SOFTWARE REQUIREMENT SPECIFICATION TOOL

KONG MENG YEOW

A project report submitted in partial fulfilment
of the requirements for the award of
Bachelor of Science (Hons) Software Engineering

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

May 2016

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :

Name : Kong Meng Yeow

ID No. : 13UEB00709

Date : 15/09/2016

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"SOFTWARE REQUIREMENT SPECIFICATION TOOL"** was prepared by Kong Meng Yeow has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Science (Hons) Software Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : ______________________________

Supervisor : Too Chian Wen

Date : 15/09/2016

ACKNOWLEDGEMENT

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Miss Too Chian Wen for her invaluable advice, guidance and her enormous patience throughout the development of the project.

In addition, I would also like to express my gratitude to my loving parent who had supported and given me encouragement throughout the period of preparing this report.

Furthermore, I would like to extend my gratitude to all my friends who given me advices, feedbacks, technical assistance and being supportive in making this project a success.

SOFTWARE REQUIREMENT SPECIFICATION TOOL

ABSTRACT

Statistics has shown that requirement phase held great responsibility for software projects that exceeded their cost and time or even failed. The main factor is because requirements were frequently written ambiguously, inconsistently, and insufficiently. Most of the time, non-functional requirements were neglected and not specified as much as functional requirements although they were both equally important. The main objective of this project was to propose and develop a software requirement specification tool to rectify the above mentioned issues. Our tool was focused on assisting user to specify both functional and non-functional requirements in a structured and consistent manner. We conducted literature review to study in depth about problems in requirement specification with natural language. The approach used to solve this problem was to use a structured natural language or requirement boilerplate to generate unambiguous and consistent requirements. Using this approach, user inputs were gathered, reformatted and represented as structured requirements. ISO 25010 quality model was referred as a guideline to support requirement specification of non-functional requirements. As an outcome of this project, we produced and deployed a web application on the Internet for users to specify their project's requirements. Last but not least, evaluation was done on the tool by requesting user to specify an existing project's requirements and then to complete a survey. In conclusion, our tool was able to help our participants to specify requirements effectively and efficiently.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

TABLE OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

In the process of creating a software product, there are a set of related activities that will be performed. According to (Hull et al. 2011), in the field of Software Engineering, there are four fundamental activities that cannot be excluded. These activities are software specification, software design and implementation, software validation and software evolution.

The first fundamental activity that will be conducted in every software process is software specification. Software specification refers to the definition of functionality and constraints on operations of the software. As an outcome of software specification, a software requirement document or software requirement specification (SRS) will be produced. SRS is an agreed statement for both system customer and software developer for the software product that will be delivered. However, many software project failed due to the problems in the process of software specification.

It was reported that incomplete or changing requirements and specifications were the main reasons that software projects went over schedule and budget. The main factor that causes projects to be cancelled was also due to incomplete requirements of product (Clancy 1995). In another report from Project Management Institute (PMI), inaccurate requirement gathering is the root cause of 38% of failed projects (Anon 2015). It is estimated that $81 billion was wasted on cancelled project and $59 billion was incurred for project extensions (Clancy 1995)

1.2 Problem Statement

When software requirements are gathered correctly, requirements will be specified in an understandable and consistent manner. When a complete set of software requirement specification (SRS) are produced, a software project is already considered a partially successful project. This is the ideal situation each project manager wished to have in their projects.

However in reality, requirements were frequently written in ambiguous sentences and inconsistent manner which confuses reader, and incomplete which fails to specify all the important and core requirements (Bures et al. 2012). In addition, non-functional requirements were often left out from requirement specification although they are as important as functional requirements (Azuma 2004).

Hence, our proposed solution to overcoming the above mentioned problems is to develop a software requirement specification (SRS) tool. Our proposed tool will emphasize in specifying a clear and consistent requirement, and produce a complete set of SRS. The proposed tool will also emphasize the elicitation and specification of non-functional requirements, which is not supported by other tools.

1.3 Proposed Solution

Our proposed solution is to provide a tool that gathers information provided by user and convert them into requirements using boilerplate. We propose to use natural language requirement boilerplates as the templates of requirements. Then, we will request required information from user and add into boilerplate in order to generate requirements. All elicited requirements will be saved and listed as a software requirement specification (SRS).

