the ones listed below:

Frequency of occurrence of a hazardous event	Risk Levels			
Frequent	Undesiderable	Intolerable	Intolerable	Intolerable
Probable	Tolerable	Undesiderable	Intolerable	Intolerable
Occasional	Tolerable	Undesiderable	Undesiderable	Intolerable
Remote	Negligible	Tolerable	Undesiderable	Undesiderable
Improbable	Negligible	Negligible	Tolerable	Tolerable
Incredible	Negligible	Negligible	Negligible	Negligible
	Insignificant	**Marginal**	**Critical**	**Catastrophic**
	Severity Levels of Hazard Consequence			

Table 1.1: Example of risk acceptance evaluation.

THR	SIL
$10^{-9} \leq \text{THR} < 10^{-8}$	4
$10^{-8} \leq \text{THR} < 10^{-7}$	3
$10^{-7} \leq \text{THR} < 10^{-6}$	2
$10^{-6} \leq \text{THR} < 10^{-5}$	1
$10^{-5} \leq \text{THR}$	0

Table 1.2: Relation between THR and SIL defined in EN50129 [24].

- software must be developed using modular programming technique[1];
- verification activities performed at each stage of the development life cycle;
- all of the used libraries and modules must be verified;
- drawing up of clear documentation.

The phase of system verification is a key step and it is driven by requirements and it aims at demonstrating that the system is their correct implementation.

The norm recommends for the development process the use of the V Model (shown in Figure 1.4) [26].

The V Model is a graphical representation of the development lifecycle of a system. It is used to produce rigorous development lifecycle models and project management models.

[1]Modular programming is a software design technique that emphasizes separating the functionality of a programme into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

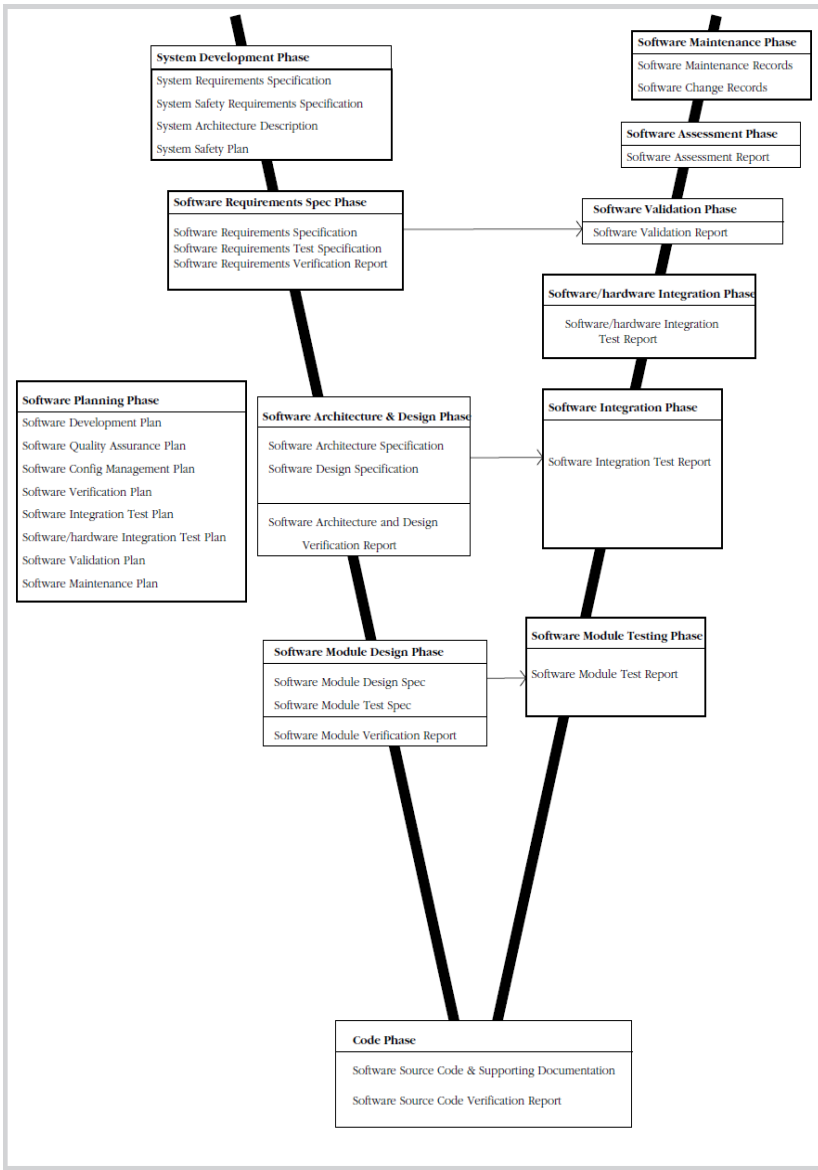

Figure 1.4: Development process using the V Model cycle.

The norm describes the activities to be performed at each step and the deliverables that have to be produced in output. The activities performed in all phases in the left side are verified in the corresponding phases placed at the same level in the opposite branch of the V. The left side of the V represents the decomposition of requirements, and creation of system specifications. The right side of the V represents integration of parts and their validation. However, Requirements need to be validated first against the higher level requirements or user needs. Below is a list and brief explanation of the represented phases in terms of the action taken by different roles played in the development cycle, namely the *Requirement Engineer*, the *Developer*, the *Verification Engineer*, the *Validation Engineer* and the *External Assessor*. According to EN-50128 [26], the roles have to be played by different actors for software rated at the highest SIL.

Software Requirements Specification Phase: inputs of this phase are the *System Requirements Specification, System Safety Requirements Specification, System Architecture Description* and *System Safety Plan*. During this phase the following activities are performed:

- The *Requirement Engineer* provides *Software Requirements Specification* that shall describe the system as a single entity, formalizing the macro-level actions and features that it shall provide and the constraints that it shall guarantee;

- The *Verification Engineer* takes as input *Software Requirements Specification* and provides the Test Cases, in the form of *Software Requirements Test Specification*, needed to verify the software at the corresponding level of detail;

- The *Validation Engineer* takes as input all artifacts mentioned in the current step, validates that the *Software Requirements Specification* has taken into account all requirements defined at higher level, and validates that *Software Requirements Test Specification* correctly covers the *Software Requirements Specification* by producing the *Software Requirements Verification Report*.

Software Architecture and Design Phase: inputs of this phase are artifacts produced during the previous phase. During this phase the following activities are performed:

- The *Requirement Engineer* provides *Software Architecture Specification* that shall describe architectural blocks composing the system. Specific actions and features are specified for each block;
- The *Requirement Engineer* provides *Software Design Specification* as specification of each block that shall be designed in order to provide the requested functionalities and all those interfaces required for information propagation between different blocks. Different Requirement Engineers can be responsible for different phases;
- The *Verification Engineer* takes as input *Software Design Specification* and provides the Test Cases, in the form of *Software Design Test Specification*, needed to verify the software at the corresponding level of detail;
- The *Validation Engineer* takes as input all of the mentioned artifacts in the current step, validates that the *Software Architecture Specification* has taken into account all requirements defined in *Software Requirements Specification*, validates that the *Software Design Specification* has taken into account all requirements defined in *Software Architecture Specification* and validates that *Software Design Test Specification* correctly covers the *Software Design Specification* by producing the *Software Architecture and Design Verification Report*.

Software Component Design Phase: inputs of this phase are artifacts produced during the previous phase. During this phase the following activities are performed:

- The *Requirement Engineer* provides *Software Component Design Specification* as specification of each component that is derived from Architecture and Design Phase; a component is defined as a stand-alone compilation unit and thus corresponds to a single source code file;
- The *Verification Engineer* takes as input *Software Component Design Specification* and provides the Test Cases, in the form of *Software Component Test Specification*, needed to verify the software at the corresponding level of detail;
- The *Validation Engineer* takes as input all artifacts mentioned in the current step, validates that the *Software Component Design Specification* has taken into account all requirements defined in *Software Design*

Specification and validates that *Software Component Test Specification* correctly covers the *Software Component Design Specification* by producing the *Software Component Verification Report*.

Code Phase: inputs of this phase are artifacts produced during the previous phase. During this phase the following activities are performed:

- The *Developer* provides the implementation of the requested software system, or of a portion of it, according to the provided requirements, thus producing *Software Source Code and Supporting Documentation*;
- The *Verification Engineer* takes as input *Software Source Code and Supporting Documentation* and verifies the *Software Source Code* compliance against the planned Source Code verification activities and provides the *Software Source Code Verification Report*.

Software Component Testing Phase: inputs of this phase are artifacts produced during the previous phase and in *Software Component Design Phase*. During this phase the following activity is performed:

- The *Verification Engineer* executes all Test Cases contained in *Software Component Test Specification* and produces the *Software Component Test Report*.

Software Integration Phase: inputs of this phase are artifacts produced during the previous phase and in *Software Architecture and Design Phase*. During this phase the following activity is performed:

- The *Verification Engineer* verifies the capability of the *Software Source Code* to execute on the System Hardware and produces the *Software Integration Test Report*.

Software/Hardware Integration Phase: inputs of this phase are artifacts produced during the previous phase. During this phase the following activity is performed:

- The *Verification Engineer* executes all Test Cases contained in *Software Design Test Specification* and produces *Software/Hardware Integration Test Report*.

Software Validation Phase: inputs of this phase are artifacts produced during the previous phase and in *Software Requirement Specification Phase*. During this phase the following activity is performed:

- The *Validation Engineer* validates that all test cases have been executed by *Verification Engineer* and that they all had positive result, thus providing the *Software Validation Report*.

Software Assessment Phase: inputs of this phase are artifacts produced during all the phases. During this phase the following activity is performed:

- The *External Assessor*, upon revision of all produced artifacts, certifies that the Software has been developed according to what is required by the applicable CENELEC norms and provides the *Software Assessment Report*.

Software Maintenance Phase: during this phase the following activities are performed:

- The *Software Maintenance Records* are persistenly recorded each time a maintenance operation is performed;

- The *Software Change Records* are persistenly recorded each time a Change Request (CR) is issued, both to implement a new functionality (enhancement CR) and to correct an issue (defect CR).

One crucial activity is the verification that one requirement document has taken into account all requirements defined in the previous phase. In order to provide a standard method to perform this activity the norm requires traceability matrices between adjacent and corresponding phases. There is one traceability matrix for each couple of phases (adjacent and corresponding). Each matrix contains all the requirements of the specification traced to their refinements in the lower specification phase and viceversa. This activity is extremely useful in order to check that all the features described in the high level requirements have been detailed in the following development phases and that all requirements have been tested in the corresponding phase of the V right branch.

Chapter 2

Requirements expressed in Natural Language and Ambiguity

2.1 Introduction

Requirements Engineering (RE) refers to the process of defining, documenting and maintaining requirements in the software development process (described in Section 1.3). Requirements define constraints on the system that need to be expressed in a form that is suitable for analysis, communication and subsequent implementation of the system. Furthermore, the number of requirements grows phase after phase in the descendent branch of the V cycle, thus a defective requirement has a higher impact when it refers to a high level phase. The traceability matrices are a useful tool to evaluate the impact of every requirement on the lifecycle. The first step to deliver the desired quality product is the full understanding of the customer needs and their documentation in a clear, complete and concise way.

As stated by Mich in her survey [87] and by Méndez et al. in the initiative NaPIRE [37], requirements are usually expressed in Natural Language, which is inherently ambiguous. An ambiguous specification can lead to two or more implementors writing interfacing code to operate under different assumptions, despite each programmer's confidence that he has programmed the correct behavior.

We may think that, being rational, human beings can usually overcome

miscommunication caused by ambiguity by using analysis, but this is not always true in an absolute way. Ambiguity is the characteristic of having more than one possible interpretation.

Figure 2.1: Example of ambiguity: what is it?

To better explain this concept we refer to Figure 2.1, which is not clear at first sight: it may have multiple interpretations and even if you argue the correct one, you may still have some doubts. With a proper explanation, Figure 2.1 becomes immediately clear, and it seems impossible that a few seconds before it wasn't.[1]

The norms listed in Section 1.1 help to create a process that ensures system safety even in presence of defective requirements. In particular, CENELEC EN-50128 asks requirements documents for railway systems to be *complete, clear, precise, unequivocal, verifiable, testable, maintainable, and feasible* – clause 7.2.4.4 of the norm [26]. To ensure that these quality attributes are met, companies developing railway products have a Verification Engineer (VE) who reviews for defects any requirements document produced during the development process. This review activity is time consuming and error prone, and an automated review assistant might help VEs in their task.

[1] Figure 2.2

NLP (Natural Language Processing) refers to all applications aiming at processing the natural language by means of a computer. While it is quite challenging to make a tool able to understand the human creativity, it is true that machines are faster and more precise than humans: NLP techniques use these strengths to achieve the best result.

Section 2.2 contains a brief summary of the literature on this topic, Section 2.3 contains the description of different ambiguity types, Section 2.4 describes some criteria to evaluate NLP techniques performances.

2.2 Literature review

NLP techniques have been largely applied to automate several requirements engineering tasks, including model synthesis [99], classification of requirements into functional/non-functional categories [23], classification of online product reviews [82], traceability [107, 33], detection of equivalent requirements [44], completeness evaluation [49], information extraction [56, 97, 81], ambiguity detection [111, 12], and its generalization, defect detection. Since in this paper we focus on defect detection, we will discuss related works in this field. Techniques developed to address the problem of defects in written requirements can be broadly partitioned into two sets. The first set of techniques suggests to use constrained NL or formal/semi-formal languages to prevent or limit defects. The second set of techniques starts from unconstrained NL and generally aims at detecting defects, either by means of manual verification, or by means of automated tools.

2.2.1 Preventing and limiting defects

In the literature, several strategies were defined to *prevent* defects by means of constrained natural languages [85, 94] or (semi-)formal approaches [87, 4, 76, 59].

Concerning the use of constrained natural languages, the EARS [85] and the Ruppâ template [94] are well known constrained formats for editing requirements. Arora et al. [6] defined an approach to check the conformance of requirements to these templates. Although the adoption of constrained natural languages is not widespread in industry, recent studies have shown that templates can be proficiently used by domain experts [86]. On the other hand, templates can limit the amount of requirements defects at the syntactic

level, but linguistic defects may still be present at the lexical, semantic and pragmatic levels. Addressing these defects requires other techniques [6].

Among the works on (semi-)formal approaches, one of the earlier contributions with a focus on defect prevention is the tool LOLITA [87], which implements an approach for translating NL requirements into object-oriented models. Similarly, Circe-Cico [4], starts from NL requirements to generate models to support requirements analysis. Logic as a tool to identify and analyze inconsistency in requirements from multiple stakeholders is suggested in [126, 59]. More specifically, these papers propose a tool, named CARL, that automatically translates NL into logic and then uses theorem proving and model checking to detect inconsistency in the requirements. The works of Kof aim to semi-automatically formalise NL requirements into message sequence charts [74] and automata [75]. More recently, Yue et al. [122] proposed a method and a tool, called *aToucan*, to automatically generate a UML software analysis model from textual, functional requirements specifications expressed in the form of use cases. A systematic study of defects in use case specifications expressed in restricted NL is presented in [124].

The idea behind the works on (semi-)formal approaches is that the formalisation process may help in identifying requirements defects, since errors in requirements would lead to inconsistencies or omissions in models, and, due to the more formal nature of models, defects are easier to detect in models than in textual requirements. However, through an analysis of two empirical studies, Kamsties [69] concludes that formalization does not help to eliminate defects from informal requirements documents. Indeed, during the formalization process the analyst makes implicit assumptions, transforming defects into errors. Therefore, even when formal modelling is applied, other techniques for defect defection shall be used as a complement.

2.2.2 Detecting defects

Approaches for defect *detection* can be categorised into manual approaches and automated ones, mostly based on NLP. Early and successful techniques for manual requirements inspection were provided in [43, 106]. Inspection checklists were developed, among others, in [5, 70], while a survey on the topic of requirements inspection was published in [7].

Automated NLP approaches for defect detection can be be categorised into those that use rule-based techniques [117, 12, 62, 61, 111, 6, 46] and those that leverage artificial intelligence techniques [28, 118, 50]. Our contribution

falls into the first category, which collects all the works in which defects are identified based on linguistic patterns.

The Ambiguity Handbook of Berry et al. [12] includes one of the most influential classification of ambiguity-related defects in requirements, and provides a large set of examples of typically dangerous words and constructions (see Section 2.3). Wilson et al. [117] define a quality model composed of quality attributes and quality indicators, and develop an automatic tool (called ARM: Automated Requirement Measurement) to perform the analysis against the quality model aiming to detect defects and to collect metrics. The tool was applied to industrial requirements from NASA [101]. Gnesi et al. [62, 42] present QuARS, a tool for defect detection based on a quality model developed by the authors. Similarly, [61] implemented a grep-like, pattern-based technique to detect defects, supported by statistical NLP techniques such as POS tagging. Kiyavitskaya et al. [73] propose a two-step approach to identify ambiguities in NL requirements. In the first step, a tool applies a set of ambiguity measures to the requirements, in order to identify potentially ambiguous sentences. In the second step, a (manually simulated) tool shows the specific parts that are potentially ambiguous in the set of sentences identified. Tjong et al. [111] developed SREE, a tool that identifies defects based on a pre-defined list of dangerous terms. Arora et al. [6] use patterns of linguistic defects as the other works, and, in addition, checks the conformance of the requirements to a given template.

Among the works that use artificial intelligence techniques, Chantree et al. [28] present a technique that helps requirements analysts to identify so-called innocuous ambiguities (i.e., linguistic ambiguities that have a single reading in practice). The focus of this work is on *coordination* ambiguities (i.e., due to the usage of coordinating conjunctions), and a set of heuristics, developed according to a data-set built by human assessors, is presented to discriminate between innocuous and nocuous ambiguities. This approach was extended for *anaphoric* ambiguities (i.e., due to the usage of pronouns) in [118]. Finally, Ferrari et al. [50] propose a graph-based technique to detect *pragmatic* ambiguities (i.e., ambiguities that depend on the context) in NL requirements defined for a specific application domain.

All these works, and in particular the ones employing rule-based techniques, were used as fundamental references to define the defect detection patterns of our study. On the other hand, all the listed works provide limited validation in real industrial contexts, as noted also in [46]. Large data-sets

annotated by experts were considered in [44]. However, their focus is solely on redundancy defects (i.e., equivalent requirements), detected by means of information retrieval techniques. The task of finding couples of equivalent requirements is radically different from the one we are dealing with in our study, in which multiple linguistic defects occurring in single requirements are considered. To our knowledge, the more general industrial work on defect detection is the one presented in [46], who experimented their tool named *Smella* on several datasets provided by three companies. Although domain experts were interviewed to assess the effectiveness of the tool, analysis of the results was performed by two researchers. Another industrial case study on defect detection was presented in [116]. Two datasets of 293 requirements in total were used as a benchmark, and term-based defect detection techniques were employed to detect ambiguities. The results were reviewed by domain experts.

The research results reported in the following chapters of this thesis (published in [100, 51]) contributes to the recent literature on the industrial application of defect detection NLP techniques [46, 116]. Compared to the other studies, in our work the techniques are implemented, tailored, and validated by domain experts. Furthermore, this is the first work that shows how rule-based NLP patterns for defect detection can be *incrementally tuned* to the needs of a company, to address the systematic – and domain-dependent – false positive cases typically raised by these techniques.

2.3 Ambiguity taxonomy

When we read a text written in natural language as rational and intelligent human beings we use our analysis capabilities to understand it. Usually it is easy as we do that in our everyday life, but it may happen that some words or phrases arise doubts which we need to solve using our background or knowledge of the context. This kind of doubts or questions are known as **ambiguity**. It exists in all disciplines where communication is based on natural language: writing, linguistic, philosophy, law, requirement engineering. Berry notices that the word *ambiguity* is ambiguous itself as it suggests to refer to something with only two possible interpretations while it would have even more; a better word it would be *multiguity* [12]. We can distinguish four broad classes of linguistic ambiguity [61, 12]:

- **Lexical ambiguity** (Subsection 2.3.1);

- **Syntactic ambiguity** (Subsection 2.3.2);
- **Semantic ambiguity** (Subsection 2.3.3);
- **Pragmatic ambiguity** (Subsection 2.3.4).

2.3.1 Lexical ambiguity

A lexical ambiguity occurs when a term has multiple meanings. It can happen in the following cases:

- *Homonymy:* Occurs when two different words are spelled the same way, but they have unrelated meanings and sometimes also different etymologies. Examples of homonyms are the pair *stalk* (part of a plant) and *stalk* (follow/harass a person) and the pair *left* (past tense of leave) and *left* (opposite of right), *bank* (estabilishment for custody, loan, exchange, or issue of money) and *bank* (in the sense of rising ground bordering a lake, river, or sea). A distinction is sometimes made between "true" homonyms, which are unrelated in origin, such as *skate* (glide on ice) and *skate* (the fish), and polysemous homonyms, or polysemes, which have a shared origin, such as *mouth* (of a river) and *mouth* (of an animal);
- *Polysemy:* Occurs when a word has several related meanings but one single etymology. For example:
  - Man:
    1. The human species (i.e., man vs. other organisms);
    2. Males of the human species (i.e., man vs. woman);
    3. Adult males of the human species (i.e., man vs. boy).

  This example shows the specific polysemy where the same word is used at different levels of a taxonomy. Example 1 contains 2, and 2 contains 3:

  - Mole:
    1. a small burrowing mammal;
    2. consequently, there are several different entities called moles. Although these refer to different things, their names derive from 1 (e.g., "A Mole burrows for information hoping to go undetected").

However, other senses of the word (i.e., the skin blemish, the breakwater, the unit of measure, and the Mexican sauce) are homonyms, not polysems, as they are each etymologically distinct.

- Book:
  1. a bound collection of pages;
  2. a text reproduced and distributed (thus, someone who has read the same text on a computer has read the same book as someone who had the actual paper volume);
  3. to make an action or event a matter of record (e.g., "Unable to book a hotel room, a man sneaked into a nearby private residence where police arrested him and later booked him for unlawful entry.").

2.3.2 Syntactic ambiguity

Syntactic ambiguity, also called structural ambiguity, occurs when a given sequence of words can be given more than one grammatical structure, and each has a different meaning. It can be distinguished as an analytical, attachment, coordination, or elliptical ambiguity:

- *Analytical ambiguity:* occurs when the role of the constituents within a phrase or sentence is ambiguous. Among the various patterns of analytical ambiguity that can occur is the structure of a complex noun group including modifier scope. For instance *"The Tibetan history teacher"* can be read as *"The (Tibetan history) teacher"*, *"The Tibetan (history teacher)"*;

- *Attachment ambiguity:* occurs when a particular syntactic constituent of a sentence, such as a prepositional phrase or a relative clause, can be legally attached to two parts of a sentence. The most popular pattern of attachment ambiguity is a prepositional phrase that may modify either a verb or a noun. For example in the phrase *"The police shot the rioters with guns"* we can intend *with guns* either as a modifier of the noun *rioters* or as a modifier of the verb *shot* leading to two different interpretations;

- *Coordination ambiguity:* occurs in two conditions:

1. when more than one conjunction, *and* or *or* is used in a sentence (e.g., *"I saw Peter and Paul and Mary saw me"*);

2. when one conjunction is used with a modifier (e.g., *"Young man and woman"*).

- *Elliptical ambiguity:* occurs when it is not certain whether or not a sentence contains an ellipsis. An example is: *"Perot knows a richer man than Trump"*. It has two meanings, that Perot knows a man who is richer than Trump is and that Perot knows a man who is richer than any man Trump knows (ellipsis present).

2.3.3 Semantic ambiguity

Semantic ambiguity occurs when a sentence has more than one way of reading it within its context although it contains no lexical or structural ambiguity. Semantic ambiguity can be viewed as ambiguity with respect to the logical form, usually expressed in predicate logic, of a sentence. Semantic ambiguity can be caused by: coordination ambiguity, referential ambiguity, and scope ambiguity. Coordination ambiguity is already discussed. Referential ambiguity is on the borderline between semantic and pragmatic ambiguity, because referential ambiguity can happen within a sentence or between a sentence and its discourse context. Thus it is discussed in Subsection 2.3.4 The quantifier operators include such words as every, each, all, some, several, a, etc., and the negation operators include not.

Scope ambiguity occurs when these operators can enter into different scoping relations with other sentence constituents. For example, in the sentence *"All linguists prefer a theory"*, the quantifiers *all* and *a* interact in two ways. When the scope of *a* includes the scope of *all*, this sentence means all linguists love the same one theory. When the scope of *all* includes the scope of *a*, this sentence means that each linguist loves a, perhaps different, theory. The same thing can happen with negations and quantifiers which can interact ambiguously. For example, the sentence *"No one has seen a pig with wings"*. It can be read as saying either that there exists no pig with wings or that there exists a mythical pig with wings that no one has ever seen.

2.3.4 Pragmatic ambiguity

Pragmatics, like semantics, investigates the meaning of language; pragmatics focuses on context-dependent meaning, while semantics on context-invariant meaning.

Pragmatic ambiguity occurs when a sentence has several meanings in the context in which it is located. The context comprises for example the sentences before and after, and the domain beyond the language (i.e., the situation, the background knowledge, the domain specific language). We distinguish referential ambiguity and deictic ambiguity. In traditional semantics, the relation between a word or phrase and the object of the real world that the word or phrase describes is called a reference. An anaphor is an element of a sentence that depends for its reference on the reference of another element, possibly of a different sentence. This other element is called the antecedent and must appear earlier in the same sentence or in a previous sentence. A referential ambiguity occurs when an anaphor can take its reference from more than one element, each playing the role of the antecedent. Anaphora include pronouns (e.g., it, they), definite noun phrases, and some forms of ellipsis. An example of a referential ambiguity is: *"The trucks shall treat the roads before they freeze"*. Ellipses can have the same effect as pronouns and definite nouns, as the following sentence shows. *"... If the ATM accepts the card, the user enters the PIN. If not, the card is rejected"*. The word *not* is here an elliptical expression that refers either to the condition specified in the previous sentence or to something written before that. Deictic ambiguity occurs when pronouns, time and place adverbs, such as now and here, and other grammatical features, such as tense, have more than one reference point in the context. The context includes a person in a conversation, a particular location, a particular instance of time, or an expression in a previous or following sentence.

2.4 NLP techniques evaluation

In order to quantitatively evaluate the performance of NLP techniques for defect detection an evaluation criterion has to be defined. These NLP techniques, when used to detect one precise defect typology, share some similarities with *binary classifiers*, in fact:

- these techniques highlight a part of a sentence that has been detected

as defective, which is very similar to the *predicted condition positive* of a binary classifier;

- sentences – or fractions of them – not highlighted by these techniques are considered as non-defective, which is very similar to the *predicted condition negative*.

Due to these similarities, evaluation criteria applicable to binary classifiers can be applied to the NLP technologies too. In order to measure the effectiveness, we define the following quantities:

- tp: number of true positive cases. The number of requirements that contain at least a defect – *condition positive* – and correctly got *predicted condition positive*;
- fp: number of false positive cases. The number of requirements fragments not containing any defect – *condition negative* – but got *predicted condition positive*;
- fn: the number of false negative cases. The number of requirements that contain at least a defect – *condition positive* – but got *predicted condition negative*;
- tn: number of true negative cases. The number of requirements that don't contain any defect – *condition negative* – and correctly got *predicted condition negative*.

