International Journal on Perceptive and Cognitive Computing (IJPCC) Vol 10, Issue 2 (2024)
https://doi.org/10.31436/ijpcc.v10i2.487

Design and Development of a Requirements Conformance Tool (RCT)

Siti Syara Aiman Seh Wali, Azlin Nordin

Department of Computer Science (DCS), Kulliyyah of Information and Communication Technology (KICT), International Islamic University Malaysia (IIUM), 53100 Gombak Kuala Lumpur, Malaysia

*Corresponding author: azlinnordin@iium.edu.my
(Received: 29th June 2024; Accepted: 25th July 2024; Published on-line: 30th July 2024)

Abstract— To guarantee that a quality requirement is well-defined, it should meet various criteria, including completeness and unambiguousness. When a requirement statement is manually written, the quality of the requirements could be affected because the majority of requirements engineers particularly the inexperienced ones, have not been adequately trained. If they are unable to transfer stakeholders' needs into the requirements, they may end up with problematic requirements. As a result, they may be unable to provide high-quality specification requirements as a reference throughout the software development process. To reduce this problem, standard boilerplates' formats were established as one of the solutions. Nevertheless, requirement engineers may still require guidance in order to adopt any boilerplates that suit their needs. In this project, we seek to increase the quality of requirements by assisting requirement engineers in comprehending boilerplates. The Requirements Conformance Tool, which uses semi-automated boilerplates, was created to help requirements engineers determine whether the requirement conforms to the chosen requirements boilerplate or not. To show the use of boilerplates, the project was constructed in Java using basic Part-of-Speech (POS) tagger.

Keywords— boilerplates, ambiguity, requirements, conformance, software development, semi-automated

I. Introduction

Before developing a project, it is crucial to write requirements to specify what should be implemented. Requirements Engineering (RE) is described as the process involved in developing the system requirements [1] which describe how a system should behave, application domain information, obstacles on the operation of the system or the specifications of the system attribute. Requirements have also been described as [2]:

- A situation or functionality needed by a user to solve a hassle or gain a goal.
- A situation or functionality that needs to be met or possessed by a system or system component to satisfy a formally imposed document.
- A documented representation of a condition or capability as in (1) or (2).

The effect of the RE on successful and customer-oriented system development cannot be disregarded. It has turned out to be the usual practice to supply the sources for RE [2]. Requirement engineers play an important role to ensure the system requirements specification is being written correctly. RE is performed to allow communication between the stakeholders and programmers. To avoid project failures, it is crucial to handle the requirements of a system carefully. Requirement of a system is written in a document called Software Requirement Specifications (SRS). SRS is an important document which contains a list of requirements, and it explains what the stakeholders' wants which are intended as a basis for developing the software design [3]. A good SRS has the capability to ensure that the system developed is successful and meets the real users' needs while being a cost-effective creation. A full requirement free from any errors are important for a successful system development.

The errors in SRS need to be discovered early during the writing requirement phase, or the cost to pay for the maintenance will be high. One of the challenging issues in the current software industry is that requirement engineers frequently develop incorrect requirements with possibilities of various requirements errors. Such issues could inherently lead to reducing the SRS quality.

The most common mistake is that the requirements are missing and not clearly formulated [3]. One of the ways to improve the quality of SRS is boilerplate [3]. Boilerplates or also known as the requirements template is a blueprint for the syntactic structure of individual requirements. In this research, we are focusing on two boilerplates which are IREB's boilerplates and Easy Approach to Requirements Syntax (EARS) boilerplates.

https://doi.org/10.31436/ijpcc.v10i2.487

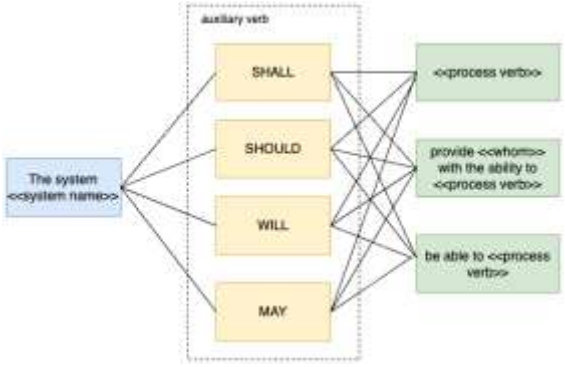

Fig. 1 Core requirement for IREB Boilerplate [2]

The International Requirement Engineering Board IREB) has come up with a step-by-step explanation of the correct approach of the requirement template. See Figure 1. It is also called Rupp's boilerplate. IREB boilerplates have three basic templates [2]. The first one is for the autonomous system activity where the users do not interact with the activity. The template is: THE SYSTEM SHALL/SHOULD/WILL/MAY <process verb>.