The usage of natural language requirement boilerplate will effectively resolve the problem of inconsistency and ambiguity of requirement. The reason is because requirement boilerplates are structured natural language patterns. Requirement generated from requirement boilerplate will always be in certain sentence structure, making it consistent. When expressing requirements in a structured and consistent manner, it is also less likely to misinterpret the real meaning of requirement.

To support non-functional requirement elicitation, ISO 25010 quality model (International Organization For Standardization ISO 2011) will be referred as guidance to generate non-functional or quality requirements. We will provide user interfaces that is designed to collect information from user in order to generate non-functional requirements.

1.4 Proposed Approach

In conducting this project, the software development approach that will be used is Rational Unified Process (RUP). RUP is an iterative and incremental software development process and it encourages following certain best practices. The product of this project will be a Software Requirement Specification Tool, which is a web application that can be deployed to cloud and accessible from any major browsers.

The tool will be built based on client-server architecture. The server side will provide a RESTful API built on top of NodeJS and ExpressJS. The REST API will serve as intermediary between database and the frontend of the website. MongoDB, a NoSQL database will be used to store all project information of the tool. On the client side, AngularJS along with Materialize CSS framework will be used as the frontend of the tool. AngularJS will provide transition between user interfaces, handling program logics and updating server when saving project, while Materialize CSS will provide a material design themed user interface and experience for user.

For the approach of specifying software requirements, we will be utilizing requirement boilerplate, which is a structured natural language template. User inputs will be gathered and mapped into boilerplate to generate both functional and non-functional requirements. We will also integrate ISO 25010 model to guide non-functional requirement elicitation and specification process in our tool.

1.5 Project Goal

The goal of this project is to assists user in requirement elicitation and specification phases and improve the quality of requirements that produced in requirement specification phase.

1.6 Project Objectives

The objectives of this project are:

1. To prepare a complete project proposal to conduct this project
2. To conduct literature reviews on every aspect of this project
3. To plan and decide the methodology to be used to conduct this project
4. To specify and model requirements that shall be fulfilled in this project
5. To analyse and design each aspect of the tool of the project
6. To code and implement the project and produce our proposed tool
7. To test and evaluate the effectiveness and efficiency of our produced tool

1.7 Project Scope

The following sections will describe the target users of this tool, modules that are covered and those which are not covered.

1.7.1. Target Users

The target users of the proposed tools are requirement engineers of software project. The users will be able to use this tool to assist them to elicit requirements and specify requirements in structured natural language format.

1.7.2. Modules Covered

The following modules shall be provided by the proposed tool in order to achieve the project objective.

The following modules are covered in this project:

1. **Boilerplate maintenance**
   The system will provide boilerplate that can be used and modified by user. Boilerplate templates are structured natural language patterns which will be used to generate consistent and unambiguous requirements.

2. **Requirement generation**
   The user shall be able to generate requirements from boilerplates by providing required information. For each required field, the system will suggest appropriate keywords extracted from knowledge base of the system. This will allow more completed set of requirement to be generated.

3. **Pre-defined requirement types**

   The system shall provide both functional and non-functional predefined boilerplates for the user. Non-functional requirement boilerplate provided will developed based on quality characteristics of ISO 25010 Quality Model (International Organization For Standardization ISO 2011). Non-functional requirement boilerplates will support requirement engineer in the elicitation of non-functional requirement.

4. **Project Maintenance**

   The system shall allow user to save or load their software requirement specification projects from server. There are two types of projects that can be created by user, which are private projects that only editable by themselves and public projects that can be edited by any user.

5. **Export**

   The system shall be allow user to export all of their specified requirements as plain HTML document (.html) or Microsoft Word Document (.doc) file.

1.7.3. Modules Not Covered

Unless explicitly mentioned, the proposed tool will not cover any modules or functionality that are not mentioned in Section 1.7.2. The following are some key functionality that will not be covered in this proposed project:

1. The system will not provide traceability matrix, requirement prioritization and related functionality. The reason is because our proposed tool will only focus to support requirement elicitation and specification.

