REBox: Collaborative Environment for Requirements Engineering

João Pedro Nunes da Costa Fernandes

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Alberto Manuel Rodrigues da Silva

Examination Committee
Chairperson: Prof. José Carlos Alves Pereira Monteiro
Supervisor: Prof. Alberto Manuel Rodrigues da Silva
Member of the Committee: Prof. Ana Cristina Ramada Paiva

October 2016
Acknowledgments

First, I would like to thank my supervisor, Professor Alberto Silva, for his guidance during the whole process. His constant availability for answering any question and his words of encouragement were always very important. Furthermore, his expertise in the field of Requirements Engineering and his suggestion on the topics to investigate were fundamental to steer the thesis in the right direction.

I also would like to thank all the colleagues at INESC-ID who helped develop and test the platform.

I would especially like to thank my family for all the support throughout the time it took to reach this point. To my father, my sister, my grandmother and especially my mother, Isabel, an enormous “thank you”. And also a very special thanks to my grandfather, who got sick and passed away during the development of this thesis, but never ever stopped encouraging me. Wherever he may be, this work is also his.

And last but certainly not least, I would like to thank you, Helena, for your unwavering support and your kind words of encouragement, but also for your tough love when I was almost about to give up. That was, without a doubt, what got me to deliver this. I hope I can now help you now as much as you helped me. Best of luck to your exam and may we be traveling the world two months from now.
Abstract

Requirements engineering is an extremely important area on the development process of software systems that requires a good communication between stakeholders to achieve a shared vision of the problems to be addressed. Therefore, an efficient collaborative tool to manage requirements specification documents is crucial to guarantee an alignment of expectations between all involved parties. Collaborative RE tools currently available already offer ways to manage those documents, but are more focused on the organization and traceability of requirements, rather than the quality and rigorousness of their representation. Furthermore, there are still RE concerns which have not yet been properly addressed, such as the lack of proposals to efficiently model variability aspects of software systems at the requirements level.

Therefore, a platform that could offer good baseline usability features, a formal structure for modeling RE concerns and a way to efficiently manage requirements variability would not only provide a well-rounded approach to RE management, but also contribute to a faster and less error-prone requirements documents creation process.

This dissertation proposes REBox, a collaborative platform for RE supported by a language designed towards rigorous requirements specifications. The platform was conceived to provide a familiar but powerful editor, as well as a set of strong reusability features. In addition, it offers an innovative approach for managing requirements variability, based on the recent OMG proposal for a domain-independent variability modeling standard: the Common Variability Language(CVL).

Keywords: Requirements Engineering, Collaborative Platforms, Reusability, Variability Modeling
Resumo

Engenharia de Requisitos é uma área extremamente importante no processo de desenvolvimento de sistemas de software que requer uma estreita colaboração entre stakeholders no sentido de alcançar uma visão partilhada dos problemas a resolver. Devido a isso, torna-se particularmente importante a utilização de uma ferramenta colaborativa para gerir documentos de especificação de requisitos, no sentido de garantir o alinhamento de expectativas através de uma melhor comunicação entre as partes interessadas. As ferramentas colaborativas atualmente disponíveis para a área de ER já oferecem a possibilidade de gerir estes documentos, mas estão mais direcionadas para garantir uma boa organização e rastreabilidade dos requisitos do que a qualidade dos mesmos. Além disso, existem questões nesta área que não foram ainda devidamente aprofundadas, como a falta de propostas para modelar eficientemente aspetos de variabilidade de sistemas de software ao nível dos requisitos.

E portanto, uma plataforma que apresente simultaneamente boa usabilidade, uma estrutura formal para modelar conceitos de ER e uma forma eficiente de gerir variabilidade ao nível dos requisitos, pode contribuir para um processo de criação de documentos de requisitos mais rápido e menos sujeito a erros.

Esta dissertação propõe REBox, uma plataforma colaborativa para ER suportada por uma linguagem desenhada para criar especificações de requisitos mais rigorosas. A plataforma oferece um editor familiar mas poderoso, bem como um conjunto de funcionalidades direcionadas para reutilização. Para além disso, propõe uma abordagem inovadora para gerir variabilidade de requisitos, baseada na recente proposta da OMG para uma norma de modelação de variabilidade: a linguagem CVL.

Palavras-Chave: Engenharia de Requisitos, Plataformas Colaborativas, Reutilização, Modelação de Variabilidade
# Contents

<table>
<thead>
<tr>
<th>List of Tables</th>
<th>xi</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>xiii</td>
</tr>
<tr>
<td>Acronyms</td>
<td>1</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Context .................................. 4
1.2 Problem Definition and Research Questions ................. 4
1.3 Proposed Solution .................................. 5
1.4 Thesis Statement .................................. 6
1.5 Research Methodology .................................. 6
   1.5.1 First Iteration .................................. 6
   1.5.2 Second Iteration .................................. 7
   1.5.3 Integration, testing and validation of the results ........ 8
   1.5.4 Publications .................................. 8
1.6 Outline .................................. 8

## 2 Related Work

2.1 Requirements Management Systems ...................... 11
   2.1.1 Dedicated RE Platforms ...................... 12
   2.1.2 RE Wikis .................................. 15
2.2 Software Variability Modeling ...................... 18
   2.2.1 The Common Variability Language ............ 19
   2.2.2 Variation Points .................................. 20
   2.2.3 Variability Specifications (VSpecs) ........ 21
   2.2.4 Constraints .................................. 21

## 3 Research Context

3.1 The RSLingo Approach .................................. 23
3.2 The RSL-IL Excel Template .................................. 24
3.3 The RSLingo Studio Tool .................................. 25
3.4 Case Study .................................. 26

## 4 The REBox Platform

4.1 Main Goals .................................. 29
4.2 Overview & Features .................................. 30
   4.2.1 Template Management System .................. 30
   4.2.2 RS Library .................................. 31
List of Tables

2.1 Requirements Tools Comparison .................................................. 12
2.2 RE Dedicated Platforms Comparison .......................................... 15
2.3 Summary of the Feature Modeling Approaches. Compiled from (Bosch et al., 2013) &
(Bosch et al., 2015) ................................................................. 19

4.1 Properties of the var.specs View ................................................ 33
4.2 Properties of the var.specs.constraints View ............................... 33

6.1 Survey average score (in a scale of 0-5) by question for the REBox Web Platform aspect. 46
6.2 Survey average score (in a scale of 0-5) by question for the REBox Variability Framework
aspect ................................................................................... 47
6.3 Survey average score (in a scale of 0-5) by question for the REBox General Approach
aspect ................................................................................... 47
6.4 Survey average score (in a scale of 0-5) for each of the questionnaire aspects .................. 47

A.1 Properties of the Goals View in the RSL-IL Template .................. 59
A.2 Properties of the Functional Requirements View in the RSL-IL Template .... 59
A.3 Properties of the Quality Requirements View in the RSL-IL Template ....... 60
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Example of a Requirements Management Menu from Spiratest.</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>SWORE use and interlinking with domain ontologies (Riechert et al., 2007).</td>
<td>16</td>
</tr>
<tr>
<td>2.3</td>
<td>Overview of ReqWiki’s Architecture (Riechert et al., 2007).</td>
<td>17</td>
</tr>
<tr>
<td>2.4</td>
<td>CVL Language Variability Model Overview [OMG, 2012].</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>An example of a VSpecs Tree [OMG, 2012].</td>
<td>22</td>
</tr>
<tr>
<td>3.1</td>
<td>Overview of the RSLingo approach at project-level (De Almeida Ferreira and Da Silva, 2012).</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Example of a Quality Requirements sheet of the RSL-IL Excel Template.</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>The REBox Web Platform Overview.</td>
<td>31</td>
</tr>
<tr>
<td>4.2</td>
<td>The REBox Variability Modeling process.</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>The REBox Requirements Variability Binding Model.</td>
<td>34</td>
</tr>
<tr>
<td>4.4</td>
<td>An example of REBox’s Variation Points definition, the associated VSpecs and Constraints and on the right side a representation of the corresponding VSpecs Tree.</td>
<td>35</td>
</tr>
<tr>
<td>5.1</td>
<td>The REBox Web Platform Domain Model.</td>
<td>37</td>
</tr>
<tr>
<td>5.2</td>
<td>Overview of the DNN CMS architecture [OMG].</td>
<td>39</td>
</tr>
<tr>
<td>5.3</td>
<td>REBox Templates Page.</td>
<td>40</td>
</tr>
<tr>
<td>5.4</td>
<td>REBox project edit screen.</td>
<td>41</td>
</tr>
<tr>
<td>5.5</td>
<td>Json.NET object serialization example.</td>
<td>42</td>
</tr>
<tr>
<td>5.6</td>
<td>REBox RSLib Module Editor screen.</td>
<td>42</td>
</tr>
<tr>
<td>5.7</td>
<td>REBox Variability points selection screen.</td>
<td>43</td>
</tr>
</tbody>
</table>
Acronyms

CMS  Content Management System. 5
CVL  Common Variability Language. 4 19
DNN  DotNetNuke. 5
DSL  Domain Specific Language. 26
M2M  Model-To-Model. 26
M2T  Model-To-Text. 26
MOF  Meta-Object Facility. 19
OCL  Object Constraint Language. 21
OMG  Object Management Group. 19
RE  Requirements Engineering. 3
SPL  Software Product Line. 11
SRS  Software Requirements Specification. 25 26
T2M  Text-To-Model. 26
Chapter 1

Introduction

**RE** defines a set of tasks to build a Requirements Specification Document, most of which demand a high level of collaboration between business and technical stakeholders in order to achieve a mutual understanding of the software system to be developed. The intrinsically abstract nature that characterizes the software conceptualization phase can often lead to a misalignment of the views amongst the various stakeholders, which ultimately leads to a system that doesn’t match the initial vision [Kappelman et al., 2009]. Correcting the consequences of those early misunderstandings is often a costly endeavor [Davis, 2013], requiring a new elicitation of the requirements and a rework of the project to accommodate the changes.

A good collaborative environment to manage these requirements specifications is, therefore, crucial to guarantee an alignment of expectations and an overall better communication of the problem amongst stakeholders. Several software tools already offer ways to organize and share those documents, ranging from simple text and spreadsheet editors to more advanced commercial platforms specifically designed for the purpose. Furthermore, recent academic works have proposed wiki-based collaborative systems with integrated semantic assistants for analyzing requirements supported by RE-specific ontologies.

However, trade-offs must be made when choosing between these tools. Widely used text and spreadsheet editors can be free and easy to learn, but lack many of the advanced capabilities present in RE management commercial systems. Likewise, these systems focus on strong traceability features but are often little more than databases, lacking any meaningful ways of analyzing the requirements in regards to their correctness, consistency, completeness and ambiguity. Semantic Wikis like the ones described before do address this lack of analysis by providing mechanisms to extract relevant information contained within the requirements and verifying it against pre-defined RE ontologies in an effort to detect inconsistencies. However, and given the difficult task of analyzing contextless, ad-hoc natural language texts, most of these wikis have not ultimately provided practical results.

Moreover, there are still concerns in the field of RE for whom none of these tools provide clear solutions, such as the lack of efficient proposals to model variability in requirements specifications. This is a relevant issue because during the development of many software projects, it is common that some of the original features will need to be adapted in order to meet specific customers’ needs. To manage this variability, companies often begin developing customer-specific versions of the original product, changing and adding features for each new version. As the number of versions grow, however, so does the effort needed to model all the variation points and respective variants (alternatives for each point). In some cases, the sheer number of these variation points increases the complexity of the project to the point of making it almost impossible to manage.

Naturally, this variability will need be reflected at the requirements specification level, but software companies prefer to express product variability in terms of features because it is closer to how designers...
understand and represent it (Bosch et al., 2015).

This happens mainly because since its first introduction, feature modeling (Kang et al., 1990) has been the most popular technique to model commonality and variability (C&V) of products of a product line. Commonalities and variabilities are modeled from the perspective of product features, “stakeholder visible characteristics of products” in a product line that are of stakeholders’ concern (Bosch et al., 2013). With the advent of software product lines for building multiple and related products in a given domain, variability models have equally increased in popularity.

Common Variability Language (CVL) (Haugen et al., 2008) comes as the OMG’s latest proposal for a Variability Modeling standard, defining itself as a domain-independent language for specifying and resolving variability. In practice, this allows the representation the system’s features variability to be independent from the DSL used to model those features. This domain-independence makes CVL an ideal language to develop a framework for modeling variability at the requirements level, incentivizing experimentation and the proposal of new approaches in the field.

1.1 Context

This research work has been conducted at the Information and Decision Support Systems Lab[1] of INESC-ID (Instituto de Engenharia de Sistemas e Computadores – Investigação e Desenvolvimento) under the supervision of Professor Alberto Rodrigues da Silva, regarding the Master Degree in Information Systems and Computer Engineering at Instituto Superior Técnico. The work results from an ongoing research initiative in area of Requirements Engineering and is based in two proposals: ProjectIT and RSLingo, which are briefly described below:

- **ProjectIT** (Da Silva et al., 2007). The ProjectIT Approach was proposed by the Information Systems Group, states that the emphasis of software development should be on project management, requirements engineering and design activities, so that production activities (such as programming and testing) can be minimized and automated as much as possible. Consequently, it focused on the RE and MDE fields of research, in an effort to enhance the productivity and rigor of Software Engineering activities. As a result, a wiki was proposed which provides an integrated environment to support tasks ranging from requirements specification, architecture definition, system design and modeling, until code generation (Ferreira and da Silva, 2009).

- **RSLingo** (De Almeida Ferreira and Da Silva, 2012). The RSLingo initiative states that natural language, although being the most common and preferred form of representation used within requirements documents, is prone to produce ambiguous and inconsistent specification documents. The proposal presents an approach to use simplified Natural Language Processing (NLP) techniques for capturing relevant information from these Ad-Hoc natural language requirements specifications and then applying lightweight parsing techniques to extract domain knowledge encoded within them. Using a set of languages and processes, the initial knowledge written in natural language can be extracted, parsed and converted to more a more formal structure, reducing its original ambiguity and creating a new and more refined software requirements specification document.

1.2 Problem Definition and Research Questions

As introduced previously, several collaborative tools for managing requirements specifications are already available, but they vary in terms of the benefits they can provide and often require users to choose

between better usability and more complex features. Furthermore, there are still concerns in the field of requirements engineering that have not yet been fully addressed by any current tools, such as defining efficient ways to model variability aspects in requirements specifications. Variability modeling proposals in the field of software development thus far have mostly been directed towards the features of the systems, rather than their requirements, which comes from the fact that features are often closer to how stakeholders perceive and understand variability.