Based on these definitions, the measures of accuracy (a), precision (p) and recall (r) are defined as [96]:

$$a = \frac{tp + fp}{tp + fp + tn + fn} \quad p = \frac{tp}{tp + fp} \quad r = \frac{tp}{tp + fn}$$

We chose to use as evaluation criteria precision p and recall r. The precision is negatively influenced by the amount of defects wrongly identified (fp). The recall is negatively influenced by the amount of undetected defects (fn). The same evaluation criteria can be used, with the proper extensions described by Powers [96], also in the general case of *multi-class classifiers*.

The above described quantitative evaluation of the performance requires an additional and non-trivial effort: in order to correctly compute the quantities tp, fp, fn and tn the condition positive and negative shall be known a priori. This knowledge in *classification* and *machine learning* problems is

known as *labeling* or *ground truth*, and it consists in the association of a *label* to each example composing the dataset used for training and test purposes.

In general the labeling (or annotation) process is conducted manually, thus it is time consuming and error-prone. In the specific case of *Requirement Engineering*, the annotation process is subjective – due to the ambiguities in NL described in Section 2.3. In order to mitigate as much as possible the subjectivity of this operation we agreed that the labeling operation has to be performed by an actor with a proven knowledge of the industrial domain under consideration. We also agreed that, in order to further decrease the subjectivity impact, in some conditions the labeling operation has to be replicated by more than one actor. When using this technique we need to define a criterion to evaluate the *inter-rater agreement* and a criterion to resolve disagreements.

A proper criterion for inter-rater agreement evaluation is the Cohen's kappa coefficient (k) [79]. It measures the agreement between two raters who individually classify N items into C mutually exclusive categories and it is defined as:

$$k = 1 - \frac{1 - p_o}{1 - p_e}$$

where p_o is the relative observed agreement among the raters (i.e., the accuracy) and p_e is the hypothetical probability of chance agreement. The hypothetical probability of chance agreement is defined as:

$$p_e = \frac{1}{N^2} \sum_{c=1}^{C} n_{c1} n_{c2}$$

where n_{ci} is the number of times rater i predicted category c. According to Landis and Koch [79], the qualitative inter-rater agreement can be defined using Cohen's kappa coefficient as in Table 2.1.

Inter-rater agreement	Cohen's kappa coefficient
No	$k < 0$
Slight	$0 \leq k \leq 0.20$
Fair	$0.20 < k \leq 0.40$
Moderate	$0.40 < k \leq 0.60$
Substantial	$0.60 < k \leq 0.80$
Almost perfect	$0.80 < k \leq 1$

Table 2.1: Interpretation of inter-rater agreement using the Cohen's kappa coefficient [79].

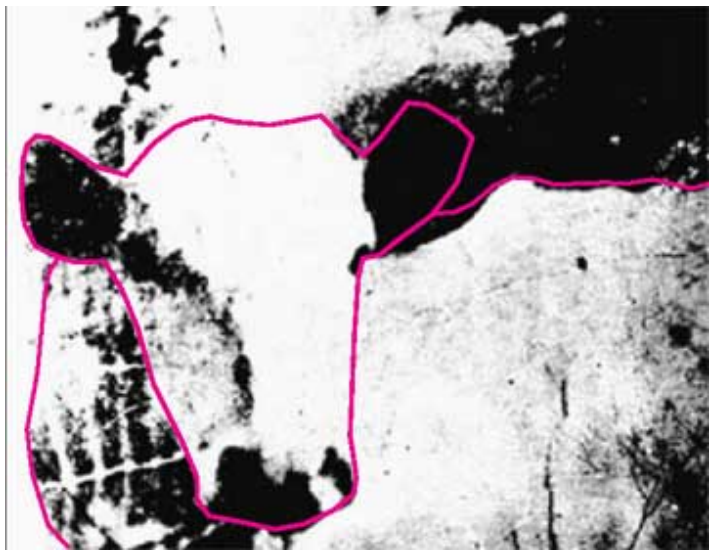

Figure 2.2: In case you're still wondering...

Chapter 3

Detecting defects: a rule-based approach

3.1 Introduction

In this chapter, we first give a background on the NLP technologies used in the study (Section 3.2). Then, we describe the defined NLP patterns (Section 3.3), and the identified discard patterns developed in order to address systematic false positive cases (Section 3.4). We also describe the tool SREE [111], and we outline how the tool was used in our study (Section 3.5). Finally, we summarize the different usage of the presented technologies in the study.

3.2 NLP technologies

In this section, we list the natural language processing (NLP) technologies included in the tool GATE [35] that was adopted to define the patterns:

- **Tokenization:** this technology partitions a document into separate *tokens* (e.g., words, numbers, spaces and punctuation);
- **Part-of-Speech (POS) Tagging:** this technology associates to each token a Part-of-Speech (e.g., noun (NN), verb (VB), adjective (JJ), etc.). Common POS taggers are statistical in nature, i.e., they are trained to predict the POS of a token based on a manually annotated

corpus. Available POS taggers normally have an accuracy around 97% [83], errors might occur in the POS associated to a token. For example, in a sentence such as *"An incident management team shall be organised"*, the token *organised* might be incorrectly tagged as an adjective (JJ), and not as a verb in past participle (VBN);

- **Shallow Parsing:** this technology identifies noun phrases (NP) – in this case we speak about Noun Chunking – and verb phrases (VP) – in this case we speak about Verb Chunking – in sentences. For example, given the sentence *"Messages are received by the system"*, a shallow parser identifies {*Messages*, *the system*} as NPs, and {*are received*} as VP;

- **Gazetteer:** this technology searches for occurrences of terms defined in a list of terms. In our case, we used it to check the presence of vague terms;

- **JAPE Rules:** this technology allows defining rules (i.e., high-level regular expressions) over tokens and other elements in a text [35]. A rule identifies sequences of elements that match the pattern. Rules are expressed in the intuitive JAPE grammar, which is similar to regular expressions. JAPE rules can be rather long to report. In this chapter, for more clarity to describe JAPE rules, we will use a more concise and intuitive pseudo-code inspired to the JAPE grammar. In JAPE, and in our rules, the symbols reported in Table 3.1 are used. Furthermore, when we use a term in capital letters, this indicates a form of *macro* that identifies terms of the specific type (e.g., NUMBER identifies numbers, while ELSE identifies the term *else* in its various orthographic forms). Although these macros differ in terms of semantics, we expect that the reader can infer their meaning.

3.3 Patterns for defect detection

This section lists the classes of language defects considered, together with the patterns (i.e., JAPE rules) defined to identify them. Patterns are defined in terms of sequences of tokens to be matched within a requirement. Hence, the output produced by one pattern when applied to a requirement is zero or n requirement fragments (i.e., contiguous sequences of tokens in the

Expression	**Meaning**
$< expr_1 > \mid < expr_2 >$	$< expr_1 >$ OR $< expr_2 >$
$< expr_1 >, < expr_2 >$	$< expr_1 >$ AND $< expr_2 >$
$! < expr >$	NOT $< expr >$
$< expr > +$	One or more elements matching $< expr >$
$< expr > *$	Zero or more elements matching $< expr >$
$< expr >?$	Zero or one element matching $< expr >$

Table 3.1: Symbols used in the JAPE grammar.

requirement) that match the pattern. The patterns were defined with the idea of identifying the most relevant defects for the Verification Engineers, and also taking into account the defect classes provided by [12]. In Table 3.2 we report the patterns in a compact version. The JAPE implementation of the patterns, together with the discard-patterns that will be introduced in Section 3.4, is available in the public repository[1]. Below, we describe the defect classes addressed by each pattern:

Defect Class	**Pattern**
Anaphoric ambiguity	P_{ANA} = (NP)(NP)+ (Split)[0,1] (Token.POS == PP | Token.POS =∼ PR*)
Coordination ambiguity	P_{CO_1} = ((Token)+ (Token.string == AND | OR)) [2] P_{CO_2} = (Token.POS == JJ) (Token.POS == NN | NNS) (Token.string == AND | OR) (Token.POS == NN | NNS)
Vague terms	P_{VAG} = (Token.string ∈ Vague)
Modal adverbs	P_{ADV} = (Token.POS == RB | RBR), (Token.string =∼ "[.]*ly$")
Passive voice	P_{PV} = (AUXVERB)(NOT)?(Token.POS == RB | RBR)? (Token.POS ==VBN)
Excessive length	P_{LEN} = Sentence.len > 60
Missing condition	P_{MC} = (IF)(Token, !Token.kind == punctuation)* (Token.kind == punctuation)(!(ELSE | OTHERWISE))
Missing unit of measurement	P_{MU_1} = (NUMBER)((Token)[0, 1](NUMBER))?(!MEASUREMENT) P_{MU_2} = (NUMBER)((Token)[0, 1](NUMBER))?(!PERCENT)
Missing reference	P_{MR} = (Token.string == "Ref")(Token.string == ".") (SpaceToken)?(NUMBER)
Undefined term	P_{UT} = (Token.kind == word, Token.orth == mixedCaps)

Table 3.2: Pattern adopted for each defect class.

- **Anaphoric ambiguity** Anaphora occurs in a text whenever a pro-

[1] `https://github.com/ISTI-FMT/QUARS_plus_plus`

noun (e.g., *he*, *it*, *that*, *this*, *which*, etc.) refers to a previous part of the text. The referred part of the text is normally called *antecedent*. An anaphoric ambiguity occurs if the text offers more than one antecedent option [118], either in the same sentence (e.g., *"The system shall send a message to the receiver, and **it** provides an acknowledge message"* - *it* = *system* or *receiver*?) or in previous sentences. The potential antecedents for the pronouns are noun phrases (NP), which can be detected by means of a shallow parser. The pattern P_{ANA} matches any sequence of two or more noun phrases (NP), followed by zero or one sentence separators (Split), followed by a personal pronoun (PP), or other types of pronouns (PR*);

- **Coordination ambiguity** Coordination ambiguity occurs when the use of coordinating conjunctions (e.g., *and* and *or*) leads to multiple potential interpretations of a sentence [28]. Two types of coordination ambiguity are considered here. The first type includes sentences in which more than one coordinating conjunction is used in the same sentence (e.g., *"There is a* 90° *phase shift between sensor 1 **and** sensor 2 **and** sensor 3 shall have a* 45° *phase shift"*). The second type includes sentences in which a coordinating conjunction is used with a modifier (e.g., *"Structured approaches and platforms"* – *Structured* can refer to *approaches* only, or also to *platforms*). Two patterns were defined, one for each type. P_{CO_1} matches exactly two occurrences (notation "[2]") of one or more Tokens followed by a coordinating conjunction. P_{CO_2} matches cases in which an adjective (JJ) precedes a couple of singular (NN) or plural nouns (NNS), joined by *and* or *or*;

- **Vague terms** Vagueness occurs whenever a sentence admits borderline cases, i.e., cases in which the truth value of the sentence cannot be decided [12]. Vagueness is associated with the usage of terms without a precise semantics, such as *minimal*, *as much as possible*, *later*, *taking into account*, *based on*, *appropriate*, etc. In our context, we use the list of 446 vague terms provided by the QuARS tool [62]. The list includes single-word and multi-word terms that were collected as source of vagueness in requirements. P_{VAG} matches any term included in the set *Vague* of vague terms;

- **Modal adverbs** Modal adverbs (e.g., *positively*, *permanently*, *clearly*) are modifiers that express a quality associated to a predicate. Adverbs

are discouraged in requirements as potential source of ambiguity [61]. We noticed that, in the requirements of the company, most of the adverbs causing ambiguity were modal adverbs ending with the suffix -*ly*. For this reason, P_{ADV} matches adverbs in normal form (RB) or in comparative form (RBR) that terminate ($ indicates string termination) with -*ly*;

- **Passive voice** The use of passive voice is a defect of clarity in requirements, and can lead to ambiguous interpretations in those cases in which the passive verb is not followed by the subject that performs the action expressed by the verb (e.g., *"The system shall be shut down"* – by which actor?). Passive voice detection is also considered in [61, 47]. To identify passive voice expressions, P_{PV} matches auxiliary verbs followed by a verb in past participle (VBN), possibly with negations and adverbs;

- **Excessive length** Longer sentences are typically harder to process than short sentences, and can be source of unclarity. It was chosen to identify all the sentences that are longer than 60 tokens. Although this is a rather weak threshold – for generic English texts, sentences are recommended not to exceed 40 tokens [36] –, we considered this value appropriate for the length of the sentences in the railway domain;

- **Missing condition** To be considered complete, each requirement expressing a condition through the *if* clause shall have a corresponding *else* or *otherwise* clause. P_{MC} checks whether an *if* clause is followed by an *else/otherwise* clause in the same sentence;

- **Missing unit of measurement** Each number is required to have an associated unit of measurement, unless the number represents a reference (see below). Hence, the patterns check whether a number has an associated unit, or a percentage value associated to it;

- **Missing reference** This defect occurs when a reference that appears in the text in the form *Ref.* <X> does not appear in the list of references of the requirements document. To detect this defect we leverage the pattern P_{MR} to extract references in the text, and then – through Java code not reported here – we check whether each number found appears in the list of references;

- **Undefined term** This pattern searches all the terms that follow the textual form used in the company for defining glossary terms (e.g., *restrictiveAspect*), which are expressed in camelCase format (i.e., *mixedcap* orthography). As for the *missing reference* case, we leverage the P_{UT} pattern to search for terms expressed in camelCase, and then we automatically search the glossary to check whether the term is present or not.

The defect classes associated to the patterns can be related to part of the broad quality criteria specified by the CENELEC norms, and reported in Section 2.1. Furthermore, they can be related to the different levels of language to which the defect belong, namely lexical, syntactic, semantic and pragmatic – see 2.3 for a discussion in the context of NL requirements. They can also be related to the level of detection, which, in our case, is either lexical or syntactic. Table 3.3 reports these relationships, using a structure similar to the one adopted by Gleich et al. [61].

Defect Class	**Criterion**	**Lev. of Language**	**Detection**
Anaphoric ambiguity	Unequivocal	Syntactic, Semantic, Pragmatic	Syntactic
Coordination ambiguity	Unequivocal	Syntactic, Semantic	Syntactic
Vague terms	Precise	Pragmatic	Lexical
Modal adverbs	Precise	Pragmatic	Syntactic
Passive voice	Clear	Semantic, Pragmatic	Syntactic
Excessive length	Clear	Semantic, Pragmatic	Lexical
Missing condition	Complete	Semantic, Pragmatic	Syntactic
Missing unit of measurement	Complete	Semantic, Pragmatic	Lexical
Missing reference	Complete	Semantic, Pragmatic	Lexical
Undefined term	Complete	Semantic, Pragmatic	Lexical

Table 3.3: Patterns associated to the different CENELEC criteria, and to the different levels of language.

3.4 Discard patterns

A set of context-aware patterns was defined along the case study based on an analysis of the false positive cases produced by the defect detection patterns

(see Section 6.4.2). For the sake of clarity, we refer to these additional patterns as *discard patterns*. Each discard pattern is associated to one defect class. The defect class is the one whose patterns generate the systematic false positive cases. The discard patterns, adapted from the JAPE rules reported in our repository, are shown in Table 3.4, and briefly described below:

Defect Class	**Discard Pattern**
Anaphoric ambiguity	D_{ANA} = ((Token.POS == PP | Token.POS = PR*) within IT_SHALL_BE_POSSIBLE)
Vague terms	D_{VAG_1} = (P_{VAG}, Token.string ==∼ "(?i)sound" | "(?i)light", Token.POS == NN | NNS) D_{VAG_2} = (P_{VAG} within IT_SHALL_BE_POSSIBLE) D_{VAG_3} = (P_{VAG} within $StophPhrasesVague$)
Modal adverbs	D_{ADV_1} = (Token.string ==∼ "(?i)manually" | "(?i)automatically") D_{ADV_2} = (P_{ADV} within INFORMATION_PURPOSES_ONLY)
Undefined term	D_{UT} = (P_{UT} contains $KnownAcronym$)

Table 3.4: Discard patterns.

- **Anaphoric ambiguity:** the pattern D_{ANA} detects the pronoun within sentences matching the pattern IT_SHALL_BE_POSSIBLE. This pattern matches the expressions *it shall be possible*, *it may be possible* and *it should be possible*, in their orthographic variants, and possibly including other terms within the pattern (e.g., *it should* ***also*** *be possible*). The JAPE notation "within" indicates that the first argument is completely included in the second argument. Each ambiguity detected through the pattern P_{ANA} is discarded when it includes D_{ANA};

- **Vague terms:** the pattern D_{VAG_1} matches all the tokens in which the terms *sound* and *light* are used as nouns, according to the annotations of the POS Tagger. The JAPE notation "(?i)" indicates that all orthographic variants of the string shall be matched. The pattern D_{VAG_2} matches the term *possible* when used within the pattern IT_SHALL_BE_POSSIBLE. The pattern D_{VAG_3} matches any vague term included in the list of stop phrases $StopPhrasesVague$, which collects the set of domain specific terms that include vague terms (e.g., *distant signalling distance*, *near miss*), according to our analysis of the false positive cases. Each vague term detected through P_{VAG} is discarded when it includes D_{VAG_1}, D_{VAG_2} or D_{VAG_3};

- **Modal Adverbs:** the pattern D_{ADV_1} matches the terms *manually* and *automatically*. The pattern D_{ADV_2} matches the term *only* within

the expression *information purposes only*. Each modal adverb detected through P_{ADV} is discarded when it includes D_{ADV_1} or D_{ADV_2};

- **Undefined term:** the pattern D_{UT} matches any unknown term annotation (P_{UT}) that contains a known acronym (i.e., a term included in the list $KnownAcronym$). Any P_{UT} annotation is discarded when it includes D_{UT}.

3.5 SREE patterns

The tool SREE [111] is a defect detection tool for NL requirements that is oriented to achieve 100% recall for the defects in its scope, even at the cost of lower precision. SREE leverages a set of dictionaries of typically defective terms (single and multi-word). A requirement that includes a term that matches one of the terms of the dictionaries is returned by SREE as a potentially defective requirement. Furthermore, the matched term is also returned. The key feature of SREE resides in searching only for *lexical* matches, without leveraging POS Taggers or other statistical tools that may, in principle, decrease the recall. The approach is analogous to the one adopted in our work for the pattern for *Vague terms* (see Section 3.3).

SREE employs ten dictionaries, and each dictionary is associated to a defect class. The defect classes, together with representative examples of the terms included – called SREE indicators [111] – are:

- **Continuance:** as follows, below, following, in addition, in particular, etc.;
- **Coordinator:** and, and/or, or;
- **Directive:** e.g., etc., figure, for example, i.e., note, table;
- **Incomplete:** TBA, TBD, as a minimum, as defined, as specified, etc.;
- **Optional:** as desired, at last, either, eventually, if appropriate, in case of, if necessary, etc.;
- **Plural:** contains a list of $11,287$ plural nouns, each ending in "s";
- **Pronoun:** anyone, he, her, this, they, which, whom, yourself, etc.;
- **Quantifier:** all, any, few, little, many, much, several, some;

- **Vague:** ( ), [], as far as, as required, eventually, mutually-agreed, etc.;
- **Weak:** can, could, may, might, ought to, preferred, should, will, would.

The complete list of terms for each dictionary, with the exception of the *plural* class, can be found in [111]. For the *plural* class, we contacted Daniel M. Berry, who kindly provided the list. In our study we adopted the dictionaries of SREE. Specifically, each SREE dictionary was imported in GATE as a separate Gazetteer. In our evaluation we apply all the SREE dictionaries, with the exception of the dictionary of the *weak* class, since this class was initially excluded from the analysis.

3.5.1 SREE-reduced

A subset of SREE was also adopted in our case study. The selection, which we call *SREE-reduced*, is composed of the terms that are specific to SREE, and are not considered in our patterns. In particular, pronouns are sources of anaphoric ambiguities, and are considered in our P_{ANA} pattern. Furthermore, the coordinators *and* and *or* are sources of coordination ambiguities and are considered in our P_{CO_1} and P_{CO_2}, while the expression *and/or* was considered in our list *Vague* of vague terms. Finally, also part of the terms included in the different SREE dictionaries are included in our *Vague* list. Therefore, *SREE-reduced* is composed of the dictionaries of SREE but excluding:

1. the dictionaries of the *coordinator*, *pronoun* and *weak* class;
2. all the terms in the other dictionaries that were already part of the *Vague* list.

3.6 NLP technologies applied to our case study

In this section we summarize the different experimental usage of the NLP technologies presented above to address the study goal. As already said, the grand objective of our case study, described in detail in the following chapters, was to investigate to which extent NLP can be practically applied to detect defects in the requirements documents. In order to improve the cost-effectiveness of the verification activity it was important to maximize both

precision and recall: for a proficient use of the provided tool by the Verification Engineers it has to show all possible defects without missing nothing and hopefully without showing too many false positive cases. Therefore we applied an iterative process (Chapter 4, Figure 4.1) in order to maximize the identified defects minimizing the false positive cases. From here descends the decision to use first our identified patterns, then the discard ones and at the end we tried to improve the obtained recall choosing within SREE dictionaries (Section 3.5) the potential words missing in our patterns (Section 3.5.1).

Chapter 4

Research methodology and Case study design

4.1 Introduction

The experience presented in this chapter and in chapters 5, 6 and 7[1] has been conducted in collaboration with Alstom, one of the major railway companies. The case study selection has been triggered by the involved company and by its need to support Verification Engineers in their task of requirements review with automated tools. Specifically, the company contacted two research institutions, namely ISTI-CNR and University of Florence, and during the case study the following roles were identified and requested:

- NLP-E: this role identifies an NLP Expert, that is, a researcher with proven knowledge of NLP techniques;
- VE: this role identifies an actor with a proven knowledge of CENELEC standards and a strong background as a Verification Engineer.

The subjects taking these roles are presented in Section 5.2.2.

This research experience shares the typical characteristics of case study research, in that the phenomenon under study is analyzed within its natural context (in particular a railway company) and the boundary between the context and the phenomenon are not clearly evident, and cannot be fully controlled [120]. It also includes iterative and improving aspects that are

[1] This experience has been published in [100, 51]

closer to action research [8], and technology transfer [63]. Overall, our empirical design can be regarded as an *exploratory* and *iterative* case study. Its design largely follows the guidelines provided in [103], adapted to the iterative context of our experience. Specifically, each iteration follows a template reference structure, which includes research question (RQ) definition, data collection procedures, and data analysis procedures. Each iteration is based on specific RQs, and its results are used as triggers to define additional RQs to be answered in the next iteration. In the following, we first outline the RQs produced, and then we describe the template structure adopted in each iteration.

4.2 Research objective and Research questions

The research objective can be decomposed into the following RQs. It should be noticed that the RQs have been generated along with the case study iterations, and were not already defined at the beginning of the study. We will keep that same order in the following enumeration.

Each RQ will be associated to one or more iterations of the case study. Given the iterative nature of research question generation, we consider it appropriate to present the research questions together with the outline of the iterations that addressed them.

- **RQ1: What is the accuracy of the NLP patterns for defect detection?**
  We want to provide a *quantitative* measure of the effectiveness of the patterns in identifying requirements defects. The assumption is that the higher the measures of accuracy, the more effective are the patterns. To this end, we want to compare the results of the application of the patterns with the defects identified by domain experts (i.e., VEs). The accuracy is measured in terms of precision and recall. The former indicates how many of the defects identified by a tool are considered as defects by VEs. The latter indicates how many of the defects identified by VEs are actually identified by a tool. Precise definitions are given in Section 2.4, and will both consider single defects and defective requirements as described in Section 4.3.3. Single defects are requirements fragments considered defective according to a specific defect class, while defective requirements are requirements that include at least one single defect.

- **RQ2: Which are the cases of inaccuracy of the NLP patterns for defect detection?**
  We want to provide a *qualitative* analysis of the effectiveness of the patterns. More specifically, we want to understand which are the specific cases in which the patterns fail in identifying defects. This is done in terms of (a) defects identified by VEs that are not detected by the patterns (i.e., *false negative* cases, which impact on recall); and (b) in terms of defects that are detected by the patterns, but are not considered as defects by the VEs (i.e., *false positive* cases, which impact on precision), as stated in Section 2.4.

- **RQ3: What is the precision of NLP patterns for defect detection when complemented with discard patterns?**
  This question was generated after answering RQ2. Indeed, it was observed that the defect detection patterns generate *systematic* false positive cases, which could be addressed with discard patterns. The application of discard patterns is expected to increase the precision of the overall approach, and this question aims at quantitatively evaluating to which extent the precision can be increased.

- **RQ4: Can a third-party tool identify additional defects?**
  We want to understand whether the usage of an additional tool can allow us to address false negative cases, and to identify additional defects not considered in the patterns. To this end, we apply the dictionaries of SREE, a tool specifically designed to achieve 100% recall on the defects considered. To answer this broad question, we decompose it into the following sub-questions.

  - **RQ4.1: What is the accuracy of SREE with respect to the NLP patterns for defect detection complemented with discard patterns?**
    We first want to understand whether SREE identifies defective requirements identified by the VEs, and not identified by the patterns (i.e., false negatives). By answering this question, we provide a quantitative evaluation of the accuracy of SREE in identifying defective requirements, in terms of recall and precision. The comparison with the patterns is useful to understand whether SREE and the patterns can be considered as complementary tools.

- **RQ4.2: What is the precision of SREE?**
  This question was generated after answering RQ4.1, and noticing that SREE generates a large number of false positive requirements. This suggested that SREE may be less precise than the patterns also at the level of single defects. Thus we wanted to further assess the precision obtained by using uniquely the tool SREE.
- **RQ4.3: Which additional defects can be identified with SREE?**
  This question was generated after answering RQ4.1. Indeed, we considered that some of the false positive requirements issued by SREE could include specific defects not considered by the patterns. Therefore the goal was to understand whether novel categories of defects can be identified using SREE.
- **RQ4.4: Which are the false positive cases for SREE?**
  This question was generated after answering RQ4.1, and as a qualitative complement to RQ4.2, to check which are the typical sources of false positives at the level of defects.

4.3 Data collection and Analysis procedures

To collect and analyze the data necessary to answer the RQs, each iteration followed a template structure. The template structure of the iterations is depicted in Figure 4.1. The template is composed of eight tasks, which are further grouped into three main phases, namely *Preparation*, *Data Collection*, and *Data Analysis*. The phases are designed to ensure a minimal intervention of NLP-E in the execution of the case study. Specifically, the contribution of NLP-E was limited to the Preparation and Data Analysis phases. The Data Collection phase was carried out by the VEs involved in the specific iterations.

