RUBRIC: A Flexible Tool for Automated Checking of Conformance to Requirement Boilerplates

Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand
SnT Centre for Security, Reliability and Trust
University of Luxembourg, Luxembourg
{firstname.lastname}@uni.lu

Frank Zimmer, Raul Gnaga
SES TechCom
9 rue Pierre Werner
Betzdorf, Luxembourg
{firstname.lastname}@ses.com

ABSTRACT
Using requirement boilerplates is an effective way to mitigate many types of ambiguity in Natural Language (NL) requirements and to enable more automated transformation and analysis of these requirements. When requirements are expressed using boilerplates, one must check, as a first quality assurance measure, whether the requirements actually conform to the boilerplates. If done manually, boilerplate conformance checking can be laborious, particularly when requirements change frequently. We present RUBRIC (ReqUirements Boil-eRplate sanIty Checker), a flexible tool for automatically checking NL requirements against boilerplates for conformance. RUBRIC further provides a range of diagnostics to highlight potentially problematic syntactic constructs in NL requirement statements. RUBRIC is based on a Natural Language Processing (NLP) technique, known as text chunking. A key advantage of RUBRIC is that it yields highly accurate results even in early stages of requirements writing, where a requirements glossary may be unavailable or only partially specified. RUBRIC is scalable and can be applied repeatedly to large sets of requirements as they evolve. The tool has been validated through an industrial case study which we outline briefly in the paper.

Categories and Subject Descriptors
D.2.1 [Software Engineering]: Requirements/Specifications

General Terms
Verification

Keywords
Requirement Boilerplates; Natural Language Processing (NLP); Text Chunking.

1. INTRODUCTION
Natural Language (NL) is the most frequently used means for requirements specification. When compared to formally-specified requirements, NL requirements tend to be easier to understand and manipulate since no special training is required. Despite their advantages, NL requirements are prone to ambiguity, meaning that even with careful specification, they may be understood in more than one way by different stakeholders [6].

Requirement boilerplates, also known as templates, molds or patterns [2, 6], provide an effective way to bring discipline to NL requirements and to mitigate many types of ambiguity in natural language statements. In addition, boilerplates make the requirements more amenable to automated analysis, e.g., semantic consistency checking and transformation to models [6].

Figure 1 shows a well-known requirement boilerplate due to Rupp [6]. The boilerplate envisages six syntactic slots: (1) an optional condition at the beginning; (2) the system name; (3) a modal (shall/should/will) specifying how important the requirement is; (4) the required processing functionality; this slot can admit three different forms based on the manner in which the functionality is to be rendered; (5) the object for which the functionality is needed; and (6) optional additional details about the object. As an example, we show in Figure 2 a requirement statement R1, which conforms to Rupp’s boilerplate. The segments corresponding to the different slots in the boilerplate have been marked, with the fixed elements of the boilerplate written in capital letters.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ESEC/FSE ’13, August 18-26, 2013, Saint Petersburg, Russia
Copyright 2013 ACM 978-1-4503-2237-9/13/08 ...$15.00.

Figure 1: Rupp’s requirement boilerplate [6]

Figure 2: An example requirement that conforms to Rupp’s boilerplate

To gain benefits from using boilerplates, it is paramount that the analysts first ensure that the boilerplates have been correctly applied. If done manually, checking boilerplate conformance for large collections of requirements is a laborious task [6], particularly when the task has to be repeated as the requirements evolve and many stakeholders are involved.

In this paper, we present RUBRIC (ReqUirements Boil-eRplate sanIty Checker). The tool’s main function is to automatically verify correct use of boilerplates in NL requirements. RUBRIC does not require a complete requirements glossary to be constructed first and provides highly
accurate results even without a glossary. RUBRIC in addition provides a range of diagnostics highlighting potentially problematic syntactic constructs in requirements, e.g., use of passive voice, conjunctions, pronouns and vague terms [2]. In the remainder of this paper, we describe the motivation behind RUBRIC, along with its main components and usage scenarios. We further outline key points from the evaluation we have performed of the tool in an industrial case study.