2. The system will only cover the elicitation and specification of system requirements. This is due to the limited amount of time available that is insufficient to apply boilerplate to all types of requirement that may be specified in a software requirement specification (SRS).

CHAPTER 2

LITERATURE REVIEW

2.1 Software Requirements

In this section, few definitions from different sources are presented and summarized. There are multiple definitions for a software requirements. The following are three main definitions referred:

1. IEEE-STD-1220-1998 (IEEE 1998) defined requirement as “a statement that identifies a product or process operational, functional, or design characteristic or constraint, which is unambiguous, testable or measurable, and necessary for product or process acceptability (by consumers or internal quality assurance guidelines)”

2. In SWEBOK v3.0 (Bourque & Fairley 2014), the author describes requirement as a property that must be exhibited by something in order to solve some problem in the real world.

3. In Software Engineering (Sommerville 2011), the author classified requirements into two main categories: (1) user requirements which are high level abstraction of expected service provided with constraints, (2) system requirements which are detailed description of system’s functionality, service, and operational constraint.

In this project, the definition from IEEE-STD-1200-1998 were chosen as main reference. The definition stated that requirements are statement which identifies **functional** and **design** characteristics. These inferred that in this project, both functional and non-functional requirements should be included. Then, a requirement should be **unambiguous**, which is one of the main problems in requirement we intended to tackle. Lastly, a requirement should be **measurable**, which also hinted that we should develop certain metrics to measure requirements.

The following subsections will clarify on what are functional and non-functional requirements.

2.1.1. Functional Requirement

Functional requirements are software requirements that describes functions, capabilities or features that the software is to execute. Functional requirements can be expressed in terms of steps or procedures that user can take to achieve result (Bourque & Fairley 2014).

(Sommerville 2011) defined functional requirements as "statements of services the system should provide, how system should react to particular input, and how system should behave in particular situations". There are also cases where functional requirement states what a system should not do instead.

(Dorfman & Thayer 1990) also have their own definition for functional requirement, which it is "a statement that identifies what a product or process must accomplish to produce required behaviour and/or results".

In this project, the definition from Sommerville will be referred. The information about functional requirement that we can extract from his definition is that functional statements should state what **should** and what **should not** be done by the system, and how should system **react to different situations**. Hence, in eliciting functional requirement, we should take into consideration of the above three aspects.

2.1.2. Non-Functional Requirement

(Sommerville 2011) also defined non-functional requirements as "constraints on services or functions provided by system, inclusive of timing constraint, development constraint and constraints imposed by standards". Non-functional requirement usually applies to the system as a whole rather than individually.

Non-functional requirements are also referred as quality requirements (Bourque & Fairley 2014). Non-functional requirement can be further classified as performance requirements, maintainability requirements, safety requirements, reliability requirements, security requirements, interoperability requirements, and etc.

In "Systems and software engineering – Vocabulary" (ISO/IEC & IEEE 2010), non-functional requirement was referred as "a software requirement that describes not what the software will do but how the software will do it".

From the definition of Sommerville, non-functional requirements could be **constraints** that imposed by **standards**. This hinted that quality models are good source of non-functional requirement. Hence in this project, we will be adopting ISO 25010 quality model into non-requirement elicitation process.

To go further into the aspect of software requirements, in the next section, the field of requirement engineering which concerns about the engineer practices for the whole life cycle of requirement will be discussed.

2.2 Requirement Engineering

2.2.1. Definition of Requirement Engineering

In “Classification of research efforts in requirements engineering” from (Zave 1995), the author presented a clear definition of requirement engineering – “Requirements engineering is the branch of software engineering concerned with the real-world goals for functions of and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families”.

In “Requirements engineering: A Roadmap” (Nuseibeh & Easterbrook 2000), the authors added their opinions in the above definition. They mentioned that requirement engineering is a motivation of developing a software system and represents reasons and what a system wants. They also added that in order to produce precise requirement specification, the basis of requirement analysis, requirement validation with stakeholder, design specification, and correctness verification are essential. Lastly, the authors also stated that there is a need of specification reusability in requirement engineering.