4.3.1 Preparation

The preparation phase consists of two tasks, described below:

- **Research Questions:** definition of RQs which are going to be answered by the iteration. If in the previous iteration, the RQs are

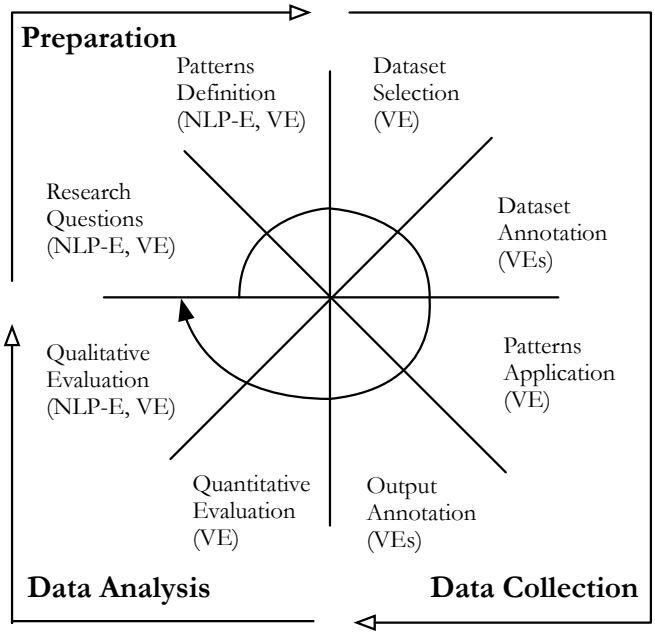

Figure 4.1: Template structure adopted in the iterations of the case-study.

considered to require another iteration to be answered, the previous RQs are kept. Furthermore, in this phase, a specific instance of the template is chosen so that this is appropriate to answer the questions. In particular, the phases of the template that will be performed are selected – not all the phases are required for each iteration;

- **Patterns Definition:** patterns are defined and implemented to support defect detection. The patterns will be employed in the iteration. In this phase, we consider the definition of defect detection patterns, the definition of discard patterns, and also the implementation of the patterns that support the dictionaries of SREE. If the patterns are defined in previous iterations, this phase is not performed.

4.3.2 Data collection procedure

Data are collected according to the following tasks:

- **Dataset Selection:** a requirements dataset is selected, to which we apply the patterns;

- **Dataset Annotation:** the dataset is manually annotated for defects. Annotations may have been performed also before the current study, as for *Large-scale Study - 1st Iteration*, see Section 5.3.2. In this case, for the sake of structure and clarity of the presentation, we consider the annotation as it would be performed during this task. If both the dataset and the annotations come from a previous iteration, this phase is not performed. The output of this phase is a set of requirements which are annotated as *accepted*, if they do not contain defects, or *rejected*, if they contain at least one defect. Furthermore, depending on the iteration considered, annotations associated to specific defects are also provided. More specifically, the annotation was performed as follows: given a requirement, this was labelled as *accepted* if it appeared to fulfill the criteria normally adopted by the company. These criteria are derived from the more general guidelines provided by the CENELEC EN-50128:2011 norm [26], and considering also the IEEE Std 1233-1998 as a reference [67][2]. In particular, a requirement was labeled as *accepted* if it was: (a) *feasible*: what is required is physically and technologically possible, can be done with available resources and is not against laws and regulations; (b) *testable*: can be demonstrated through repeatable tests or is at least verifiable through inspection; (c) *complete*: stand-alone, no missing references, undefined terms, to-be-defined parts, or missing conditions; (d) *clear and unambiguous*; (e) *uniquely identifiable*; (f) *consistent*: no internal contradiction and no contradiction with other requirements. The requirement was labeled as *rejected* in case it did not fulfill one of the criteria. In case the requirement was marked as *rejected* for criterion (c) or criterion (d), the VE involved stated whether the rejection was due to one or more linguistic defect classes associated to the patterns listed in Section 3.3. In this case, the VE involved labelled as $defective(i)$ each requirement fragment that included the i-th defect. Different defective requirement fragments can thus overlap;

- **Patterns Application:** the patterns are applied on the annotated dataset, and potentially defective requirements are produced as output;

[2]The standard is currently replaced by ISO/IEC/IEEE 29148:2011 [68].

- **Output Annotation:** in each iteration, this task is considered as mutually exclusive with the Dataset Annotation task. Indeed this task is mainly oriented to assess the precision of the output of the patterns, and has been introduced when doubts were raised about the quality of the original annotations, or whenever further assessment was required. This task is performed as follows. For each requirement fragment labelled as defective according to pattern i, each VE annotated the fragment as $defective(i)$, if the VE considered the defect as a true defect[3]. Overall, if a fragment was annotated as $defective(i)$ by at least one VE, the fragment was marked as $defective(i)$ in the annotated set used for the evaluation. This choice (instead of the unanimity one) enlarges the number of defective fragments, but prevents us to miss some possible ambiguous fragment.

4.3.3 Data analysis procedure

Data analysis is performed according to the following tasks:

- **Quantitative Evaluation:** the accuracy of the patterns in detecting defects is evaluated. In particular, we evaluate the values of precision and recall (see Section 2.4) of the patterns with respect to the golden standard (i.e., the performed annotations). Evaluation measures for single defects and for entire requirements are provided, and defined as follows:

  - *Evaluation Measures by Defect:* To measure the effectiveness of the patterns, we first provide a set of measures that focus on single defective fragments identified by the patterns. Given the pattern associated to the i-th defect, we consider the amount of true positive tp^D as the number of requirements fragments labeled as $defective(i)$ and correctly identified by the pattern; the amount of false positive fp^D as the number of requirements fragments wrongly identified as defective by the pattern; the amount of false negative fn^D as the number of requirements fragments labeled as $defective(i)$ that are not discovered by the pattern. Based on these definitions, we define the measure of precision (p^D) and recall (r^D) using the same equations provided in Section 2.4;

[3]In this context, we consider as a pattern i also a dictionary from *SREE-reduced*, as defined in Section 3.5

  - *Evaluation Measures by Requirement:* to have a view of the effectiveness of the patterns applied together, we provide a set of measures that focus on the number of requirements, instead of on the number of defective fragments. Here, we consider the amount of true positive tp^R as the number of requirements labeled as *rejected* for which at least one of the patterns correctly identified a defective requirement fragment; the amount of false positive fp^R as the number of requirements wrongly identified as defective (i.e., at least one of the patterns triggered a defect while the requirement was marked as *accepted*); the amount of false negative fn^R as the number of requirements marked as *rejected* for which none of the patterns triggered a defect. The measures of precision p^R and recall r^R are defined using the same equations provided in Section 2.4.

  Depending on the iteration, different evaluation measures are used, among those listed above;

- **Qualitative Evaluation:** cases of inaccuracy of the patterns are evaluated and classified. In particular, the results produced by the patterns are inspected, and classes of inaccuracy cases are provided; for each class a representative example has been extracted. The interaction between the involved subjects was performed by means of on-line calls, and shared documents.

4.4 Validity procedure

The validity procedure adopted aims to ensure the validity of the data used in the study.

To ensure the validity of the annotations performed on the datasets during the Output Annotation task, the annotation process is independently performed by two VEs. The inter-rater agreement is computed by means of the Cohen's kappa coefficient (see Section 2.4 and Table 2.1). In case of disagreement, if at least one of the annotators considered a requirement as defective, the requirement was considered defective in the final set used during the analysis. This validity procedure was not followed in the *Pilot Study*, due to its preliminary nature (Section 5.3.1). Furthermore, it was not

followed during the Dataset Annotation task. Specific threats associated to this aspect are discussed in Section 6.9.

Second, we ensure the validity of the quantitative results reported, by replicating part of the study.

Third, to limit the researcher bias, the intervention of NLP-E was limited to the Preparation and Data Analysis phases, while Data Collection was entirely performed by the VEs involved in each iteration.

NLP-E never had access to the datasets used, but solely to the quantitative results produced.

Chapter 5

Experimentation

5.1 Introduction

This chapter describes the execution of the case study. We first describe the characteristics of the case and the subjects involved, and then we describe the different iterations performed in relation to the RQs presented in Section 4.2.

5.2 Case and Subjects description

5.2.1 The company and its process

The company produces signalling equipment for both railway and urban transport applications. In order to efficiently produce such systems, the company develops a set of different products aimed to provide generic functionalities; specific projects based on their product lines are then developed in order to satisfy customer's specific needs. These needs are usually expressed in requirements released by the customer to the companies tendering for contract. The requirements are then elaborated and refined by the company, without relying on standard editing guidelines. The company, for both products and projects, applies the V-model for life-cycle management according to the CENELEC standard [26] as described in Chapter 1. As dictated by the standard, a requirements' review activity is performed by the Validation Team, according to the criteria reported in Section 4.3.2.

5.2.2 Subjects

The case study has been conducted by a team composed as follows:

- NLP-E: a researcher of ISTI-CNR who covered the NLP-E role described in Section 4.1;
- VE-A: the author of this thesis and a former Verification Engineer who covered the VE role described in Section 4.1. This actor, during the development of the case study, acquired strong knowledge on NLP techniques;
- VE1: a Verification Engineer covering the VE role described in Section 4.1. This actor, during the development of the case study, acquired strong knowledge on NLP techniques;
- VE2: a Verification Engineer with strong experience in tender requirements review covering the VE role described in Section 4.1.

VE-A, VE1 and VE2 belonged to different groups within the company, but they were subject to the same company practices. VE1 and VE-A voluntarily participated to the study. VE2 participated to the study since the requirements reviewed by him before this work was conceived (*D-Large*, see below) were used in the case study.

5.2.3 Datasets

The datasets made available for this research activity consist of:

- **Pilot Dataset (*D-Pilot*):** this dataset consists of 241 system requirements. This dataset was randomly selected from the requirements documents of a wayside Automatic Train Protection (ATP) system and an interlocking (IXL) system belonging to the same product. ATP systems are embedded platforms that enforce the rules of signaling systems, by adding an on-board automatic control over the speed limit allowed to trains along the track. Instead, IXL systems controls the movement of trains in the railway yard, by setting signal statuses, and moving railway switches. This dataset is composed by the following requirements types: functional, architectural, interface and performance;

ID	Iteration Name	Nature	RQs	Patterns	Dataset
0	Pilot	Exploratory	RQ1 RQ2	Def. Det. Patterns	D-Pilot
1	Large-scale - 1st	Exploratory	RQ1 RQ2	Def. Det. Patterns	D-Large
2	Large-scale - 2nd	Explanatory	RQ1 RQ2	Def. Det. Patterns	D-Large
3	Large-scale - 3rd	Improving	RQ3	Def. Det. Patterns + Discard Patterns	D-Large
4	Large-scale - 4th	Improving	RQ4.1	SREE	D-Large
5	Large-scale - 5th	Explanatory	RQ4.2 RQ4.3 RQ4.4	SREE-reduced	D-Large

Table 5.1: Outline of the different iterations performed.

- **Large-scale Dataset (*D-Large*):** this dataset consists of 1866 requirements. The requirements belong to a requirements document concerning a system-of-systems that includes an interlocking system, an ATP, a CTC (Centralised Traffic Control) and an Axle Counter. Interlocking and ATP systems have been briefly described above. CTC systems monitor and dispatch trains. Axle Counters are embedded systems located along the railway line, which detect the passing of a train between two points on a track. The requirements were originally written by the customer in international English language and refined by the company. No particular glossary restrictions are applied and no guideline was provided. This dataset is composed by the following requirements types: functional, architectural, interface and ergonomical.

In all these datasets safety requirements are not included, since they are handled by an independent safety assessment process, which produces separately the safety requirements documents.

5.3 Iterations

The execution of the case study consists in a set of iterations: each iteration is aimed at answering one or more RQs, and, although the overall case study is exploratory, each iteration has a different flavor, which range from *exploratory*, to *explanatory* and to *improving*. Furthermore, in each iteration,

ID	Res. Quest.	Pat. Def.	Data. Sel.	Data. Ann.	Pat. App.	Out. Ann.	Quant. Eval.	Qual. Eval.
0	VE1 NLP-E	VE1 NLP-E	VE1	VE1	VE1	-	VE1	VE1 NLP-E
1	VE1 NLP-E	-	VE1	VE2	VE1/ VE-A	-	VE1/ VE-A	VE1 NLP-E
2	VE1 NLP-E	-	-	-	-	VE1 VE-A	VE1/ VE-A	VE1/VE-A NLP-E
3	VE-A NLP-E	VE-A NLP-E	-	-	VE-A	-	VE-A	-
4	VE-A NLP-E	VE-A NLP-E	-	-	VE-A	-	VE-A	VE-A NLP-E
5	VE-A NLP-E	VE-A NLP-E	-	-	VE-A	VE1 VE-A	VE-A	VE-A NLP-E

Table 5.2: Tasks performed and subjects involved in each iteration.

different tasks of the template are performed. Tables 5.1 and 5.2 give an outline of the different iterations. Overall, the case study consists of six iterations. The first one is a *Pilot Study*, based on a preliminary requirements dataset (*D-Pilot*), while the others belong to the *Large-scale Study*, based on a larger requirements dataset (*D-Large*). Table 5.1 shows the nature of the iteration, the associated RQs, the patterns and dataset used. Iterations from 0 to 2 were dedicated to investigate the accuracy of NLP patterns (RQ1, RQ2), with different levels of insight. Iteration 3 was dedicated to improve the precision of the patterns (RQ3). Iteration 4 and 5 were focused on the application of the SREE dictionaries (RQ4.1-4). Table 5.2 shows the tasks performed together with the subjects who participated to the task.

As already stated in Section 4.4, the role of NLP-E has been involved during Research Questions, Patterns Definitions and Qualitative Evaluation in order to limit as much as possible the researcher influence in the case study execution. Here, we briefly summarize the rationale, execution and results of each iteration, with reference – explicit or implicit – to Table 5.1 and 5.2. We do not provide all the justifications for the content of the tables, since extensive details are given in the subsequent subsections.

- ***Pilot Study:*** this iteration was oriented to have a first understanding of the applicability of NLP patterns for defect detection in the context of the company. To this end, the defect detection patterns (Def. Det. Patterns in Table 5.1, reported in Section 3.3) were defined by VE1

under the guidance of NLP-E, with the objective of maximizing recall [11]. Then, they were applied by VE1 on a limited dataset of the company, i.e., *D-Pilot*, which was previously annotated for defects by VE1. A recall of 88.33% (r^R) and a precision of 64.24% (p^R) were obtained, and the recall r^D for single defects reached 100% for the majority of the patterns.

- ***Large Scale Study - 1st Iteration:*** given the encouraging result of the previous iteration, the defect detection patterns were applied by VE1 on *D-Large*, annotated for defects by VE2. The goal was now to understand whether the approach was applicable on a larger set of requirements of the company, annotated by a subject who did not participate to the definition of the patterns. Furthermore, the tasks named Patterns Application and Quantitative Evaluation, originally performed by VE1, were replicated by VE-A (VE1/VE-A in Table 5.2), to confirm the validity of the produced data. In this iteration, the results were acceptable in terms of recall ($r^R = 85.39\%$), but particularly poor in terms of precision, with $p^R = 5.81\%$. A non-systematic Qualitative Evaluation performed by VE1 suggested that many potential *linguistic* defects were ignored by VE2 in his annotation, thus leading to the low value of precision observed.
- ***Large Scale Study - 2nd Iteration:*** this iteration aimed at systematically explaining the poor results of the previous one. In particular, we were interested in understanding whether the false positive cases produced according to the annotations of VE2 could be considered as true positives (i.e., defects), if an additional annotation was performed with a focus on linguistic defects. Therefore, the output of the Pattern Application task from the previous iteration was considered – as shown in Table 5.2, the tasks from Patterns Definition to Patterns Application were not performed again. The Output Annotation task was carried out by VE1 and VE-A, and their agreement was assessed. Quantitative Evaluation was performed by VE1, and then replicated by VE-A. The precision obtained was $p^R = 77.37\%$, and the average precision at defect level – average of p^D for the different defects – was 72.81%. This confirmed the effectiveness of the patterns for linguistic defects. The Qualitative Evaluation, also replicated, was supported by NLP-E, and allowed the identification of classes of *systematic* false positive cases, which could be potentially discarded with additional patterns.

- ***Large Scale Study - 3rd Iteration:*** based on the Qualitative Evaluation of the previous iteration, we wanted to understand to which extent the precision could be further increased through additional patterns, designed to discard false positive cases (Discard Patterns in Table 5.1, reported in Section 3.4). VE-A took the lead in this activity due to other company-related commitments of VE1, and defined a set of discard patterns under the guidance of NLP-E. With these patterns, the precision p^R further increased to 83.16%, and the average p^D reached 81.36%.

- ***Large Scale Study - 4th Iteration:*** this iteration aimed at understanding whether the defect-detection capabilities of the approach could be complemented with the usage of an additional tool, namely SREE (see Section 3.5). To have a general, initial indication, we considered the annotations performed by VE2 on *D-Large* (annotations already used in *Large Scale Study - 1st Iteration*), and we checked whether SREE was able to identify *requirements* that were annotated as defective by VE2, but were not identified by our patterns. To this end, the performance of SREE, in terms of p^R and r^R, were compared with those of the defect detection patterns complemented with discard patterns. VE-A performed all the tasks included in this iteration. The Quantitative Evaluation task showed that SREE achieved higher recall with respect to our patterns ($r^R = 96.63\%$ vs 85.39%), but at the cost of lower precision ($p^R = 5.45\%$ vs 6.24% – i.e., 351 additional false positive requirements). SREE was therefore recognised as an appropriate complement to our patterns, i.e., undetected defective requirements could be identified, but further investigation was required to explain its poor performance in terms of precision.

- ***Large Scale Study - 5th Iteration:*** this iteration was driven by the low value of precision obtained with SREE at the level of requirements, and was oriented to have a fine-grained assessment of the performance of SREE. Specifically, we wanted to assess the precision of SREE at the level of the single *defects* in its scope. VE-A used a subset of the SREE dictionaries, i.e., *SREE-reduced* (see Section 3.5), including solely those terms that were specific to SREE and were not already considered in our patterns. The Output Annotation task was performed in parallel by VE1 and VE-A on the single defects produced by *SREE-reduced*, and their agreement was assessed. Although the average p^D for the

different defects resulted to be only 11.29%, the Qualitative Evaluation, performed by VE-A and NLP-E, showed that several novel classes of *defects* discovered were not considered by our patterns. This confirmed the complementary role of SREE with respect to our patterns.

In the following subsections, we report how each specific iteration was executed. The reader should refer to Table 5.1 and Table 5.2 to have a structured summary of the information provided in each subsection.

5.3.1 Pilot Study

Fig. 5.1 gives an outline of the iteration. The iteration involved NLP-E and VE1, and aimed to address RQ1 and RQ2. In this iteration, all the tasks of the template are performed, with the exception of Output Annotation. This iteration was *exploratory*, in that it aimed to assess the accuracy of NLP patterns on a limited dataset of the company. The tasks performed are as follows:

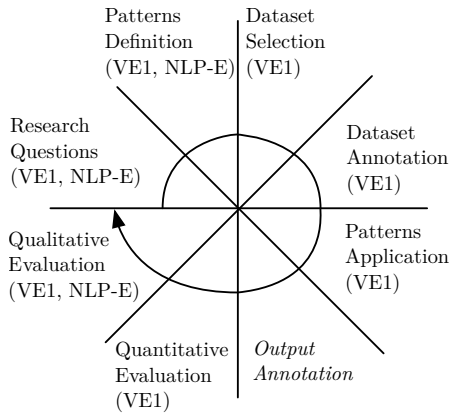

Figure 5.1: Structure of the Pilot Study.

- **Research Questions:** RQ1 and RQ2 were defined in collaboration between NLP-E and VE1. In this iteration, the underlying goal was to establish whether the patterns were able to achieve a recall value close to 100%. Defect detection techniques shall favor recall over precision since the cost of undetected *true* defects is much higher than the cost of manually discarding false positive cases [11].

- **Patterns Definition:** NLP-E considered that assessing the effectiveness of a domain-generic tool for defect detection (e.g., QuARS [62]) would have required a strong *expertise* in the domain of the requirements documents. In addition, he considered that, if the tool had provided too many false positive cases, e.g., *innocuous ambiguities* [28], the company would not have considered the tool as appropriate for its needs. Hence, it was decided to let VE1 develop the tool *in-house*, with the support of NLP-E.

  VE1 was initially required to study some papers [12], [62], [61], [111] and [6]. Then, she was required to perform the tutorials provided by GATE (General Architecture for Text Engineering, see [35]), which was the generic NLP tool selected to be tailored to support defect detection. The tool was chosen since it was considered sufficiently easy to use for an engineer, and sufficiently powerful for the task. After this training, VE1 and NLP-E met to define the defect classes on which to focus. Priority was given to those defect classes that were considered more relevant from the point of view of VE1 – taking into account the defect classes provided in [12], and in the other papers she had studied – and whose identification was considered feasible by NLP-E. VE1 autonomously implemented the patterns, under the supervision of NLP-E.

  The patterns developed are reported in Section 3.3.

- **Dataset Selection:** *D-Pilot* was selected by VE1 under the guidance of representatives of the company.

- **Dataset Annotation:** the dataset was manually annotated by VE1. After this task, 120 requirements were marked as *rejected*, while 121 were marked as *accepted*[1].

- **Patterns Application:** the task was then carried out using the support of GATE.

- **Quantitative Evaluation:** VE1 provided NLP-E with a table with the results of the evaluation. The measures used are for defects, tp^D, fp^D, fn^D, p^D, r^D, and for requirements, tp^R, fp^R, fn^R, p^R, r^R.

[1]The dataset appears balanced since VE1 continued to randomly select new requirements from the original requirements considered, until a balanced number of accepted and rejected requirements was obtained.

- **Qualitative Evaluation:** VE1 evaluated false positive and false negative cases, and provided representative examples. VE1 and NLP-E interacted so that NLP-E could tailor the cases and examples for reporting.

5.3.2 Large-scale Study - 1st Iteration

Fig. 5.2 gives an outline of the iteration. The iteration involved NLP-E, VE-A, VE1 and VE2. This iteration is still based on RQ1 and RQ2, in that it aims to further answer the RQs with a case modification – in terms of dataset used and annotator –, and the nature of the iteration is still *exploratory*. All the tasks, with the exception of Patterns Definition and Output Annotation are performed. The patterns were the one used in the previous iteration. To confirm the validity of the produced data, VE-A replicated part of the tasks. The parts replicated by VE-A are represented in dashed line in Fig. 5.2. The tasks performed are as follows.

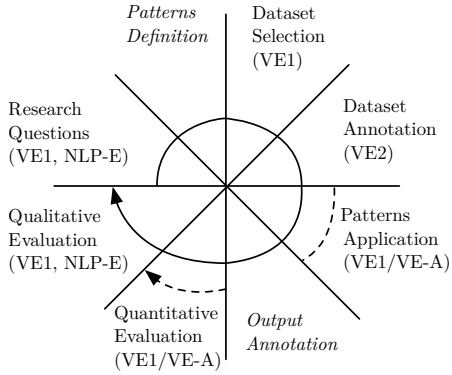

Figure 5.2: Structure of the Large-scale Study – 1st Iteration.

- **Research Questions:** the research questions RQ1 and RQ2 were kept from the previous task. The objective of this iteration was to perform an assessment of the patterns on a larger requirements dataset of the company, previously validated by VE2, to understand to which extent the approach could be applicable more widely within the company.
- **Dataset Selection:** *D-Large* was selected by VE1, under the guidance of representatives of the company.

- **Dataset Annotation:** the defects of the document were previously annotated by VE2, following the criteria of the company already outlined in Sect. 4.3.2. Since this task was performed before this work was conceived, the annotation of the defective fragments was not performed by VE2, who just marked requirements as *accepted* or *rejected*, and described the reasons for rejection in a specific requirements validation document. From the 1866 requirements, 1733 were marked as *accepted*, while 93 were marked as *rejected*.

- **Patterns Application:** the task was initially carried out using the support of a tool developed by VE1 on top of GATE to facilitate the analysis of the results. In the replication, the task was performed by VE-A, but using solely the support of GATE.

- **Quantitative Evaluation:** the measures adopted to evaluate the effectiveness of the patterns in identifying defective requirements are tp^R, fp^R, fn^R, p^R and r^R. Intuitively, these measures indicate whether the application of the different patterns simultaneously allows the identification of requirements that were marked as *rejected* by VE2. Since VE2 did not annotate fragments, for this analysis we do not consider evaluation measures for the single defects as in the *Pilot Study*.

- **Qualitative Evaluation:** given the poor results obtained from the Quantitative Evaluation (see Sect. 6.3), especially in terms of precision, this task was performed by VE1 as a non-systematic inspection of the false negative and false positive cases. The inspection of the false positive cases was oriented to understand whether these cases included defective requirements not initially annotated by VE2. This evaluation triggered the *Large-scale Study – 2^{nd} Iteration*, which aimed to more rigorously explain the poor results.

5.3.3 Large-scale Study - 2nd Iteration

Fig. 5.3 gives an outline of the iteration. The iteration involved NLP-E, VE-A and VE1 and was performed to provide a more informed answer to RQ1 and RQ2. The iteration has an *explanatory* nature, in that its underlying goal was to explain whether the false positive cases identified in the previous iteration could be considered as true positive cases, from the point of view of more strict annotators. To confirm the validity of the produced data, VE-A

replicated part of the tasks. The parts replicated by VE-A are represented in dashed line in Fig. 5.3. The tasks performed are as follows.

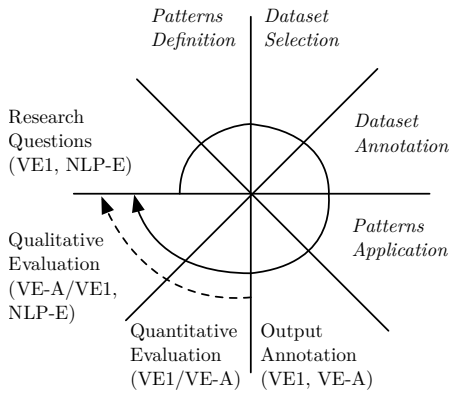

Figure 5.3: Structure of the Large-scale Study – 2nd Iteration.

- **Research Questions:** RQ1 and RQ2 were considered not sufficiently answered by the previous iteration, and the iteration was designed to understand to which extent the low value of precision observed was due to inaccuracies in the annotation process performed by VE2.

- **Output Annotation:** a second annotation process was performed on the requirements marked as defective by at least one of the patterns. In this annotation process, two VEs (VE1 and VE-A) independently annotated the output of the patterns. The agreement between annotators was estimated with the Cohen's Kappa, resulting in $k = 0.82$, indicating an almost perfect agreement[2].

- **Quantitative Evaluation:** since in this analysis we focus solely on the output produced by the patterns, we consider neither the amount of false negative cases, nor the measure of recall. Hence, we consider tp^D, fp^D, p^D, for each defect class i, and tp^R, fp^R, p^R, as measures of the precision over requirements.

[2]According to [79], the following qualitative measures are associated to the different ranges of the Cohen's Kappa: $k < 0$, no agreement; $0 \leq k \leq 0.20$, slight; $0.21 \leq k \leq 0.40$, fair; $0.41 \leq k \leq 0.60$, moderate; $0.61 \leq k \leq 0.80$ substantial; and $0.81 \leq k \leq 1$ almost perfect agreement.

- **Qualitative Evaluation:** the task was performed by VE1 first, and was later reviewed VE-A, to give a first categorisation of the false positive cases. The categorisation was refined by NLP-E based on the examples given by the VEs, with a particular focus on systematic categories of false positives, which could be potentially discarded with additional patterns.

5.3.4 Large-scale Study - 3rd Iteration

Fig. 5.4 gives an outline of the iteration. This iteration involved NLP-E and VE-A, was aimed at answering RQ3, and had an *improving* nature. Indeed, the goal of this iteration was to understand whether the performance of the patterns in terms of precision could be improved with discard patterns. To implement the foreseen improvement of the patterns, VE-A was actively involved in the activity. Indeed, at this stage, VE1 was committed to a mentoring program within the company, to disseminate the best practices for requirements quality learned throughout the experience. The task performed in this iteration are as follows.