And so, the main problems addressed in this dissertation are summarized by the following research questions:

**RQ1:** How do collaborative tools that are currently available for managing software requirements specification documents compare amongst themselves?

**RQ2:** Which RE concerns should be addressed when designing a new RE management tool and its features?

**RQ3:** What are the standards for modeling variability aspects of systems at the RE level?

### 1.3 Proposed Solution

This research proposes REBox, a collaborative platform for managing requirements specifications with a focus on reusability and variability modeling.

REBox is a web-based platform that offers a management system for requirements specifications. Its reusability features include a Document Template System for uploading/managing SRS templates and a Requirements Specifications Library (RSLib) for creating and reusing modular sets of reusable requirements. It also offers an innovative variability modeling framework that leverages the concepts of the CVL language to model variability aspects in the context of Requirements Engineering. The framework defines a process for applying the language’s central variability notions - the Variation Points - to the views of a structured software requirements specification document, modeled using a rigorous requirements specification language: RSL-IL (De Almeida Ferreira and Da Silva, 2013a).

The web platform has been built using the DotNetNuke (DNN) .NET web Content Management System (CMS) and employs the Google Drive cloud storage service. The Google technology provides both the Requirements Specification Document (RSDoc) editor through Google Sheets as well as the collaborative aspect through its synchronization system and distributed architecture. In addition, REBox uses a series of Google provided APIs for automating processes and managing data, which helped the development of its reusability features and variability framework.

The final solution is a collaborative platform that allows a decentralized editing of requirements specifications through a familiar but powerful editor, that uses the RE-specific RSL-IL language to provide a rigorous model for its requirements specifications.

Given the proposed solution, the main research goals of this dissertation can be summarized as follows:

**RG1:** Investigate the current tools and technologies used for Software Requirement Specification documents management and also the latest proposals in the field of Requirements Variability Modeling;

**RG2:** Design and implement a collaborative platform for managing requirements specifications documents;
RG3: Develop a requirements variability modeling framework based on the CVL language;

RG4: Evaluate the quality of the proposed system and its features system through a pilot-user test session.

1.4 Thesis Statement

This dissertation thesis states that there is possible to develop a collaborative environment for RE based on a simple but extensible technology, that allows for both an efficient management of requirements documents, and a more rigorous definition of the concerns encoded in those requirements. In addition, it also states that the current proposals for variability modeling in software feature models can be extended to the field of RE to clearly express variation points between multiple requirements documents in a software product line.

In particular, I claim that this can be achieved by using the Google Drive cloud technology to provide an accessible and familiar collaborative spreadsheet editor, allied with the rigorous RSL-IL structured Excel representation. Furthermore, the set of powerful APIs provided by Google to manage the data encoded in those sheets, allowed the development of several requirements reusability features, including a variability modeling framework that is based in the recently proposed domain-independent CVL variability language.

1.5 Research Methodology

This research work has been conducted in an iterative and gradual way following the Action Research methodology (Baskerville, 1999). This methodology suggests a cyclical process composed of five steps executed in a certain scope, known as client-system infrastructure. The five steps are the following: (1) Diagnosing - identification of the problem domain, (2) Action Planning - planning and definition of the proposed solution, (3) Action Taking - implementation of the solution, (4) Evaluating - evaluation/assessment of the solution developed and (5) Specifying Learning - lessons learned during the cycle and preparation of the next one. The research work in this dissertation had two key iterations:

1.5.1 First Iteration

The first iteration lasted from September 2015 to April 2016. This iteration covered the background investigation, planning and implementation of the main proposal of this dissertation: the REBox collaborative platform. The Action Research steps that comprised this phase are described below.

Diagnosing - This step consisted in the review of the current collaborative technologies used in the area of Requirements Engineering. This review started by focusing on the more simple text and spreadsheet editors, and then the more advanced commercial platforms. Lastly, some of the more relevant academic proposals for collaborative RE Wikis were also reviewed. At the end of this research, a comparative study was developed analyzing the pros and cons of every type of tool.

Action Planning - The planning step consisted in the definition of the architecture of the REBox system, including its base technologies, its features and the integrations with other systems.
**Action Taking** - The action taking step consisted on the implementation of the proposed platform, using the DNN .NET CMS technology. After the web platform was setup and functional (views, data model, user and permission systems), the work shifted to programming the Google Drive API calls for uploading and sharing the files to the cloud server and then embedding the spreadsheet editor view (Google Sheets) in the REBox project management views. With the integration working, the last phase consisted in implementing the Google Sheets API calls for data manipulation (reading and writing data from and to the editor). These new functions allowed the development of the RSLib system and laid the work for the variability modeling framework to be developed in the next iteration.

**Evaluating** - The evaluation of the work was done by developing the Billing System RSL-IL template, especially the Goals, Functional Requirements and Quality Requirements views, and using it to test the platform’ features and the Google Drive integration.

**Specifying Learning** - After the first iteration was completed and the tests provided positive results, it came the conclusion that the data manipulation functions built to integrate with the Google Sheets API could be easily extended for automating other processes in the context of RE, which led to the second iteration of the project.

1.5.2 Second Iteration

The second iteration lasted from May 2016 to August 2016. Regarding the particular concern of reusability, it was identified a lack of clear RE-specific proposals for modeling variability aspects in in requirements documents in the context of software product lines, and so the second iteration consisted lasted the next 4 months and consisted in investigating the current feature variability modeling proposals, and designing a framework for adapting it to the context of the REBox platform using the previously developed data manipulating functions with the Google Sheets API. This iteration originated the submission of a paper describing Requirements Variability Modeling Process. Again, the Action Research steps that comprised this phase are described below.

**Diagnosing** - This step consisted in investigating the area of feature modeling, which had its first proposal in 1990 and has since been adopted as the main approach for modeling software variability aspects in the context of product lines. The original proposal has received a significant number of incremental extensions, the last one being the OMG’s proposal for a domain-independent language in the form of the CVL language. This language was chosen as the base foundation for the requirements variability framework in REBox, and, as such, constituted the major focus of investigation during this step.

**Action Planning** - The planning step consisted in the design of the requirements variability modeling framework based on the CVL language. The main work consisted in mapping the concepts of the language such as the Variation Points and the VSspecs, as well as the main processes, such as the Variability Binding and the Materialization to the new framework and the views of the RSL-IL template.

**Action Taking** - This step consisted in extending the API calls and data management functions created in the first iteration to the new processes defined by the requirements variability modeling process.
Evaluating - The evaluation of the model in this phase was primarily done by comparing it to the latest proposals in the field. However, not many literature was yet written, particularly referring the concrete use of the CVL language, and so the comparison was primarily made with other feature modeling approaches for software product lines that somewhat also included references to the requirements specification phase. Furthermore, the new variability modeling features were also tested using the Billing System RSL-IL template.

Specifying Learning - The tests effectuated on the new developments were positive and left some ideas for future work, such as the extension of the variability modeling supported views from simply requirements (Goals, Functional and Quality) to other concerns in the scope of RE, such as Stakeholders, Use Cases, etc..

1.5.3 Integration, testing and validation of the results

The last month of the work was reserved mainly for integrating the work done in the two iterations and then conducting an user evaluation session to test and validate the results of the developed platform and its variability modeling framework.

1.5.4 Publications

Earlier this month, a paper was submitted for the RE track of the SAC’ 2017 conference, titled “Variability Aspects in Requirement Specifications: Model and Implementation based on the CVL Standard”. The paper described the REBox Requirements Variability Model and it is currently being evaluated and awaiting approval.

1.6 Outline

The remainder of this dissertation is organized as follows:

Chapter 2 - This chapter provides an overview of the main concepts and related work that underlie this research, particularly focusing on two main topics: Requirements Management Systems and Software Variability Modeling.

Chapter 3 - This chapter presents the context of the work proposed in this dissertation.

Chapter 4 - This chapter presents the REBox web platform architecture and design choices, as well as the main technologies used. It also describes its main features, including an in-depth view of the platform’s Variability Modeling approach, based on the CVL language.

Chapter 5 - This chapter presents and discusses the evaluation performed on the REBox system and its results.

Chapter 6 - This chapter presents the main conclusions of this work along with the future development perspectives.

In addition, there are the following appendixes:
Appendix A - Contains a representation of the RSL-IL Excel Template Requirements Views and their Properties.

Appendix B - Contains the user session guide used during the work evaluation.

Appendix C - Contains the detailed User Session questionnaire results.
Chapter 2

Related Work

This chapter overviews the main technologies and concepts that constitute the base for this research work: Requirements Management Systems and Variability Modeling in the context of Software Product Lines (SPL). Section 2.1 presents a comparative survey on the current collaborative systems and technologies used for managing Requirements Engineering Projects. Section 2.2 introduces the concept of Variability Modeling in the context of SPLs and presents some of the most important languages and processes proposed in this research field, with a particular focus on the latest OMG proposal for a variability modeling standard: the Common Variability Language.

2.1 Requirements Management Systems

In an effort to build a RE platform, it’s important to understand what tools are currently being used to create and manage requirements specifications, what purposes each one serves and what are the underlying architectures. Three main categories were identified, each with advantages and disadvantages, presented in Table 2.1.

- **Text and Spreadsheet editors:** Although lacking many of the fundamental features of dedicated RE tools, and often being disincentivized for big projects with many requirement dependencies that quickly become very hard to track, these kind of editors, such as Word or Excel, have always been a popular choice to manage requirements specifications due to their widespread availability and simple interface, making them a good baseline tool for nontechnical stakeholders. Until recently, one of the main problems with this desktop software was to maintain synchronized and updated versions of the documents in decentralized projects requiring multiple concurrent changes. Recent cloud storage services, however, such as Google Drive or OneDrive, have helped on that regard by offering collaborative environments to manage these kind of documents with automatic synchronization and lightweight versioning tools.

- **Dedicated RE Platforms:** These platforms present a general set of functionalities with a strong emphasis on stakeholder collaboration, requirement traceability, quality assurance and overall enterprise workflow integration. Many are now fully web-based, while some still maintain a desktop core component and provide collaboration through browser clients with reduced functionalities. To guarantee concurrent use and data synchronization, most operate under a client-server architecture, with rich clients main-
taining the business logic, and servers keeping the databases. Some of the main problems associated with the more advanced platforms are their complexity, often presenting a steep learning curve for non-technical stakeholders, and also high costs when compared with other open-source solutions (Decker et al., 2007). However, from this project’s point of view, their main shortcoming lies in the fact that they still treat requirements as black-box elements, ignoring their natural-language semantics, and thus becoming exposed to all of the problems that often come from working with unprocessed natural language written requirements, such as incorrectness, inconsistency, incompleteness and ambiguity.

**RE-Specific Wikis:** Wikis provide a free and collaborative environment for creating, modifying and deleting content, generating an expandable collection of interlinked web pages. Because of that, many projects have used wikis as a baseline tool for developing extensions aimed at supporting intrinsically collaborative areas, such as Requirements Engineering. (West and West, 2008) (Lai et al., 2012). With the recent development of the Semantic Web paradigm, many wikis have been developed to take advantage of the new technologies that were introduced. Data in a wiki page can now be captured or identified and can be related to data in other wiki pages. Consequently, navigation, search, retrieval, and presentation can be improved. Another point of interest is the better means to support interoperability. With semantic wikis the interoperability among each other or with external services is increased (Hoenderboom and Liang, 2009). Ontology languages are also amongst the new semantic web technologies that allow for formal descriptions of knowledge in various domains. With help of the new data capture and interoperability features, and with the use of underlying RE ontologies, some new RE wiki projects have proposed and/or developed automatic data extraction methods that not only provide new ways to increase requirements traceability but also allow the development of tools to process that data using NLP pipelines in an effort to increase requirement quality.

<table>
<thead>
<tr>
<th>PROS</th>
<th>CONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Text and spreadsheet editors</strong></td>
<td>Low cost (often free)</td>
</tr>
<tr>
<td></td>
<td>Easy to use</td>
</tr>
<tr>
<td></td>
<td>Lightweight collaboration and versioning support tools</td>
</tr>
<tr>
<td><strong>Dedicated RE Platforms</strong></td>
<td>Collaboration-focused</td>
</tr>
<tr>
<td></td>
<td>Strong traceability features</td>
</tr>
<tr>
<td></td>
<td>Good quality assurance with automatic test generation and defect tracking</td>
</tr>
<tr>
<td></td>
<td>Good developer tools</td>
</tr>
<tr>
<td></td>
<td>Low cost (mostly open-source)</td>
</tr>
<tr>
<td></td>
<td>Good collaborative built-in features</td>
</tr>
<tr>
<td></td>
<td>Extensible Architecture</td>
</tr>
<tr>
<td></td>
<td>Allow for integrated NLP assistants</td>
</tr>
</tbody>
</table>

Table 2.1: Requirements Tools Comparison

### 2.1.1 Dedicated RE Platforms

As discussed in section 3.1, dedicated RE platforms offer a set of functionalities strongly focus on collaboration, requirement traceability and project life-cycle integration. Below are presented some of the most prominent platforms currently available on the market.

**IBM Rational DOORS:** Originally developed by Quality Systems and Software (QSS) in 1991, it had its first commercial release in 1993. It was later acquired by Telelogic in 2000, which in turn was acquired by IBM in 2008. Rational Dynamic Object Oriented Requirements Systems (DOORS) is a very
mature and fully-fledged software that uses its own programming language similar to C and C++ - the DOORS eXtension language (DXL) - which offers its users a programmable API to customize a large set of activities, from creating multi-level traceability views to developing integrations with other tools. It operates under a Client-Server paradigm, with all the business logic being managed Client-Side, while the Server keeps a central Database. To provide user collaboration, it also comes with a lightweight optional web-based client - DOORS Web Access - that supports review and common editing tasks in a web browser.

Recently, IBM released another platform with similar capabilities: Rational DOORS Next Generation. Contrary to its predecessor, this is a fully web-based tool that is currently being improved to include all the features already present in DOORS. It runs on the IBM Rational Jazz Integration Server and can be installed and integrated with the Rational solution for Collaborative Lifecycle Management (CLM), providing a foundation of capabilities that allow distributed, multidisciplinary teams to collaborate on all aspects of the development process.

Both products provide strong traceability tools to manage requirements across the development lifecycle, support requirements-driven development and testing, and link to design and modeling resources.