2. MOTIVATION AND RELATED WORK

The motivation for RUBRIC comes from an observed need when attempting to introduce boilerplates for documenting the requirements of a large satellite system. A thorough inspection of the requirements of the system to ascertain boilerplate conformance would take several hours. This level of effort combined with the frequent changes made by the numerous stakeholders involved in the project posed a challenge for maintaining boilerplate conformance.

Prior to developing RUBRIC, we conducted a review of existing requirements engineering tools, guided by a recent and comprehensive tools survey [4]. The goal of our review was to determine whether we could use already-existing solutions for boilerplate conformance checking. We found two tools that directly addressed this task: DODT [5] and RQA [7]. DODT provides means for bringing requirements into conformance with a boilerplate via requirements transformation; RQA provides features for defining boilerplates and automatically checking conformance to them.

Neither tool was an ideal match for our needs: DODT requires a high-quality domain ontology for the system to be developed first. In our project and arguably in most other industrial projects, developing such an ontology is too costly to be practical. As for RQA, while an investigation of the underlying technologies was not possible due to the tool’s proprietary nature, we observed that the tool’s ability to correctly identify different segments of requirements statements was adversely impacted when the glossary terms were left unspecified. This constraint too posed a problem, because we wanted to provide feedback to analysts about boilerplate usage at early stages, when such feedback is most useful and easiest to apply. Subsequently, we could not assume the completeness of the glossary. Furthermore, the literature suggests that incompleteness in glossaries is a pervasive issue in many projects [8]. Therefore, a high level of reliance on the glossary for analyzing boilerplate conformance can have negative consequences not only in our motivating project but in other projects as well. RUBRIC in contrast, and as stated earlier, does not require a requirements ontology or glossary to be built first and yet manages to yield highly accurate results.

3. TOOL OVERVIEW

Figure 3 shows an overview of the process implemented by RUBRIC. The process consists of three main steps: (1) Text Chunking, (2) Boilerplate Conformance Checking, and (3) Natural Language Best Practices Checking. In the first step, requirement statements are segmented into constituent parts (noun phrases, verb phrases, etc.) and annotated with appropriate tags. The annotated requirements, along with the boilerplates’ syntax grammar rules (e.g., Rupp’s Boilerplate syntax in Figure 5) are used in the second step to produce boilerplates conformance results. The third step needs the annotated requirements from the first step and the NL requirements’ best practices rules (e.g., see [2]) to generate feedback for potentially problematic constructs in the requirements. In the rest of this section, we elaborate the three main steps shown in Figure 3.

3.1 Text Chunking Process

RUBRIC has been developed as an application for the GATE workbench (http://gate.ac.uk/). GATE is an opensource Natural Language Processing (NLP) framework. Building on the NLP modules available in GATE, the Text Chunking step in Figure 3 decompose a sentence into non-overlapping segments. The main segments of interest are Noun Phrases and Verb Phrases. A Noun Phrase (NP) is a segment that can be the subject or object for a verb. A Verb Phrase (VP), sometimes also called a verb group, is a segment that contains a verb with any associated modal, auxiliary, and modifier (often an adverb).

Similar to most NLP applications, a text chunker is a pipeline of NLP modules, run in a sequence over an input document. The (abstract) pipeline for chunking is shown in Figure 4. This pipeline can be instantiated in many ways, as there are alternative implementations for each step in the pipeline. The first module, the Tokenizer, breaks up the input into tokens. A token can be a word, a number or a symbol. The Sentence Splitter divides the text into sentences. The POS Tagger annotates each token with a part-of-speech tag. These tags include among others, adjective, adverb, nouns, verb. Next is the Name Entity Recognizer, where an attempt is made to identify named entities, e.g., organizations and locations. In a requirements document, the named entities can further include domain keywords and component names. The main and final step is the actual Chunker. Typically (but not always), NP and VP chunking are handled by separate modules, respectively tagging the NPs and VPs in the input. When a glossary of terms is available, one can instruct the NP Chunker to treat occurrences of the glossary terms in the input as named entities, thus reducing the likelihood of mistakes by the NP Chunker.

The input requirements after going through text chunking have annotations for tokens, sentences, parts-of-speech, named entities, NPs, and VPs. These annotations are used for checking conformance to boilerplates and generating feedback about adherence to requirements writing best practices.