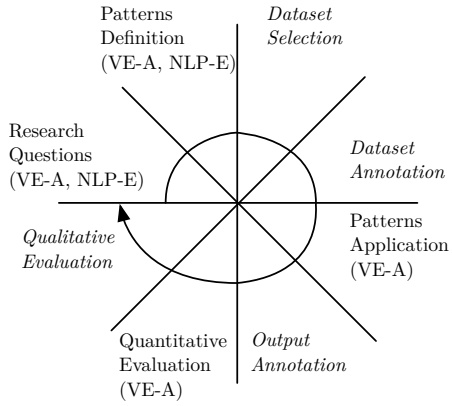

Figure 5.4: Structure of the Large-scale Study – 3rd Iteration.

- **Research Questions:** the Qualitative Analysis performed in the previous iteration allowed NLP-E, VE1 and VE-A to observe that a set of systematic false positive cases could be addressed with specific patterns designed to discard these cases (see Sect. 6.4.2). Therefore RQ3 was

defined, and the iteration was designed to define, apply and evaluate the discard patterns in conjunction with the defect detection patterns.

- **Patterns Definition:** VE-A performed a self-training, analogous to the one performed by VE1 (i.e., a study of the selected literature, and a tutorial on GATE) during the *Pilot Study*. Afterwards, VE-A implemented the discard patterns, under the supervision of NLP-E. The discard patterns are reported in SectION 3.4.
- **Patterns Application:** the patterns were applied by means of GATE.
- **Quantitative Evaluation:** the evaluation was performed by VE-A considering the annotations produced in the previous Output Annotation task. As in the previous iteration, the evaluation measures used are tp^D, fp^D, p^D, for each defect class i, and tp^R, fp^R, p^R.

The Qualitative Evaluation was not performed, since the goal was only to assess whether the discard patterns could improve the performance of the overall approach in terms of precision.

5.3.5 Large-scale Study – 4$^{\text{th}}$ Iteration

Fig. 5.5 gives an outline of the iteration. The iteration involved NLP-E and VE-A, and aimed to give an answer to RQ4.1. In the context of the case study, this analysis was performed to understand whether the dictionaries of SREE could be used to identify additional requirements defects that could not be identified with our patterns. The nature of the iteration was again *improving*, and consisted of the following tasks.

- **Research Questions:** the iteration was designed to compare the defect detection capabilities of SREE with respect to our patterns, and in particular, whether SREE actually allows to achieve higher values of recall. Therefore, RQ4, and its first refinement, RQ4.1, were defined by NLP-E and VE-A.
- **Patterns Definition:** under the guidance of NLP-E, each SREE dictionary, as reported in Section 3.5, was imported in GATE by VE-A as a separate Gazetteer. As mentioned, in our evaluation we apply all the SREE dictionaries, with the exception of the dictionary of the *weak* class (see Section 3.5).

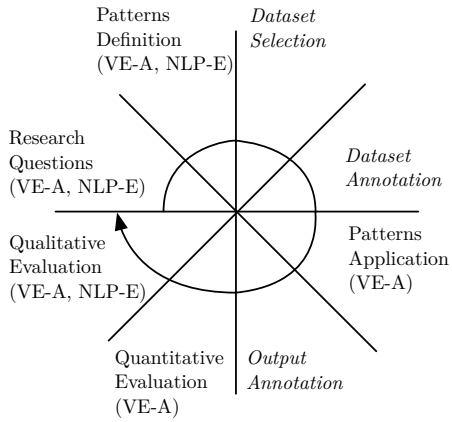

Figure 5.5: Structure of the Large-scale Study – 4^{th} Iteration.

- **Patterns Application:** the patterns implementing the SREE dictionaries were applied by VE-A by means of GATE.
- **Quantitative Evaluation:** the annotations considered for these requirements are those of VE2 only, from *Large-scale Study – 1^{st} Iteration*. Indeed, in this phase, we are interested in understanding whether the dictionaries of SREE applied altogether are able to detect defects, identified by VE2, that our patterns were not able to detect. To this end, SREE is compared with our patterns according to the values of tp^R, fp^R, fn^R, p^R, r^R. The patterns considered include the defect-detection patterns, plus the discard patterns.
- **Qualitative Evaluation:** this task was performed by VE-A with the support of NLP-E in a non systematic way, to observe defective requirements that could be detected by SREE.

5.3.6 Large-scale Study – 5^{th} Iteration

Fig. 5.6 gives an outline of the iteration. The iteration involved NLP-E, VE-A and VE1, and aimed to answer RQ4.2, RQ4.3 and RQ4.4. The iteration had an *explanatory* nature. Indeed, from the previous iteration, a high amount of false positive requirements was returned by SREE with respect to our patterns. This suggests that SREE may be less precise also at the level of defects. On the other hand, these false positive requirements may

conceal defects that were not considered by VE2. Therefore, it was decided to evaluate the potential degree of precision for the single defects identified by SREE. The tasks performed in this iteration are as follows.

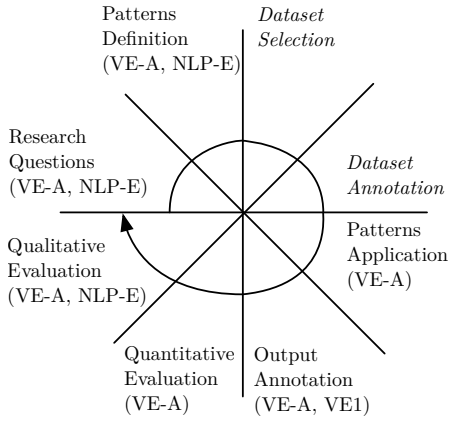

Figure 5.6: Structure of the Large-scale Study – 5$^{\text{th}}$ Iteration.

- **Research Questions:** NLP-E and VE-A considered that further investigation was required to answer RQ4, and its refinement RQ4.2, 4.3 and 4.4 were defined. Specifically, with RQ4.2 we wanted to assess which was the precision of SREE at the level of single defects, since low precision was observed at the level of requirements, after answering RQ4.1. Furthermore, we wanted to systematically study the specific defects that could be detected with SREE, and that could not be detected with our patterns (RQ4.3). With RQ4.4, we wanted to provide a qualitative evaluation of the false positive cases at the level of single defects.

- **Patterns Definition:** to evaluate the false positive cases issued by SREE at the level of defects, a selection of the SREE dictionaries was adopted for the analysis, which we call *SREE-reduced* (see Sect. 3.5). Indeed, we recall that, to address RQ4, this analysis was oriented to understand to which extent the SREE dictionaries could complement our patterns.

- **Patterns Application:** the patterns were applied by means of GATE.

- **Output Annotation:** a second annotation process was performed on the requirements marked as defective by at least one of the patterns derived from the dictionaries of SREE. VE-A and VE1 independently vetted the output derived from the application of *SREE-reduced*, and decided whether the defects issued were true positive or false positive cases. For each SREE defect class associated to one *SREE-reduced* dictionary, all the requirements labelled as defective according to the dictionary were considered. An exception is the *plural* class, for which a sample of 50 requirements labelled as defective was randomly chosen. The annotator agreement was estimated with the Cohen's Kappa, resulting in $k = 0.79$, indicating substantial agreement.

- **Quantitative Evaluation** the values of tp^D, fp^D and p^D were used for each single defect class of SREE considered.

- **Qualitative Evaluation:** true positive and false positive cases were analyzed and classified by VE-A, under the supervision of NLP-E, for each dictionary of *SREE-reduced*. True positives were analyzed to answer RQ4.3, while false positives were analyzed to answer RQ4.4.

Chapter 6

Results

6.1 Introduction

This chapter focuses on the results presentation and evaluation. The organization is made according to the RQs and the analyzed datasets.

6.2 RQ1, RQ2: Pilot Study

6.2.1 RQ1: What is the accuracy of the NLP patterns for defect detection?

In Table 6.1 we report the results of the different evaluation measures to establish the accuracy of the patterns. We see that, although the patterns for *anaphoric ambiguity* and *coordination ambiguity* are both based on shallow parsing, which normally has a typical accuracy of 90-95% [71], we achieve the objective of 100% recall. Similarly, for *modal adverbs* and *passive voice*, we achieve 100% recall, although these patterns employ POS tagging, which has an accuracy around 97% [83]. Two of the patterns that employ only lexical-based pattern matching, namely *missing reference* and *undefined term*, also achieve 100% recall. Lower values of recall are instead achieved for the patterns associated to *vague terms* (67.74%), *excessive length* (60.06%), *missing unit of measurement* (50%) and *missing condition* (97.05%).

Defect Class	tp^D	fp^D	fn^D	p^D	r^D
Anaphoric ambiguity	22	8	0	73.33%	100%
Coordination ambiguity	16	8	0	66.66%	100%
Vague terms	21	16	10	56.75%	67.74%
Modal adverbs	28	14	0	66.66%	100%
Passive voice	343	60	0	85.11%	100%
Excessive length	200	30	133	86.95%	60.06%
Missing condition	66	14	2	82.5%	97.05%
Missing unit of measurement	2	2	2	50%	50%
Missing reference	10	0	0	100%	100%
Undefined term	208	76	0	73.23%	100%
Requirements	tp^R	fp^R	fn^R	p^R	r^R
	106	59	14	64.24%	88.33%

Table 6.1: Results for single defects and requirements for the *Pilot Study*.

6.2.2 RQ2: Which are the cases of inaccuracy of the NLP patterns for defect detection?

Vague terms

By inspecting the ten false negative defects for vague terms, VE1 found that they were all due to the absence of the quantifier *some* in the list of vague terms provided by QuARS. Hence, requirements such as the following were not marked as defective by the pattern: *In case the boolean logic evaluates the permissive state, the system shall activate* ***some*** *redundant output* – which output shall be activated? The problem was resolved by simply adding the term *some* to the list of vague terms. Since also p^D was particularly low (56.75%), VE1 inspected the false positives and saw that they were due to domain-specific terms, namely ***raw*** *data*, ***hard*** *disk*, ***short****-circuit*, ***logical*** *or*, ***logical*** *and*, ***green*** *LED*. These terms were used to discard false positives in future analysis.

Excessive length

By inspecting the false negative cases for excessive length, VE1 saw that they were due to a limitation of the GATE Tokenizer. For nested bullet point lists, the Tokenizer considers each item as a separate sentence. Hence,

very long and deeply nested bullet point lists were not considered as sentences of excessive length. However, VE1 also argued that the length of a sentence, and the hard readability due to complex nested lists are different kinds of defects. Hence, she decided not to change the pattern for excessive length, and to consider the problem of nested lists as a defect that, at the moment, was left uncovered.

Missing unit of measurement

Concerning the two false negative cases for missing unit of measurement, VE1 observed that these were due to the presence of ranges of numerical values (e.g., *[4,20]*) without the specification of the unit of measurement. To address these cases, the pattern was adjusted.

Missing condition

The two false negative cases for missing condition appeared to be due to the presence of multiple *if* statements in the same sentence, with one *else* statement only, as in the following case: *"**If** the initialization starts, **if** the board is plugged in and **if** the operator has sent the running command the system shall start, **else** it shall go in failure mode"*. For requirements as the one presented, it is difficult to understand which specific *if* is covered by the *else* statement. Since the large majority of missing condition defects were identified (66 out of 68), and considering that the norm EN-50128 requires anyway a manual review [26], VE1 decided not to add additional rules for this defect class. It could be noticed that the specific defect could be detected also with techniques that check the *readability* of the text [34], an emerging topic in requirements [48], which is however outside of the scope of this paper.

False negative requirements

It is also useful to look at the values of false negative cases fn^R and recall r^R for the requirements. These 14 false negative cases not only include those already discussed, but also cases of defective requirements that could not be identified with our patterns – but which were annotated following the guidelines of the company. In particular, interesting cases are those in which we have *inconsistent requirements* (e.g., 1: *"The system shall accept only read access to file X"*; 2: *"The system shall accept read and write access to*

file X".) that violate guideline (f), which asks requirements to be *consistent*. Other cases are those for which we have problems of *testability* (guideline (b)), as in the case of *under-specified statements* (e.g., *"The system shall go in error mode when an internal asynchronism has been detected"*; asynchronism among which components?), or *incomplete statements* (e.g., *"The system shall make available its internal status"*; through which interface?). Finally, other cases are those associated to other defects of completeness of the requirements document, as in the case of requirements for which it is expressed only the best-case scenario, and not the worst-case (e.g., *"The system shall go at runtime state from power off state in 3 minutes in the best case."*; which is the requirement for the worst case?). Although some false negative cases were found, the evaluation of the patterns was considered successful in terms of recall by VE1. Hence, we decided to experiment the use of the patterns on a larger requirements dataset.

6.3 RQ1, RQ2: Large-scale Study – 1st Iteration

6.3.1 RQ1: What is the accuracy of the NLP patterns for defect detection?

In Table 6.3 we report the output of the patterns on the dataset in terms of defects identified (**D**), and in terms of defective requirements (**R**) – the other columns of the table will be discussed in Section 6.4.

We see that the majority of the defects are due to *passive voice*. This is in line with the results of [47]. The use of passive voice appears to be a sort of writing style of these requirements, since 824 out of 1866 (44%) include this defect. However, the most interesting – and disappointing – aspect comes from the evaluation presented in Table 6.2. The number of false positive requirements is extremely high, and the precision is only 5.81%. This value is comparable with the precision obtained through a random predictor (for which $p^R = r^R = 93/1866\% = 5\%$, see [3]). Hence, it appears not acceptable if the tool needs to be used in a real-world setting. Furthermore, also the value of r^R (85.39%) is slightly lower if compared with the one obtained in our preliminary study, for which $r^R = 88.33\%$.

tp^R	fp^R	fn^R	p^R	r^R
76	1232	13	5.81%	85.39%

Table 6.2: Results for the *Large-scale Study – 1st Iteration.*

6.3.2 RQ2: Which are the cases of inaccuracy of the NLP patterns for defect detection?

In this iteration, we give general observations of false negative cases, which impact the value of r^R, and false positive cases, which impact on p^R. Given the low value of p^R observed, the evaluation of false positives, and their classification was systematically performed during *Large-scale Study – 2nd Iteration* (Section 6.4).

False negative cases

As for the preliminary analysis, the false negative cases are due to requirements that include defects that were not considered by any of the patterns, but that violate one or more criteria adopted by the company. Interesting examples are requirements that do not fulfill the criterion of *testability* (guideline (b)) (e.g., *"The system shall be in continuous operation for 24 hours a day and 7 days a week"*); requirements that are not *feasible* (guideline (a)) (e.g., *"The core of the system shall use TCP/IP protocol in order to communicate with peripheral boards"* – in this case, this requirement was considered not feasible since the only communication protocol that was considered applicable was UDP); requirements that include *inconsistent* statements (guideline (f)) (e.g., *"The brake symbol shall be able to show the following colors: Green when the brake is not active, Grey when the brake is not active"*). Overall, these cases show that there is a variety of defects of semantic nature that are hardly identifiable with the applied NLP techniques – which focus on lexical and syntactic aspects –, and hence require a human expert to accurately assess them.

False positive cases

VE1 inspected the output of the tool, and saw that part of the false positive requirements were actually defective. For example, the following requirement marked as *accepted*, was evidently defective due to several vague terms (highlighted in bold): *"**Depending on** the technical or functional*

solution selected, there shall be time parameters in the control system, that the Purchaser shall be able to adjust during operation in order for the registration/deregistration to be made ***as effectively as possible****".*[1] In other terms, the intuition was that the initial annotation (by VE2) actually tolerated several linguistic defects, and marked as *rejected* only those requirements that appeared to include severe conceptual defects. When consulted, the VE2 observed that he also had an in-depth knowledge of the design of the requirements, which allowed him to disambiguate, or tolerate, certain defects. To assess how many of the false positive cases could be considered as linguistic defects from the point of view of a more strict annotator that did not have prior knowledge of the project, a second annotation process was performed to evaluate the false positive cases (*Large-scale Study – 2^{nd} Iteration*, Section 5.3.3).

6.4 RQ1, RQ2: Large-scale Study – 2nd Iteration

6.4.1 RQ1: What is the accuracy of the NLP patterns for defect detection?

Table 6.3 reports the results of this phase. For each defect class, the precision reaches an average value of **72.81%** for what concerns the number of defects (average of different p^D). Overall p^R resulting from the application of all the patterns together, raises from the 5.81% of Table 6.2, to **77.37%**.

6.4.2 RQ2: Which are the cases of inaccuracy of the NLP patterns for defect detection?

From the results presented in the previous section, there is still a significant amount of false positive cases that should be noticed. Part of these cases are *systematic*, and they can be discarded with additional patterns. Here we will discuss relevant examples of false positive cases for each class, specifically focusing on the systematic cases, and mentioning non-systematic ones when this is considered relevant.

[1] The requirement was not rejected since it was clarified by other subsequent requirements. This violates the guideline (c) that require requirements to be stand-alone, but the defect was not considered crucial.

Defect Class	**D**	**R**	tp^D	fp^D	p^D
Anaphoric ambiguity	391	342	198	193	50.64%
Coordination ambiguity	261	215	190	71	72.80%
Vague terms	857	580	392	465	45.74%
Modal adverbs	478	379	333	145	69.67%
Passive voice	1317	824	888	429	67.43%
Excessive length	13	13	13	0	100%
Missing condition	185	147	127	58	68.65%
Missing unit of measurement	0	0	0	0	-
Missing reference	2	1	2	0	100%
Undefined term	61	57	49	12	80.33%
Average					**72.81%**
Requirements			tp^R	fp^R	p^R
			1012	296	**77.37%**

Table 6.3: Results for the *Large-scale Study – 2nd Iteration.*

Anaphoric ambiguity

The majority of the false positive cases for anaphoric ambiguities are due to the usage of the pronoun *it* in its impersonal form, especially in the expression *"**It** shall be possible [...]"*. This expression, and its variants – *it shall also be possible*, *it should be possible*, etc. – is often used as a preamble in the requirements of the company. These cases are systematic sources of false positives, and appropriate patterns can be defined to discard them.

The remaining, non-systematic cases, include situations in which the referent of the pronoun is disambiguated by the context, as in the following requirement: *"**Trains** that arrive on the automatically controlled **stretches** shall continue to be directed to **their** correct destinations"*. The pronoun *their* is clearly referred to the trains, since only trains and not stretches have a destination attribute, but the pattern P_{ANA} recognises two nouns (i.e., *trains* and *stretches*), to which the pronoun may refer. To detect these non-systematic false positive cases, machine learning approaches, such as those applied in Yang et al. [118] should be applied.

Coordination ambiguity

The false positive cases for coordination ambiguity, in line with those identifyied in Chantree et al. [28], are non-systematic cases, in which the potentially ambiguous fragment is disambiguated by the context. For example, consider a requirement such as: *"It shall be possible to print out the **whole timetable or part** of it"*, in which the fragment in bold is detected by means of pattern P_{CO_2}. In this requirement, it is clear that the adjective *whole* refers solely to the noun *timetable*. Similarly, consider the following requirement: *"A train can consist of one, **two or three cars for services between Station A and** Station B"*, in which the fragment in bold is detected by means of pattern P_{CO_1}. Also in this case, it is clear that the conjunctions *and* and *or* refer to their nearby terms. However, these cases are non-systematic, and can hardly be detected by means of rule-based patterns. Other heuristics, such as those presented by Chantree et al. [28] should be used.

Vague terms

A large number of false positive cases (465) is identified for this defect. These cases can be partitioned into the following typical situations:

1. **Lexical Ambiguity:** the vague term is *lexically ambiguous* [12]. For example, the term *light*, considered as adjective, is vague, but when playing the role of noun, as in the requirement *"Yellow Stop **lights** do not have to be monitored"*, is not vague. Cases such as the one in this example can be systematically detected by applying POS tagging, and considering a term as vague only if it plays the role of adjective. A similar systematic case, which can be addressed with the same approach, is the case of the term *sound*, as in the requirement fragment *"Blue arrows, and their associated **sound**, shall not be presented to the driver [...]"*;

2. **Domain-specific Term:** a vague word is part of a domain-specific multi-word term, as for the term *distant* of the following example: *"The operator shall use "**distant** signalling distance" to apply the brake"*. Another interesting case is the term *near* in the typical railway expression ***near** miss* – indicating an unplanned event that has the potential to cause, but does not actually result in human injury. To discard

these cases, techniques for multi-word term identification [15] may be applied. Otherwise, a list of stop phrases to be ignored can be defined based on the false positives identified. In our case, this second option will be chosen;

3. **Accepted Expressions:** the term *possible* is used in the phrase *"It shall be **possible** [...]"*, considered an accepted requirement preamble within the company, as previously mentioned;

4. **Internal Clarification:** the vague term is later clarified with the specification of numerical quantities, as in the following fragment: *"[...] for a **short** stretch (maximum 3 meters) on tramcars [...]"*. In this case, the term *short* is clarified by the phrase *maximum 3 meters*;

5. **Domain Clarification:** the vague term is clarified by the domain, as in the case of the term *adjacent* in the following requirement: *"In the case of a train passing **adjacent** to a level crossing, each train shall register its own priority"*. Physical adjacency among elements in the railway line is a well defined concept in the domain. However, we found also cases in which the term *adjacent* was considered vague, as in the fragment ***adjacent** track*, in which it is not specified whether the referenced track is on the left-hand or on the right-hand side.

The first three cases can be systematically detected. By contrast, the last two are hard to be detected in a systematic manner. Indeed, apart from case 4, patterns that check numerical quantities nearby the vague term can be defined, but it is not sure how "nearby" should be intended. In addition, these false positive cases are rather easy to discard, and, for this reason, patterns will not be defined to address these cases.

Modal adverbs

For modal adverbs the great majority of the false positive cases are due to the usage of the terms *manually* and *automatically*. These terms are not considered defective in the context of the requirements, since they are used to distinguish between the duties of the system (*automatically*), and the duties of the operator (*manually*). The remaining false positive cases are due to the usage of the term *only*. Consider the following requirement: *"In case there are two coupled points the system shall select **only** the point with identifier equal to 1"*. Here, the term *only* is used to distinguish between

multiple choices. Since the term *only*, especially when misplaced, may be ambiguous [12], the usage of this term cannot be regarded as a systematic source of false positives. An exception in this sense is the occurrence of *only* in the fragment *information purposes* ***only***, an expression frequently used in the requirements. When *only* occurs in this fragment, it can be considered as a systematic false positive case.

Passive voice

For the false positive cases of this class, we can identify four typical situations, listed below:

1. **Irrelevant Actor:** the actor performing the action is sometimes considered as not relevant, as in the requirement: *"Air conditioning units* ***are installed*** *in some of the technical equipment areas"*. This sentence provides information about a certain environment, and the reader does not need to know who installed the air conditioning units. Similar cases are those in which the passive voice *is connected*, or *is disconnected* are used;

2. **Implicit Actor:** the actor – often, the system or the operator – can be inferred from the context, as in *"Error signals shall* ***be displayed*** *in the MMI above the speedometer"* (the actor is the system), or *"The emergency brake restore shall be performed with the green signal"* (the actor is the operator);

3. **Explicit Actor:** the actor is actually expressed, as the passive voice is used in conjunction with prepositions (e.g., *by*, *from*), after which the actor is clarified, as in the following example: *"All views shall* ***be developed*** *by the Supplier in consultation with the Purchaser"*;

4. **Intransitive Verb:** the passive voice is used with intransitive verbs, such as the verb *log-in* (e.g., *"If a workstation fails and the operator* ***is still logged in*** *[...]"*).

The first two cases cannot be identified systematically. However, the latter two can be, in principle, identified with appropriate patterns, which detect the prepositions *by* and *from* in conjunction with passive voice (case 3), or which identify intransitive verbs (case 4). However, since the number of these cases was considered negligible, these two discard patterns were not implemented.

Missing condition

False positives for this defect class occur when the term *if* is not used to express a condition over the system behaviour. For example, the requirement *"The system shall check if there is a train in the route"* does not require an *else* statement. In other cases, the *else* condition is expressed in another requirement (e.g., *1: "If the precondition satisfies all initialization check the system shall set its internal state to running"; 2: "In case an initilization check fails, the system shall set its internal state to failure"*). These cases can hardly be detected with patterns, and require the knowledge of the context to be disambiguated.

Undefined term

The entirety of the false positive cases for undefined terms are due to the identification of units of measures, or known acronyms in their plural forms (e.g., kVA, dB, LEDs). A list of known unit of measurement and known acronyms can easily be defined to discard these cases.

6.5 RQ3: Large-scale Study – 3rd Iteration

6.5.1 RQ3: What is the precision of NLP patterns for defect detection when complemented with discard patterns?

Table 6.4 reports the results obtained when applying the discard patterns. To ease the comparison with Table 6.3 we report, for the improved rows, the previous results in italics. We notice a substantial increase, in terms of p^D. In particular, compared with the results of Table 6.3, p^D increases by 22.69% for anaphoric ambiguity, by 24.89% for vague terms, by 11.75% for modal adverbs, and by 17.67% for undefined term. Overall, the average p^D raises to 81.36% (an increase of 8.55% with respect to Table 6.3), and also p^R increases by a non negligible 5.79%. This increase of precision saves, in principle, a considerable amount of checks to the Verification Engineer, who has to vet a lower number of requirements. More specifically, if we look at the values of fp^R in Table 6.4 (296) and in Table 6.3 (205), we see that 91 requirements do not have to be vetted after the introduction of the discard patterns.

Defect Class	**D**	**R**	tp^D	fp^D	p^D
Anaphoric ambiguity	270 *391*	251 *342*	198 *198*	72 *193*	73.33% *50.64%*
Coordination ambiguity	261	215	190	71	72.80%
Vague terms	555 *857*	384 *580*	392 *392*	163 *465*	70.63% *45.74%*
Modal adverbs	409 *478*	330 *379*	333 *333*	76 *145*	81.42% *69.67%*
Passive voice	1317	824	888	429	67.43%
Excessive length	13	13	13	0	100%
Missing condition	185	147	127	58	68.65%
Missing unit of measurement	0	0	0	0	-
Missing reference	2	1	2	0	100%
Undefined term	50 *61*	47 *57*	49 *49*	1 *12*	98% *80.33%*
Average					**81.36%** *72.81%*
			tp^R	fp^R	p^R
Requirements			1012 *1012*	205 *296*	**83.16%** *77.37%*

Table 6.4: Results for the *Large-scale Study – 3rd Iteration.*

As noticed in Section 6.4, the majority of the remaining false positive cases cannot be systematically detected, and require the judgment of a human assessor. These types of situations can be potentially addressed through statistical techniques (e.g., [28] and [118]). Typical examples have already been reported in Section 6.4.

6.6 RQ4.1: Large-scale Study – 4th Iteration

6.6.1 RQ4.1: What is the accuracy of SREE with respect to the NLP patterns for defect detection complemented with discard patterns?

Table 6.5 compares the performance of the SREE dictionaries and our patterns against the initial annotations of VE2. From the table, we see that SREE outperforms our patterns by 11.24% in terms of recall on the requirements, while its precision is 0.79% lower. Hence, SREE dictionaries may contain terms that help to identify defective requirements that were not detected through our patterns, and were therefore part of the false negative cases issued. On the other hand, a 0.79% gap in terms of precision, implies that 351 additional false positive requirements (fp^R) are generated by SREE with respect to our patterns. The obtained results confirm that SREE meets its aim to reach 100% recall, even at the cost of a lower precision.