At the same time, the authors described requirement engineering as a process of discovering purpose and intention of stakeholders for the developed software system. Requirement engineering is a process that identifies the need of stakeholder and document them in the form which is “amendable to analysis, communication, and subsequent implementation”.

In “Requirement Engineering” (Hull et al. 2011), the author presented a clear definition where requirements engineering is “the subset of systems engineering concerned with discovering, developing, tracing, analysing, qualifying, communicating and managing requirements that define the system at successive levels of abstraction.”.

In essence, requirement engineering concerns about process and activities regarding to "requirements" in order to produce a precise SRS. Requirement engineering also looks into the area of "specification reusability". Requirement engineering is the field of knowledge which this project's goal aligned to – to reduce the problem in requirement elicitation and specification.

In our project, the idea of reusability is adopted in our deliverable, which we will strive to increase reusability of requirement by breaking down keywords of requirement and save them into database. This will allow the saved knowledge to be reused in future.

2.2.2. Requirement Engineering Activities

There are many models that could visualize the main activities in requirement engineering. Two of the main process model will be presented in this section.

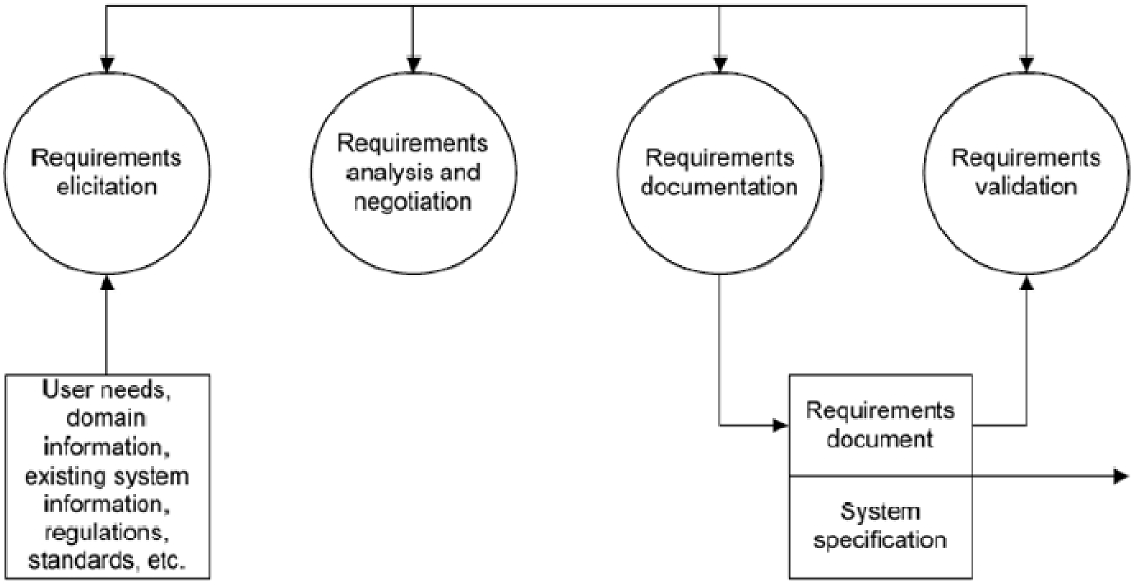

Requirements elicitation
Requirements analysis and negotiation
Requirements documentation
Requirements validation
User needs, domain information, existing system information, regulations, standards, etc.
Requirements document
System specification

Figure 2.1: Linear Requirement Engineering Process Model
(Kotonya & Sommerville 1998)