The second template is for the user interaction where the system provides a functionality to the user which requires them to interact with the system. The template is: THE SYSTEM SHALL/SHOULD/WILL/MAY provide <whom?> with the ability to <process verb>.

Finally, the third template is meant for interface requirements where the system is performing an activity, and it depends on the neighbouring system. The template is: THE SYSTEM SHALL/SHOULD/WILL/MAY be able to <process verb>.

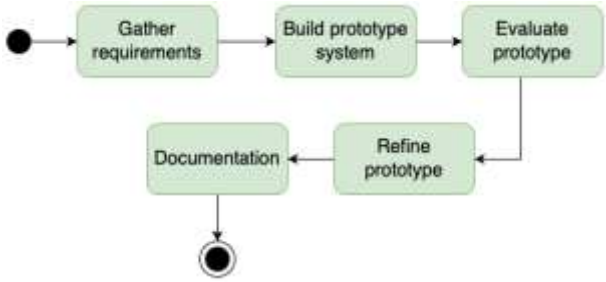

Fig. 2 Core requirements for EARS Boilerplate

In this research, there are six phases in this model. The first phase is to gather the requirements. All the information needed is collected and analysed through literature review. There are two tools that have been discovered in the process of collecting data which is the SRS tool [6] and Rubric tool [7]. The next phase is building the prototype. Once the team has finished the research on how to improve the quality of boilerplates, the prototype will be built.

The prototype is built using Java Eclipse where it focuses on creating new requirements and improving the existing requirements. The third phase is evaluating the prototype. The prototype is evaluated by making sure that it is functioning before it is shown to the supervisor. The supervisor will provide the feedback required to further improve the prototype which marks the fourth phase of this model. The fifth phase is where the feedback is taken into account, and improvements will be made to refine the prototype. The last phase is documentation. If there are no more alterations to be made, the prototype will be documented in a report.

II. LITERATURE REVIEW

In this paper, we analyzed two of the study papers that discussed requirement tools used to improve the quality of requirements. The papers are:

A. *Software Requirements Specification Tools*

The SRS tool focuses on requirement management feature. The tool provides the functionalities such as generating the requirements based on the boilerplates and modifying the boilerplate to the desired format where the user will be given the option to choose.

B. *RUBRIC*

Rubric Tool is a natural language processing (NLP) tool for automatic checking of conformance to the requirement boilerplates. Besides, the tool can also detect the problematic constructs in natural language requirements (see Figure 3).

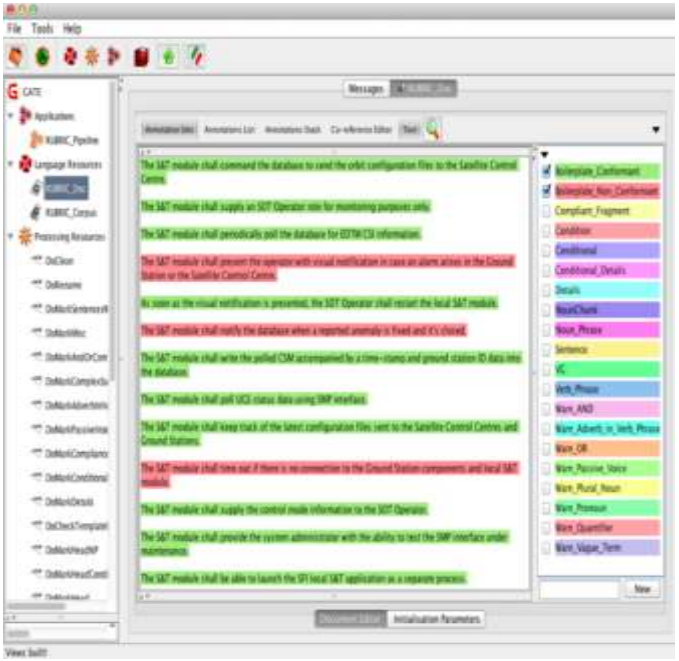

Fig. 3 RUBRIC [7]

https://doi.org/10.31436/ijpcc.v10i2.487

Based on the comparison in Table 1, we analyzed two existing tools. This is because these tools are in line with our tool. The chosen tools are Software Requirements Specification Tool [6] and RUBRIC Tool [7]. However, the Specification Tool is not published yet. After reviewing the three tools, we made a simple comparison between them as well as our proposed tool, Requirement Conformance Tool (RCT).