Let us first analyze the false negative cases of our patterns that are detected through the SREE dictionaries, and then we will investigate the issue of precision.

Tool	tp^R	fp^R	fn^R	p^R	r^R
SREE	86	1492	3	5.45%	96.63%
Patterns	76	1141	13	6.24%	85.39%

Table 6.5: Results for the *Large-scale Study – 4^{th} Iteration*, SREE vs Patterns.

A representative example of the requirements detected through SREE, and not with our patterns, is the following one: *"**Normal** and abnormal changes in the status of the Facility shall warrant special treatment [...]"*. VE2 in his initial annotation rejected the requirement, and stated that *"Normal and abnormal changes" are not defined and shall be agreed.* SREE iden-

tifies this requirement as defective, since its dictionary for the *vague* class includes the term *normal*. On the other hand, it is worth noticing that SREE dictionaries do not include the term *abnormal*, which is also a defective term, according to the statements of VE2. A similar case is the following requirement: *"When the driver follows indications as to the* ***maximum*** *speed limit the Facility shall not cause braking that produces jolty and uneven driving"*. The requirement was marked as rejected, because it does not fulfill the criterion of *testability* (guideline (b)). This is due to the presence of the adjectives *maximum*, *jolty* and *uneven*. Here, the SREE dictionaries correctly detects the vague term *maximum*, but do not detect the defective terms *jolty* and *uneven*. Hence, although including the SREE dictionaries in our patterns can help to increase the recall, novel terms may be needed in the future to address other, previously unseen, defects.

Another interesting aspect concerns other requirements that:

1. are marked as defective by SREE;
2. are marked as rejected by VE2 in the initial annotation;
3. the cause of the rejection is not the defect indicated by SREE.

Exemplary cases are mostly related to the usage of plurals, which have 3377 occurrences in 1250 requirements (see Table 6.6, discussed in Section 6.7). An example is the following requirement: *"It shall be possible to turn* ***trains*** *at the intended turning points without restriction"*. SREE identifies the source of the defect in the plural term *trains*. However, VE2 marked the requirement as rejected because it violates the criterion of *testability* (guideline (b)). This is due to the expression *without restriction*, which does not allow the definition of a finite number of tests to verify the requirement.

Of course, there are entire defect classes considered by our patterns, which are not detected by the dictionaries of SREE, such as *passive voice*, *missing condition*, *missing reference*, *missing unit of measurement*, etc. Given these observations, SREE dictionaries can be considered as complementary to our patterns. Still, SREE and our patterns altogether are insufficient to detect all the potential defects, and should be complemented with additional terms (e.g., *jolty* and *uneven*).

As mentioned, a high amount of false positive requirements was returned by SREE with respect to our patterns. This suggests that SREE may be less precise also at the level of defects. On the other hand, these false positive

cases may conceal defects that were not considered by VE2 during the initial annotation.

Therefore, it was decided to evaluate the potential degree of precision for the *single defects* identified by SREE. An analysis of the false positive cases was performed at the level of the single defects, similar to the one applied on the output of our patterns during *Large-scale Study – 2nd Iteration*.

6.7 RQ4.2, RQ4.3, RQ4.4: Large-scale Study – 5th Iteration

6.7.1 RQ4.2: What is the precision of SREE for the defects in its scope?

Table 6.6 reports the results of the analysis of the false positive defects. The average value of p^D is 11.29%, which indicates that a large amount of false positive cases are issued, which is much lower, compared with the 81.36% obtained through our patterns (Section 6.6)[2].

6.7.2 RQ4.3, RQ4.4: Which additional defects can be identified with SREE, and which are the false positive cases?

Below, we provide an analysis of the true positive and false positive cases.

Continuance

The continuance class includes terms that, when present, indicate a reference between a statement and a previous one (e.g., *in addition*, *in particular*), or a subsequent one (e.g., *following*, *below*). True positive cases occur when the referred statement is absent, and, therefore, the requirement is incomplete. The number of these cases is not negligible, and are all associated to the terms *as follows* and *below*. False negative cases occur anytime the referred statement appear in the requirement. These cases occur especially when the

[2]The value of p^R that considers the analysis of the false positive cases for the SREE dictionaries cannot be provided, since we analyzed only a subset of the defects for the *plurals* class. However, the average value of p^D gives a clear indication of the precision of SREE at the level of defects.

Defect Class	**D**	**R**	tp^D	fp^D	p^D
Continuance	181	155	41	140	22.65%
Directive	123	102	0	123	0%
Optional	102	92	26	76	25.49%
Incomplete	32	31	2	30	6.25%
Plural	3377	1250	*6*	*125*	*4.58%*
Quantifier	308	264	25	283	8.11%
Vague	931	665	111	820	11.92%
Average					**11.29%**

Table 6.6: Results for the *Large-scale Study – 5^{th} Iteration.*

referent is a *previous* statement, and the terms *in addition* and *in particular* are used.

Directive

The directive class includes terms that indicate the presence of a reference to an element in the requirement (e.g., *e.g.*, *i.e.*) or in the document (e.g., *figure*, *table*). As for the continuance class, true positives may occur when the referred element is absent, while false positive occur when the referred element is present. In the considered requirements, no true positive case was identified.

Incomplete

The incomplete class includes terms that may indicate a form of internal incompleteness of the requirement (e.g., *TBD*, *to be defined*). The dictionary of this class raises a limited number of defects (32). Indeed, expressions as *TBA* and *TBD* do not occur in the requirements, and the great majority of the false positive cases occur when the term *in addition* is used – a term that is included also in the continuance class. Another typical case of false positive is the following requirement fragment: *"[...] functions shall be performed in a secure way,* ***as defined*** *in the CTC security requirements"*. Here, the requirement is not incomplete, since it refers to another document in which the required information is available. Instead, the cases evaluated as true positives are similar to the following one: *"All alarms [...] shall be shown in track plan views* ***as specified*** *above"*. Here, the problem is with the term

above, rather than with the term *as specified*, since the involved subjects could not find the referred information in the document. However, the defect was considered a true positive by the involved subjects, since the tagging of the term *as specified* allowed the identification of the defect.

Optional

The optional class includes terms that indicate subjective optionality. Anytime an expression such as *if needed*, *if necessary*, *if appropriate* occurred, this was marked as a true positive case. Similarly, many true positive cases occur with the term *either*, as in the requirement: *"A cable run shall be laid on **either** side of the track"*.

False positive cases occur when terms such as *either*, or *neither*, are used in the expressions *either [...] or* , or *neither [...] nor*. Another typical, systematic false positive case occurs with the usage of the term *in case of*, when this expresses a condition that depends on actions that are external to the system, as in: *"**In case of** a restart of the system [...]"*.

Plurals

Plurals are ambiguous when they are used to describe a property of a set, and it is not clear if the property is that of each element or of the whole set [13], as in the requirement fragment *"[...] **printers** shall have a sound [...]"*. In the considered sample of 50 defective requirements for the plurals class, cases such as this one were extremely rare. A large amount of false positive cases was instead observed.

Typical false positive cases belong to two classes. The first class includes lexically ambiguous verbs used in third person singular form (e.g., *means*, *passes*, *leaves*). The second class includes cases in which the plural term indicates a set of objects or subjects, such as *trains*, *boards*, *tracks*, *operators*, etc., and it is clear from the context that the requirement refer to all the elements in the set, as in the following fragment: *"Control **orders** that are executed by **operators** shall be registered [...]"*. Since the requirements are high-level system requirements, the use of plurals in the form exemplified is rather common, and accepted by the involved subjects.

Quantifier

Quantifiers that express quantities in a vague form such as *few*, *little*, *many*, are included in the quantifier class. The occurrence in the requirements of these vague terms was always considered by the involved subjects as a true positive defect. False positive cases are due to universal quantifiers, such as *all* or *any*. Indeed, although, as noted by [13], these terms may be source of ambiguity (e.g., *"All lights have a switch"* – one switch for each light, or a common switch?), in the considered requirements these terms are not used in ambiguous forms. Instead, non ambiguous requirements fragments such as the following are common: *"[...]* ***all*** *equipped tramcars [...] shall be able to operate on* ***all*** *track networks [...]"*.

Vague

The vague class includes additional terms with respect to the *Vague* dictionary of our patterns. Part of these terms appear to be useful to identify extremely vague requirements that were not identified through our patterns. A representative example is the following requirement, which includes two vague expressions: *"Communication shall* ***as far as*** *possible be redundant, with separate cable runs, for the* ***various*** *communication links."* False positive cases are mainly due to the usage of terms such as *also*, and *but*, which are rather frequent in the requirements, but are not considered sources of vagueness by the involved subjects. Indeed, the presence of these terms sometimes indicates that a requirement includes more than one statement, as in the fragment: *"The [...] system shall not be reused* ***but*** *shall be dismantled [...]"*. However, since the considered requirements are high-level system requirements, the VEs accepted these situations.

6.8 General Observations

From this analysis, we see that additional defects, which were not previously considered by the involved subjects, are actually detected thanks to SREE. This confirms that SREE may play a complementary role with respect to our patterns. On the other hand, the value of precision of SREE, at the level of defects, is poorer than the precision of our patterns (i.e., a larger number of false positive cases is issued). However, this numerical difference should be considered with care. Indeed, there are two main reasons that explain

and justify this result:

1. **SREE Philosophy:** the philosophy of SREE, as we interpret it through its usage, is to identify terms that, when present, may also indicate that a defect *may* be present. If the defect is not present, it is easy for the analyst to vet the requirement. Representative examples in this sense are the terms in the *continuance* class: terms such as *as follows* and *below* were judged as particularly useful by the VEs to detect incomplete requirements, although their occurrence was not always associated to a defect. The VEs said that vetting the false positive cases was straightforward for this class. Hence, the low value of precision was sufficiently counter-balanced by the usefulness of the terms included in the defect class;

2. **Subset of SREE:** a subset of SREE dictionaries was used, instead of the whole SREE. Hence, the comparison cannot be considered complete. However, our goal in this case study was not to identify the best tool for defect detection, but rather to investigate whether additional defects could be found by means of the SREE dictionaries. This goal also mitigates a potential annotators' bias that may have occurred in the evaluation of the false positives of SREE dictionaries. Although this bias cannot be totally eliminated in the context of our case study, our patterns, as well as the SREE dictionaries, are available for the research community, who can independently compare the different strategies.

6.9 Threats to Validity

In this section, we discuss threats to validity according to the structure recommended in [103].

6.9.1 Construct Validity

Objective and widely used metrics (i.e., precision and recall) were used in this work to assess the accuracy of the adopted NLP technologies. To derive measures of precision and recall, subjective evaluations were performed by VE-A, VE1, and VE2 during the Dataset Annotation and Output Annotation tasks. In the Pilot Study, only VE1 annotated the dataset, and no

countermeasure was taken to assess the validity of the annotation, given the preliminary nature of the study. Similarly, in the *Large-scale Study – 1st Iteration*, only VE2 annotated *D-Large*, and the same annotation was used for the *Large-scale Study – 4th Iteration*. On the one hand, also in the real-world context of the company, requirements review is performed by a single subject, and the subjectivity threat can be considered as partially mitigated by the realism of this annotation. On the other hand, the Output Annotation on *D-Large*, was independently performed by VE1 and VE-A, and the inter-rater agreement was computed by means of the Cohen's Kappa. The agreement resulted in $k = 0.82$ (almost perfect) for *Large-scale Study – 2nd Iteration*, and $k = 0.79$ for *Large-scale Study – 5th Iteration* (substantial). Therefore, we believe that the threat is further mitigated by these measures of agreement, at least for those requirements that were produced as output by the NLP patterns. Therefore, construct validity threats are mitigated for *Large-scale Study – 2nd, 3rd and 5th Iteration*, while they are only partially mitigated for *Pilot Study*, and *Large-scale Study – 1st and 4th Iteration*, in which only one subject was involved in the annotation process.

6.9.2 Internal Validity

The main threats to the internal validity of the study are due to the personal objectives of the involved subjects, which may have had an impact on the results. Indeed, the annotations performed by VE1 and VE-A in the tasks in which they were involved may be biased by their need to show that the implemented patterns were successful, hence annotating as defective also requirements that were not. In the case of the *Pilot Study*, this threat is mitigated by the fact that the annotation was performed *before* applying the patterns, and hence without exactly knowing their output. In the *Large-scale Study* iterations, the threat is mitigated by (a) by the pragmatics of the case study, and (b) the independent Output Annotation process performed. Indeed, since VE1 works as VE in the company, she is also interested on improving her job, besides showing that the implemented technology is effective. VE-A may be less keen to this type of integrity, since she is not part of the company anymore. However, since the Output Annotation task was always performed independently by the two VEs, we argue that this threat is sufficiently controlled. Furthermore, as noticed in Section 6.8, since this threat cannot be totally mitigated, we share our patterns so that other researchers can apply them to their contexts, and check their effectiveness. It

should be noted that the annotations of VE2 are not subject to this threat, since they were performed before this work was conceived. Validity issues related to the discrepancies between the annotations performed by VE2 compared to the ones of VE-A and V1, are discussed in Section 6.9.3, since we argue that the annotations represent different contexts, from which different generalization criteria may apply.

Another internal validity threat is associated to the tool-suite initially used by VE1 in the *Large-scale Study – 1st and 2nd Iterations*, to compute the data for the case. Indeed, she used an internally developed tool on top of GATE to produce the results. To mitigate potentially unsound manipulation of the data by this prototype tool, part of *Large-scale Study – 1st and 2nd Iterations*, were replicated by VE-A, with the support of GATE only. Discrepancies in the results were observed, and root causes were analyzed. The rest of the analysis were performed by means of GATE only. Since GATE is a widely used tool – see the list of companies using GATE[3] and, e.g., [6] and [38], for relevant scientific works in which GATE was employed –, we believe that the results produced with its support are correct.

6.9.3 External Validity

Our discussion on the external validity of the study is loosely based on the principles of case-based generalization [115], and of similarity-based generalization [60]. Specifically, we describe the main architectural aspects of our study, i.e., domain, requirements, subjects, that can be considered as a term of comparison for other studies. In this way, other researchers and practitioners can reason by analogy, and possibly profit from our results [60].

- *Domain:* our study covers a company of a specific domain, i.e., the railway domain. In Europe, railway companies have to follow the general guidelines of the CENELEC norms [26], and their work practices at process level can be considered comparable. Furthermore, the railway domain is characterized by a limited number of suppliers, who often deal with the same customers – i.e., the national or private railway companies, who provide infrastructure, and services to passengers. This increases the homogenisation of processes and, in part, requirements documents. While we cannot generalise our results for any type

[3] `https://gate.ac.uk/commercial.html`

of domain, we argue that similar results may be obtained in other railway companies. On the other hand, the following limitations to the external validity of our results shall be considered.

- *Requirements:* the requirements considered in the study have been selected by VE1, with the support of the company, as benchmarks to represent typically defective requirements of the firm. VE1 and VE-A admits that, depending on the subjects involved in the production of requirements, the documents may have different degrees of quality, and the documents belonging to the study are requirements of lower quality than average. Furthermore, along the process, system requirements such as those analyzed are normally refined into lower level requirements. Hence, the results produced shall be considered representative for (a) system requirements, (b) requirements with a poor degree of quality. Since the requirements concerns several types of railway signalling systems, they are sufficiently representative of the types of product developed in railways.
- *Subjects:* Overall, three VEs were involved in this study. The sample is limited, but it shall be considered that all the VEs are normally subject to the same company practices and process, and can therefore be considered representative VEs for the company. Considering the characteristics of the railway domain mentioned above, they can be considered, to a certain extent, also representative of VEs in railways. The task of annotating is all but simple; the high number of requirements to be tagged and the repetitiveness of the task may cause a loss of focus after a while. Discrepancies were observed between the annotations performed by VE2 on *D-Large* during *Large-scale Study – 1st Iteration*, and the annotations on the output of the patterns performed by VE1 and VE-A, during *Large-scale Study – 2nd Iteration*. In principle, the discrepancies may be associated to the different degree of experience of the subjects. VE1 and VE-A had 3 and 2 years experience, respectively, while VE2 had 10 years of experience. We believe that the discrepancies observed are only partially associated to the experience. Instead, we believe that the discrepancies are due to the differences in terms of contextual knowledge, and goals. VE2 had in in-depth knowledge of the project that allowed him to disambiguate, or tolerate, certain defects, and focused on severe conceptual problems. Instead, VE1 and VE-A did not have any prior knowledge

on the project, and focused on linguistic aspects, given the research-based, exploratory nature of their work.

For these reasons, the different iterations have different degrees of external validity – notwithstanding the construct validity threats already discussed. Specifically, *Pilot Study*, and *Large-scale Study – 2^{nd}, 3^{rd}, and 5^{th} Iterations* can be considered representative for those cases in which the annotation is performed by VEs who do not have prior knowledge of the project of the requirements, and focus on *linguistics* defects. Instead, *Large-scale Study – 2^{nd} and 4^{th} Iterations* are representative for those cases in which the annotation is performed by a VE who has an in-depth knowledge of the project, and focuses on *conceptual* defects.

As mentioned, our results can be generalised to other domains only to a limited extent. Our work focusses on a single railway company, and railway companies have a well-defined processes to follow, that is not shared by other context. The degree of rigour of the railway process is comparable to the one employed in the avionic sector, in which the DO-178C norm applies for software development [102]. However, the products developed in railways and avionics are highly different, and use domain specific terminology. Many of our patterns are domain independent, but, given the large variability of NL, and of domain specific NLs, the generalisation of our results to other domains requires further research.

6.9.4 Reliability

The results provided are mainly quantitative, and we argue that a common understanding on their meaning was achieved when the values of precision and recall had to be computed. Concerning the qualitative data, these were provided by the VEs and were refined with the support of NLP-E. We argue that this interaction increased the reliability of the qualitative results.

Chapter 7

Lessons learned and future research issues

7.1 Introduction

In this chapter we present a set of lessons learned from our experience and the future research directions emerged.

7.2 Domain-customisable NLP Tools

Our experience shows that NLP technologies are available for requirements analysts with limited NLP training, and that these technologies can be proficiently used for the detection of several typical requirements defects. Rule-based NLP patterns tend to generate large numbers of false positives [28, 118]. If the results come from a tool that the requirements analyst cannot control, the analyst is likely to distrust the tool. Instead, if the analyst understands the inherent principles of the tool – and implementing the tool is a proper way for understanding its principles –, they can understand its weaknesses and use it at its best. Furthermore, it is also important that domain experts develop the tools since, to reduce the amount of false positive cases, tailoring the patterns for the specific needs of the domain is required. If the VEs implement the patterns, they can customise them according to the language used in the domain to take into account terms such as *raw data*, *hard disk* (Section 6.2), and phrases such as *it shall be possible* (Sec-

tion 6.4). The introduction of the discard patterns, to remove systematic false positive cases, allowed an increase of the average p^D from 72.81% to 81.36% (Section 6.4). It should be noticed that, if a company defines a set of patterns to be applied for defect detection, a maintenance cost should be taken into account since, as any software tool, patterns may need to evolve. While for COTS tools the software house who develops them takes care of their evolution, and maintenance costs, the company has to take the burden of maintenance in case of internally developed tools.

7.2.1 Requirements language counts

Looking at the large number of passive voice defects in *Large-scale Study – 2^{nd} Iteration*, it appeared that the use of passive voice was a form of writing style. As a consequence, the patterns generated a large number of detected defects (i.e., 1317). This tells us that, to effectively use NLP, one cannot simply implement appropriate defect detection patterns: one should change also the language adopted in the requirements, to make it more error free, so that the VEs can focus on a smaller amount of defects. For this reason, we argue that NLP tools should be firstly used by the requirements engineers, to limit the amount of poor writing style, and only *afterwards* by the VEs. However, this is not always practicable, especially in those cases in which requirements are produced by the customer, and assessed by the company who has to develop the product. As acknowledged by the company, the requirements considered in this study are particularly rich in defects, also with respect to other requirements of the company. However, it is worth noting that, after taking inspiration from the work of Terzakis and Gregory [110], VE1 is currently involved in a mentoring program within the company, to educate the requirements engineers towards the production of higher quality requirements.

7.2.2 Requirements level counts

During the analysis of the false positive cases of SREE, a large number of plurals (3377) was identified, which were tolerated in most of the cases. Furthermore, also the presence of conjunctions such as *also* and *but*, which indicate non-atomic requirements, was tolerated in these requirements. This was motivated by the level of the requirements. The considered dataset was composed of high-level system requirements for which, according to the VEs,

a certain degree of generality can be accepted. These requirements will be refined into lower-level technical requirements, along the development lifecycle, for which a greater degree of precision is expected. Also the cost of a defective requirement is different depending on its level: high level defects have a greater impact on the lifecycle, while low level defects have a smaller one. Specifically, the detected defects generate change requests for the product that have a different management cost (in terms of hours of rework and required money) depending on the impact on the development process. As we notice in a recent work [48], this suggests that requirements at different degrees of abstractions may need different treatments.

More specifically, patterns to check presence of plurals, as well as *also* and *but* conjunctions, may need to be applied for low-level requirements, while they do not need to be used for high-level ones.

7.2.3 Validation criteria count

Considering the *Large-scale Study – 1^{st} and 2^{nd} Iterations*, we saw that a large part of the false positive cases encountered in the *Large-scale Study – 1^{st} Iteration* could be associated with a weaker validation performed by VE2. In fact he was not focused on linguistic defects, but more on severe conceptual defects, also given his in-depth knowledge of the project. For this reason, the results obtained in terms of precision were extremely poor. When changing criteria, p^R varied from 5.81% to 77.37% (Section 6.4). Hence, to perform an appropriate validation of rule-based NLP patterns, it is advisable to start from an annotated dataset that has been defined *knowing* the classes of defects that will be checked by the patterns, and specifically stating that the focus is on *linguistic* defects. Otherwise, the results might be misleading. This observation might appear counter-intuitive, since we suggest to adapt human operators to tools. However, when dealing with the complexity of NL, we argue that the adaptation between humans and NLP tools should be bi-directional.

7.3 NLP is only a part of the answer

In our large-scale study, several false negative cases occurred, which can hardly be detected with NLP. These are examples of conceptual defects that require a human with knowledge of the domain and of the specific project.

In recent years, NLP technologies have seen radical progress [64]. Linguistic tasks at the semantic level such as question-answering became possible. However, the *pragmatic* nature of ambiguity [53], and the contextual knowledge needed to understand a requirements document, make the problem of automatic defect detection in requirements hardly solvable with current technologies. Therefore, NLP represents only a part of the answer to defect detection, while the other part is represented by human analysts with domain expertise. It should also be considered that relying on a tool for defect detection may also change company practices, in that a VE may rely too faithfully on the tool's output. This reasonable hypothesis requires further empirical investigation, but its potential implications should be considered when introducing an automated tool to support practices that are normally manually conducted.

7.4 Statistical NLP vs Lexical techniques

Our patterns make use of POS tagging and shallow parsing, which are statistical techniques that can hamper the objective of 100% recall [11]. However, in Section 6.2, we showed that 100% recall was achieved for those patterns that used these techniques, while it was *not* achieved for the pattern adopted for *vague terms*, which uses a lexical based approach. Hence, we argue that the argument in favour of a "dumb" lexical-based defect detection approach instead of an approach that leverages statistics-based techniques [11] should be partially revised. If one wants to use lexical-based detection approaches, then one should use only defect indicators belonging to closed word classes (e.g., pronouns, conjunctions). Instead, if one uses open word classes (e.g., adjective, adverbs), the problems are not different from those that *might* emerge with statistical techniques. As statistical techniques may fail, also lists of dangerous adjectives and adverbs may fail, because they might not include words that were not considered until they appear in the requirements (e.g., the word *some*, as noted in Section 6.2, or the words *jolty* and *uneven*, as noted in Section 6.6).

7.5 Implication for practice and future research

In this section we draw conclusions presenting the implication for practice and introducing some future research directions.

7.5.1 Implication for practice

Overall, the experience was considered extremely useful by the company. In particular, VEs say that, after studying the literature on defect identification, and implementing the patterns, also their way of judging requirements defects became stricter. This is also one of the reasons why requirements previously marked as *accepted*, were afterwards *rejected* in the former annotation process. This implies that, while on the one hand tools have to be adapted to company practices, also company practices can be modified by tools. In our study, we also observed that an increase in the performance can be obtained by *incrementally* tuning the patterns based both on the defects encountered in practice, and through the inclusion of other defect-detection criteria from the research literature – in particular, the SREE dictionaries. Therefore, regardless of the NLP technologies used to detect defects, technologies need to be adapted to the specific language of the company, to be fruitfully used.

It should also be observed that, based on the lessons learned from the current study, one of the participating VEs is now involved in a mentoring program within the company, oriented to teach requirements engineers how to write linguistically clear requirements. The idea is that requirements engineers should be aware of linguistic defects, so that the work of VEs can focus on conceptual ones. In this sense, we argue that, by working with NLP techniques for defect detection, one can have an effect also in terms of organizational learning. Furthermore the company has recently begun to use NLP techniques to detect defects also in the application condition of their products. Another relevant implication for practice concerns the complementary role of NLP techniques, and human analysis. We observed that part of the conceptual defects present in the requirements could not be detected with the patterns, but some ignored linguistic defects could be identified by the patterns. This suggests that, although human analysts cannot be replaced, tools can help them to perform a better job.

7.5.2 Ongoing and future research

After this experience some new research directions emerged and are presented below.

Dataset extension

In order to achieve more general results there is the need of increasing the dataset dimension and especially its variety. In particular, it is necessary to collect a dataset composed of requirements coming from different levels and different projects. Alstom expressed the willingness to continue the collaboration providing other requirements from four different levels of software development life cycle with increasing detail level (see Section 1.3):

- **Software Requirements Specification:** 58 requirements;
- **Software Architecture Specification:** 231 requirements;
- **Software Design Specification:** 439 requirements;
- **Software Component Specification:** 574 requirements.

The defects annotation process is in progress.

The experimentations on this dataset would allow to understand to which extent the NLP patterns have to be tuned to analyze requirements at different levels of abstractions, and to understand which patterns are appropriate for which level.

Furthermore we want to extend our dataset using requirements documents from different domains in order to assess to which extent the adaptation of NLP patterns to the language of a company can lead to improved results in terms of defect detection accuracy. We are looking forward to use other repositories, for example the one introduced in [52]. A major problem in conducting a solid experimental analysis on these further data is their annotation by independent reviewers, which is scarcely available.

Language and cost: research directions

It would be of interest to study to which extent *language errors* – a defect not considered here, but mentioned in [12] – may impact on the quality of the requirements. The VEs noticed that large part of the requirements considered were not expressed in correct English, since they were written by Italian editors, who tended to use Italian syntactic constructions. However, apparently, these language errors did not have an impact on the subsequent phases of the process, since the readers of the requirements were also Italian.

Another interesting research direction regards the evaluation of the cost of using NLP techniques for defect detection, compared to the cost of manual

review. Cost-based evaluation approaches suitable for our context have been recently discussed by [10].

Machine learning techniques for NLP

We agree that the usage of NLP rule-based approach for defect detection in NL requirements has numerous advantages, in particular: (a) it is relatively easy to define rules and to apply them; (b) this approach provides hints on the localization of detected defects.

We also agree that this approach suffers from one major drawback: it lacks the capability to generalize (i.e., the rules need to be tailored for each specific company, project, requirements level, etc.).