**Spiratest:** Developed by Inflectra, this web-based platform focus on a core RE environment with strong quality assurance and test management tools. It offers the ability to create, edit and delete project scope and requirements in a hierarchical organization that resembles a typical scope matrix. In addition, each requirement is mapped to one or more test cases that can be used to validate that the functionality works as expected. It also provides a bug tracker tool to tack defects and trace them back to the respective test case and the underlying requirement. It supports external Add-Ons development with an open SDK, and offers integrations with IDEs such as VisualStudio and Eclipse, as well as other testing and bug tracking tools.

![Figure 2.1: Example of a Requirements Management Menu from Spiratest.](image)

**TestTrack:** Developed by Seapine Software, TestTrack had its first release in 1996. It implements a client-server architecture, where the server keeps a central database while the client executes the business logic. It offers four types of clients: a cross-platform GUI, developed using the QT framework, that provides all the end-user and administrative operations; a Web UI, containing a more collaborative-oriented subset of the core functionalities; a SOAP SDK that allows for data-retrieval applications to extend built-in functionalities using independent language and platforms; and a plugin interface for many...
popular IDEs (Eclipse, VisualStudio, Excel), providing third-party applications the possibility to execute some of the platform’s functionalities.

TestTrack provides a similar toolset to many of the other analyzed platforms, including strong end-to-end traceability features on all the project’s artifacts, automatic test-case generation and the ability to import and organize requirements from text files. The most interesting functionality, however, is the possibility to reuse Requirements over multiple projects through a built-in templating system, provided they follow a set of 6 best-practices: 1) They must be well documented in a searchable repository; 2) Subsequently tuned with improved information, such as exception-handling; 3) Domain-agnostic, keeping them generic enough to be applicable to other systems; 4) Not excessively granular, which can exponentially increase the final number of requirements, making them impossible to manage; 5) Developed using a pattern that should contain helpful background information or links to other resources; and 6) Linked with other dependent requirements that need to be reused together.

TraceCloud: TraceCloud was created in 2008, and offers both a web based subscription system, and as an on-site installation option. It was developed using J2EE technologies (HTML, JavaScript, JSP, Servlets) and uses AJAX for asynchronous server communication. It also provides a JSON based REST API for external integrations.

TraceCloud’s main design focus was based on performance and scalability, achieved mainly through optimizations in the client-server communication protocols. Its features allow for cross-project requirements reusability through a requirements publishing system, full traceability and change impact analysis with automatically generated reports, and good import/export tools from and to Excel or Word formats.

Polarion REQUIREMENTS: Developed by Polarion Software since 2005, REQUIREMENTS is a fully web-based platform with a strong focus on multidisciplinary team collaboration and strong requirement traceability thought multi-directionally linked items. As most other web-based applications, it runs a rich Client with most of the dynamic content cached in the browser. It uses an Apache Server with SVN integration for data management as back-end. Since it stores most of its XML data in subversion, it makes use of the Apache Lucene indexing framework to implement an index for all its read operations, and recently introduced an SQL layer for more sophisticated querying. Polarion also provides a set of developer tools for creating connectors, web services and extensions to the main platform, such as an SDK and a Java API for web services.

Its main features include a Google Docs-esque document editor - Polarion LiveDocs - that allows for concurrent and synchronized editing of documents and artifacts; a branching system for maintaining a common trunk of requirements to reuse in multiple projects, and branches to keep the more project-specific ones; and a strong versioning system, allowing for full traceability of every artifact modification along the whole project cycle.

\[2\]http://www.seapine.com/papers/best-practices-for-requirements-reuse
<table>
<thead>
<tr>
<th>Developer:</th>
<th>Rational DOORS</th>
<th>Spiratest</th>
<th>TestTrack</th>
<th>TraceCloud</th>
<th>Polarion REQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture:</td>
<td>Desktop w/ Web extension, Web-based</td>
<td>Web-based</td>
<td>Desktop w/ Web extension</td>
<td>Desktop, Web-based</td>
<td>Web-based</td>
</tr>
<tr>
<td>Developer Support:</td>
<td>DOORS eXtension Language (DXL) API</td>
<td>SOAP &amp; REST API IDE Integration</td>
<td>SOAP SDK IDE Integration</td>
<td>REST API</td>
<td>Java SDK, Java Web API</td>
</tr>
<tr>
<td>Traceability Features:</td>
<td>Full requirements linking to internal and external artifacts</td>
<td>Full Requirements traceability though the QA and Testing steps</td>
<td>Traceability Matrix, Requirements flagging System snapshots</td>
<td>Traceability Matrix, Traceability Trees, Traceability Reports</td>
<td></td>
</tr>
<tr>
<td>Quality Assurance:</td>
<td>Test tracking through custom modules</td>
<td>Automated Test Cases &amp; Bug Tracker Tool</td>
<td>Automated Test Cases Impact analysis reports</td>
<td>Automated Test Cases Testing Framework MSOffice integration</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: RE Dedicated Platforms Comparison

2.1.2 RE Wikis

As discussed in section 3.1, Wikis provide a good baseline tool for developing projects in areas such as Requirements Engineering. Below are presented some of those projects and the functionalities they propose.

**SoftWiki:** The Softwiki project was developed between 2006 and 2009 by teams from the University of Leipzig and the University of Duisburg-Essen, on a funding program granted by the German Ministry of Education and Research[^3]. It produced several papers on the general subject of Semantic Requirements Engineering, ([Riechert et al., 2007](#) ([Lohmann et al., 2009](#)) and more specifically on Softwiki, an ontology-based wiki for distributed requirements engineering ([Lohmann et al., 2008](#)).

The SoftWiki project focuses on semantic RE collaboration aspects, in an effort to allow very large and spatially distributed groups of stakeholders to collect, semantically enrich, classify, and aggregate software requirements using semantic web technologies to interlink and exchange requirements data ([Lohmann et al., 2008](#)).

Each requirement within the SoftWiki has its own URI making it a unique instance on the semantic web and allowing it to be linked to other resources using semantic web standards such as RDF and OWL. To ensure a shared conceptual foundation and semantic interoperability, the team developed the SoftWiki Ontology for Requirements Engineering (SWORE) ([Riechert et al., 2007](#)), which defines the core concepts of requirement engineering and the way they are interrelated. The requirements can also be linked to external resources in an evolutionary process, where the requirements are successively linked to each other and to further concepts in a collaborative way. Whenever a requirement is formulated, reformulated, analyzed, or exchanged, it might be semantically enriched by the respective participant.

In regards to architecture, Softwiki has a central platform for semantic collaboration based on the OntoWiki tool ([Auer et al., 2006](#)) extended to support requirements engineering according to the SWORE ontology. The central platform can be interoperable with other tools by exporting its requirements in an RDF format according to the SWORE schema or though a SPARQL endpoint.

**SmartWiki:** SmartWiki was proposed in 2009 ([Knauss et al., 2009](#)) by a team of researchers from the Leibniz University in Germany, though never got a concrete implementation. The paper conceptualized a collaborative wiki environment aimed at refining requirement specifications by generating context-sensitive feedback and performing automated consistency checks on its content.

SmartWiki fundamental concepts are the following:

- **Heuristic Feedback**: To achieve better requirement documents, the system uses heuristics to find items for improvement. It then implements a “critique” system to provide information to the user on what should be improved. Two types of feedback were proposed: **Direct Feedback** - Based on the Heuristic Requirements Assistant HeRA tool, the system detects common problems in natural language requirements, like the use of passive voice, or the occurrence “weak words” that lead to imprecise requirements, and provide direct feedback in the form of non-intrusive JavaScript popups. **Asynchronous Feedback** - Performed by bots that collect metrics and publish the results in special wiki pages;

- **Glossary Support**: According to the authors, the glossary is an important artifact to establish a common language between domain experts, but it’s often the case that project members are unaware of all the terms already present in a given glossary. SmartWiki uses a system to highlight terms defined in a glossary after being written, and provides an additional tooltip containing it’s definition;

- **Project Management Support**: Wikis already provide a good support for distributed and collaborative work, though a set of tools such as versioning, diffs and searching mechanisms. However, the authors state that not all users make full use of those tools and propose the creation of overview pages with the most important informations in that regard so it can more easily reach the users;

- **Quality Management Support**: To guarantee a better quality management of the system, SmartWiki uses **Quality Gates** - significant milestones and decision points within a project, usually located be-
between the project’s phases. At each gate, certain project results are evaluated against predefined criteria and “gatekeepers” (usually members of the quality management team) decide if the project may proceed or not;

**ReqWiki**: ReqWiki is web-based system for software requirements engineering currently being developed by the Semantic Software Lab from Montréal, Canada. It is based on an open source wiki platform that includes NLP assistants to aid users on the development of requirements specification documents. It combines wiki technology for collaborative use and semantic knowledge representation for formal queries and reasoning with natural language processing assistants. ReqWiki was first proposed in 2012 ([Sateli et al., 2012](#)) and further described in the next year ([Sateli et al., 2013](#)) and its design philosophy is based in guaranteeing 7 main requirements:

- **R1**: Collaborative Environment;
- **R2**: Consistency;
- **R3**: Semantic Support;
- **R4**: Quality Assurance;
- **R5**: Seamless Integration;
- **R6**: Service Independence;
- **R7**: Support for Standard Methodologies.

![Figure 2.3: Overview of ReqWiki’s Architecture](http://www.semanticsoftware.info/semantic-assistants-architecture)

The first requirement (R1) is fulfilled by the intrinsic collaborative characteristics of the Wiki technology that serves as ReqWiki’s base, while the second, third, fourth and fifth (R2, R3, R4, R5) are all consequences of the Semantic Data Model and NLP web services that are built on top of it. The sixth requirement (R6) is accomplished through its service-oriented *Semantic Assistants Framework*[^2][^3], while the seventh (R7) is guaranteed through the support of common RE artifacts, such as Unified Process(UP) and Use Case Models.

ReqWiki was implemented using the MediaWiki engine and uses the Semantic MediaWiki[^4] extension to provide its semantic support capabilities. This extension generates semantic metadata for every new

[^2]: http://www.semanticsoftware.info/semantic-assistants-architecture
[^3]: https://www.semantic-mediawiki.org/wiki/Semantic_MediaWiki
[^4]: https://www.semantic-mediawiki.org/wiki/Semantic_MediaWiki
created artifact which is then used to populate ReqWiki’s underlying ontology. Once this ontology is fully populated from the user input, it can be used for a set of purposes, such as automatically generating traceability matrices to trace complex relationships.

ReqWiki’s NLP capabilities are provided by analysis pipelines deployed in the GATE Architecture\(^6\) and brokered through the Semantic Assistants Framework\(^2,3\) as web services. These services can be executed over any body of text or artifact within the page either on demand by a user or proactively based on events. After being executed, they return their results back to the client to be presented on the wiki page. An example of an NLP service available is the Writing Quality pipeline that performs grammar and spell checking on the content and provides suggestions for improvements, where possible.

### 2.2 Software Variability Modeling

Given the interest in developing a framework for modeling requirements variability, it becomes important to understand the current proposals in the field. Feature modeling is the most prominent form of variability modeling in software and is also the base for the CVL language, and as such its especially important to understand its philosophy and its evolution from the first proposal until now.

Since its first introduction in 1990, feature modeling (Kang et al., 1990) has been the most popular technique to model commonality and variability (C&V) of products of a product line. Commonalities and variabilities are modeled from the perspective of product features, “stakeholder visible characteristics of products” in a product line that are of stakeholders’ concern (Bosch et al., 2013). This happens because features are abstract concepts effectively supporting communication among diverse stakeholders of a product line, and therefore, it is natural and intuitive for people to express commonality and variability (C&V) of product lines in terms of features. Also, it has been recognized that the C&V information codified by a feature model is most critical for developing reusable software assets.

The original Feature-Oriented Domain Analysis (FODA) model, was relatively simple, with features organized using “consists of” and “generalization/specialization” relationships using an AND/OR graph. Features are typed as mandatory, alternative, or optional features to represent C&V. Attributes of a feature may also be documented. However, after the introduction of FODA, and given its success, many researchers have extended the feature model by adding new concepts for their researches, thereby resulting many variations and extensions. The most prominent extensions are presented in Table\(^2,3\).

---

\(^6\)https://gate.ac.uk/
<table>
<thead>
<tr>
<th>Variability Approach</th>
<th>Authors</th>
<th>Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORM (Feature-Oriented Reuse Method)</td>
<td>Kang et al., 1998</td>
<td>Introduces different Viewpoints</td>
</tr>
<tr>
<td>FeatuRSEB (Featured Reuse-Driven Software Engineering Business)</td>
<td>Griss et al., 1998</td>
<td>Notation Changes; Requires Notation</td>
</tr>
<tr>
<td>Generative-Programming (GP)</td>
<td>Czarnecki and Eisenecker, 2000</td>
<td>Refines the alternative relationship to XOR and OR</td>
</tr>
<tr>
<td>-</td>
<td>Van Gurp et al., 2001</td>
<td>Introduces External Features; Provides a binding time notation; Redefines generalization and specialization relationships</td>
</tr>
<tr>
<td>GP-Extended</td>
<td>Barth et al., 2002</td>
<td>Feature cardinality</td>
</tr>
<tr>
<td>Cardinality-Based Feature Model</td>
<td>Czarnecki et al., 2004</td>
<td>Introduces Feature Cardinality, Feature-Group and Group-Cardinality</td>
</tr>
<tr>
<td>PLUSS (Product Line Use Case Modeling for Systems and Software Engineering)</td>
<td>Eriksson et al., 2005</td>
<td>Notation changes; Constraint Notation</td>
</tr>
<tr>
<td>OVM (Orthogonal Variability Modeling)</td>
<td>Pohl et al., 2005</td>
<td>Internal and External variation points; Traceability between artifacts</td>
</tr>
<tr>
<td>CVL (Common Variability Language)</td>
<td>Haugen et al., 2008</td>
<td>Different kinds of variation points; Language for expressing constraints</td>
</tr>
</tbody>
</table>

Table 2.3: Summary of the Feature Modeling Approaches. Compiled from (Bosch et al., 2013) & (Bosch et al., 2015)

As it can be seen, the FODA extensions in Table 2.3 emphasize mainly notational improvements, new types of features, cardinalities and feature attributes, and extended relationships to define more accurately the constraints and relationships between features. The last one - CVL- is presented in in the next section.

2.2.1 The Common Variability Language

The Common Variability Language (CVL) is the upcoming consortium proposal for a domain independent language for specifying and resolving variability. It facilitates the specification and resolution of variability over any instance of any language defined using any MOF-based meta-model. (OMG, 2012)

Figure 2.4 shows an overview of CVL and how it is combined with other modeling languages. The Variability Model and the Resolution Models are defined in CVL, while the Base Model and Resolved Models can be defined in any MOF-defined language. This is what makes CVL a domain-independent language, which can work in many different contexts.