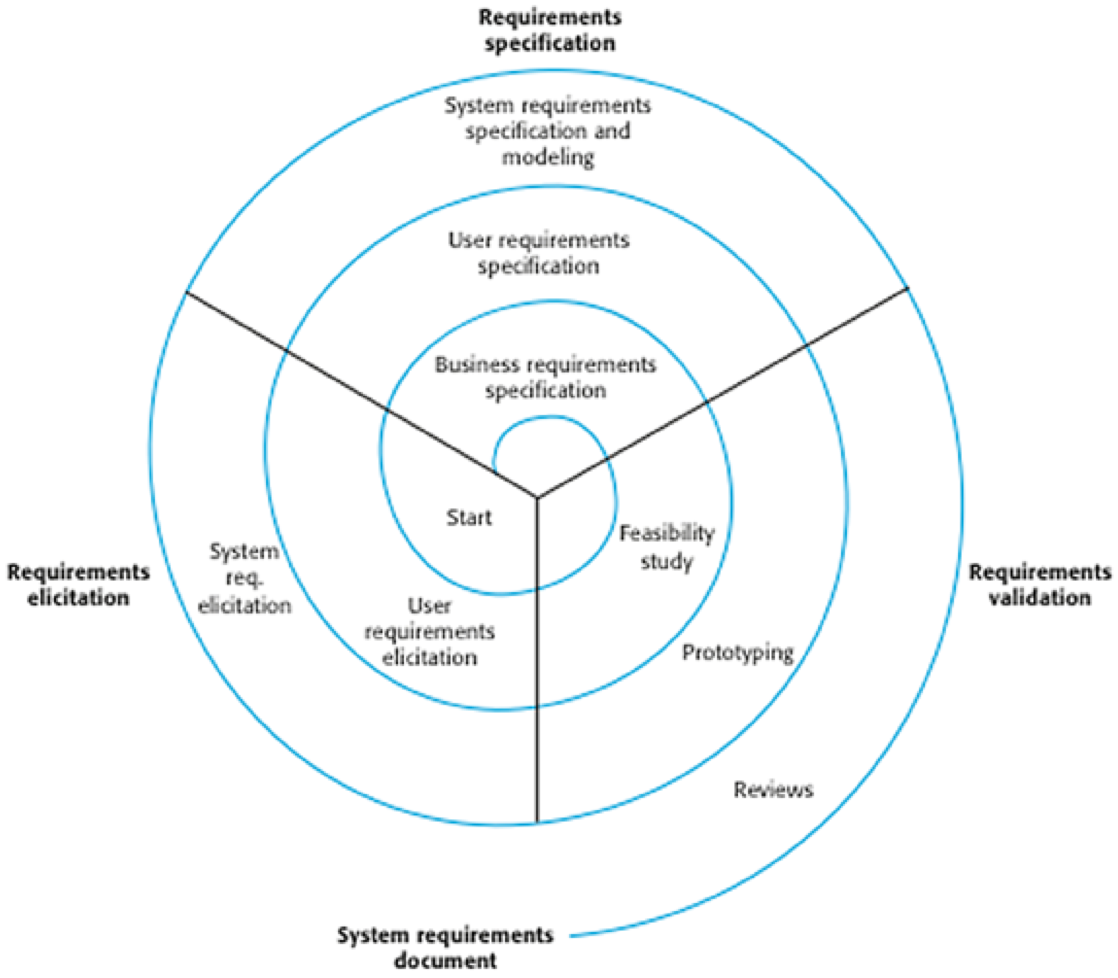

Requirements specification
System requirements specification and modeling
User requirements specification
Business requirements specification
Requirements elicitation
System req. elicitation
Start
User requirements elicitation
Feasibility study
Prototyping
Reviews
Requirements validation
System requirements document

Figure 2.2: Spiral Requirement Engineering Process Model (Sommerville 2011)

There are 4 major activities in requirement engineering, which are requirement elicitation, requirement specification, requirement validation, and requirement management.

Requirement elicitation is the phase where requirements are identified, listed and classified. The sources of requirements are mainly from the stakeholder of the system. There are mainly three classes of stakeholders, which are clients (who pay for the system), developer (who design, code and maintain), and end users (who uses system) (Nuseibeh & Easterbrook 2000).

There are many elicitation techniques that can be used. For instance, the most traditional methods are survey, questionnaires, interviews and analysis of existing documents. There are also other techniques such as group elicitation, prototyping, cognitive techniques and ethnography.

Requirement specification is the process of writing down user and system requirements in a requirements document (Sommerville 2011). There are few ways to write a requirement specification. The most frequent used methods are by using natural language sentences or structured natural language, which the latter ones refers to natural language in standard form or template. The others are design description language which uses language like programming language, graphical notation such as use case and sequence diagrams, and mathematical specifications such as notations.