It would be of interest to evaluate the feasibility of machine learning techniques for NLP defect detection and, in case this approach results to be feasibile, to compare its performance with the rule-based one. As emerged from our early stages investigations in this area [45] it is particularly important to have a various and balanced dataset. The required dataset dimension depends on the selected algorithm. We performed a preliminary test in this direction using the approach described below:

- each requirement was vectorized by extracting *bag-of-word* (BoW) features, performing the following steps:
  1. tokenization of text using white spaces as separators;
  2. removal of the punctuation and the words of length 1;
  3. removal of morphological and inflexional endings from the words by running the Porter stemming algorithm [95];
  4. removal of all the stop words[1];
  5. creation of the histogram of the unigrams and bigrams[2].
- we randomly split the dataset into a training set containing the 90% of the requirements and a test set containing the remaining 10%. Since the dataset is unbalanced (i.e., the defective requirements are only the 4.8%) we stratified the splitting so that train and test sets have the same proportion of defective requirements;

[1]Stop words are the most common words in a language and are usually stored in lists.
[2]A unigram is the occurrence of one word, while a bigram denotes the occurrence of two words in sequence.

- we trained a linear SVM on the training set.

We measured a precision of 6.9% and a recall of 87.2% on the test set, thus obtaining quantitative results close to the ones discussed in the main matter. The major drawback of this simple early-stages sperimentation is that a pure classifier – SVM in this case – is unable to provide hints on the defect localization. This drawback may be overcome applying different machine learning algorithms (e.g., multi-instance learning, meta learning, Recurrent Neural Networks (RNNs) [54, 98, 108, 30]).

Chapter 8

Public and private transport integration model with STPN

8.1 Introduction

By the year 2030, urban mobility will have changed due to sociodemographic evolution, urbanization, increase of the energy costs, implementation of environmental regulations, and further diffusion of Information and Communication Technology (ICT) applications. The demand for public and collective modes of transport will increase considerably. Part of the answer will come from the public transport that will evolve as an integrated combination of buses, cars, metros, tramways and trains [41, 1]. In general, right-of-way (ROW) is the defining characteristic of public transportation modes and we can list three ROW types:

1. *Exclusive:* Transit vehicles operate on fully separated and physically protected ROW. Tunnels, elevated structures, or at-grade tracks are such examples. This ROW type offers very high capacity, speed, reliability and safety. All heavy rail transit systems, like the Metrorail of the Washington Metropolitan Area Transit Authority, belong to this category.

2. *Semi-Exclusive:* Transit ways are longitudinally separated from other traffic, such as private vehicles and pedestrians. Light rail transit (LRT) systems, like the Florence tramway in Italy, are mostly built according to this ROW type.

3. *Fully-Shared:* Transit vehicles share ROW with other traffic, for examples buses, taxi and cars. This ROW type requires the least infrastructure investment, but operations are relatively unreliable due to roadway congestion.

Exclusive ROW need major investment, thus often semi-exclusive or fully-shared modes are chosen. The drawback of this choice is that the different transportation modalities may end in a conflict due to physical constraints concerned with the urban structure itself. For example, this is the case of an intersection between a public road and a tramway right-of-way, where traffic lights priority given to trams may trigger road congestion, while an intense car traffic can impact on trams' performance. These situations can be anticipated and avoided by accurately modeling and analyzing the possible congestion events. Typically, modeling tools provide simulation facilities, by which various scenarios can be played to understand the response of the intersection to different traffic loads. Simulation techniques are used to support early verification of design choices, but can analyze a limited, yet high, number of different scenarios, and encounter difficulties in the evaluation of rare events. Only modeling techniques and tools that support the analysis of the complete space of possible scenarios are able to find out such rare events [20, 14].

In this chapter, we present an analytical approach to model and evaluate a critical intersection for the Florence tramway, where frequent traffic blocks used to happen. This work has been funded by Fondazione Cassa di Risparmio di Firenze, with the kind help of GEST[1], the company running the Florence tramway, in providing important data on which to base the study.

8.2 Analysis of a conflict between public and private transport in Florence

Figure 8.1 shows the route of line 1, which has been put in service in 2010 and links Santa Maria Novella central station to Scandicci (Florence suburbs). This line has overall good performance, with trams running regularly from the end of the line in Scandicci to almost the other end in the city center, but there is a consistent source of delay just a few meters short of the last

[1] `https://www.ratpdev.com/en/references/italy-florence-tramway`

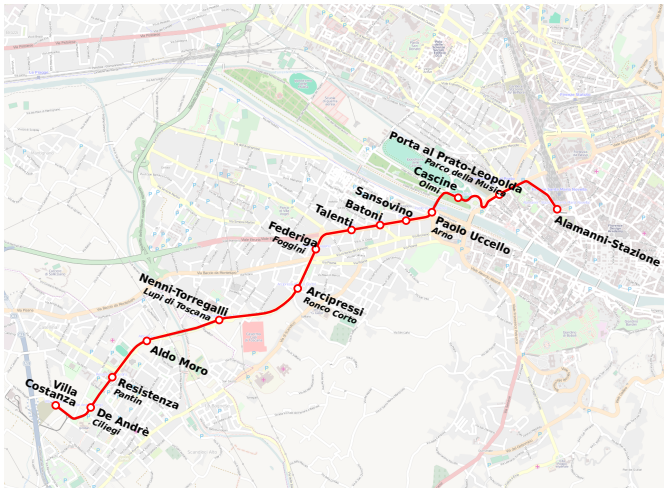

Figure 8.1: Map of tram route from Villa Costanza (Scandicci) to Alamanni-Stazione (Santa Maria Novella station). The route is 7720 meters long with 14 tram stops.

scheduled stop, near Santa Maria Novella train station [32]. The root cause for these issues is the Diacceto-Alamanni intersection, where both via Iacopo da Diacceto, a street with dedicated tracks for tramways, and via Luigi Alamanni, a street for private transport, head to Santa Maria Novella train station.[2]. An aerial view of this intersection is shown in Figure 8.2. The darker stripe that crosses the tracks represents the (unidirectional) private traffic flow from Alamanni street that is the source of the analyzed conflict.

Taking this intersection as a case study, we exploit the ORIS tool to evaluate the probability of a traffic block, leveraging regenerative transient analysis based on the method of stochastic state classes to analyze a model of the intersection specified through Stochastic Time Petri Nets (STPNs). Note that ORIS supports the analysis of models with multiple concurrent temporal parameters associated with a general (i.e., non-exponential) distribution. In particular, the model of the Diacceto Alamanni intersection in-

[2]Actually, the construction works of the new tramway lines (due to be opened soon) have consistently changed the geometry of the Diacceto-Alamanni intersection, partially removing the car traffic. Anyway, the analysis presented in this work refers to a relevant scenario, typical of intersections between a public road and a tramway right-of-way, which will occur more frequently in Florence as new tramway lines will be built.

Figure 8.2: Aerial view of the Diacceto-Alamanni intersection.

cludes temporal parameters associated with a deterministic value (e.g., tram interleaving period), a uniform distribution (e.g., tram delay time), and an exponential distribution (e.g., private vehicles arrival rate). The reported experience shows that the frequency of tram rides impacts on the road congestion, and hence compensating measures (such as sychronizing the passage of trams in opposite directions on the road crossing) should be considered.

The remainder of the paper is structured as follows. Section 8.3 summarises related works. Section 8.4 provides a short introduction to STPNs, the method of stochastic state classes, and the ORIS tool. Section 8.5 presents the realized model and Section 8.6 the obtained results. Finally, Section 8.7 concludes the chapter.

8.3 Related Works

Earliest research on integrated control for traffic management at network level can be traced back to the 1970s. The first railway timetables were planned based on the experience and knowledge of dispatchers in resolving train conflicts [65]. This manual scheduling practice proved its low efficiency with the increase of traffic congestion and exacerbated train delays.

An integrated policy for priority signals at intersections is required, given that trams operate in a *semi-exclusive* ROW environment. In the literature, we can find two different streams of studies: the first aiming at optimizing tram schedules without considering their effects on other traffic flows; the second aiming at manipulating the tram schedule so that trams always clear the intersection during green phases, thus reducing influences on other traffic flows. In [105], the tradeoffs between tram travel times and roadway

traffic delays are explored. Literature counts several works applying different simulation techniques. Microscopic models, i.e., models in which each vehicle is modeled by itself as a particle, can be divided according to the representation of road structure in greater detail. In the continuous road model group, a base structure of road space is modeled as a continuous one dimensional (1D) link. The behavior of car agents is often implemented by applying car-following theories [92, 109, 125]. In the cell-type road model group, road space is discretized by homogeneous cells in which the behavior of car agents is expressed using transition rules such as cellar automata [72, 112]. In a queuing model group, road networks are modeled as queuing networks [57, 2]. Most commercial microscopic traffic simulators employ the continuous road model. In addition, several researchers have proposed simulation frameworks for mixed traffic of two or more models. For example, Yang et al. [119] proposed a framework for pedestrian road crossing behavior in Chinese cities in which they determined the criterion used by pedestrians to decide whether to start crossing a road after considering vehicle flows. Meanwhile, Zeng et al. [123] modeled pedestrian-vehicles interactions at crosswalks in order to minimize pedestrian-vehicle collisions.

Dobler and Lämmel [40] integrated multi-modal simulation modules to the existing framework of MATSim, a large scale traffic simulation framework based on the queuing model [29]. Their integration approach was based on locally replacing simple queue structures with continuous 2D space at sections with higher traffic flows. The behavior rules of agents in the 2D space are based on the social force model (SFM). Krajzewicz et al. [77] introduced pedestrian and bicycle agent models into SUMO, which is a widely used traffic simulator belonging to the continuous road model group [78]. Finally Fujii et al. [55] introduced an agent-based framework for mixed-traffic of cars, pedestrians and trams by using the simulator MATES [121].

8.4 Background

In this section, we provide some background on STPNs (Sect. 8.4.1), the method of stochastic state classes (Sect. 8.4.2), and the ORIS tool (Sect. 8.4.3).

8.4.1 Stochastic Time Petri Nets

An STPN is a tuple $\langle P, T, A^-, A^+, A^\cdot, m_0, F, W, E, U\rangle$ where: P is the set of places; T is the set of transitions; $A^- \subseteq P \times T$, $A^+ \subseteq T \times P$ and $A^\cdot \subseteq P \times T$ are the sets of precondition, postcondition, and inhibitor arcs, respectively: $m_0 : P \to \mathbb{N}$ is the initial marking; $F : T \to [0,1]^{[EFT_t, LFT_t]}$ associates each transition t with a Cumulative Distribution Function (CDF) $F(t) : [EFT_t, LFT_t] \to [0,1]$, where $EFT_t \in \mathbb{Q}_{\geq 0}$ and $LFT_t \in \mathbb{Q}_{\geq 0} \cup \{\infty\}$ are the *earliest* and *latest firing time*, respectively; $W : T \to \mathbb{R}_{>0}$ associates each transition with a weight; E and U associate each transition t with an enabling function $E(t) : \mathbb{N}^P \to \{true, false\}$ and an update function $U(t) : \mathbb{N}^P \to \mathbb{N}^P$, which associate each marking with a boolean value and a new marking, respectively.

A place p is an *input*, an *output*, or an *inhibitor* place for a transition t if $\langle p, t\rangle \in A^-$, $\langle t, p\rangle \in A^+$, and $\langle p, t\rangle \in A^\cdot$, respectively. A transition t is *immediate* (IMM) if $EFT_t = LFT_t = 0$ and *timed* otherwise; a timed transition t is *exponential* (EXP) if $F_t(x) = 1 - e^{-\lambda x}$ over $[0, \infty]$ with $\lambda \in \mathbb{R}_{>0}$, and *general* (GEN) otherwise; a general transition t is *deterministic* (DET) if $EFT_t = LFT_t > 0$ and *distributed* otherwise; for each distributed transition t, we assume that F_t is the integral function of a Probability Density Function (PDF) f_t, i.e., $F_t(x) = \int_0^x f_t(y)dy$. IMM, EXP, GEN, and DET transitions are represented by thick white, thick gray, thick black, or thin black bars, respectively.

The state of an STPN is a pair $\langle m, \tau\rangle$, where m is a marking and $\tau : T \to \mathbb{R}_{\geq 0}$ associates each transition with a time-to-fire. A transition is *enabled* by a marking if each of its input places contains at least one token, none of its inhibitor places contains any token, and its enabling function evaluates to true; an enabled transition t is *firable* in a state if its time-to-fire is equal to zero. The next transition t to fire in a state $s = \langle m, \tau\rangle$ is selected among the set of firable transitions $T_{\mathrm{f},s}$ with probability $W(t)/\sum_{t_i \in T_{\mathrm{f},s}} W(t_i)$. When t fires, s is replaced with $s' = \langle m', \tau'\rangle$, where:

- m' is derived from m by: removing a token from each input place of t, which yields an intermediate marking m_{tmp}; adding a token to each output place of t, which yields a second intermediate marking m'_{tmp}; and, applying the update function $U(t)$ to m'_{tmp};

- τ' is derived from τ by: *i*) reducing the time-to-fire of each *persistent* transition (i.e., enabled by m, m_{tmp} and m') by the time elapsed

in s; *ii*) sampling the time-to-fire of each *newly-enabled* transition t_n (i.e., enabled by m' but not by m_{tmp}) according to F_{t_n}; and, *iii*) removing the time-to-fire of each *disabled* transition (i.e., enabled by m but not by m').

8.4.2 The method of stochastic state classes

The method of stochastic state classes [114, 66] permits the analysis of STPNs with multiple concurrent GEN transitions. Given a sequence of firings, a *stochastic state class* encodes the marking and the joint PDF of the times-to-fire of the enabled transitions and the absolute elapsed time τ_{age}. Starting from an initial stochastic state class, the *transient tree* of stochastic state classes that can be reached within a time $t_{\max}$ is enumerated, enabling derivation of continuous-time transient probabilities of markings (*forward transient analysis*), i.e., $p_m(t) := P\{M(t) = m\}\ \forall\ 0 \leq t \leq t_{\max},\ \forall\ m \in \mathcal{M}$, where $M(t)$ is the *marking process* describing the marking $M(t)$ of an STPN for each time $t \geq 0$ and $\mathcal{M}$ is the set of reachable markings.

If the STPN always reaches within a bounded number of firings a *regeneration*, i.e., a state satisfying the Markov condition, its marking process is a Markov Regenerative Process (MRP) [31], and its analysis can be performed enumerating stochastic state classes between any two regenerations. This results in a set of trees that permit to compute a local and a global kernel characterizing the MRP behavior, enabling evaluation of transient marking probabilities through the numerical solution of Markov renewal equations (*regenerative transient analysis*). Trees also permit to compute conditional probabilities of the Discrete Time Markov Chain (DTMC) embedded at regenerations and the expected time spent in any marking after each occurrence of any regeneration [84], supporting derivation of steady-state marking probabilities according to the Markov renewal theory (*regenerative steady-state analysis*).

While stochastic state classes support quantitative evaluation of an STPN model, the set Ω of behaviors of the STPN can be identified with simpler and more consolidated means through non-deterministic analysis of the underlying TPN model. In this case, the state space is covered through the method of *state classes* [113, 39], each made of a marking and a joint support for τ_{age} and the times-to-fire of the enabled transitions. In this approach, enumeration of state classes starting from an initial marking provides a representation for the continuous set of executions of an STPN, enabling verification

of qualitative properties of the model, e.g., guarantee, with certainty, that a marking cannot be reached within a given time bound (*non-deterministic transient analysis*).

8.4.3 ORIS overview

ORIS [16][3] is a software tool for qualitative verification and quantitative evaluation of reactive timed systems. ORIS supports modeling and evaluation of stochastic systems governed by timers (e.g., interleaving or service times, arrival rate, timeouts) with general probability density functions (PDFs). The tool adopts Stochastic Time Petri Nets (STPNs) as a graphical formalism to specify stochastic systems, and it efficiently implements the method of stochastic state classes, including regenerative transient, regenerative steady-state and non-deterministic analysis.

The software architecture of ORIS decouples the graphical editor from the underlying analysis engines. Given the many variants of Petri net features, ORIS was developed with extensibility in mind: new features can be defined by implementing specific interfaces, so that they can be introduced in the graphical editor and made available to the analysis engines. In turn, analysis engines implement a specific interface that allows them to cooperate with the graphical interface, i.e., to collect analysis options from the user, to start/stop analysis runs, to record and display analysis logs, and to show time series and tabular results. The available analysis engines include:

Non-deterministic Analysis, to produce a compact representation of the dense set of timed states that can be reached by the model. Non-deterministic analysis based on the theory of Difference Bound Matrix (DBM) supports the identification of the boundaries of the space of feasible timed behaviors [19]. The state space is displayed as a directed graph, where edges represent transition firings while nodes are *state classes* [113] comprising a marking and a DBM zone of timer values. This analysis is useful to debug STPNs models and ensure that their state space M is finite.

Transient and Regenerative Analysis, to compute transient probabilities in Generalized Semi-Markov Processes (GSMPs) and Markov Regenerative Processes (MRPs), respectively. These methods evaluate trees where edges are labeled with transitions and their firing probabilities, while nodes are *stochastic state classes* [66] comprising a marking, the PDF of timers,

[3]ORIS is available for download at the webpage `https://www.oris-tool.org/`

and their support (a DBM zone). For a given *time limit* T, the enumeration proceeds until the tree covers the transition firings of the STPN by time T with probability greater than $1 - \epsilon$, where $\epsilon > 0$ is an *error* term. While standard transient analysis enumerates a single, very large tree of events, regenerative analysis avoids the enumeration of repeated subtrees rooted in the same *regeneration point* (where all general timers are reset or have been enabled for a deterministic time). A *time step* Δt is used to select equispaced time points where transient probabilities are evaluated (directly or by solving Markov renewal equations).

Regenerative Steady-State Analysis, to compute steady-state probabilities in MRPs (and thus Semi-Markov Processes (SMPs) and Continuous Time Markov Chains (CTMCs)) with irreducible state space. This method uses trees of stochastic state classes between regeneration points to compute steady-state probabilities of markings: expected sojourn times in each tree are combined with the steady-state probability of regenerations at their roots [84]. As for transient analysis, this method can be applied to STPNs allowing multiple general timers enabled in each state.

Transient Analysis under Enabling Restriction, to compute transient probabilities in MRPs that allow at most one general transition enabled in each state [58].

ORIS engines support instantaneous (transient or steady-state) and cumulative (transient) rewards. A *reward* is a real-valued function of markings $r : M \to \mathbb{R}$ that is evaluated by substituting place names with the number of contained tokens in order to compute the instantaneous expected reward $I_r(t) = \sum_{i \in M} r(i) p_i(t)$ at each time t, its steady-state value $\overline{I}_r = \lim_{t \to \infty} I_r(t) = \sum_{i \in M} r(i) \overline{p}_i$ or its cumulative value over time $C_r(t) = \int_0^t I_r(t) dt$. In addition, the user can specify a *stop condition*, i.e., a Boolean predicate on markings such as $(\mathtt{p_0} == 1) \&\& (\mathtt{p_1} == 1)$, that is used to halt the STPN. This feature can be used to compute first-passage probabilities [66] or reach-avoid objectives equivalent to bounded until operators [90].

8.5 Diacceto-Alamanni: an STPN model

In this section, we describe the STPN model of the Diacceto-Alamanni intersection. Figure 8.3 shows the model which is composed of the following

two submodels:

- tramway submodel (blue box);
- private traffic submodel (red box).

8.5.1 Tramway submodel

The portion of the tramway submodel in the dotted blue box represents the direction from Santa Maria Novella train station (Alamanni-Stazione), while the one in the dashed blue box represents the opposite direction. GEST provided the interleaving period of trams, which is equal to 220 s; transition *period*, which models tram departures, fires a new token periodically and is enabled with continuity until place *KO* receives a token. Places *p0* and *p1* represent a tramway departing from Alamanni-Stazione and Villa Costanza, respectively. Transitions *delayFromSmn* and *delayFromScndc* represent the delays cumulated by the two trams, respectively; note that 120 s is an upper bound on the maximum delay observed in the available data set and, given that data are few and their distribution is unknown, this parameter is modeled using a uniform distribution [9].

When the tramway is approaching the intersection, dedicated wayside systems (i.e., two loops placed under the railway tracks) are activated (places *Loop01.001.1* and *Loop01.001.2*) and the corresponding traffic lights are set to red (places *setRedFromSmn* and *setRedFromScnd*). The traffic lights are in fact set to red 5 s before the arrival of the tram at the intersection; this parameter has been provided by GEST and is modeled by the DET transitions *crosslightAnticipationSmn* and *crosslightAnticipationScnd*. Places *crossingFromSmn* and *crossingFromScnd* represent the arrival of the tram at the intersection, while transitions *leavingFromSmn* and *leavingFromScndc* account for the time needed to free the intersection. Specifically, the minimum and the maximum time needed to free the intersection are set equal to 6 s and 14 s, respectively, based on the fact that in the data set provided by GEST this temporal parameter has mean value nearly equal to 10 s and a standard deviation approximately equal to 4 s. Also in this case, given that available data are few, this parameters is modeled by a uniform distribution over the interval $[6, 14]$ s [9].

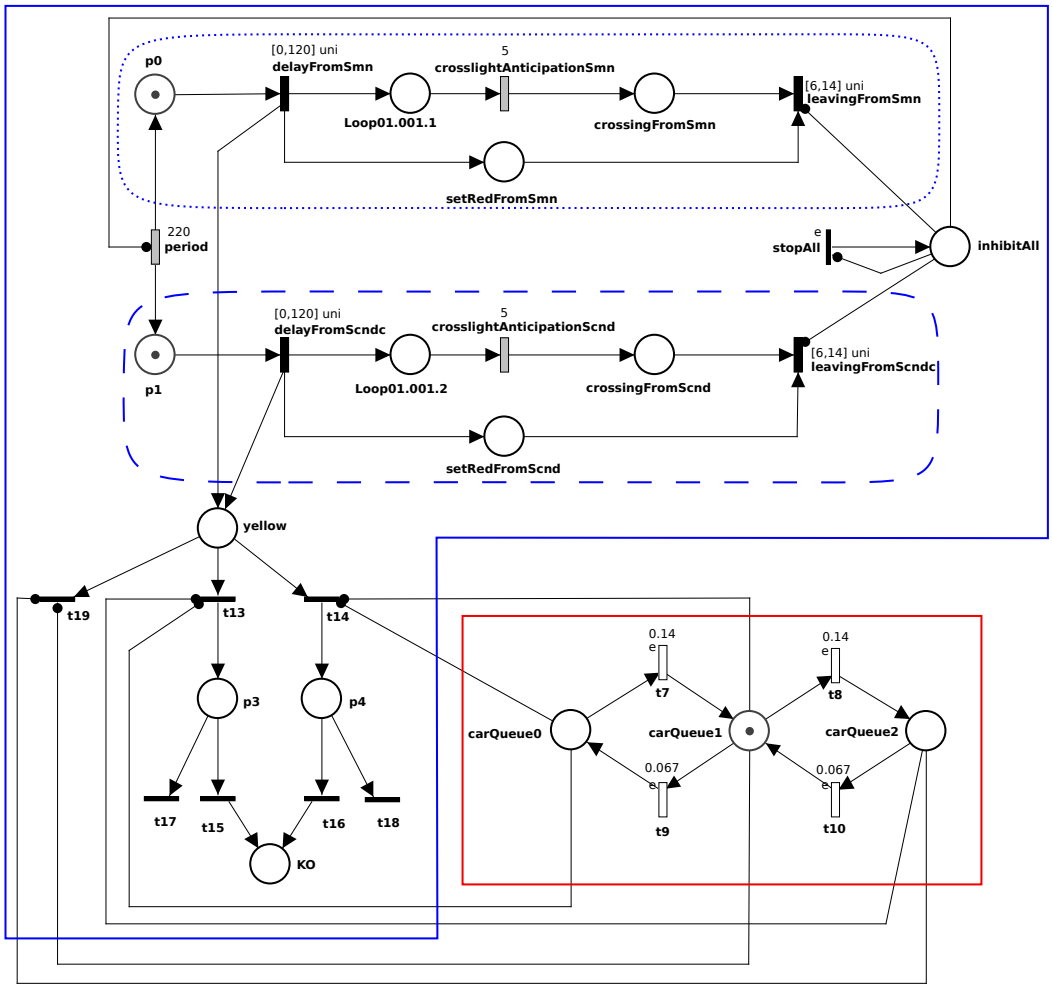

Figure 8.3: Intersection model. The tramway submodel is highlighted by the blue box, the private traffic queue submodel is highlighted by the red one. Transitions associated with an enabling function are marked by a label “e”.

8.5.2 Private transport submodel

We model private traffic as a birth-death process with three levels of traffic congestion: specifically, places *carQueue0*, *carQueue1*, and *carQueue2* model the condition of low, moderate, and high volume of traffic, respectively. Since we lack data on car traffic in Florence, we assume that the average traffic density is approximatively 1000 cars per hour, which is a typical value for a high traffic flow on a single lane [89], and we consider the case that the arrival/departure of two cars increases/decreases the traffic congestion level, respectively, and that the time needed to occupy the intersection is nearly half the time needed to leave it. According to this, the EXP transitions *t7* and *t8* have rate equal to $0.14\,\mathrm{s}^{-1}$, while the EXP transitions *t9* and *t10* have rate equal to $0.067\,\mathrm{s}^{-1}$.

Intuitively, the number of cars in the queue increases when the private traffic light is set to red and decreases otherwise. In order to model this behavior, transitions *t7* and *t8* are associated with an enabling function that evaluates to true when at least one token is present in place *setRedFromSmn* or in place *setRedFromScnd* (i.e., *setRedFromSmn+setRedFromScnd>0*). Conversely, transitions *t9* and *t10* are associated with an enabling function that evaluates to true when no token is present in places *setRedFromSmn* and *setRedFromScnd* (i.e., *setRedFromSmn+setRedFromScnd==0*).

8.5.3 Interaction between the tramway submodel and the private transport submodel

Road congestion may cause cars to stand for a while on the tracks after the private traffic light has turned to red, thus blocking trams. Place *yellow* models the private traffic light set to yellow, while place *KO* actually models the case that a tram ride is blocked by private vehicles on the lane. When place *KO* receives a token, transition *stopAll* becomes enabled (given that it is associated with an enabling function *KO>0*) and fires, depositing a token in place *inhibitAll*. This finally disables transitions *period*, *leavingFromSmn*, and *leavingFromScndc*, due the inhibitor arcs from *KO* to each of these transitions.

Transitions *t13* through *t19* model the possibility that a tram ride is blocked by private vehicles stopping on the tracks. If the traffic congestion level is low (i.e., *carQueue0* > 0), the tram runs regularly and transition *t19* is enabled, so that no token is deposited in place *KO*. If traffic con-

gestion increases to a moderate level (*carQueue1* > 0) or to a high level (*carQueue2* > 0), transition *t13* or transition *t14* becomes enabled and fires, respectively. In the former case (*p3* > 0), transitions *t15* and *t17* fire with probability 0.3 and 0.7, respectively, given that they have weight equal to 30 and 70, respectively; in the latter case, transitions *t16* and *t18* fire with probability 0.4 and 0.6, respectively, given that they have weight equal to 40 and 60, respectively. In doing so, the probability of a traffic block is 0.3 and 0.4 in the case of moderate and high traffic congestion, respectively. These parameters have been estimated from tram delays observed in the data set provided by GEST.