Below are described the concepts depicted in Figure 2.4

Base Model - The Base model upon which variability is defined using CVL. This model is not part of CVL and can be an instance of any metamodel defined via MOF.

Variability Model - The Variability Model consists in a collection of variation points, VSpecs, and constraints that used to specify variability over the base model. These concepts and the integrations between them constitute the core of the CVL language and are presented in sections 2.3.1 - 2.3.3.
Resolution Model - The Resolution Model consists in a collection of VSpec resolutions, which resolve the VSpecs of a variability model.

Materialization - Materialization is the process of transforming a base model into a product model by applying variation points. This process is driven by a resolution model which provides resolutions for the VSpecs defined against the base model.

Resolved Model - Lastly, the Resolved Model is a new model derived from the base model by materialization according to a resolution model.

2.2.2 Variation Points

Variation points are specifications of concrete variability in the base model and are part of variability models. They define specific modifications to be applied to the base model during materialization. Variation points refer to base model elements via base model handles and are bound to VSpecs. Binding a variation point to a VSpec means that the application of the variation point to the base model during materialization depends on the resolution for the VSpec. The nature of the dependency is specific to the kind of variation point.

There are four ways that variation points define modifications of the base model:

- **Existence** - an indication that the existence of a particular object, link, or value in the base model is in question;

- **Substitution** - an indication that a single object or an entire model fragment may be substituted for another. Object substitution involves two objects and means redirecting all links in which one is involved to the other and then deleting the former. Link-end substitution involves a link-end and an object and means redirecting the link-end to the object (hence substituting it for the former object
at this end). Fragment substitution involves identifying a placement fragment in the base model via boundary elements, thereby creating a conceptual “hole” to be filled by a replacement fragment of a compatible type;

- **Value assignment** - an indication that a value may be assigned to a particular slot of some base model object;

- **Opaque variation point** - an indication that a domain specific (user defined) variability is associated with the object(s), where the semantic of domain specific variability is specified explicitly using a suitable transformation language, such as QVT;

Each variation point references a VSpec, which in CVL is called binding: Binding provides the linkage between variability realization and variability abstraction. Each variation point is bound to a single VSpec.

### 2.2.3 Variability Specifications (VSpecs)

Variability abstraction provides constructs for specifying and resolving variability on an abstract level, i.e. without specifying the exact nature of the variability with respect to the base model. The central concept in this phase is the variability specification (VSpec). VSpecs are technically similar to features in feature modeling, and can be seen as abstract variability specifiers. A VSpec is an indication of variability in the base model. The specifics of the variability, i.e. what base model elements are involved and how they are affected, is not specified, which is what makes VSpecs abstract. In CVL, the effect on the base model is specified by binding variation points to VSpecs, which refer to the base model. VSpecs may be arranged as trees, where the parent-child relationship organizes the resolution space by imposing structure and logic on permissible resolutions.

There are four kinds of VSpecs:

- **Choice** - A choice is a VSpec whose resolution requires a yes/no decision. Nothing is known about the nature of the choice in the level of a VSpec tree, except of course what is suggested by its name;

- **Variable** - A variable is a VSpec whose resolution involves providing a value of a specified type. This value is meant to be used in the base model, but similar to choices, it is unknown in this level exactly where and how;

- **Variability Classifier** - a variability classifier, which can be abbreviated as VClassifier, is a kind of VSpec whose resolution means creating instances and then providing per-instance resolutions for the VSpecs in its sub-tree. Each VClassifier has an instance multiplicity which indicates how many instances of it may be created under its parent in permissible resolutions;

- **Composite VSpec** - A Composite VSpec, or CVSpec is a VSpec whose resolution requires resolving a set of VSpecs, identified through its type, which is a variability interface. A variability interface is just a collection of VSpecs that may be organized in tree structures.

### 2.2.4 Constraints

In CVL, constraints are used to express intricate relationships among VSpecs that cannot be directly captured by hierarchical relations. To achieve this, it introduces two approaches based on the Object Constraint Language ([OCL](Object Management Group, 2014)) with different levels of complexity: a
basic constraint language, consisting of a restricted subset of OCL, and a second, more complete language, including the complete OCL specification.

The Basic Language defines two main types of constraints:

- **Propositional**: The Propositional constraints expresses an implication between two or more VSpecs, and can often be described as a variation of the expression $A \implies B$. More complex expressions can be achieved by combining multiple constraints using conjunction (AND);

- **Arithmetic**: The Arithmetic constraints involve expressions to restrict values of variables, such as $\text{var} > 30$. 

![Figure 2.5: An example of a VSpecs Tree (OMG, 2012).](image)
Chapter 3

Research Context

This chapter provides an overview of the context that underlies this dissertation, namely the RSLingo research initiative, which defines a series of processes and languages towards the development of more formal requirements specifications. The work developed in this initiative over the last years has also originated a set of tools that were relevant in the design of the REBox platform. Both the RSLingo initiative and its tools are presented below in sections 3.1 through 3.3. Lastly, section 3.4 presents the main case study that has been used to develop the RSL-IL language and that was adapted for the context of the dissertation.

3.1 The RSLingo Approach

In the field of Requirements Engineering, natural language is often the preferred choice in most projects for allowing users without knowledge in formal modeling languages to easily express their needs and views of the system (Foster et al., 2004). Yet, despite being the most common form of representation used within requirements documents due to its familiarity, natural language possesses intrinsic characteristics that often lead to incorrect, inconsistent, incomplete and ambiguous requirements (Firesmith, 2003). RSLingo proposes an Information Extraction approach based on linguistic patterns that are frequently used to express RE-specific concerns to improve the quality of requirements specifications written in ad-hoc natural language (De Almeida Ferreira and Da Silva, 2012).

Although being the preferred language to write requirements specifications, the human effort required to produce this type of documents is substantial. The high amount of information and the manual nature of RE often makes writing and validating requirements in natural language a very time consuming and error-prone task.

Thus, the automation of such a task could be used for (1) domain analysis, such as identifying relevant concepts and relations; (2) verification and consistency checking and (3) producing transformations, like the automatic generation of representations of diagrams or reports.

The full processing of natural language text is still too complex to be automatically performed by computers in a broad general-purpose context. However, simplified NLP techniques can be used when the only goal is to extract relevant information from the text. To capture this information and translate it to formal domain knowledge, the RSLingo approach proposes the use of two languages. The first one is RSL-PL (“PL” standing for Pattern Language), whose purpose is to encode the RE-specific linguistic patterns found in natural language-written requirements specifications (De Almeida Ferreira and Da Silva, 2013b). The second one is RSL-IL (“IL” standing for Intermediate Language), which can be considered as a Domain Specific Language (DSL) for RE, according to the perspective of the “domain experts”, and
serves as the way to formally document requirements (De Almeida Ferreira and Da Silva, 2013a).

The definition of the linguistic patterns encoded in RSL-PL and the mapping between these patterns and the equivalent RSL-IL formal structures comprises the first stage of the RSLingo approach - **RSLingo’s Process Level**. The main asset to be produced in this phase is precisely the \( RSL-PL \rightarrow RSL-IL \) Mapping.

**Figure 3.1:** Overview of the RSLingo approach at project-level (De Almeida Ferreira and Da Silva, 2012).

The second stage - **RSLingo’s Project Level** - is illustrated in figure 3.1 and consists in applying the concepts and the languages previously defined, as well as using a set of tools during the execution of a specific software project. This stage foresees two main roles: the **Requirements Engineer** and the **Business Stakeholder**, both of which contribute with textual inputs to the initial requirements specification, written in natural language. This step should be accompanied by the use of a **Glossary** establishing a common vocabulary for key domain-problem terms. These two artifacts, along with the \( RSL-PL \rightarrow RSL-IL \) Mapping produced in the last step and a set of **General Lexical Resources** used to provide disambiguation tasks are then used by the RSLingo toolset (Ferreira and da Silva, 2009) to populate a repository of RSL-IL expressions that represent a formal subset of the original requirements specification written in natural language. These formal expressions can then be used, as proposed above, to perform automatic verifications and consistency checking, as well as transformations to “Controlled NL” requirement specifications or even Requirement-derived Models.

### 3.2 The RSL-IL Excel Template

The RSLingo approach foresees the whole process of knowledge extraction and conversion to a more rigorous representation as a way to help business stakeholders without knowledge in the area of Requirements Engineering in having a deeper insight on the problem at hand, and thus better understanding the implications of the set of stated natural language statements that represent their real requirements (De Almeida Ferreira and Da Silva, 2012).

However, to a technical user already familiar with the concepts of requirements engineering, the **RSLingo’s Process Level** phase can be omitted, as there is already a much better understanding of the RE concerns and a deeper knowledge of its documentation process. As such, the REBox system doesn’t focus on the automatic extraction part of the process, but instead on the formal representation of RE-specific concerns provided by the RSL-IL language. The language defines several constructs that
are logically arranged into viewpoints according to the specific RE concerns they address. For example, the people and organizations that can influence or will be affected by the system are represented in the Stakeholders viewpoint. Likewise, the objectives of business stakeholders regarding the value that the system will bring are expressed in the Goals viewpoint. This multi-view architecture defined in the RSL-IL language has since been adapted to an Excel format: the “RSL-IL Excel SRS Template” [1].

In the field of Requirements Engineering, a Software Requirements Specification (SRS) is a document that describes multiple technical concerns of a software system, and serves as a way to share the system view amongst its stakeholders throughout different stages of the project life-cycle. It usually follows a previously defined template (that should be customized to the needs of the organization involved) prescribing the use of multiple modular artifacts corresponding to different views. The RSL-IL Excel SRS Template is based on the multi-view architecture defined in the RSL-IL language and is currently being used by other RSLingo-related research projects to document and model system requirements and other RE concerns in a set of different views using the RSL-IL language. The template is illustrated in figure 3.2.

![Figure 3.2: Example of a Quality Requirements sheet of the RSL-IL Excel Template.](image)

Regarding the REBox platform, even though the whole template can be uploaded and used to model requirement specifications documents, only the main requirements views - Goals, Functional Requirements and Quality Requirements - are supported for reusability and variability purposes. These views and their properties can be conferred in Appendix A.

### 3.3 The RSLingo Studio Tool

The RSLingo Studio is a software tool that is currently being developed as an adaptation of the homonymous RSLingo4Privacy Studio. RSLingo4Privacy [2] is a multi-language approach that intends to improve the specification and analysis of privacy policies, supported by several processes and respective

---

tools. This approach is backed by its own requirements specification language - the \textit{RSL-IL4Privacy} (Rodrigues et al., 2016) that, despite sharing the same background and technologies as RSL-IL, was recently defined independently and with the only purpose of supporting the rigorous specification of privacy policies with multi-representations.

In the context of this dissertation and to avoid confusion, we will refer exclusively to the RSLingo Studio tool and assume it already implements all the same features as its counterpart, only using the RSL-IL language and focused on the management of requirements specifications instead of privacy policies.

The main feature that RSLingo Studio provides is a textual editor that allows creating and editing RSL-IL files. This editor was developed using the Xtext framework\textsuperscript{3}, which allows the development of textual domain specific languages (DSLs) through their definition using Xtext’s grammar. From that grammar definition it is possible to automatically generate the language infrastructure (e.g. parser and typechecker) and a fully customizable Eclipse\textsuperscript{4} plugin containing the DSL editor with helpful features like syntax highlighting, error checking, auto-completion or source-code navigation. Furthermore, the SRLingo Studio tool supports multiple representations of a requirements document. It relies on several transformations to create a mapping between these representation by using RSL-IL as a common and intermediate language. This multi-transformation approach involves Text-To-Model\textsuperscript{T2M} Model-To-Text\textsuperscript{M2T} and Model-To-Model\textsuperscript{M2M} transformations and currently supports the following document types: ad-hoc and controlled NL text, Excel, Word and JSON. The transformations and the document types associated are described below:

- **Text-To-Model**: Studio performs a T2M transformation during the import process of an ad-hoc NL text file. This transformation involves the execution of automatic text classification and extraction processes. The classification process identifies the set of statements and the second process extracts the relevant elements from the original statements into their equivalent representation in RSL-IL. These processes are implemented using RapidMiner\textsuperscript{5}, which is a popular open source platform for predictive analytics and data mining. The implementation of this transformation is a complex task that involves the integration and tuning of feature models and tools.

- **Model-To-Text**: These transformations occur when a RSL-IL file is exported to Word and controlled NL text files.

- **Model-To-Model**: M2M transformations are used both during the import and export of a SRS in RSLingo Studio. The import of an Excel file generates its corresponding RSL-IL representation, while the reverse transformation uses the internal RSL-IL representation to generate a RSL-IL Excel file based on the existing template (Silva et al., 2015). Other transformations consist in the export of a SRS specified in RSL-IL for JSON.

Regarding REBox, the M2M transformations allow for a direct integration through the import/export capabilities using the RSL-IL excel template. This integration is further explained in chapter 4.

#### 3.4 Case Study

During this research work, a case study was defined in order to test and evaluate the REBox platform, based on a running example used in the RSLingo initiative - the \textit{Billing System}. Throughout this dissertation, these case study is used to exemplify various aspects of its design and features, contributing to

\textsuperscript{3}https://eclipse.org/Xtext/index.html
\textsuperscript{4}https://eclipse.org
\textsuperscript{5}http://rapidminer.com
a better understanding and simplicity of explanation. A brief description of the case study is presented
below.

The Billing System

This system was originally defined for the purpose of showing how to specify a simple business in-
formation system based on the RSL-IL language. It provides a complete set of features for managing
customers, products and invoices and can be configured to support the work of different business man-
agers (operators, product operators, managers, and sys-admins) in order to allow them collaboratively
manage and participate in the processes related the creation, approval, issue of invoices, control their
respective payments, and produce several reports. It also implements a simple workflow associated to
the invoice's lifecycle, namely considering the following states: created, waiting-for-approval, approved,
issued, paid, and deleted.
Chapter 4

The REBox Platform

This chapter presents an in-depth view of the main proposal of this dissertation: the REBox collaborative platform for managing software requirements specifications. Section 4.1 presents REBox’s main goals based on this thesis statement. Section 4.2 starts by giving an overview of REBox’s components and their interactions and then describes the platform main features in bigger depth. Finally, section 4.3 extensively details the system’s requirements variability modeling framework.