TABLE I
COMPARISON BETWEEN EXISTING TOOLS AND PROTOTYPE

Aspect	**Tools**		
	SRS Tool [6]	**Rubric Tool [7]**	**RCT Tool**
Requirement specification approach	Any boilerplate	IREB Boilerplate	IREB Boilerplate
Import requirement from text file	No	No	Yes
Export Requirement	Yes	No	Yes
Guiding in writing requirement	No	No	Yes
Support functional and non-functional requirements	Yes	Yes	Yes

For our tool, we focused on guiding requirement engineers to generate requirement statements so that they will not mistakenly use wrong words or structure. From the above comparison, it is observed that all the required engineering tools are using boilerplates. Nonetheless, the features and focuses provided in each of the tools are different.

In summary, our tool allows requirement engineers to export the generated requirements and also import the existing requirements to test whether the requirements are conformant to the boilerplate or not, which is an added advantage for our tool. As a future enhancement, the tool may include various existing templates, such as EARS boilerplates, to assist in producing requirements. This method could make the tool more adaptable to its users.

III. TOOL OVERVIEW

Prior to categorizing the requirement according to the IREB boilerplate, a set of possible rules of POS tag for each requirement statement have been studied and developed. Based on existing literature on the boilerplates and its structures, we derived the respective rules in Table 2. These rules highly depend on the three categories of the boilerplates i.e. (1) requirements for system activity, (2) requirements for user interaction, or (3) requirements for interface. For each of these categories, it will be used to detect different patterns for the conformance matching algorithm.

TABLE 2
RULES DEVELOPED TO CATEGORIZED REQUIREMENTS

User
DT(.*)MDVBDT(.*)INDTNNTOVB(.*)
Interface
DT(.*)MDVBJJTOVB(.*)
System
DT(.*)MDVB(.*)

In this table, (.*) (.*) can be NN, NNP, NNS, NNNNP, NNPNN and (.*) can be any part-of-speech tags based on what the requirement engineer chooses to write. The list of part-of-speech tags used are summarized as below.

TABLE 3
ABBREVIATIONS USED IN RULES

POSTAGGER	CATEGORY
DT	Determiner
NN	Noun, singular or mass
NNP	Proper noun, singular
NNS	Noun, plural
MD	Modal
VB	Verb, base form
IN	Preposition or subordinating conjunction
TO	To
JJ	Adjective

Figure 4 provides an overview of the process implemented in RCT. The process consists of three main steps: (1) Text Splitting, (2) POS Tagger to identify the tag meaning of each word, (3) Structure checking, and (4) Boilerplate Conformance Checking. In the first step, requirement statements are read in line and segmented into

https://doi.org/10.31436/ijpcc.v10i2.487

constituent parts (noun phrases, verb phrases, etc.) and annotated with appropriate tags by using POS Tagger.

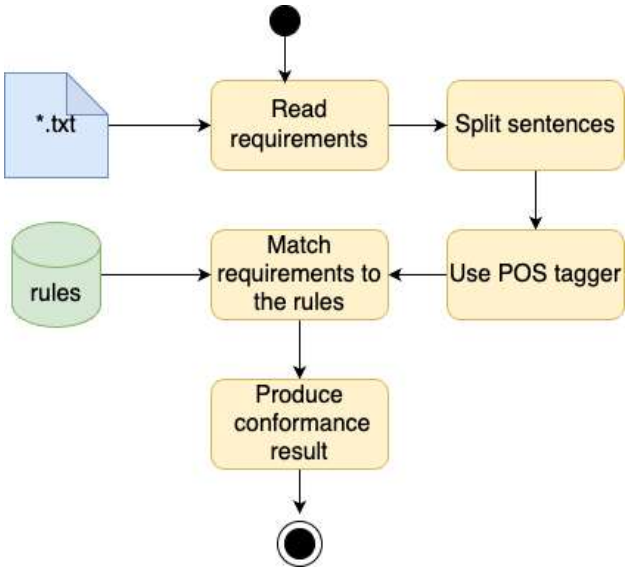

Fig. 4 Process in RCT

The annotated requirements, along with the IREB boilerplates' syntax grammar rules and structure checking are used in the next step to produce requirements that are conformant to the boilerplates results based on the position of the part-of-speech tags in the array.

IV. ANALYSIS AND DESIGN ALGORITHM FOR RCT

This section describes the algorithm used in identifying the conformance of the requirements to the chosen boilerplate template. In this approach (as can be seen in Figure 5), an abbreviated requirement array signifies that each statement is pushed into a three-dimensional array and placed in the first array. The location of the abbreviation indicates that each word of the sentence will be pushed into the second array. The third array will be populated with the position of the string modal verb array, with each word segmented using POS tagger. If the requirements satisfy the rules, they will be described as conforming to the boilerplates; otherwise, they will be stated as a faulty or non-compliance requirement, with an indication of which segment of the requirement source.