8.6 Analysis and Results

In this section, we report the results obtained from the analysis of the model of Sect. 8.5. In all the experiments, we performed regenerative transient analysis of the model through the ORIS tool using the following parameters:

- *Time limit* $T = 7200\,\mathrm{s}$ (corresponding to 2 h);
- *Time step* $\Delta t = 20\,\mathrm{s}$.
- *Error* $\epsilon = 0.01$;

The first experiment has been performed with average traffic density equal to 1000 cars per hour (i.e., the EXP transitions *t7* and *t8* have rate equal to $0.14\,\mathrm{s}^{-1}$, and the EXP transitions *t9* and *t10* have rate equal to $0.067\,\mathrm{s}^{-1}$, as shown in Figure 8.3) and crosslight anticipation equal to 5 s (i.e., the value of the DET transitions *crosslightAnticipationSmn* and *crosslightAnticipationScnd* is 5 s, as also shown in Figure 8.3). Figure 8.4 shows the probability of the private traffic queue status in a time interval of 2 h, obtained computing the instantaneous rewards "*carQueue0* > 0", "*carQueue1* > 0", and "*carQueue2* > 0". As we can see, the queue status tends to saturation quite rapidly.

Figure 8.5 shows the KO probability for different values of the crosslight anticipation parameter, obtained computing the instantaneous reward "$KO > 0$". We observe that the probability of reaching the KO state increases every 220 s for all the displayed curves, due to periodic tram departures. We also note that the probability of reaching the KO state increases when the

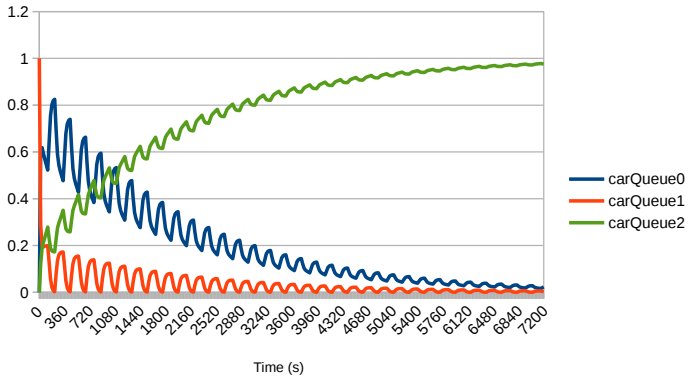

Figure 8.4: Transient probability of the traffic queue status.

crosslight anticipation is higher: intuitively, when the anticipation time increases, the time during which private traffic should flow away from the intersection decreases, thus degrading the queue status and consequently increasing the KO probability.

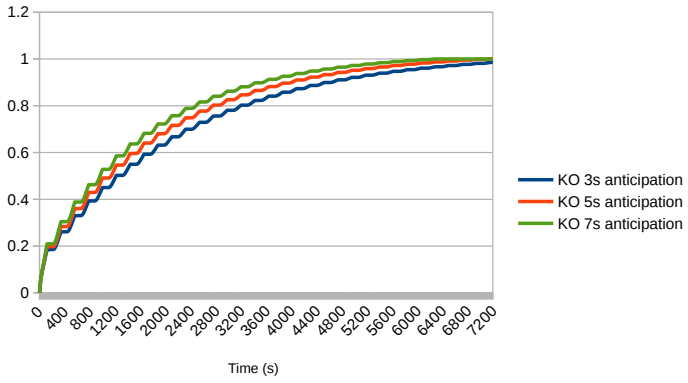

Figure 8.5: Transient probability of the KO state for different values of the crosslight anticipation parameter.

Finally, Figure 8.6 shows the KO probability (obtained computing the instantaneous reward "$KO > 0$") for different values of the private traffic density. The probability of reaching the KO state increases when the traffic

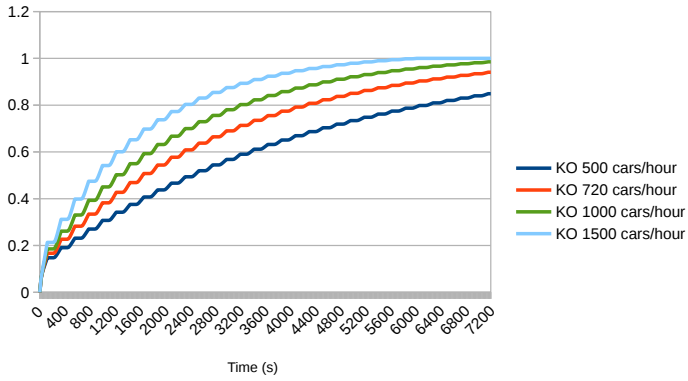

Figure 8.6: Transient probability of the KO state for different values of traffic density.

density is higher and reaches 0.7 in less than half an hour with extremely congested private traffic (i.e., 1500 cars per hour), while the same value is reached in more than a hour with moderately congested private traffic (i.e., 500 cars per hour).

We also argue that, for the planning of both tram timetables and traffic light timings, it is important to consider the correlation between the time of red signal, the time of green signal, and the tram headway, pointing out the need of an integrated management of the different transport systems in order to have a more robust and higher quality service. Furthermore, a more detailed analysis is needed to accurately model the behavior of private traffic during the day.

8.7 Implication for practice and future research

In this section we draw conclusions presenting the implication for practice and introducing some ongoing and future research directions.

8.7.1 Implication for practice

Modeling and analysis of complex intersections for the integration of private and public transport supports the evaluation of the perceived availability of public transport and the identification of robust traffic light plans and tram

timetables. In this chapter, we presented an analytical approach to model and evaluate a critical intersection for the Florence tramway. Specifically, we used the ORIS tool to evaluate the probability of a traffic block, leveraging regenerative transient analysis based on the method of stochastic state classes to analyze a model of the intersection specified through Stochastic Time Petri Nets (STPNs). The analysis results showed a correlation between the frequency of tram rides, the traffic light plan, and the status of the queue of private vehicles, pointing out that the frequency of tram rides impacts on the road congestion. Therefore, compensating measures should be considered, such as synchronizing the passage of trams in opposite directions on the road crossing.

8.7.2 Ongoing and future research

Within the context of modeling techniques to optimize the integration of public and private traffic, our work will go towards the following directions:

- analyze other road/tramway intersections, also considering the new tramway lines that will be opened in Florence, so as to to compare differences and similarities and generalize the modeling methodology;
- improve the scalability of the approach by combining numerical solution of the tramway submodel through the ORIS tool with analytical evaluation of the traffic congestion level, which could permit to model private traffic more accurately (e.g., considering a larger number of congestion levels) without incurring in the state space explosion problem;
- evaluate to which extent the behavior of passengers and pedestrians as well as the weather conditions perturb the tramway performance, including them in the model of the road/tramway intersection [88];
- compare the results obtained with analytical approach with the ones obtained by microsimulation techniques.

Chapter 9

Conclusion

The work presented in this thesis is the result of a research activity aimed at investigating on methods to improve the product lifecycle cost-effectiveness in the context of railway domain. In particular it focused on different aspects: a main topic on system *verification* and a side project in *performance evaluation*.

For what concerns system *verification* the focus was on the activity of requirement review performed by the Verification Engineers in order to detect defects in requirements expressed in natural language. This activity is time consuming and error prone and we investigated to which extent an automatic tool could help the Verification Engineers in this task.

In Chapters ranging from 3 to 7 is presented a case study experience performed in collaboration with Alstom and ISTI-CNR. We first identified a set of typical defects classes and, for each class, a Verification Engineer of the company implemented a set of defect-detection patterns by means of the GATE tool for text processing. A pilot study on 241 requirements is presented, as well as a large-scale study on 1866 requirements. After a refinement of the patterns, a precision of 83.16% and a recall of 85.39% are obtained. Recall can be increased by using term-based defect detection tools such as SREE [111], although at the cost of a lower precision. This experience led to two publications [100, 51].

This is one of the first works in which defect detection NLP techniques are applied on a very large set of industrial requirements annotated by domain experts. We contribute with a comparison between traditional manual techniques used in industry for requirements analysis, and analysis performed

with NLP. Our experience shows that several discrepancies can be observed between the two approaches. The analysis of the discrepancies offers hints to improve the capabilities of NLP techniques with *company specific* solutions, and suggests that also company practices need to be modified to effectively exploit NLP tools.

For what concerns system *performance evaluation* we focused on the problem of interaction and integration between public and private traffic in the actual and future cities. This activity was funded by the Fondazione Cassa di Risparmio di Firenze and focused on the analysis of a critical intersection for the Florence tramway. Specifically, we used the ORIS tool to evaluate the probability of a traffic block, leveraging regenerative transient analysis based on the method of stochastic state classes to analyze a model of the intersection specified through Stochastic Time Petri Nets (STPNs).

The experience showed that an analytical analysis is possible on the statistical behaviors of undesired events often not considered in the simulation scenarios.

The future research directions related to the two topics have been detailed in Section 7.5.2 and Section 8.7.2 and here we observe that in both the areas there is the need of capturing and understanding the "human factor" influence; in one case it is related to the way human beings communicate (the natural language), in the other it is related to what they use to behave (when travelling by public transport or driving a vehicle). With the increase of transport systems complexity, high quality and performance demand, thus new challenges are introduced, but all of them need to consider the 'human factor" playing behind as it fills the gap between the ideal situation and the reality.

Appendix A

Appendix A: Stochastic Discrete Time Petri Nets

Petri Nets are a family of formalisms used in the modeling and analysis of synchronous, asynchronous and distributed systems. There are countless variations and extensions, that allow to deal with both qualitative and quantitative aspects of systems; some formalisms take into account time and are purely nondeterministic, whereas some others enable a stochastic characterization of the system.

A.1 Petri Nets

One of the simplest models is the original *Petri Net* (PN), introduced in the seminal work [93]. In the following Petri Nets are presented with modern notations.

A.1.1 Syntax

A *Petri Net* (PN) is a tuple:

$$PN = \langle P; T; A^-; A^+; A^\bullet; w^-; w^+; w^\bullet; M \rangle$$

- P and T are disjoint sets of *places* and *transitions*.
- $A^- \subseteq P \times T$ is a set of *precondition* arcs.
- $A^+ \subseteq T \times P$ is a set of *postcondition* arcs.

- $A^\bullet \subseteq P \times T$ is a set of *inhibitor arcs*.
- $w^- : A^- \to \mathbb{N}$ associates each precondition arc with a *multiplicity*.
- $w^- : A^+ \to \mathbb{N}$ and $w^- : A^\bullet \to \mathbb{N}$ are defined analogously.
- $M : P \to \mathbb{N}$ is a *marking*, and associates each place with a number of *tokens*.

Petri Nets are essentially biparted graphs in which one of the two node types, the place, is associated with a natural number. A graphical representation of a Petri Net is given in Figure A.1; places are represented as circles and transitions as black rectangles; preconditions are arrows going from a place to a transition; postconditions are arrows going from a transition to a place; an inhibitor arc is a line going from a place to a transition, terminated by a small black circle; the marking is represented by decorating every place with the corresponding number of tokens; weights are natural numbers written near the arc they refer to, and when no multiplicity is specified it is assumed to be 1.

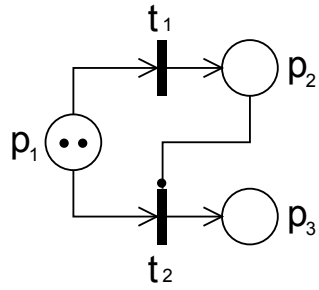

Figure A.1: A simple Petri Net.

A place p is called an *input place* for a transition t if $(p,t) \in A^-$; it is called an *output place* if $(t,p) \in A^+$; it is called an *inhibitor place* if $(p,t) \in A^\bullet$. It is a good idea, in order to simplify notations, to extend the domain of w^- and $w^\bullet$ to the entire set $P \times T$, and the domain of w^+ to the entire set $T \times P$; these extensions are denoted with the symbol δ:

$$\delta^- : P \times T \to \mathbb{N}, \delta^-(p,t) = \begin{cases} w^-((p,t)) & \text{if } (p,t) \in A^- \\ 0 & \text{otherwise} \end{cases}$$

$$\delta^+ : T \times P \to \mathbb{N}, \delta^+(t,p) = \begin{cases} w^+((t,p)) & \text{if } (t,p) \in A^+ \\ 0 & \text{otherwise} \end{cases}$$

$$\delta^\bullet : P \times T \to \mathbb{N}, \delta^\bullet(p,t) = \begin{cases} w^\bullet((p,t)) & \text{if } (p,t) \in A^\bullet \\ +\infty & \text{otherwise} \end{cases}$$

A.1.2 Semantics

The *state* of a PN consists of its marking:

$$s = \langle M \rangle$$

The state evolves according to two rules.

Enabling

A transition t is *enabled* if and only if, for each of its precondition arcs $(p,t) \in A^-$ it is true that $M(p) \geq w^-((p,t))$, and for each of its inhibitor arcs $(p,t) \in A^\bullet$ it is true that $M(p) < w^\bullet((p,t))$. This is equivalent to requiring that $\delta^\bullet((p,t)) > M(p) \geq \delta^-(p,t)$ for any place p.

Firing

An enabled transition t can *fire*, leading to a transition relation $\xrightarrow{t}$ between states such that $s = \langle M \rangle \xrightarrow{t} s' = \langle M' \rangle$ if and only if, for each place p:

$$M'(p) = M(p) - \delta^-(p,t) + \delta^+(t,p)$$

In other words, when a transition fires, tokens are removed from its input places and are added to its output places. The amount of tokens is specified by the multiplicities δ^- and δ^+.

A.1.3 State-Space generation

Given an initial state S_o, the succession relation $\xrightarrow{t}$ identifies a set of reachable states $\mathcal{S}$ and a transition system $TS = \langle S_o, \mathcal{S}, \xrightarrow{t} \rangle$. Derivation of the TS requires algorithms for the symbolic firing of transitions; for example methods using relational product that can be efficiently implemented.

A.2 Discrete-Time Stochastic Petri Nets

This section describes a simplified version of the Petri Net formalism first introduced in [17].

A.2.1 Syntax

A *Discrete-Time Stochastic Petri Net* (*dtSPN*) is a tuple:

$$dtSPN = \langle P; T; A^-; A^+; A^\bullet; w^-; w^+; w^\bullet; M; \mathcal{D}; \mathcal{C} \rangle$$

- P and T are disjoint sets of *places* and *transitions*.
- $A^- \subseteq P \times T$ is a set of *precondition* arcs.
- $A^+ \subseteq T \times P$ is a set of *postcondition* arcs.
- $A^\bullet \subseteq P \times T$ is a set of *inhibitor arcs*.
- $w^- : A^- \to \mathbb{N}$ associates each precondition arc with a *multiplicity*.
- $w^- : A^+ \to \mathbb{N}$ and $w^- : A^\bullet \to \mathbb{N}$ are defined analogously.
- $M : P \to \mathbb{N}$ is a *marking*, and associates each place with a number of *tokens*.
- $\mathcal{D}$ associates each transition t with a *static* probability mass function $\mathcal{D}_t$. The extrema of the support of $\mathcal{D}_t$ are the *static earliest and latest firing time* and are denoted $EFT^s(t)$ and $LFT^s(t)$ respectively.
- $\mathcal{C} : T \to \mathbb{N}$ is a *competitiveness* function.

The graphical representation of a dtSPN is very similar to the one of a PN; the only additional elements are annotations for probability mass functions and competitiveness. An example of dtSPN is shown in Figure A.2. The family of Petri Net formalisms is quite homogeneous and it is often the case that syntactical differences can be handled by *decorating* the basic PN syntax with extended elments. Decorating the semantics, however, is way harder; but can be a very efficient way to enable code reuse and favor the development of multi-formalism environments. An example of such a framework which is actively maintained is [22].

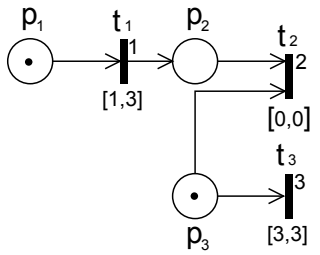

Figure A.2: A simple Discrete-Time Stochastic Petri Net. Note that when enabled t_2 fires deterministically in zero time, t_3 fires deterministically in 3 time units and t_1 has a nondeterministic firing time; when only the interval $[EFT_t, LFT_t]$ is specified for a transition t, the static distribution $\mathcal{D}_t$ is assumed to be uniform, hence transition t_1 could execute in 1, 2, or 3 time units with probability 1/3.

A.2.2 Semantics

The *state* of a dtSPN consists of a marking M and a vector $\vec{ttf}$ of integer *times to fire*, one for each transition t, denoted ttf_t:

$$s = \langle M, \vec{ttf} \rangle$$

The state evolves according to four rules.

Enabling

A transition t is *enabled* if and only if, for each of its precondition arcs $(p,t) \in A^-$ it is true that $M(p) \geq w^-((p,t))$, and for each of its inhibitor arcs $(p,t) \in A^\bullet$ it is true that $M(p) < w^\bullet((p,t))$. This is equivalent to requiring that $\delta^\bullet((p,t)) > M(p) \geq \delta^-(p,t)$ for any place p.

Firability

A transition t is *fireable* if it is enabled and its time to fire ttf_t is equal to 0. The set of fireable transitions is called the *attempting set*. Note that for a disabled transition t the time to fire is irrelevant, and could be thought as to be $+\infty$.

Step selection

If the attempting set is empty, time advances by one tick. Otherwise a *firing set* is derived from the attempting set through *repetitive stochastic selection*; until the attempting set is not empty, any of its transitions t_* is randomly selected with probability $Prob\{t_*\}$, it is removed from the attempting set, and it is added to the firing set if and only if it is not in *conflict* with the firing set. A set $T' \subseteq T$ of transitions is *nonconflicting* if $\sum_{t \in T'} \delta^-(p,t) \leq M(p)$ for any place p, otherwise it is *conflicting*. This definition is adapted from [18], but is compatible with the semantics of [17]. The probability of selecting t_* depends on its competitiveness and on those of the other transitions still in the attempting set:

$$Prob\{t_*\} = \frac{\mathcal{C}(t_*)}{\sum_{t \in \text{Attempting set}} \mathcal{C}(t)}$$

Note that, since the repetitive stochastic selection stops only when the attempting set is empty, the firing set that is derived is effectively *maximal*: two transitions that can fire simultaneously will do so.

Progress

If time advances the marking remain unchanged and the time to fire of any enabled transition is reduced by one time unit. Otherwise a firing set ϕ fires, and the marking is updated for each place p by taking into account the preconditions and postconditions of all the transitions in the firing set:

$$M'(p) = M(p) - \sum_{t \in \phi} \delta^-(p,t) + \sum_{t \in \phi} \delta^+(t,p)$$

In doing so, a temporary marking M_{tmp} is also derived, which accounts for decrementing token counts on input places:

$$M_{tmp}(p) = M(p) - \sum_{t \in \phi} \delta^-(p,t)$$

$$M(p) = M_{tmp} + \sum_{t \in \phi} \delta^+(t,p)$$

Transition which are enabled in M' are either *persistent* or *newly enabled*:

- A transition t is newly enabled if it is enabled in M' but not in M_{tmp} or if $t \in \phi$;

- A transition t is persistent if it is enabled in both M' and in M_{tmp}.

A persistent transition t has its time to fire ttf_t unchanged; a newly enabled transition t has its time to fire ttf_t resampled nondeterministically according to the probability mass function $\mathcal{D}_t$. Time to fire is irrelevant for transitions not enabled in M, because they will be resampled as soon as they are newly enabled.

This semantics describes a transition relation $\overset{ev,\mu}{\rightarrow}$, where μ is a probability measure and ev is either a *time advancement* event, also called a *tick* event, or a *firing* event. For the time advancement event $\mu = 1$, while for the firing event μ depends on both the result of repetitive stochastic selection as well as on the determination of the times to fire resampled for newly enabled transitions.

As an example, consider the dtSPN of Figure A.2; in this model a firing event could happen after one, two or three ticks; if it happens after one or two, then it involves only t_1, that is, $\phi = \{t_1\}$; after three ticks, t_1 and t_2 must fire simultaneously, that is, $\phi = \{t_1, t_2\}$.

A.2.3 Maximal step semantics

The expression *maximum firing strategy* was introduced in [18] and denotes an execution semantics for Petri Nets such that, intuitively, all transitions that can fire *compete* to do so, and eventually some of them execute simultaneously; in [17] the terminology *maximal step semantics* is used to denote a similar behavior, with an additional stochastic characterization via stochastic repetitive selection. Throughout the rest of the dissertation, the expression "maximal step semantics" is used.

Maximal step semantics is inherently more complex than *interleaving semantics*, in which transitions fire one at a time. Under interleaving semantics, if in state s a set $E \subseteq T$ of transitions are enabled, then there are exactly $|E|$ possible firing events that can lead to other states, one for each enabled transition. Under the maximal step semantics instead, any subset $\phi \subseteq E$ can lead to a corresponding firing event; the only restriction is that subsets must be maximal, so that if $\phi_1 \subseteq \phi_2$ is a set derived through repetitive stochastic selection, then ϕ_2 certainly is not. However this restriction does not prevent the number of possible firing sets from growing exponentially in the worse case.

A.2.4 Stochastic states

The evolution of a dtSPN is regarded as a discrete time stochastic process. The states of this process are called *stochastic states* to distinguish them from the states introduced in Section A.2.2. A stochastic state consists in a marking M and a vector $\vec{\mathcal{P}}$ of probability mass functions for the times to fire of the transitions enabled by M.

$$s = \langle M, \vec{\mathcal{P}} \rangle$$

The evolution across stochastic states is described through a succession relation $\overset{ev,\mu}{\Rightarrow}$ where ev is an event and μ a measure of probability. ev could be $tick$, $defer$ or a firing set ϕ denoting a firing event. Given two stochastic states $S = \langle M, \vec{\mathcal{P}} \rangle$ and $S' = \langle M', \vec{\mathcal{P}}' \rangle$, $S \overset{ev,\mu}{\Rightarrow} S'$ if and only if the following property holds: if the marking is M and the vector of times to fire is a random variable distributed according to $\vec{\mathcal{P}}$, then ev is a possible next event, which occurs with probability μ, and which leads to a new marking M' and a new vector of times to fire distributed according to $\vec{\mathcal{P}}'$.

In addition to *tick* and *firing* events, in stochastic state-space generation an additional fictious *defer* event is considered. *defer* accounts for the instantaneous choice that no fireable transition will fire before the advancement of time. With this convention, the set of outgoing events for a given state can take one of the following two forms:

- a single *tick* event;
- several *firing* events, one for each possible firing set, plus an optional *defer* event.

A.2.5 Stochastic State-Space generation

Given an initial stochastic state S_o, the succession relation $\overset{ev,\mu}{\Rightarrow}$ identifies a set of reachable stochastic states $\mathcal{S}$ and a timed stochastic transition system $STS = \langle S_o, \mathcal{S}, \overset{ev,\mu}{\Rightarrow} \rangle$. Derivation of the STS requires algorithms for the detection of the outcoming events from a stochastic state, for the calculus of their probability, and for the derivation of successor stochastic states.

Tick event

Tick is an outcoming event for $S = \langle M, \vec{\mathcal{P}} \rangle$ if and only if $\mathcal{P}_t(0) = 0$ for every transition t enabled under marking M or, in other words, if $EFT_t > 0$. In

the transition from S to $S' = \langle M', \vec{\mathcal{P}}' \rangle$ through a *tick* event, marking is not changed, while time to fire of enabled transitions is updated in the following way:

$$\mathcal{P}'_t(n) = \mathcal{P}_t(n+1) \tag{A.1}$$

The probability of the $\mu = tick$ event is:

$$\mathcal{P}_{tick}(S) = 1 \tag{A.2}$$

Defer event

Defer is an outcoming event for S if and only if $\mathcal{P}_t(0) < 1$ for each enabled transition and $\mathcal{P}_{t_o}(0) > 0$ for at least one enabled transition t_o or, in other words, if $LFT_t > 0$ and $EFT_{t_o} = 0$. In the transition from S to $S' = \langle M', \vec{\mathcal{P}}' \rangle$ through a *defer* event, marking is not changed, while time to fire of enabled transitions is updated by conditioning the time to fire to being higher than zero:

$$\mathcal{P}'_t(n) = \begin{cases} 0 & \text{it } t \text{ is enabled in } S \text{ and } n = 0 \\ \frac{\mathcal{P}_t(n)}{1-\mathcal{P}_t(0)} & \text{it } t \text{ is enabled in } S \text{ and } n \neq 0 \end{cases} \tag{A.3}$$

The probability of the $\mu = defer$ event is:

$$\mathcal{P}_{defer}(S) = \prod_{t \in fireable(S)} (1 - \mathcal{P}_t(0)) = \prod_{t \in Enabled(S)} (1 - \mathcal{P}_t(0)) \tag{A.4}$$

where $Enabled(S)$ is the set of transitions that are enabled in state S and $fireable(S) \subseteq Enabled(S)$ is the subset of those transitions for which $\mathcal{P}_t(0) \neq 0$.

Firing event

The *firing* of a set $\phi = \{t_n\}_{n=1}^{H}$ is an outcoming event for S if and only if: ϕ contains only enabled and nonconflicting transitions, every transition $t \in \phi$ can fire before time advances and, and every enabled transition $t_{nc} \notin \phi$ which is not conflicting with ϕ can be delayed, *i.e.*

$$\mathcal{P}_t(0) > 0 \ \forall t \in \phi \tag{A.5}$$

$$\mathcal{P}_{t_{nc}}(0) < 1 \ \forall t_{nc} \notin \phi, \text{ enabled and not conflicting with } \phi. \tag{A.6}$$

In the transition from S to $S' = \langle M', \vec{\mathcal{P}}' \rangle$ through the *firing* of set ϕ, marking is changed according to the marking update rule described in Section A.2.2, while the components of vector $\vec{\mathcal{P}}$ are updated differently for transitions that are persistent or newly enabled after the firing of ϕ: newly enabling involves a resampling of the time to fire, while the time to fire of persistent transitions is conditioned to be higher than zero:

$$\mathcal{P}'_t(n) = \begin{cases} \mathcal{D}_t(n) & \text{if } t \text{ is newly enabled} \\ 0 & \text{it } t \text{ is enabled in } S \text{ and } n = 0 \\ \frac{\mathcal{P}_t(n)}{1-\mathcal{P}_t(0)} & \text{it } t \text{ is enabled in } S \text{ and } n \neq 0 \end{cases} \tag{A.7}$$

The probability of the $\mu = \phi$ event is:

$$P_\phi(S) = \sum_{\sigma \in Att(S)} P_{\phi|\sigma} \cdot P_\sigma(S) \tag{A.8}$$

where $Att(S)$ is the set of attempting sets that are feasible in S, $P_\sigma(S)$ is the probability that σ is the attempting set for a $\vec{ttf}$ distributed like $\vec{\mathcal{P}}$, and $P_{\phi|\sigma}$ is the probability that ϕ is the firing set given that the attempting set is σ. Due to the independence of times to fire of different transition,

$$P_\sigma(S) = \prod_{t_i \in \sigma} \mathcal{P}_{t_i}(0) \prod_{t_j \in fireable(S) \setminus \sigma} (1 - \mathcal{P}_{t_{i_j}}(0)) \tag{A.9}$$

where $fireable(S) \setminus \sigma$ denotes the set of fireable transitions that are not included in the attempting set. $P_{\phi|\sigma}$ can be seen as a sum over different possible orderings:

$$P_{\phi|\sigma} = \sum_{seq \in Perm(\sigma)} P_{\phi|seq} \cdot P_{seq|\sigma} \tag{A.10}$$

where $Perm(\sigma)$ is the set of permutations of the elements of σ and seq denotes any such permutations. $P_{seq|\sigma}$ is the probability that the transitions

in σ are selected in the order implied by seq, and can be expressed as

$$P_{seq|\sigma} = \prod_{i=1}^{|seq|} \frac{\mathcal{C}(t_{seq_i})}{\sum_{h=i}^{|seq|} \mathcal{C}(t_{seq_h})} \tag{A.11}$$

and $P_{\phi|seq}$ is 1 if and only if every transition $t \in \phi$ appears in seq before any other transition which has conflict with t itself, and is 0 otherwise.