4.1 Main Goals

The REBox system’s main conceptual goals can be summarized as follows:

Create a collaborative environment for managing requirements specification documents: The platform major purpose is to provide a collaborative editor for software requirements documents. Simple spreadsheet editors, such as Excel, have always been a common choice to manage requirements specifications due to their widespread availability and simple interface, despite lacking most management features offered by dedicated RE tools. Their biggest problem until recently, however, was the high effort needed to maintain synchronized and updated versions of the documents in decentralized projects, often requiring a significant effort to maintain track of the latest changes. The recent rise of cloud storage services, however, such as Google Drive or OneDrive, have helped on that regard by offering collaborative mediums to manage these type of documents with automatic synchronization and lightweight versioning tools. As such, the REBox platform makes use of various Google Drive API services to provide both the SRS editor and a collaborative environment.

Integrate with RSLingo concepts and technologies: As stated before, this includes in particular the RSL-IL language and its current Excel representation. Although the platform allows for the usage of any format of SRS document, all the features that involve data manipulation expect the current format of the RSL-IL template. Besides the RSL-IL language, REBox also aims to integrate its features with other tools currently being developed in the scope of the RSLingo field of investigation.

Provide a set of Reusability and Variability mechanisms: To complement the document editor, the system makes use of the data manipulating capabilities provided by the integration with the Google APIs to offer a set of features to increase the efficiency of its processes. The main feature is the variability modeling approach based on CVL that will be described in depth in chapter 4.3. The other reusability-focused features are Templating System for requirements specification documents and a Requirements
Library system for requirements reuse. The former allows for any previously developed template in a spreadsheet format to be uploaded to the platform and later used to create new projects based on its structure, while the latter allows the creation of libraries of coarse-grained and potentially generic requirements that can afterwards be edited if necessary and added to any new project.

4.2 Overview & Features

Given the main goals that defined the conception of the REBox system and the lessons learned from analyzing the related work in regards to collaborative platforms, the REBox RSDoc editor was designed in line with the following ideas: 1. provide an environment that combines some of the best features of each of the current types of collaborative tools (presented in chapter 2.1) and 2. provide the basis to build strong reusability features, such as the newly developed requirements Variability Framework. To achieve this, the technology chosen was the Google Drive system and its Google Sheets editor. The reasoning for this choice is presented below.

The collaborative tools described in the related work section can be broken into three types: Spreadsheet Editors, Dedicated RE platforms and RE Wikis. REBox, through Google Sheets, wholly incorporates the first type - the spreadsheet editor - and all its benefits, namely its familiarity to most users and its focus on usability. Regarding the second type - RE Dedicated Tools - their main strength is often related to strong management and productivity features. Although not as extensively, REBox also strives to tackle this, through its Templating System, Requirement Library and even though the Google Sheets, who besides its editing capabilities, also provides lightweight versioning and rollback mechanisms. Finally, regarding the third type - RE Wikis - they propose innovative ideas to analyze and validate the quality of requirements based their syntax and semantics. Again, REBox is often not as ambitious as some proposals, but guarantees textual validation though its integration with the RSLingo Studio system.

Figure 4.1 provides an overview of the REBox platform, its internal features and the way it integrates with the Google Drive Service and the RSLingo Studio tool. The next sections describe each of those components in detail.

4.2.1 Template Management System

The template management system, like its name implies, offers the possibility to create and manage reusable requirements templates that can be later utilized to bootstrap new projects. Like the RSDoc editor, this system also makes full use of the APIs provided by Google. The process of creating a template in the platform encompasses two steps. The first one consists on the manual upload of a spreadsheet file containing the template into the platform. To note that this document doesn’t need to hold the finalized template (it can even be empty), as REBox allows it to be further edited once its uploaded. Once the file is uploaded, it is saved into the local filesystem and the second step begins. This time, the file is automatically re-uploaded, this time to the Drive cloud storage server and then shared with the users of the platform. From this point on, it can be accessed and edited concurrently like any other document, and it can be used as the base structure for new project created in REBox.
4.2.2 RS Library

REBox’s Requirements Specifications Library offers the possibility to create modules of reusable requirements that can then be added to any RSProject. Requirements reusability across multiple projects is already one of the main ideas being explored by some of the platforms presented in the Related Work in regards to increasing the efficiency of RE activities. To that end, REBox RS Library serves as an open “database” of modular, agnostic and coarse-grained requirements, that can be grouped by areas of interest, such as Security or Multi-Language, etc.. These groups of requirements can be created manually, extracted from an existing project or uploaded directly into the platform. Furthermore, after created, the user can still add and remove requirements to the module, as well as edit the existing ones, allowing for an iterative process of refinement. The process of adding a Requirements Module to a project is very simple, since the new requirements only need to be “appended” to the existing ones.

4.2.3 RSLingo Studio Integration

As described in chapter 3.3, the RSLingo Studio tool is another system currently being developed in the scope of the RSLingo initiative. Like REBox, it also provides an editor for creating and editing RSL-IL requirement specification documents. However, while REBox focuses on a more user-friendly spreadsheet representation, Studio focuses on the textual representation of the language. By defining a DSL containing the language’s grammar, it used the xText framework to generate a parser and typechecker for RSL-IL. This tools, allied to features like syntax highlighting, error checking, auto-completion and source-code navigation, provide very powerful means for automatically analyzing and validating the correctness of a RSL-IL document. Therefore, the integration with the RSLingo Studio tool would provide a very big improvement in terms of requirements quality validation. From the start of the project that the integrating with other RSLingo technologies was defined as one of the main focuses, and indeed, the choice to use the Google Drive technology allows the possibility not only to upload, but also to download the spreadsheet files. Those files can then be manually uploaded to the RSLingo Studio tool and
converted into the textual RSL-IL representation though its M2M transformations. The user can then receive visual feedback about the correctness of the document. This solution, however functional, could be improved in terms of automation. Time constraints and the fact that the RSLingo Studio itself is also not yet fully implemented led to the postpone of that feature, but the conceptual idea and technologies are described in section 7.2.

### 4.3 Requirements Variability Modeling Approach

The REBox process of variability modeling closely follows the proposed CVL approach, as it can be seen in Figure 4.2. A detailed description of the process is presented below.

#### 4.3.1 Variation Points Configuration

At the start of the variability modeling process, the user selects the SRS document to be used as the Base Model. In REBox, the Base model is an instance of a requirements specification document created using the RSL-IL language. This document can either be part of a project already present in the platform, or can be manually uploaded into the system from an outside source, provided it conforms to the RSL-IL language. The system then automatically parses the document and extracts the requirements from the RSL-IL supported views, namely the **Goals**, **Functional Requirements** and **Quality Requirements**. It is then, regarding these extracted requirements, that the user will define the variation points. As defined by CVL, variation points are specifications of concrete variability in the base model, defining specific modifications to be applied to it during the materialization process. Variation points can take various types, which will define the nature of the dependency between a VSpec and the corresponding Variability Point in the materialization phase.
In REBox, variation points can be defined on two levels: (1) the requirements themselves and (2) their properties. Variation points defined over a requirement are of the *Existence* type, meaning that in the materialization phase, they will be bound to a *Choice* VSpec. On the other hand, variation points defined over a requirement property are of the *Value Assignment* type, and will be bound to a *Variable* VSpec.

### 4.3.2 VSpecs Binding

After the user defines all the variation points over the base model, the REBox system automatically generates the first part of the variability model: the *var.specs* view. In CVL each variation point references a single VSpec, in a process defined as *Binding*, which provides the linkage between variability realization and variability abstraction. Figure 4.3 details the variability binding model used by the REBox approach. The *var.specs* view contains all the information about the newly generated VSpecs and its full list of attributes are presented in Table 4.1. To note that this view introduces a new concept associated to VSpecs: the *VarToken*. VarTokens are nothing more than “children” VSpecs that are associated with a particular attribute of the requirement that server as its variation point. A VSpec can have any number of VarTokens, depending on the number of attributes that were considered as variation points in the last step. Regarding the type, “parent” VSpecs that are associated with requirements are defined as *Choices*, meaning that in the materialization phase they will require a binary yes/no decision to define if that particular requirement is going to exist in the resolved model. Contrarily, the VarTokens are of type *Variable*, determining that in the materialization phase these VSpecs will require providing a resolution value of their specified type.

Furthermore, the user is then able to define constraints over the VSpecs that were created. In CVL, *constraints* are used to express intricate relationships among VSpecs that cannot be directly captured by hierarchical relations. To achieve this, it relies on a simplified version of the OCL [Object Management Group, 2014] language. The user can utilize both the parent VSpecs and their children VarTokens when building the logical expressions that serve to express these constraints. A more in-depth example of this process is shown in chapter 4.4.

After all the constraints are created, the system then generates the *var.specs.constraints* view, shown in Table 4.2. With both views created, the Variability Model is finalized and saved.
4.3.3 The Materialization Process

The final step in the variability modeling approach is the Materialization process. As stated in the CVL proposal, this process consists of transforming a base model into a new document by applying variation points. Materialization is driven by a resolution model which provides resolutions for the Vspects defined against the base model. In REBox, after the Variability Model is created in the previous steps, its information is stored in the system’s database. From that moment on, whenever a new requirements specification document creation process starts, the user is given the option to, instead of creating a new empty document, generate a new one by defining a set of resolution values for an existing variability model. When this happens, the platform crosses the information contained in the Variability Model with the resolution values provided by the user and checks every constraint for possible inconsistencies. If nothing is found, a new Resolved Model is generated.

4.3.4 Example - The Billing System Reliability Features

Figure 4.4 provides a concrete example of Variability Model generation process described above. This example describes the billing system set of reliability-focused quality requirements, which specify its expected uptime over an year, required maintenance procedures and fault logging details.

The first table of the image shows an exact representation of a RSL-IL Excel template Quality Requirements view. The view defines a set of attributes to characterize the requirement, such as an unique Id, a name, a type and sub-type, a quantitative metric and its value (if applicable), the stakeholder who proposed the requirement, its parent requirement(s), a description and finally its priority in the context of the current project.

The second table presents an example of the “Variation Points Config” step defined in Figure 4.2. As described before, this step starts with the automatic extraction of the Goals, Functional Requirements
and Quality Requirements from the base model defined in the RSL-IL language and consists in the manual configuration of the model's Variation Points. To define a requirement as a variation point, the user just needs to check the “VarPoint?” field on that requirement's row. This action will mark that requirement as an Existence variation point, meaning that it be linked to to a Choice VSpec in the Binding phase. Furthermore, field “VarTokens” lets the user select which of that requirement’s attributes will also be modeled as children variation points of the type Value Assignment. In the example, from the five initial requirements, two were selected as variation points: QR,R,1 and QR,R,3. The first one has two attributes modeled as children variation points: “Description” and “Priority”. However, not all of the content of the “Description” field is to be considered for variability purposes, but only the substring “99%”, denoted by the suffix .value#99%.

The third table shows a simplified version of the .var.specs view (for the sake of space, some attributes were omitted; the full list is displayed in Table 4.1). This view is automatically generated by the REBox system after the user selects the variation points in the previous step. This process represents the Binding phase of CVL, where each of the those points are linked to their abstract representation in the variability model. To the right is an illustrative example of the corresponding VSpecs tree representation, which shows the six VSpecs created based on the input provided in the second table. The parent VSpec corresponds to the system itself. On the second row, the QR,R,1 and QR,R,3 requirements are represented as Choice VSpecs, meaning that in the Materialization phase, both resolutions will require a binary yes/no decision. These elements are linked to their parent by a dashed edge, representing a False “IsImpliedByParent” value, making their materialization value independent of the one their parent takes. Finally, on the last row are shown thee children VSpecs of type Variable, generated by the VarTokens defined in the second table and their corresponding types.

The fourth table presents an equally simplified version of the .var.specs.constraints view (again, the full list of fields can be viewed in Table 4.2). Like the name implies, this table defines the Constraints associated with this particular variability model. Unlike the previous table however, the constraints are...
not automatically generated by the system (nor could they). It is up to the user to, based on the VSpecs defined in the `.var.specs` table and the context of the system, build the needed logical expressions to capture intricate relationships among VSpecs that cannot be simply expressed by the hierarchical relations in a VSpec tree. The example defines a *Propositional* constraint determining that if in the materialization phase, the annual uptime of the system is set as greater than or equal to 99.9%, then at the same time, the value defined for the *Priority* VSpec has to be “Must”. To note that in this example both the VSpecs that composed the constraint were children of the same VSpec (`QR_R_1`, with Id `VS_1`, as identified in the example by the prefix “VS_1:`”). As such, the constraint is defined in the context of the parent node, denoted by the solid edge. It is possible, however, to combine multiple parent VSpecs and their children (e.g. `VS_1 → VS_2:Description = ‘‘Something’’`, meaning that if the `QR_R_1` is resolved as true, then the `Description` of `QR_R_3` must be resolved to the string “Something”). In this case, the context of the constraint will be the whole model and therefore it should not be connected to any specific VSpec.
Chapter 5

The REBox Web Platform Implementation

This chapter provides an overview of the REBox’s web platform implementation. Section 5.1 gives an
description of the system’s domain model. The main technologies used in the development of the web
platform are described in section 5.2. Lastly, section 5.3 presents an example of a typical use case
scenario of the platform, from an implementation perspective.

5.1 Overview

Figure 5.1 shows the REBox platform domain model. Below is a brief description of its components.

![Class Diagram]( Diagram.png)

Figure 5.1: The REBox Web Platform Domain Model.
RSProject - The main class of the system, each new RE project developed in REBox has one corresponding Project object. The Project contains several customizable metadata and is linked with one User that acts as his owner. Each Project must also have one associated RS Document, and any number of RS Library Modules.

User - The most fundamental component of the system alongside the Project. Each user object contains the information of user of the platform. At creation-time, each user object is associated with an individual set of access permissions that will determine what parts of the system are accessible to it.

Permission - Permissions are related with CMS Roles, granting customizable access restrictions within the platform. They fall into two main categories: view and edit, and can be customized in a page by page basis.

RS_doc - The RS Doc class contains the information about a software requirements document that is represented in the context of REBox by a Google Sheets spreadsheet. Each RS Doc is associated with a project, and contains a set of requirements. Furthermore, it can also any any number of Variability Models associated with it.

RS Library - The RS Library class aggregates the Requirements Modules represented in the RS Module class.

RS Module - The RS Module contains a set of modular and coarse-grained requirements, usually organized by type, and can be associated with a RSDocument by appending its requirements to the ones contained in the document.

RS Doc Templates - Each RS Doc Template object will represent the Excel document with pre-filled fields that can then be imported by a newly created RS Doc, saving the time of manually retyping the information and thus increasing the efficiency of the document setup process.

Requirement - The Requirement class, like the name implies, represents a requirement within the system. It has 3 types: ReqGoal, ReqFunctional and ReqQuality, each containing the corresponding RSL-IL view’s attributes. The class is directly associated with the RS Module and with the RSDoc.