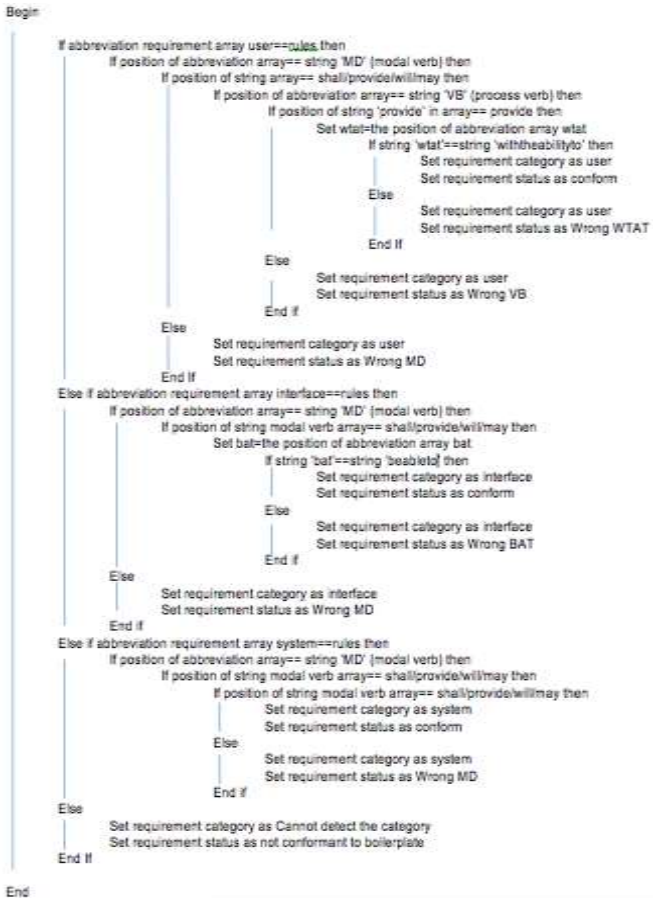

Fig. 5 Algorithm for Matching Requirements

V. RCT PROTOTYPE

This section elaborates a few of the snapshots of the RCT prototype. The screen shot in Figure 6 shows the RCT's main page where requirement engineers can choose to generate new requirements or to import existing requirements from the text files. Additionally, the requirement engineer should enter the project name before they want to write the requirements.

Fig. 6 Boilerplates Option

https://doi.org/10.31436/ijpcc.v10i2.487

The screen in Figure 7 shows the boilerplates option that requirement engineers can choose before writing the requirements. When they choose this feature, they will be able to pick any of these options i.e. (1) requirements for system activity, (2) requirements for user interaction, or (3) requirements for interface. Each of these options will lead into different requirements boilerplates.

Fig. 7 Generate Requirements

The subsequent Figure 8 demonstrates the fields that the requirement engineer should enter in order to generate the requirement based on the chosen boilerplate. This include the option for system name, auxiliary verb and the process verb. The typical process for defining project requirements is for project managers to decide ahead of time what kind of auxiliary verbs will be used and what they will signify. For example, in this Software Requirements Specification (SRS) sample in [9], the authors define the following auxiliary terms:

1. 'Must'- indicates requirements strictly to be followed to conform to the document and no deviation is allowed. Must is synonymous with "shall."
2. 'Should'- indicates that a possibility among a set of possibilities is recommended as particularly suitable
3. 'May' - indicates a course of action permissible within the limits of the document.

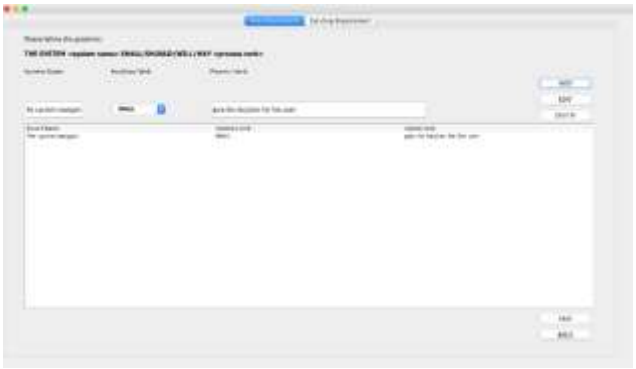

Fig. 8 Adding Requirements

Figure 9 shows the requirement is added to the table. The requirement engineers are able to add more requirements, edit, and delete the requirements by interacting with the respective buttons.