3.2 Boilerplate Conformance Checking (BCC)

The BCC step takes as input an annotated requirements document (produced by the Text Chunking step) as well as the syntax rules for the boilerplates of interest. The main purpose of BCC is to tag each requirement statement as either conformant or non-conformant. BCC can further mark optional parts (e.g., conditions and additional details) that
3.3 Checking NL Best Practices

RUBRIC detects and warns about several potentially problematic constructs that are discouraged by requirements writing best practices. Berry et al. [2] give a detailed treatment of these best practices. The detection and markup of the problematic constructs is performed using JAPE in a similar manner to the BCC rules described earlier in Section 3.2. Figure 7 shows and exemplifies some of the problematic constructs that RUBRIC detects and generates warnings for.

![Figure 6: JAPE script for autonomous requirements](image)

Figure 6: JAPE script for autonomous requirements

![Figure 7: A (non-exhaustive) list of potentially problematic constructs detected by RUBRIC](image)

Figure 7: A (non-exhaustive) list of potentially problematic constructs detected by RUBRIC

As an example, Figure 8 shows the JAPE script for tagging the use of pronouns. Pronouns can lead to referential ambiguity if used improperly [2]. The rule in the figure matches any Token in the text whose part of speech is personal pronoun (PRP), possessive pronoun (PRPS), wh-phrase (WP), or possessive wh-phrase (WPS). If matched, the pronoun will be labeled with Warn_Pronoun. At a later stage, a requirements analyst can review these warnings to determine which constructs are indeed ambiguous and take appropriate remedial action.

![Figure 8: JAPE script for detecting pronouns](image)

Figure 8: JAPE script for detecting pronouns

4. DIAGNOSTICS

RUBRIC leverages GATE’s visual markup environment for providing diagnostics about boilerplate conformance and requirements writing best practices. Figure 9 shows a snapshot of RUBRIC. The left panel shows the various GATE resources (Applications, Language Resources, and Processing Resources). The centre panel displays the contents of the selected resource, here a requirements document. The right panel shows the annotations list for the current document. The highlighted sections in the document mark regions annotated with selected tags from the right panel.

5. EVALUATION

We have rigorously evaluated RUBRIC over an industrial case study, where 380 requirements were specified using Rupp’s boilerplate by industry experts [1]. The case study was conducted in collaboration with our industrial partner.
6. IMPLEMENTATION AND AVAILABILITY

The core of RUBRIC consists of 28 JAPE scripts with approximately 500 lines of script code. RUBRIC further includes 9 customizable lists containing keywords used by the scripts, e.g., boilerplate fixed terms, vague terms, conjunctions, acceptable modals. To facilitate using RUBRIC in a production environment, we have implemented a plugin to export requirements from the Enterprise Architect tool (http://www.sparxsystems.com.au). This plugin is approximately 1000 lines of C# code. Similar plugins can be developed for other requirements management tools.

RUBRIC was publicly released in June 2013. The tool (along with a screen cast illustrating its use) is available at: sites.google.com/site/rubricnlp/

7. CONCLUSION

We presented RUBRIC – a natural language processing tool for automatic checking of conformance to requirement boilerplates and detecting problematic constructs in natural language requirements. Our current implementation supports only Rupp’s boilerplate [6] but can be generalized to other boilerplates as well. Our evaluation indicates that RUBRIC is scalable and highly accurate in distinguishing requirements that conform to Rupp’s boilerplate from those that do not. The accuracy in general shows little sensitivity to the presence or absence of a requirement glossary.

In the future, we plan to conduct more studies to quantify the gains from using RUBRIC in industrial settings in terms of time, effort, and quality resulting from the use of boilerplate conformant requirements. We further need to examine the effectiveness of our approach for the case where the requirements are not consciously written to conform to a given boilerplate. Another topic for investigation is providing instant guidance to analysts to help them conform to boilerplates as they are writing the requirement statements.

8. ACKNOWLEDGEMENT

We gratefully acknowledge funding from the National Research Fund - Luxembourg (FNR/P10/03 - Verification and Validation Laboratory).

9. REFERENCES