Variability Model - The Variability Model Class represents a Variability Model document created in the platform and aggregates a series of Variation Points Constraints objects. A Variability Model is always liked to one RS Document, which represents its Base Model in the CVL notation.

5.2 Technologies

Below are presented the main technologies used to build REBox. Smaller technologies, plugins and code libraries are introduced alongside the working example in section 5.3.

The technology chosen to implement REBox was the DotNetNuke web CMS framework, using C# and Javascript. The decision was based in two factors: The first was to take advantage of the current in-house knowledge in that technology, stemming from various projects currently being developed
under the supervision of prof. Alberto Silva at INESC-ID. The second reason was based on the modular nature of the framework, that allows for transversal modules, such as user management, to be easily reused from those other projects, provided minimal adaptations. Furthermore, the DNN framework already provides basic set of out-of-the-box functionalities, such as security and content management.

REBox was developed as a DNN extension, following a MVC (Model-View-Controller) paradigm. In DNN, third-party extensions can very easily be added into the system through an installation process. Also, the REBox extension contains all the necessary data to be installed in any DNN-running server, making deployment of the platform extremely easy.

Figure 5.2 provides a view of the DNN architecture. Third-party extensions, such as REBox, are installed over the DNN Web Application Framework.

Another advantage of the DNN framework is the fact that it offers a set of free Telerik .NET Controls, which provide a wide range of functionalities. These controls were used, for example, to create the Grids in all the menus and the File Explorer in the Templates page.

5.3 Working Example

This section will loosely follow the major steps defined in the User Session Guide (Appendix B) to provide a view of the platform’s implementation. To note that this section is not intended to provide a detailed explanation of the functionalities (which was done in chapter 4), but rather give a view of the design and implementation choices made when encoding the platform.

Template Upload

1 http://www.dnnsoftware.com/platform/start/architecture
The process starts with the manual upload of a spreadsheet (Excel) file into the platform. The page was built using the Telerik ASP.NET File Explorer control. This controller’s baseline functionality is directed at receiving files and saving them in a local filesystem. The user can then visually navigate the filesystem like in Windows Explorer and double-click any file to download it. Figure 5.3 shows an example of the Templates menu.

![REBox Templates Page](image)

After the file is uploaded to the local filesystem, it has to be re-uploaded to Google Drive and shared with the other users, so it can be edited. To do that, the platform calls the Drive API, specifically the `InsertMediaUpload` request, which receives the file to be uploaded and its `mime type`. When file has finished uploading to Drive, a unique key is generated and returned to REBox, the `DriveFileId`. This identifier is then saved in the local database as a pointer to the remote version. From this moment, whenever a user wants to edit the template or use it to create a new document, the `DriveFileId` is used to again call the Drive service and retrieve the file’s URL.

**Project Creation and Management**

The project creation process is relatively straightforward. In the creation menu, the only noteworthy option that the user has to select is the “Project Type”, which has the options “Simple Project” and “Variability Derived Project”. The difference here is the following: Simple Projects are created based on a Template uploaded to the Template Management System described above, while Variability Derived projects are created based on a previously defined Variability Model plus a set of resolution values. This step will only focus on the “Simple Project”; Variability Models are explained further below.

When the project is created what happens is simple: first the system makes a `CopyRequest` to Drive. This request will produce a copy of the file of the template selected to be the base of the project. Afterwards, the Drive service generates an `DriveFileId` (just like when creating a new file) and returns

---

3[http://demos.telerik.com/aspnet-ajax/fileexplorer/examples/overview/defaultcs.aspx](http://demos.telerik.com/aspnet-ajax/fileexplorer/examples/overview/defaultcs.aspx)
it to REBox. The second step consists of creating a database entry for the newly created project and associate the DriveFileId to it, so it maintains a pointer to its Drive URL.

The Editing Page is also very simple. As seen in Figure 5.4, it consists in the Google Sheets Editor embedded into a REBox page. This is accomplished through a simple iframe element linking to the URL of the Drive file containing the project, retrieved through its DriveFileId.

![REBox project edit screen.](image)

**Figure 5.4: REBox project edit screen.**

### RSLib Module Creation

The creation and management of the a module of reusable requirements is a little more complex. To start, the user must select the type of the Module from the three currently supported types (Goals, Functional Requirements and Quality Requirements). Next, he must select if he wants to import the requirements from an already existing file or not. If he choses to import, he must then select from which file. The process of extracting the requirements data from the file is described below:

Using the Google Sheets API, the system makes a `ValuesResource.GetRequest` request to Drive using the Id of the file to import and the Id of the sheet containing the requirements. The service then returns the requested values to REBox for processing; The retrieved values come in an unstructured format, which is a List of Lists of Objects (or a matrix of objects). Each object in the matrix corresponds to a value of a cell of the spreadsheet. To make sense of this values, the structure of the RSL-IL file must be known a priori. To give an example, the system knows that a Quality Requirements sheet has the following columns (by order): Id, Name, Type, SubType, Metric, Value, Source, PartOf, Description, Priority and Progress and that the first row where values start to appear is the row 11 (the first 10 are headers or just blank rows). This information is maintained in the `RSLILTemplateVariables` file described in section 5.2. And so, with this precise knowledge, the system then can iterate the response, row by row, and populate a list of objects of the type `SheetsQualityReqRow`. The system defines a class for each of the three requirements types rows: `SheetsGoalsReqRow`, `SheetsFunctionalReqRow` and
SheetsQualityReqRow. Each of these classes contain exactly the attributes of that type of requirements defined by the RSL-IL language and that are described in Appendix A. And so, this process begins with a matrix of unclassified values and ends with a structured list of objects of the correct type of requirement, containing all the extracted values;

With the list of objects containing the information of the rows of the Quality Requirements sheet, the platform then uses the Json.NET⁴ to serialize the list into a Json string representation, like shown in Figure 5.5:

```csharp
QualityRequirementsRow row = new QualityRequirementsRow();
row.Id = "QR_C_1";
row.Name = "Multi-Language Support";
row.Type = "Cultural";
...
string json = JsonConvert.SerializeObject(QualityRequirementsRow);
{
  "Id": "QR_C_1",
  "Name": "Multi-Language Support",
  "Type": "Cultural",
}
```

Figure 5.5: Json.NET object serialization example.

This process of serialization is done so the data can be sent to the client and displayed to the user through the JQuery plugin JsGrid⁵. This plugin offers an editable client-side grid that was chosen for providing a simple way for the users to work with data extracted from the RSL-IL documents throughout the platform. An example of the plugin is shown in Picture 5.6.

Figure 5.6: REBox RSLib Module Editor screen.

When the user finishes editing the requirements and saves the RSLib Module, the Json containing

⁴http://www.newtonsoft.com/json
⁵http://js-grid.com/
the serialized objects is saved to the Database for future use. Again, this serialized version of the extracted data is useful as it can be saved in the form of a single string, and de-serialized anytime the system has to use it.

This concludes the creation phase. The process of adding those requirements to a project is described below:

First, the user selects the project to whom he wishes to add the requirements and then the module to add. Before inserting, the user can edit them again if needed; The requirements insertion is done in an “append” fashion, meaning that the values are added to the end of the document. For this, REBox makes a new request to the Sheets Service from the type `ValuesResource.UpdateRequest`, which appends the values to the end of the selected sheet.

### Variability Model Creation

The process of creating a Variability Model is very similar to the one described for the RSLib Module in regards to the code execution and the technologies used. The process starts when the user selects a project from which he wants to create a variability model. At this point, the system makes the same Google Sheets API call to extract the requirements contained within the `reqs.goals`, `reqs.functional` and `reqs.quality` sheets of the document. It then resorts to the same process of serialization to a Json format, and uses the JsGrid plugin to show them to the user. The result of this process is exemplified in Figure 5.7 which shows the Variability Points selection page, containing the three types of requirements (the Goals and Quality Requirements panels are minimized).

![REBox Variability points selection screen.](image)

*Figure 5.7: REBox Variability points selection screen.*

After the Variability Points are selected, the next step is to define the constraints. The only noteworthy
aspect of this step, from an implementation perspective, is the fact that REBox does not allow the logical expression to contain the implication sign ( \( \Rightarrow \) ). This happens because the parser used to evaluate the expression, the Fast Lightweight Expression Evaluator (FLEE) does not support that notation. As such, and because the logical expressions used to create constraints require defining implications, a workaround was created using the mathematical equivalence: \( A \Rightarrow B \iff \neg A \lor B \). Consequently, a typical constraint expression in REBox will consist of \( \neg A \lor B \), meaning: if A is true, then B must also be true.

**Resolved Model Creation**

The resolved model creation happens when the user chooses to create a new project based on a Variability Model previously created. The user first chooses a series of resolution values for each variation point (the process here is very similar to the one described in the last step), and then submits those values to the platform. The process of generating the resolved model then have to combine 3 pieces: the Base Model, the Variability Model and the Resolution Values. Furthermore, the constraints defined in the last step also have to be verified against all the resolution values to test for consistency. All this moving parts made the implementation of this step relatively complex. The main steps are described below:

Starting after the user selects the resolution values and submits them to create a new resolved model, the first step that REBox has to perform is comprised by the task of indexing the VarSpecs in a dictionary so that their information can be accessed easily given an unique key (in this case, their ID). After this, the constraints are evaluated one by one. This is done using the FLEE parser, which can parse strings containing logical expressions and return a boolean containing the value of the evaluation. Since the string containing the logical expression only contains the reference to the value to be evaluated (e.g. \( VP.1:Description.TextValue#Should \)), the system must use reflection to find the values associated with that reference.

If all constraints are evaluated as positive, the system then extracts all the requirements from the Base Model, so that it can also index them. This is done because, again, there is a need to efficiently search through all of them to make the necessary changes (that are encoded in the VarSpecs + Resolution Values). After this, the system iterates the VarSpecs, makes the necessary changes to the requirements in the base model, and saves those changes in a newly created project, the Resolved Model Project.

---

6https://flee.codeplex.com/
Chapter 6

Evaluation

This chapter presents and discusses the evaluation performed to the REBox platform. Section 6.1 presents the results of a test session of REBox by a set of pilot users. This assessment focused on three aspects of the system: (1) the overall Usability and the quality of its Features, (2) its Requirements Variability Framework and (3) the General Approach. Section 6.2 compares REBox’s variability framework with other related works and section 6.3 summarizes the chapter describing the main conclusions obtained from these two evaluation approaches.

6.1 User Session Assessment

To better evaluate the REBox platform, receive feedback from people not directly involved in this research work and detect potential bugs and user limitations, we decided to conduct a pilot user session. This session involved a group of 7 participants in total with ages ranging from 22 to 28 years and with at least a Bachelor of Science degree. All participants had previous knowledge and experience within the field of RE and half had professional experience.

The user session was conducted under the following conditions:

- Session took place in the laboratory (controlled environment);
- The assigned tasks were performed without previous use and learning (for the first time);
- The user must had a computer with a web browser and Internet access;
- Direct Observation, i.e., while users performed the assigned tasks, their behavior and performance could be logged;
- Users were free to think out loud and share ideas if they wanted.

Based on these conditions, participants received a 20 minutes training explaining the fundamental concepts of the REBox platform and its features, particularly its variability modeling framework. Following that, they were given a script (see Appendix B) describing the Billing System presented in Section 3.3 and its corresponding RSL-IL Excel template. The work consisted in uploading the template to the web platform and use it to test all the system’s features with a time limit of 40 minutes. In the end, participants were asked to fill in a questionnaire to rate the REBox platform, its variability framework and the overall approach. The analysis of the results gathered from these questionnaires is described below.
Questionnaire Analysis

The questionnaire used in the user session focused on analyzing the quality of three aspects about REBox: (1) the overall Usability and the quality of its Features, (2) its Requirements Variability Framework and (3) the General Approach. The answers were classified in a scale of: 0 (N/A – Do not know), 1 (Very Low), 2 (Low), 3 (Medium), 4 (High) and 5 (Very High).

The REBox Web platform usability and features aspect included four questions:

QP.1. How do you rate the overall usability of the Web platform?

QP.2. How do you rate the usefulness of the main RE productivity features? (RSDoc Spreadsheet Editor, Template Manager, RSLibrary)?

QP.3. How easy to learn (or how familiar) was the main document editing tool (Google Sheets)?

QP.4. How suitable is the platform for a collaborative management of Requirements Specifications?

Table 6.1 summarizes the average score for the answers regarding the REBox Web Platform aspect, broken down by question.

In general, we can observe that all questions had very positive scores, implying that the platform was successful in accomplishing its goals. The lowest score was given to question QP.1, showing that the usability of the platform was the least positive aspect. This is somewhat understandable and the reasons are related to time constraints in the development of the project. This often led to the choice of sometimes foregoing the usability aspect and focusing on the functionality. This topic is further discussed in the chapter 7.2 (Future Work). Regarding the other three questions, the responses were very positive, showing that the users considered the platform’s features to be useful and easy to learn. Furthermore, the platform itself was considered very suitable for the collaborative management of requirements specifications.

<table>
<thead>
<tr>
<th>REBox Web Platform</th>
<th>QP.1</th>
<th>QP.2</th>
<th>QP.3</th>
<th>QP.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>4</td>
<td>4.29</td>
<td>4.29</td>
<td>4.71</td>
</tr>
</tbody>
</table>

Table 6.1: Survey average score (in a scale of 0-5) by question for the REBox Web Platform aspect.

The REBox Requirements Variability Modeling Framework aspect included five questions:

QV.1. How easy to understand was the overall variability modeling process?

QV.2. How easy to understand were the concepts(VarPoints, VSpecs, VarTokens, Constraints, etc.)?

QV.3. How do you rate the usefulness of the framework?

QV.4. How do you rate the simplicity of the Variability Model creation phase?

QV.5. How do you rate the simplicity of the Resolved Model creation phase?

Table 6.2 summarizes the average score for the answers regarding the Variability Framework aspect, broken down by question. Similarly to the previous aspect, we can observe that all answers had a generally positive score. However, the results were slightly worse that the ones received by the platform, especially regarding questions QV.1 and QV.2. This means that the users found both the Variability
Modeling process and its concepts sometimes hard to understand, which is relatively understandable, since they are specific to the proposal and not familiar to users, even those with knowledge in RE. The conclusion here is that the platform must do a better job of conveying the process and its terms, perhaps through informative tooltips or even a detailed user guide section. The rest of the questions obtained better results, showing that users consider the framework useful and relatively simple to use.