A.3 Stochastic Preemptive Time Petri Nets

The formalism of dtSPNs is a simplified version of the one described in [17], which the authors called *Stochastic Preemptive Time Petri Nets* (*spTPNs*). These are a discrete and stochastic variant of continuous time *Preemptive Time Petri Nets* [21]. The main feature of preemptive Petri Nets is the concept of *preemptable resource*; any transition t is associated with a marking dependent *priority*, $Prio(t, M)$, and a set $Req(t)$ of *requested resources*. An enabled transition t_o is *progressing* if and only if every resource in $Req(t_o)$ is not in the $Req(t_1)$ of any other enabled transition t_1 with higher priority; transitions that are enabled but not progressing are said to be *suspended*. Only progressing transitions can be fireable and be part of the attempting set. Intuitively, a suspended transition is similar to a disabled one, because it cannot fire, but when it becomes progressing again its time to fire is not resampled, in contrast to disabled transitions which become newly enabled. Preemptive time Petri Nets are a very powerful formalism for the analysis of real-time systems. An example of spTPNs is shown in Figure A.3. The reader should refer to [17] and [21] for additional information.

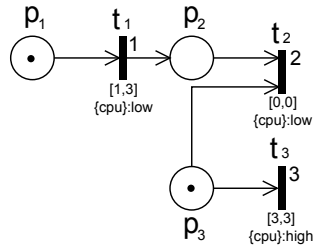

Figure A.3: A simple Stochastic Preemptive Time Petri Net

.

Publications

This research activity has led to the following publications.

International Journals

1. Alessio Ferrari, **Gloria Gori**. Benedetta Rosadini, Jacopo Trotta, Stefano Bacherini, Alessandro Fantechi, Stefania Gnesi "Detecting requirements defects with NLP patterns: an industrial experience in the railway domain", *Empirical Software Engineering*, vol. in press, 2018 [DOI:10.1007/s10664-018-9596-7]

International Conferences and Workshops

1. Benedetta Rosadini, Alessio Ferrari, **Gloria Gori**, Alessandro Fantechi, Stefania Gnesi, Jacopo Trotta, Stefano Bacherini "Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain", in *Proceedings of the 23rd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'17)* , Essen (Germany), 2017
2. Laura Carnevali, Alessandro Fantechi, **Gloria Gori**, Enrico Vicario "Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain", submitted and accepted in *Proceedings of the 12th International Conference on Verification and Evaluation of Computer and Communication Systems*, Grenoble (France), 2018

Bibliography

[1] ACEA, "The 2030 Urban Mobility Challenge," European Automobile Manufacturers Association, Tech. Rep., May 2016.

[2] A. Agarwal and G. Lämmel, "Modeling seepage behavior of smaller vehicles in mixed traffic conditions using an agent based simulation," *Transportation in Developing Economies*, vol. 2, no. 2, p. 12, 2016.

[3] S. A. Alvarez, "An exact analytical relation among recall, precision, and classification accuracy in information retrieval," Computer Science Department, Boston College, Tech. Rep. BCCS-02-01, 2002.

[4] V. Ambriola and V. Gervasi, "On the systematic analysis of natural language requirements with Circe," *Automated Software Engineering*, vol. 13, no. 1, pp. 107–167, 2006.

[5] B. Anda and D. I. Sjøberg, "Towards an inspection technique for use case models," in *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*. ACM, 2002, pp. 127–134.

[6] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated Checking of Conformance to Requirements Templates Using Natural Language Processing," *IEEE Transactions on Software Engineering*, vol. 41, no. 10, pp. 944–968, 2015.

[7] A. Aurum, H. Petersson, and C. Wohlin, "State-of-the-art: software inspections after 25 years," *Software Testing, Verification and Reliability*, vol. 12, no. 3, pp. 133–154, 2002.

[8] R. L. Baskerville and A. T. Wood-Harper, "A critical perspective on action research as a method for information systems research," *Journal of information Technology*, vol. 11, no. 3, pp. 235–246, 1996.

[9] S. Bernardi, J. Campos, and J. Merseguer, "Timing-failure risk assessment of uml design using time petri net bound techniques," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 1, pp. 90–104, 2011.

[10] D. M. Berry, J. Cleland-Huang, A. Ferrari, W. Maalej, J. Mylopoulos, and D. Zowghi, "Panel: Context-Dependent Evaluation of Tools for NL RE Tasks: Recall vs. Precision, and Beyond," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Sept 2017, pp. 570–573.

[11] D. M. Berry, R. Gacitua, P. Sawyer, and S. F. Tjong, "The case for dumb requirements engineering tools," in *Proceedings of the 18th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'12)*, ser. LNCS, vol. 7195. Springer, 2012, pp. 211–217.

[12] D. M. Berry, E. Kamsties, and M. M. Krieger, "From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity," 2003.

[13] D. M. Berry and E. Kamsties, "The syntactically dangerous all and plural in specifications," *IEEE Software*, vol. 22, no. 1, pp. 55–57, 2005.

[14] M. Biagi, L. Carnevali, M. Paolieri, and E. Vicario, "Performability evaluation of the ERTMS/ETCS - Level 3," *Transportation Research Part C: Emerging Technologies*, vol. 82, pp. 314–336, 2017.

[15] F. Bonin, F. Dell'Orletta, G. Venturi, and S. Montemagni, "A contrastive approach to multi-word term extraction from domain corpora," in *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC'10)*, 2010, pp. 19–21.

[16] G. Bucci, L. Carnevali, L. Ridi, and E. Vicario, "Oris: a tool for modeling, verification and evaluation of real-time systems," *International Journal for Software Tools for Technological Transfer*, vol. 12, no. 5, pp. 391–403, September 2010.

[17] G. Bucci, L. Sassoli, and E. Vicario, "Correctness verification and performance analysis of real time systems using stochastic preemptive Time Petri Nets," in *IEEE Trans Software Engineering*, vol. 31, no. 11, 2005, pp. 913–927.

[18] H.-D. Burkhard, "Ordered Firing in Petri Nets," *Elektronische Informationsverarbeitung und Kybernetik*, vol. 17, no. 2/3, pp. 71–86, 1981.

[19] L. Carnevali, L. Grassi, and E. Vicario, "State-density functions over dbm domains in the analysis of non-markovian models," *IEEE Transactions on Software Engineering*, vol. 35, no. 2, pp. 178–194, 2009.

[20] L. Carnevali, F. Flammini, M. Paolieri, and E. Vicario, "Non-Markovian Performability Evaluation of ERTMS/ETCS Level 3," in *Lecture Notes in Computer Science 9272*, EPEW 2015. Springer, 2015, pp. 47–62.

[21] L. Carnevali, L. Ridi, and E. Vicario, "Putting Preemptive Time Petri Nets to Work in a V-Model SW Life Cycle," *IEEE Trans. Softw. Eng.*, vol. 37, no. 6, pp. 826–844, Nov. 2011.

[22] ——, "Sirio: A Framework for Simulation and Symbolic State Space Analysis of non-Markovian Models," in *Proceedings of the 2011 Eighth International Conference on Quantitative Evaluation of SysTems*, ser. QEST '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 153–154.

[23] A. Casamayor, D. Godoy, and M. Campo, "Functional grouping of natural language requirements for assistance in architectural software design," *KBS*, vol. 30, pp. 78–86, 2012.

[24] CENELEC, "EN-50129: Railway Applications - Safety related electronic railway control and protection systems," Tech. Rep., 1994.

[25] ——, "EN-50126: Railway Applications - The specication and demonstration of dependability, reliability, availability, maintainability and safety." Tech. Rep., 1996.

[26] ——, "EN-50128:2011: Railway Applications - Communication, signalling and processing systems - Software for railway control and protection systems," Tech. Rep., 2011.

[27] ——, "EN-50159: Railway Applications - Communication, signaling and processing systems, Part 1: Safety-Related Communication in Closed Transmission Systems." Tech. Rep., 2011.

[28] F. Chantree, B. Nuseibeh, A. N. De Roeck, and A. Willis, "Identifying Nocuous Ambiguities in Natural Language Requirements," in *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*. IEEE, 2006, pp. 56–65.

[29] D. Charypar, K. Axhausen, and K. Nagel, "Event-driven queue-based traffic flow microsimulation," *Transportation Research Record: Journal of the Transportation Research Board*, pp. 35–40, 2007.

[30] K. Cho, B. Van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *CoRR*, vol. abs/1406.1078, 2014.

[31] H. Choi, V. G. Kulkarni, and K. S. Trivedi, "Markov regenerative stochastic petri nets," *Perform. Eval.*, vol. 20, no. 1-3, pp. 333–357, May 1994.

[32] I. Ciuti, "Jean-Luc Laugaa: "Ingorgo-trappola alla stazione, un rischio anche per la linea 2"," Repubblica.it, 2014. [Online]. Available: http://goo.gl/QxrXR4.

[33] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," in *ICSE (1)*. ACM, 2010, pp. 155–164.

[34] K. Collins-Thompson, "Computational assessment of text readability: A survey of current and future research," *ITL-International Journal of Applied Linguistics*, vol. 165, no. 2, pp. 97–135, 2014.

[35] H. Cunningham, "GATE, a general architecture for text engineering," *Computers and the Humanities*, vol. 36, no. 2, pp. 223–254, 2002.

[36] M. Cutts, *The plain English guide*. Oxford University Press, 1996.

[37] D. Méndez Fernández and S. Wagner and M. Kalinowski and A. Schekelmann and A. Tuzcu and T. Conte and R. Spinola and R. Prikladnicki, "Naming the Pain in Requirements Engineering: Comparing Practices in Brazil and Germany," *IEEE Software*, vol. 32, no. 5, pp. 16–23, 2015.

[38] L. Derczynski, D. Maynard, G. Rizzo, M. van Erp, G. Gorrell, R. Troncy, J. Petrak, and K. Bontcheva, "Analysis of named entity recognition and linking for tweets," *Information Processing & Management*, vol. 51, no. 2, pp. 32–49, 2015.

[39] D. L. Dill, "Timing assumptions and verification of finite-state concurrent systems," in *Automatic Verification Methods for Finite State Systems*, J. Sifakis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 197–212.

[40] C. Dobler and G. Lämmel, "Integration of a multi-modal simulation module into a framework for large-scale transport systems simulation," in *Pedestrian and Evacuation Dynamics 2012*. Springer International Publishing, 2013, pp. 739–754.

[41] ERTRAC, "ERTRAC Road Transport Scenario 2030+ "Road to Implementation"," European Road Transport Research Advisory Council, Tech. Rep., October 2009.

[42] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, "The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool," in *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*. IEEE, 2001, pp. 97–105.

[43] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182–211, 1976.

[44] D. Falessi, G. Cantone, and G. Canfora, "Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques," *IEEE Transactions on Software Engineering*, vol. 39, no. 1, pp. 18–44, 2013.

[45] A. Fantechi, P. Frasconi, G. Gori, F. Orsini, and M. Papini, "Defect detection and machine learning for requirement engineering: new roadmaps," 2018, Poster presented at NLP4RE'18 and REFSQ'18.

[46] H. Femmer, D. M. Fernández, S. Wagner, and S. Eder, "Rapid quality assurance with requirements smells," *Journal of Systems and Software*, vol. 123, pp. 190–213, 2017.

[47] H. Femmer, J. Kučera, and A. Vetrò, "On the impact of passive voice requirements on domain modelling," in *Proceedings of the 8th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'14), Art. 21.* ACM, 2014.

[48] A. Ferrari, F. Dell'Orletta, A. Esuli, V. Gervasi, and S. Gnesi, "Natural Language Requirements Processing: a 4D Vision," *IEEE Software (to appear)*, 2017.

[49] A. Ferrari, F. dell'Orletta, G. O. Spagnolo, and S. Gnesi, "Measuring and improving the completeness of natural language requirements," in *Proceedings of the 20th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'14).* Springer, 2014, pp. 23–38.

[50] A. Ferrari and S. Gnesi, "Using collective intelligence to detect pragmatic ambiguities," in *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE'12).* IEEE, 2012, pp. 191–200.

[51] A. Ferrari, G. Gori, B. Rosadini, A. Fantechi, S. Gnesi, I. Trotta, and S. Bacherini, "Detecting requirements defects with NLP patterns: an industrial experience in the railway domain," *Empirical Software Engineering*, 2018.

[52] A. Ferrari, G. O. Spagnolo, and S. Gnesi, "PURE: A Dataset of Public Requirements Documents," in *Requirements engineering Conference.* IEEE, 2017.

[53] A. Ferrari, P. Spoletini, and S. Gnesi, "Ambiguity and tacit knowledge in requirements elicitation interviews," *Requirements Engineering*, vol. 21, no. 3, pp. 333–355, 2016.

[54] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," *arXiv:1703.03400 [cs]*, Mar. 2017, 00052 arXiv: 1703.03400.

[55] H. Fujii, H. Uchida, and S. Yoshimura, "Agent-based simulation framework for mixed traffic of cars, pedestrians and trams," *Transportation Research Part C: Emerging Technologies*, vol. 85, pp. 234–248, 2017.

[56] R. Gacitua, P. Sawyer, and V. Gervasi, "On the effectiveness of abstraction identification in requirements engineering," in *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE'10).* IEEE, 2010, pp. 5–14.

[57] C. Gawron, "An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model," *International Journal of Modern Physics C*, vol. 9, no. 3, pp. 393–407, 1998.

[58] R. German, *Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets*. John Wiley & Sons, Inc., 2000.

[59] V. Gervasi and D. Zowghi, "Reasoning About Inconsistencies in Natural Language Requirements," *ACM Transactions on Software Engineering and Methodology*, vol. 14, no. 3, pp. 277–330, 2005.

[60] S. Ghaisas, P. Rose, M. Daneva, K. Sikkel, and R. J. Wieringa, "Generalizing by similarity: Lessons learnt from industrial case studies," in *Proceedings of the 1st international workshop on conducting empirical studies in industry*. IEEE Press, 2013, pp. 37–42.

[61] B. Gleich, O. Creighton, and L. Kof, "Ambiguity detection: Towards a tool explaining ambiguity sources," in *Proceedings of the 16th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'10)*, ser. LNCS, vol. 6182. Springer, 2010, pp. 218–232.

[62] S. Gnesi, G. Lami, and G. Trentanni, "An automatic tool for the analysis of natural language requirements," *International Journal of Computer Systems Science and Engineering*, vol. 20, no. 1, pp. 53–62, 2005.

[63] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin, "A model for technology transfer in practice," *IEEE software*, vol. 23, no. 6, pp. 88–95, 2006.

[64] G. Goth, "Deep or shallow, NLP is breaking out," *Communications of the ACM*, vol. 59, no. 3, pp. 13–16, 2016.

[65] A. Higgins, E. Kozan, and L. Ferreira, "Optimal scheduling of trains on a single line track," *Transportation Research Part B: Methodological*, vol. 30, no. 2, pp. 147 – 161, 1996.

[66] A. HorváTh, M. Paolieri, L. Ridi, and E. Vicario, "Transient analysis of non-markovian models using stochastic state classes," *Perform. Eval.*, vol. 69, no. 7-8, pp. 315–335, 2012.

[67] IEEE, "IEEE Guide for Developing System Requirements Specifications," *IEEE Std 1233, 1998 Edition*, pp. 1–36, Dec 1998.

[68] ISO, IEC, IEEE, "ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes –Requirements engineering," *ISO/IEC/IEEE 29148:2011(E)*, pp. 1–94, Dec 2011.

[69] E. Kamsties, "Understanding Ambiguity in Requirements Engineering," in *Engineering and Managing Software Requirements*. Springer Berlin Heidelberg, 2005, pp. 245–266.

[70] E. Kamsties, D. M. Berry, and B. Paech, "Detecting ambiguities in requirements documents using inspections," in *Proceedings of the 1st Workshop on Inspection in Software Engineering (WISE'01)*, 2001, pp. 68–80.

[71] N. Kang, E. M. van Mulligen, and J. A. Kors, "Comparing and combining chunkers of biomedical text," *Journal of biomedical informatics*, vol. 44, no. 2, pp. 354–360, 2011.

[72] B. S. Kerner, S. L. Klenov, and D. E. Wolf, "Cellular automata approach to three-phase traffic theory," *Journal of Physics A: Mathematical and General*, vol. 35, no. 47, pp. 9971–10 013, 2002.

[73] N. Kiyavitskaya, N. Zeni, L. Mich, and D. M. Berry, "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications," *Requirements Engineering*, vol. 13, no. 3, pp. 207–239, 2008.

[74] L. Kof, "From textual scenarios to message sequence charts: Inclusion of condition generation and actor extraction," in *Proceedings of the 16th IEEE International Requirements Engineering Conference, (RE'08)*. IEEE, 2008, pp. 331–332.

[75] ——, "Translation of textual specifications to automata by means of discourse context modeling," in *Proceedings of the 15th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'09)*, ser. LNCS, vol. 5512. Springer, 2009, pp. 197–211.

[76] ——, "From requirements documents to system models: A tool for interactive semi-automatic translation," in *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE'10)*. IEEE, 2010, pp. 391–392.

[77] D. Krajzewicz, J. Erdmann, J. Härri, and T. Spyropoulos, "Including pedestrian and bicycle traffic into the traffic simulation SUMO," in *ITS 2014, 10th ITS European Congress, 16-19 June 2014, Helsinki, Finland*, 2014.

[78] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner, "Sumo (simulation of urban mobility) - an open-source traffic simulation," in *4th Middle East Symposium on Simulation and Modelling*, 2002, pp. 183–187.

[79] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, pp. 159–174, 1977.

[80] J.-C. Laprie, *Dependability: Basic Concepts and Terminology.*, ser. Dependable Computing and Fault-Tolerant Systems, V. Springer, Ed., 1992, vol. 5.

[81] X. Lian, M. Rahimi, J. Cleland-Huang, L. Zhang, R. Ferrari, and M. Smith, "Mining Requirements Knowledge from Collections of Domain Documents Dependable Computing and Fault-Tolerant Systems," in *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE'16)*. IEEE, 2016, pp. 156–165.

[82] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," in *Proceedings of the 23rd IEEE International Requirements Engineering Conference, (RE'15)*. IEEE, 2015, pp. 116–125.

[83] C. D. Manning, "Part-of-speech tagging from 97% to 100%: is it time for some linguistics?" in *Proceedings of the 12th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing'11)*, ser. LNCS. Springer, 2011, vol. 6608, pp. 171–189.

[84] S. Martina, M. Paolieri, T. Papini, and E. Vicario, "Performance evaluation of fischer's protocol through steady-state analysis of markov regenerative processes," in *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2016, pp. 355–360.

[85] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (EARS)," in *Proceedings of the 17th IEEE International Requirements Engineering Conference (RE'09)*. IEEE, 2009, pp. 317–322.

[86] A. Mavin, P. Wilksinson, S. Gregory, and E. Uusitalo, "Listens learned (8 lessons learned applying EARS)," in *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE'16)*. IEEE, 2016, pp. 276–282.

[87] L. Mich, "NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA," *NLE*, vol. 2, no. 2, pp. 161–187, 1996.

[88] E. Nagy and C. Csiszár, "Analysis of Delay Causes in Railway Passenger Transportation," *Periodica Polytechnica Transportation Engineering*, vol. 43, no. 2, pp. 73–80, 2015.

[89] J. Ondráček, J. Schwarz, V. Ždímal, L. Andělová, P. Vodička, V. Bízek, C.-J. Tsai, S.-C. Chen, and J. Smolík, "Contribution of the road traffic to air pollution in the Prague city (busy speedway and suburban crossroads)," *Atmospheric Environment*, vol. 45, no. 29, pp. 5090–5100, 2011.

[90] M. Paolieri, A. Horvath, and E. Vicario, "Probabilistic model checking of regenerative concurrent systems," *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 153–169, 2016.

[91] Y. Papadopoulos and J. McDermet, "The Potential for a Generic Approach to Certification of Safety-Critical Systems in the Transportation Sector," *Journal of reliability engineering and system safety*, vol. 63, no. 1, pp. 47–66, 1999.

[92] G. Peng, X. Cai, C. Liu, B. Cao, and M. Tuo, "Optimal velocity difference model for a car-following theory," *Physics Letters A*, vol. 375, no. 45, pp. 3973 – 3977, 2011.

[93] C. A. Petri, "Communication with automata," Ph.D. dissertation, 1966.

[94] K. Pohl and C. Rupp, *Requirements engineering fundamentals.* Rocky Nook, Inc., 2011.

[95] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[96] D. M. W. Powers, "Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.

[97] T. Quirchmayr, B. Paech, R. Kohl, and H. Karey, "Semi-automatic Software Feature-Relevant Information Extraction from Natural Language User Manuals," in *Proceedings of the 23rd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'17).* Springer, 2017, pp. 255–272.

[98] M. Ren, S. Ravi, E. Triantafillou, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel, "Meta-Learning for Semi-Supervised Few-Shot Classification," Jan. 2018, 00000.

[99] M. Robeer, G. Lucassen, J. M. E. van der Werf, F. Dalpiaz, and S. Brinkkemper, "Automated extraction of conceptual models from user stories via NLP," in *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE'16).* IEEE, 2016, pp. 196–205.

[100] B. Rosadini, A. Ferrari, G. Gori, A. Fantechi, S. Gnesi, I. Trotta, and S. Bacherini, "Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain," in *Proceedings of the 23rd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'17)*, ser. LNCS, vol. 10153, 2017, pp. 344–360.

[101] L. H. Rosenberg, F. Hammer, and L. L. Huffman, "Requirements, testing and metrics," in *In 15th Annual Pacific Northwest Software Quality Conference*, 1998.

[102] RTCA Inc. and EUROCAE, "DO-178C: Software Considerations in Airborne Systems and Equipment Certification," Tech. Rep., 2012.

[103] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case study research in software engineering: Guidelines and examples.* John Wiley & Sons, 2012.

[104] F. Schreiber and A. Morzenti, "Problematiche di certificazione del software safety critical. Parte I - Generalitá, elicitazione e specifica dei requisiti." *Ingegneria Ferroviaria*, vol. LIV, no. 10, pp. 709–746, 1999.

[105] J. Shi, Y. Sun, P. Schonfeld, and J. Qi, "Joint optimization of tram timetables and signal timing adjustments at intersections," *Transportation Research Part C: Emerging Technologies*, vol. 83, pp. 104–119, 2017.

[106] F. Shull, I. Rus, and V. Basili, "How perspective-based reading can improve requirements inspections," *IEEE Computer*, vol. 33, no. 7, pp. 73–79, 2000.

[107] H. Sultanov and J. H. Hayes, "Application of reinforcement learning to requirements engineering: requirements tracing," in *Proceedings of the 21st IEEE International Requirements Engineering Conference (RE'13)*. IEEE, 2013, pp. 52–61.

[108] F. Sung, L. Zhang, T. Xiang, T. Hospedales, and Y. Yang, "Learning to Learn: Meta-Critic Networks for Sample Efficient Learning," *arXiv preprint arXiv:1706.09529*, 2017, 00000.

[109] T. Tang, Y. Wang, X. Yang, and Y. Wu, "A new car-following model accounting for varying road condition," *Nonlinear Dynamics*, vol. 70, no. 2, pp. 1397–1405, 2012.

[110] J. Terzakis and S. Gregory, "RAMP: requirements authors mentoring program," in *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE'16)*. IEEE, 2016, pp. 323–328.

[111] S. F. Tjong and D. M. Berry, "The Design of SREE: A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned," in *Proceedings of the 19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'13)*, ser. LNCS, vol. 7830. Springer, 2013, pp. 80–95.

[112] O. K. Tonguz, W. Viriyasitavat, and F. Bai, "Modeling urban traffic: A cellular automata approach," *IEEE Communications Magazine*, vol. 47, no. 5, pp. 142–150, 2009.

[113] E. Vicario, "Static Analysis and Dynamic steering of time dependent systems using Time Petri Nets," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 728–748, 2001.

[114] E. Vicario, L. Sassoli, and L. Carnevali, "Using stochastic state classes in quantitative evaluation of dense-time reactive systems," in *IEEE Trans Software Engineering*, vol. 35, no. 5, 2009, pp. 703–719.

[115] R. Wieringa and M. Daneva, "Six strategies for generalizing software engineering theories," *Science of computer programming*, vol. 101, pp. 136–152, 2015.

[116] M. Wilmink and C. Bockisch, "On the Ability of Lightweight Checks to Detect Ambiguity in Requirements Documentation," in *Proceedings of the 23rd*

International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'17), ser. LNCS, vol. 10153. Springer International Publishing, 2017, pp. 327–343.

[117] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt, "Automated analysis of requirement specifications," in *Proceedings of the 19th international conference on Software engineering*. ACM, 1997, pp. 161–171.

[118] H. Yang, A. N. D. Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Analysing anaphoric ambiguity in natural language requirements," *Requirements Engineering*, vol. 16, no. 3, pp. 163–189, 2011.

[119] J. Yang, W. Deng, J. Wang, Q. Li, and Z. Wang, "Modeling pedestrians' road crossing behavior in traffic system micro-simulation in china," *Transportation Research Part A: Policy and Practice*, vol. 40, no. 3, pp. 280–290, 2006.

[120] R. K. Yin, *Case study research: Design and methods*. Sage publications, 2013.

[121] S. Yoshimura, "MATES : Multi-agent based traffic and environmental simulator-theory, implementation and practical application," *Computer Modeling in Engineering and Sciences*, vol. 11, no. 1, pp. 17–25, 2006.

[122] T. Yue, L. C. Briand, and Y. Labiche, "aToucan: an automated framework to derive UML analysis models from use case models," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 24, no. 3, p. 13, 2015.

[123] W. Zeng, P. Chen, H. Nakamura, and M. Iryo-Asano, "Application of social force model to pedestrian behavior analysis at signalized crosswalk," *Transportation Research Part C: Emerging Technologies*, vol. 40, pp. 143–159, 2014.

[124] H. Zhang, T. Yue, S. Ali, and C. Liu, "Towards mutation analysis for use cases," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. ACM, 2016, pp. 363–373.

[125] L.-J. Zheng, C. Tian, D.-H. Sun, and W.-N. Liu, "A new car-following model with consideration of anticipation driving behavior," *Nonlinear Dynamics*, vol. 70, no. 2, pp. 1205–1211, 2012.

[126] D. Zowghi, V. Gervasi, and A. McRae, "Using default reasoning to discover inconsistencies in natural language requirements," in *Proceedings of the 8th Asia-Pacific Software Engineering Conference (APSEC'01)*, 2001, pp. 133–140.