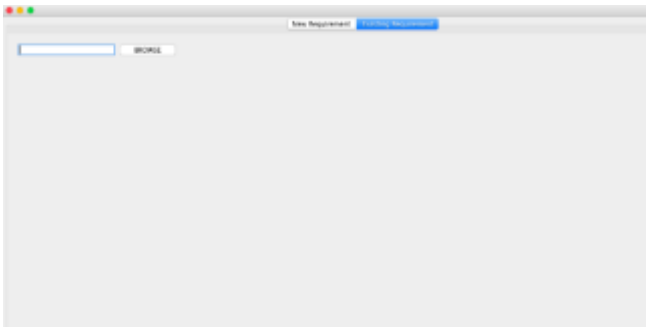

Fig.9 Upload Existing Requirement from Text File

Figure 9 depicts when the requirement engineers choose to import an existing requirement file. The requirements engineer could browse and import the text file to the tool. Based on the tool's process as depicted in Fig. 4, the result will display if the requirements conform to the boilerplates or not.

VI. IMPLEMENTATION

The Requirement Conformance Tool (RCT) was created using the Java programming language and includes a part-of-speech (POS) tagger for analyzing language components. However, the prototype has a few drawbacks. It is currently limited to IREB boilerplates and only considers the syntactic structure of requirements, not their semantic content. Furthermore, the tool requires conformance to a specified template for requirement creation prior to importing text files, and it runs as a standalone application rather than a web-based platform. This indicates that the tool is a standalone solution that cannot benefit from the web-based implementation's features, such as accessibility and collaboration.

VII. CONCLUSION

This tool provided guidance for the users to assist them in producing quality requirements using requirements boilerplates. Nonetheless, the current implementation supports only IREB boilerplate but in the future, this project is expected to generalize the tool to include other types of boilerplates as well. RCT aims to distinguish which existing requirement that has been written is conformed to the IREB boilerplate.

The challenge with the prototype is that the team lacks NLP skills, which has made it difficult to code the tool by utilizing the text chunking pipeline. The text chunking pipeline includes a tokenizer, a sentence splitter, a POS tagger, Named Entity Recognition, and an NP chunker.

This coding was completed in a restricted amount of time, thus the team devised an alternative method that uses only available JAVA libraries and imports only POS tagger libraries. This research effort has been a learning process, with many new discoveries. In addition, another limitation of this work is that the tool has not been validated for its usefulness, and its applicability. We plan to further evaluate this tool to the relevant potential users and get their feedback for future improvement.

ACKNOWLEDGMENT

The authors would like to extend our appreciation to the Kulliyyah of Information and Communication Technology (KICT), International Islamic University Malaysia (IIUM) as well as the Computer Science Department for the opportunity to work for this project.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest.

REFERENCES

[1] G. Kotonya., & I. Sommerville, *Requirements engineering: processes and techniques*. Wiley Publishing. 1998.

[2] K. Pohl. *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated. 2010.

[3] U. Anuar, S. Ahmad, & N.A. Emran. A simplified systematic literature review: Improving Software Requirements Specification quality with boilerplates. In *Software Engineering Conference (MySEC), 2015 9th Malaysian* (pp. 99-105). IEEE.

[4] C. Arora, Sabetzadeh, M., Briand, L. C., & Zimmer, F.. Requirement boilerplates: Transition from manually-enforced to automatically-verifiable natural language patterns. In *Requirements Patterns (RePa), 2014 IEEE 4th International Workshop on* (pp. 1-8). IEEE.

[5] R.S. Weinberg. Prototyping and the systems development life cycle. *Information System Management*, *8*(2), 1991, 47-53.

[6] K. Meng Y. Yeow "*Software Requirement Specification Tool.*" in Final Year Project report, pp. 29-30, 2016

[7] C. Arora, Sabetzadeh, M., Briand, L., Zimmer, F., & Gnaga, R. (2013, August). RUBRIC: A flexible tool for automated checking of conformance to requirement boilerplates. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (pp. 599-602). ACM.

[8] R.A. Carter, A.I. Antón, A. Dagnino, & L. Williams,. Evolving beyond requirements creep: a risk-based evolutionary prototyping model. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on* (pp. 94-101). IEEE.

[9] T. Hedberg Jr., M. Helu, and M. Newrock. (2017). 'Software requirements specification to distribute manufacturing data', National Institute of Standards and Technology, Gaithersburg, MD, NIST AMS 300-2, Dec. 2017. doi: 10.6028/NIST.AMS.300-2.