<table>
<thead>
<tr>
<th>REBox Variability Framework</th>
<th>QV.1</th>
<th>QV.2</th>
<th>QV.3</th>
<th>QV.4</th>
<th>QV.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>3.48</td>
<td>3.43</td>
<td>4.57</td>
<td>4</td>
<td>4.14</td>
</tr>
</tbody>
</table>

Table 6.2: Survey average score (in a scale of 0-5) by question for the REBox Variability Framework aspect.

Finally, the REBox General Approach aspect included two questions:

**QA.1.** How do you rate the productivity with REBox comparing to the traditional Requirements Specification process?

**QA.2.** How likely would you use this platform on your own Requirements Engineering projects?

Table 6.3 summarizes the average score for the answers regarding the REBox general approach aspect, broken down by question.

We can observe that the score obtained for both questions is highly positive, showing that users considered REBox to have a good productivity levels and expressed interest in using the platform for their own RE projects.

<table>
<thead>
<tr>
<th>REBox General Approach</th>
<th>QA.1</th>
<th>QA.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>4.43</td>
<td>4.43</td>
</tr>
</tbody>
</table>

Table 6.3: Survey average score (in a scale of 0-5) by question for the REBox General Approach aspect.

As can be seen in Table 6.4, the results were generally encouraging with positive scores in all three analyzed aspects. Nevertheless, it was observed that the Variability Modeling Framework and namely the process and its concepts had the lowest score and possibly need to be refined in order to improve their simplicity or at least be better explained to the user.

Regarding the relatively low number of participants, it can be stated that the sample was too small to extract any meaningful conclusion from the results. However, studies have noted that a group of 5 testers is enough to uncover over 80% of the usability problems [Nielsen and Landauer, 1993] and so the user session represented a representative example of what could be expected from a bigger number of participants. Also, since the survey focused on the usability of the platform, framework and approach, we believe 7 participants is a reasonable number for an exploratory assessment, at least in order to identify challenges on the usability of these aspects. The complete results of the questionnaires can be consulted in Appendix C.

<table>
<thead>
<tr>
<th>REBox</th>
<th>Platform</th>
<th>Variability Model</th>
<th>General Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>4.32</td>
<td>3.92</td>
<td>4.43</td>
</tr>
</tbody>
</table>

Table 6.4: Survey average score (in a scale of 0-5) for each of the questionnaire aspects.
6.2 Related Work

This section provides a comparative analysis of REBox’s Variability Modeling Approach to other related proposals. The Web platform component was not focused here since there are no academic proposals for collaborative platforms for RE. However, the comparison between REBox and other similar platforms is presented in chapter 3.

Regarding requirements variability, some research has already been conducted in the context of SPLs, leading to proposal of models (Moros et al., 2008) and languages (Alférrez et al., 2010) to address its challenges. Furthermore, with the advent of CVL, proposals have been made to leverage its concepts to software product lines. However, and given that CVL is a relatively recent language, not much research has yet been conducted to defined clear ways of applying its concepts and models to the specific constructs of the RE field.

Regarding requirements variability in the context of SPLs, the work presented in (Moros et al., 2008) proposes to integrate that variability directly into its requirements metamodel, REMM (Requirements Engineering MetaModel). Like RSL-IL, REMM encompasses a comprehensive set of RE concepts and relationships between them. Furthermore, REMM allows requirements engineers to define optional requirements by means of OR parent-child traces and supports the definition of parameterized requirements, both being variability mechanisms that are often modeled using separate feature models. With this approach, they are provided by a unique metamodel, making it possible to keep all the information about the requirements and their variability points connected, and to make explicit the relationships between the requirements parameters (variability points) by means of the explicit relationships existing between requirements.

Another proposal suggests the use of a compositional strategy. VML4RE (Variability Modeling Language for Requirements) (Alférrez et al., 2010) presents a multi-view composition language for SPL requirements with two main contributions. First, the VML4RE language - a Domain Specific Language (DSL) capable of composing requirements with feature models; and second, a requirements process that uses: 1) a feature model to perform the variability identification; 2) use cases and activity diagrams to describe the SPL domain requirements; and 3) a VML4RE model to relate the feature model with the requirement models.

Additionally, projects to leverage CVL concepts for variability management in Software Process Lines have also been proposed. The work presented in (Rouille et al., 2012) uses CVL to define an SPL and automatically derive a process according to that project’s requirements. It defines its base model as a base process model and then build a variability abstraction model (VAM) that specifies the projects requirements variability. The main benefits of this approach is the ability to capture the variability requirement in the VAM and its materialization in the base model using the VRM, which defines a binding between the projects requirements variability and the processes variability.

A more recent proposal introduces a multimodel approach for specifying requirements variability on SPLs (Blanes et al., 2014). This proposal is based on the Feature-Driven Requirements Engineering approach (FeDRE) (De Oliveira et al., 2013), which provides support to the requirements specification of SPLs. It proposes an extension of FeDRE to enable the specification of requirements variability in multiple models at the domain engineering stage by applying model-driven engineering principles. Furthermore, the proposal defines an integration with CVL in order to obtain the product requirements from the domain requirements by applying model transformations at the application engineering stage.
6.3 Summary

This chapter presented the results of the evaluation performed to the REBox platform.

The first part of the evaluation focused on the assessment of the platform by test users that were not previously involved in the research work. This assessment was done by conducting a session where the participants had to make full use of the platform’s functionalities. In the end of the session, the participants filled in a questionnaire focused on three aspects: (1) the Platform, (2) the Variability Modeling Framework (3) the General Approach. The results obtained from the questionnaires were generally encouraging with positive scores in all three analyzed aspects. The least positive aspects were regarding the variability modeling process and its concepts, which were not considered very intuitive and need to be better explained to the users.

Lastly, the variability framework was compared with related proposals, with different kinds of approaches and addressed concerns. Yet, whether using the newly proposed CVL language or not, most of them still focus greatly on the mapping between requirements and the feature modules of the product. The REBox variability approach main focus, however, is to correctly model the variability of the RE concepts, regardless of the software features they define. It achieves this by using the RSL-IL rigorous language to encode its requirements specifications, which encompasses a comprehensive set of RE concepts and relationships between them. Furthermore, it makes full use of REBox web platform capabilities to greatly automate this process and provide distributed access to its data.
Chapter 7

Conclusion

Requirements engineering strives for a good communication between stakeholders in order to achieve a shared vision of the problems to be addressed. An efficient collaborative tool to manage requirements specification documents is, therefore, crucial to guarantee an alignment of expectations between all involved parties. Collaborative RE tools currently available already offer ways to manage those documents, but are more focused on the organization and traceability of requirements, rather than the quality and rigorousness of their representation. Furthermore, there are still RE concerns which have not yet been properly addressed, such as the lack of proposals to efficiently model variability aspects of software systems at the requirements level. Variability modeling proposals in the field of software development thus far have mostly been directed towards the features of the systems, rather than their requirements, which comes from the fact that features are often closer to how stakeholders perceive and understand variability.

Fortunately, CVL comes as the OMG’s latest proposal for a Variability Modeling standard, defining itself as a domain-independent language for specifying and resolving variability. In practice, this allows the representation the system’s features variability to be independent from the DSL used to model those features. This domain-independence makes CVL an ideal language to develop a framework for modeling variability at the requirements level, incentivizing experimentation and the proposal of new approaches in the field.

This research has proposed REBox, a collaborative web-based platform for managing requirements specification documents with an innovative approach for modeling variability aspects in RE using the CVL language. To promote the efficiency of its processes, it offers a set of reusability features, including a Document Template System for uploading/managing SRS templates and a Requirements Specifications Library (RSLib) for creating and reusing modular sets of reusable requirements. Its variability modeling framework defines a process for applying the CVL language’s central variability notions, the Variation Points, to the views of a structured software requirements specification document. The language used to model that rigorous requirements specification in REBox is RSL-IL.

The web platform has been build using the DotNetNuke .NET web CMS and employs the Google Drive cloud storage service. This technology provides both the Requirements Specification Document (RSDoc) editor though Google Sheets, as well as the collaborative environment though its synchronization system and distributed architecture. In addition, it uses a series of Google provided APIs for automating processes and managing data. Thanks to this, the REBox variability modeling approach can automatically extract the information present in a specification document, modify it and generate new documents and views as needed. This allowed to define an automatized process for applying CVL concepts in the context of requirements engineering.

The work presented in this dissertation was developed over the last 12 months using the Action...
Research methodology and evaluated in order to gage its overall usefulness and the quality of its processes. The results were obtained though a user session in which the participants were asked to follow a User Guide containing a set of instructions that implied making use of the full capabilities of the platform.

Following that, a questionnaire was conducted to assess the opinion of the users in regards to three main aspects: (1) the Web Platform and its features, (2) the Requirements Variability Modeling framework and (3) the General Approach. The results were positive and showed evidences that demonstrate REBox's usefulness, feasibility and support the thesis behind this dissertation. Additionally, it is important to note that all the goals established in Section 1.2 have been accomplished during the course of this research work.

The remaining of this chapter is organized as follows: Section 7.1 describes the main contributions of this research work and section 7.2 indicates possible future research directions.

7.1 Main Contributions

We believe that the REBox platform provides a well-rounded collaborative RE platform which combines some of the strong points of other tools available, while offering an innovative way to model variability aspects in the field of requirements engineering using the CVL language.

Furthermore, it is based on an extensible technology, the Google Drive and its APIs, making it very easy to develop new features regarding the extraction and analysis of data from RE documents.

Further contributions of this work can be aligned with the research questions formulated in Section 1.3. Thus, this research work also provides the following contributions aligned with the research questions:

**RQ1**: Provides a comparative study on the currently available collaborative RE tools, divided in three groups: (1) text/spreadsheet editors, (2) enterprise RE management platforms and (3) RE Wikis;

**RQ2**: Provides a new collaborative RE platform the combines some of the main benefits from each group: (1) a very familiar and easy-to-use spreadsheet editor, (2) advanced reusability features such as a Template Manager and a Modular Requirements Library, and (3) a rigorous DSL for RE that allows the modeling of concerns such as requirements correctness and consistency.

**RQ3**: Provides an innovative approach for managing variability in RE documents by adapting the latest standard in feature variability modeling, the CVL language.

7.2 Future Work

This section presents some of the main directions that can be followed in a future research related with this work. Either due to time restrictions, or because of their complexity, these proposals were at some point conceptualized, but did not get the chance to be implemented. Nevertheless, It is important to emphasize that none of them undermine the achievement of the goals of this work.

**Enhance several usability aspects of the REBox platform** - Regarding the web platform, and due to natural time constraints, the visual frontend and some usability features were overlooked when trying to guarantee that at least the whole project and its features worked as intended. With this said, there are clear ways to make the project more user-friendly, some of which were inclusively suggested by the test users. Perhaps the most relevant suggestion was related to making the process of selecting VarTokens...
(variation points related to the property of a requirement) more accessible. Right now it has to be done by writing the name of the attribute in plain text on a textbox, but future work should focus on presenting all the selectable attributes in the form of checkboxes, increasing usability and preventing eventual typing errors. Furthermore, several other minor graphical enhancements can easily be made to improve the overall experience, like creating a landing page explaining some of the platform’s functionalities.

**Improve the integration process with RSLingo Studio** - As described in section 4.2.4, the integration with RSLingo studio can be very fruitful for enhancing the rigor of RSL-IL requirements specifications. This being said, the current “Ad-hoc” integration is not ideal. In the future, a more advanced integration could make use of an interface recently developed by the xText tool for extending its text editors to web applications using the Ace\(^1\) web editor. In this case, the language-related features offered by the RSLingo Studio Tool, such as syntax highlighting and error checking would be realized through HTTP requests to a server-side component and shown directly in a REBox web page.

**Extend the Variability Model RSL-IL supported views** - Future work should also focus on expanding the current RSL-IL language supported views, allowing to progress from simply modeling variability at the requirements level (Goals + Functional + Quality), to all the other viewpoints, like the Stakeholders, Entities, Use Cases and several others, allowing for a much wider scope of variability points within the spectrum of requirements engineering concerns.

**Define dependency mechanisms for variation points** - As it stands, any requirement can be defined as a Variation Point during the Variability Model creation. This means that a “parent” requirement that has other dependent requirements (e.g. 2 and 2.1; 2.2; etc.) can be considered variable while its children do not. The consequence of this is only seen in the materialization phase when existence values need to be chosen for each requirement. If a “False” is chosen, the requirement is removed but its dependent requirements are not, creating an inconsistency. Future work should define a “dependency” mechanism (possible tied to the RSL-IL “PartOf” attribute) that automatically forces every “children” requirement to also be considered a Variation Point and follow the same resolution values as its parent.

---

1. [https://ace.c9.io/#nav=about](https://ace.c9.io/#nav=about)
Bibliography


Alan Davis. *Just enough requirements management: where software development meets marketing*. Addison-Wesley, 2013.


OMG. Common Variability Language (CVL) OMG Revised Submission. (Cvl), 2012.


Appendix A

RSL-IL Template Requirements Views

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type/Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Unique Identifier of the Goal</td>
<td>String</td>
</tr>
<tr>
<td>Name</td>
<td>Descriptive name of the Goal</td>
<td>String</td>
</tr>
<tr>
<td>type</td>
<td>Type of the Goal</td>
<td>Concrete; Abstract</td>
</tr>
<tr>
<td>Source</td>
<td>The Stakeholder who defined the goal</td>
<td>&lt;Stakeholder Ids&gt;</td>
</tr>
<tr>
<td>PartOf</td>
<td>Type of dependency of the current goal with other goal(s)</td>
<td>&lt;Goals Ids&gt;</td>
</tr>
<tr>
<td>Description</td>
<td>Description of the Goal</td>
<td>String</td>
</tr>
<tr>
<td>Priority</td>
<td>The level of importance and priority of the goal</td>
<td>Must; Should; Could; Won’t</td>
</tr>
<tr>
<td>Progress</td>
<td>The current status of the development process of the Goal</td>
<td>NotPlan; Plan; OnDesign; OnDevelop; OnTest; OnDeploy; Concluded</td>
</tr>
</tbody>
</table>

Table A.1: Properties of the Goals View in the RSL-IL Template

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type/Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Unique Identifier of the Requirement</td>
<td>String</td>
</tr>
<tr>
<td>Name</td>
<td>Descriptive name of the Requirement</td>
<td>String</td>
</tr>
<tr>
<td>type</td>
<td>Type of the Requirement</td>
<td>Functional; Behavioral; Data</td>
</tr>
<tr>
<td>Source</td>
<td>The Stakeholder who defined the Requirement</td>
<td>&lt;Stakeholder Ids&gt;</td>
</tr>
<tr>
<td>PartOf</td>
<td>Type of dependency of the current Requirement with other Requirement(s)</td>
<td>&lt;Requirement Ids&gt;</td>
</tr>
<tr>
<td>Description</td>
<td>Description of the Requirement</td>
<td>String</td>
</tr>
<tr>
<td>Priority</td>
<td>The level of importance and priority of the Requirement</td>
<td>Must; Should; Could; Won’t</td>
</tr>
<tr>
<td>Progress</td>
<td>The current status of the development process of the Requirement</td>
<td>NotPlan; Plan; OnDesign; OnDevelop; OnTest; OnDeploy; Concluded</td>
</tr>
</tbody>
</table>

Table A.2: Properties of the Functional Requirements View in the RSL-IL Template
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type/Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Unique Identifier of the Requirement</td>
<td>String</td>
</tr>
<tr>
<td>Name</td>
<td>Descriptive name of the Requirement</td>
<td>String</td>
</tr>
<tr>
<td>Type</td>
<td>Type of the Requirement</td>
<td>appearance, security, usability, performance, operational, maintenance, cultural, legal, other</td>
</tr>
<tr>
<td>Sub-Type</td>
<td>Type of the Requirement</td>
<td>usability.ease-of-use, usability.easy-of-learn, usability.accessibility, . . . security.privacy, security.integrity, . . .</td>
</tr>
<tr>
<td>Metric</td>
<td>Metric to define quantitatively the requirement</td>
<td>String (Depends on the specific Type and Sub-Type)</td>
</tr>
<tr>
<td>Value</td>
<td>The intended value or value range to achieve</td>
<td>String</td>
</tr>
<tr>
<td>Source</td>
<td>The Stakeholder who defined the Requirement</td>
<td>&lt;Stakeholder Ids&gt;</td>
</tr>
<tr>
<td>PartOf</td>
<td>Type of dependency of the current Requirement with other Requirement(s)</td>
<td>&lt;Requirement Ids&gt;</td>
</tr>
<tr>
<td>Description</td>
<td>Description of the Requirement</td>
<td>String</td>
</tr>
<tr>
<td>Priority</td>
<td>The level of importance and priority of the Requirement</td>
<td>Must; Should; Could; Won’t</td>
</tr>
<tr>
<td>Progress</td>
<td>The current status of the development process of the Requirement</td>
<td>NotPlan; Plan; OnDesign; OnDevelop; OnTest; OnDeploy; Concluded</td>
</tr>
</tbody>
</table>

Table A.3: Properties of the *Quality Requirements* View in the RSL-IL Template
Appendix B

User Session Guide
User Session Guide


The objective of this pilot-user test session is to perform an evaluation of REBox platform by users that are not familiarized with it, in order to detect bugs or user limitations. The evaluation method involves a very simple case study application: “the Billing System”. The creation of the requirements document and the usage of the REBox platform by the users will be used to assess the system usability and to study further improvements to both the platform and the variability framework. The case study application description is as follows:

Case Study - The Billing System

This “Billing System” was originally defined for the purpose of showing how to specify a simple business information system based on the RSL-IL language.

It provides a complete set of features for managing customers, products and invoices and can be configured to support the work of different business managers (operators, product operators, managers, and sys-admins) in order to allow them collaboratively manage and participate in the processes related the creation, approval, issue of invoices, control their respective payments, and produce several reports. It also implements a simple workflow associated to the invoice's lifecycle, namely considering the following states: created, waiting-for-approval, approved, issued, paid, and deleted.

Test Conditions:

This user test session will be conducted under the following conditions:

- Tests are conducted in the laboratory (controlled environment);
- The tasks must be performed without previous use and learning (for the first time);
- The user must have a computer with a running web browser;
- Direct Observation, i.e., while users perform the assigned task, their behaviour and performance can be logged;
- Users can think out loud and share ideas if they want;
- The evaluator does not interact with the users until the tests are finished (except in case of blocking errors);
- The session will last 50 minutes (at most).
- The user must fill a survey in the end – available at: https://goo.gl/forms/twOAvtvijSY7nRqe53
**Instructions:**

1. Download the Excel file named “BillingSystem.xlsx” from https://drive.google.com/drive/folders/0B2sdf9tUjq8IWndMaHB2NGg3RWM
2. Rename the file by adding your name as a prefix (e.g. joaofernandes_BillingSystem.xlsx)
3. Open your web browser and enter the site: http://inescrebox.azurewebsites.net/
4. Login the platform using the credentials that were previously provided to you;

**5. Template Upload:**
   5.1. Enter the “Templates” tab;
   5.2. Select “+ Upload”;
   5.3. Choose the Excel file that you renamed previously and click “Upload”;
   5.4. Wait for the upload to finish and confirm that the file uploaded correctly;

**6. Project Creation:**
   6.1. Enter the “Projects” tab;
   6.2. Select “+ Create Project”;
   6.3. As the name, enter your own name followed by “REProject” (e.g. joaofernandes_REProject);
   6.4. Write a small description of the project (or alternatively, just write “Test”);
   6.5. Select yourself as the project owner;
   6.6. In the option “Document Type” select “Simple Project”;
   6.7. In the option “Template” select the template you uploaded previously;
   6.8. Click “Save”;
   6.9. You will be redirected to the Projects Page. Confirm that a new project was created with the name that you provided and click in the Edit button;
   6.10. Explore the spreadsheet document, particularly the **reqs.goals**, **reqs.functional** and **reqs.quality** sheets. If you want to increase the size of the document for better reading, choose the “Maximize” option;

**7. Requirements Library Module Creation:**
   7.1. Enter the “RS Library” tab;
   7.2. Select “+ Create RS Module”;
   7.3. As the name, enter your own name followed by “RSLibModule” (e.g. joaofernandes_RSLibModule);
   7.4. Again, write a small description if you want or simply just type “Test”;
   7.5. Select yourself as the RSLib owner;
   7.6. In the option “Requirements Type” select “Quality Requirements”;
   7.7. In the option “Import from Template” select “Yes”;
   7.8. In the option “Template to Import” select the template named “Cultural RSLib Module”;
   7.9. Click “Next”;
   7.10. Check the imported requirements and add a new one by clicking the “+” sign;
   7.11. Enter the following values:
       7.11.1. ID: QR_C_3
       7.11.2. Name: Default Timezone
       7.11.3. Description: The system should use GMT as its default Timezone
   7.12. (Important) Click the “+” sign in the new row to add the requirement;
   7.13. Click “Save”;
   7.14. You will be redirected to the RS Library Page. Confirm that a new module was created with the name that you provided.
   7.15. Enter the “Projects” tab;
7.16. Locate the project you created in point 6 and click the button under the “Actions” row.
7.17. In the option “RS Module Type” select “Quality Requirements”;
7.18. In the option “RS Module” select the module you created previously;
7.19. Click “Add Module”;
7.20. You will be redirected to the Projects Page. Again, click the Edit button on the project you created previously;
7.21. Go to the `reqs.quality` sheet and verify if all the new Cultural Requirements were added to the bottom of the document (Google often loses formatting on new cells, so don’t worry if the cells are not formatted correctly);

8. Variability Model Creation:
8.1. Go back to the “Projects” tab;
8.2. This time, locate the project you created previously and click the button under the “Actions” row.
8.3. As the name, enter your own name followed by “VarModel” (e.g. joaofernandes_VarModel);
8.4. Again, write a small description if you want or simply just type “Test”;
8.5. Select yourself as the VarModel owner;
8.6. Click “Next”;
8.7. Verify that all the requirements from the `reqs.goals`, `reqs.functional` and `reqs.quality` sheets were imported (if you want, you can minimize/maximize the panels by clicking on their names);
8.8. In the “Non-Functional Requirements” panel,
  8.8.1. Locate the requirement with the Id “QR_C_2_2” (penultimate) and:
    8.8.1.1. Check the checkbox “VarPoint”;
    8.8.1.2. In the VarTokens input, write: “Priority”.
  8.8.1.3. **(Important)** Click the icon in the row to save the changes!
  8.8.2. Locate the requirement with the Id “QR_C_3” that you created and:
    8.8.2.1. Check the checkbox “VarPoint”;
    8.8.2.2. In the VarTokens input, write: “Description.TextValue#GMT”.
    8.8.2.3. **(Important)** Again, click the icon in the row!
8.9. Click “Next”;
8.10. Verify that two VarSpecs were created based on the two Variation Points selected previously;
8.11. In the “Constraints” tab, add a new constrain with the following information:
  8.11.1. **Name:** “Constraint1”;
  8.11.2. **Type:** Logical Expression;
  8.11.3. **VarPoints:** “VP_1:Priority, VP_2:Description.TextValue#GMT” (copy and paste this string);
  8.11.4. **Constraint:** “NOT(VP_2:Description.TextValue#GMT = EDT) OR VP_2:Priority = Must” (copy and paste this string);
  8.11.5. **Description:** “Test Constraint”;
  8.11.6. **(Important)** Click the “+” icon in the new row to add the constraint;
8.12. Click “Save”;
8.13. You will be redirected to the Variability Models Page. Confirm that a new Variability Model was created with the name that you provided and click in the Edit button;
8.14. Explore the spreadsheet document, particularly the `var.specs` and `var.specs.constraints` sheets. If you want to increase the size of the document for better reading, choose the “Maximize” option;
9. **Resolved Model Creation:**
   9.1. Go back to the “Projects” tab;
   9.2. Select “+ Create Project”;
   9.3. As the name, enter your own name followed by “ResolvedProject” (e.g. joaofernandes.ResolvedProject);
   9.4. Again, write a small description if you want or simply just type “Test”;
   9.5. Select yourself as the project owner;
   9.6. In the option “Document Type” select “Variability Derived Project”;
   9.7. In the option “Variability Model” select the one with your name that you created in the last step;
   9.8. Click “Next”;
   9.9. In the first VarSpec (VP_1):
       9.9.1. Existance : True;
       9.9.2. Resolution Values: “Should”;
   9.10. In the second VarSpec (VP_2):
       9.10.1. Existance : True;
       9.10.2. Resolution Values: “EDT”;
   9.11. Click “Save”;
       9.11.1. An error will occur with the constraint, so change the Resolution Values of the first VSpec to “Must”;
   9.12. Click “Save” again;
   9.13. You will be redirected to the Projects Page. Confirm that a new project was created with the name that you provided and click in the Edit button;
   9.14. Explore the spreadsheet document, particularly the *reqs.quality* sheet. Verify that the requirements QR_C_2_2 and QR_C_3 have had their values substituted by the new values you provided. (Again, if you want to increase the size of the document for better reading, choose the “Maximize” option);

**For more information:**

Refer to the two informative documents in the folder: https://drive.google.com/drive/folders/0B2sdf9tUjq8lWndMaHB2NGg3RWM

- The presentation document for the user session (REBox_Presentation.pptx);
- The paper: *Variability Aspects in Requirement Specifications: Model and Implementation based on the CVL Standard*.pdf

And please, don’t forget to fill in the “REBox Pilot-User Test Session Survey” available at: https://goo.gl/forms/twOAv2vjSY7nRqe53

**Thank you for your help!**
Appendix C

User Session Questionnaire Results
Summary

Age

<table>
<thead>
<tr>
<th>Age</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>

Gender

<table>
<thead>
<tr>
<th>Gender</th>
<th>Number</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>6</td>
<td>85.7%</td>
</tr>
<tr>
<td>Female</td>
<td>1</td>
<td>14.3%</td>
</tr>
</tbody>
</table>

Degree

<table>
<thead>
<tr>
<th>Degree</th>
<th>Number</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSc</td>
<td>5</td>
<td>71.4%</td>
</tr>
<tr>
<td>MSc</td>
<td>2</td>
<td>28.6%</td>
</tr>
<tr>
<td>PhD</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

Number of years of professional experience

<table>
<thead>
<tr>
<th>Years</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Knowledge and previous experience with Requirements Engineering

Yes 7 100%
No 0 0%

How do you rate the overall usability of the Web platform? [REBox Web Platform]

0 (N/A – Do not know) 0 0%
1 - Very Low 0 0%
2 - Low 0 0%
3 - Medium 2 28.6%
4 - High 3 42.9%
5 - Very High 2 28.6%

How do you rate the usefulness of the main RE productivity features? (RSDoc Spreadsheet Editor, Template Manager, RSLibrary)? [REBox Web Platform]

0 (N/A – Do not know) 0 0%
How easy to learn (or how familiar) was the main document editing tool (Google Sheets)? [REBox Web Platform]

- Very Low: 0 (0%)
- Low: 0 (0%)
- Medium: 0 (0%)
- High: 5 (71.4%)
- Very High: 2 (28.6%)

How suitable is the platform for a collaborative management of Requirements Specifications? [REBox Web Platform]

- Very Low: 0 (0%)
- Low: 0 (0%)
- Medium: 0 (0%)
- High: 5 (71.4%)
- Very High: 2 (28.6%)

0 (N/A – Do not know)
How easy to understand was the overall variability modeling process? [REBox Variability Modeling Framework]

How easy to understand were the concepts? (VarPoints, VSpecs, VarTokens, Constraints, etc.) [REBox Variability Modeling Framework]
How do you rate the usefulness of the framework? [REBox Variability Modeling Framework]

0 (N/A – Do not know) 0 0%
1 - Very Low 0 0%
2 - Low 0 0%
3 - Medium 0 0%
4 - High 3 42.9%
5 - Very High 4 57.1%

How do you rate the simplicity of the Variability Model creation phase? [REBox Variability Modeling Framework]

0 (N/A – Do not know) 0 0%
1 - Very Low 0 0%
2 - Low 0 0%
3 - Medium 1 14.3%
4 - High 5 71.4%
5 - Very High 1 14.3%
How do you rate the simplicity of the Resolved Model creation phase? [REBox Variability Modeling Framework]

How do you rate the productivity with REBox comparing to the traditional Requirements Specification process? [REBox General Approach]

How likely would you use this platform on your own Requirements Engineering projects? [REBox General Approach]
Additional Comments (Suggestions, Problems, Bugs)

Nice work! It has few usability bugs and some concepts that are only clear for experts (as it should be probably). I just leave some minor issues: - Add a close option to the maximixed window (e.g. Project Edit -> Maximize -> Close option); - In Module Setup, disable or hide the "Template to Import" option (Module Setup -> Import from Template 'No' -> Disable Template to Import); - The association of VarPoints in the Variability Model Setup could be more clear than inserting plain text into a textbox (Variability Model Setup -> Associate

I enjoy very much the result of your research, Congratulations!!