Requirement validation is the process of checking and verifying requirements. The frequent used techniques includes requirement reviews, prototyping and test-case generation. Lastly, requirement management is the process that keep track of requirements and managing the changes to requirement, which is relatively important for large projects.

In the following section, the problems which lies in both requirement elicitation and requirement specification will be discussed.

2.2.3. Problems in Requirement Elicitation and Specification

The most common requirement gathering and elicitation technique is by interviews (Christel & Kang 1992). Interview are very useful to address organizational factors and contextual factors of a system. If done well, interview is very efficient as information gathered are representable as multiple stakeholder’s opinion, which saves time and money.

However, interview outcomes are lacking of organization and expression methods. There are no standardized procedures available to structure questions and collected data, lack of tool support, time consuming, and requires manual work. The requirement elicited are mainly dependent on requirement analyst who conduct and analyse the interview result. Integration of information from different sources with different interpretation and terminology is a very troublesome and error-prone work. Lastly, analyst also need to make decision whether a collected piece of information is a requirement or simply design information.

To resolve this, our proposed tool is a great add-on for requirement engineers to elicit and specify requirement when conducting an interview. This is because with help from the tool, they can directly record down requirements elicited or use the tool to guide their interview.

Another mainly used requirement elicitation technique is use case modelling. Use case modelling is one of the best approach to express functional requirements. However, there are criticism on the over emphasis of using use case modelling due to its simplicity. (Firesmith 2007). Lastly, use case modelling only emphasis on functional requirement and are not suitable for non-functional requirements.

To overcome this, our proposed tool will focus more on supporting the elicitation of non-functional requirement. This will make our proposed tool a good complement to use case modelling in requirement elicitation and specification.

In another report (Christel & Kang 1992), there are 3 major problems highlighted in the phase of requirement elicitation, which are:

1. **Problems of Scope**, where requirements are either overly or insufficiently addressed. For instance, design information should be added only if necessary. Ideally, requirement elicitation should begin by determining boundary and objectives of the system.

2. **Problem of Understanding**, which both users and analyst might not clear and understand about requirements and need for system. This is mainly due to difference in background, experience, language used, and messy information. The usage of natural language introduces ambiguity to requirement elicited, making it prone to misinterpretation and difficulty to understand.

3. **Problem of Volatility**, which is mainly due to the every changing needs of the user. Secondarily, it is caused by the revision of overemphasized requirements elicited in earlier stage.

The goal of this project is to reduce the above mentioned problems. For our proposed solution, we will be using requirement boilerplates, which is a type of structured natural language. In the next section, we will be discussing about natural language and problems in requirement specification.

2.3 Natural Language

2.3.1. Natural Language in Requirement Specification

Requirement specification can be done in 3 different formalities: (1) formal, such as notations, (2) semi-formal, which are graphical representations like use case modelling, and (3) informal, which is natural language that is mostly used.

Natural language is the native usage of communication language. In most of the time, natural language is used as medium of documentation. As most of the stakeholders are not from IT domain, natural language is frequently used by stakeholders to express their requirements. Although there are other methods of requirement specification, natural language does not affect professionalism and quality of requirement (Ibrahim et al. 2015).

From a survey conducted by Neill and Laplante (Neill & Laplante 2003), 51% of the respondents are using informal representation, which proofs that even the professionals in the industries are still using natural language despite many other methods available for requirement specification.

According to Pohl and Rupp (Pohl & Rupp 2015), there are three perspectives which a requirement can be specified: (1) Data perspective, referring to structure of input or output data and dependencies or system context, (2) Functional perspective, which process input data and output data to system context, and (3) Behavioural perspective, referring to states transitions and effect of system to its environment. And according to them, natural language is very suitable to document all of these three perspective.

Hence, it can be seen that natural language is the prominent method in requirement specification. However, there lies many problems in the usage of natural language, which will be discussed in the next section.

2.3.2. Problems of Natural Language in Requirement Specification

Requirements are usually expressed in natural language as it easily understood by different parties. However, the challenge in natural language in requirement engineering is to completely capture the need of stakeholder and express it unambiguously (Hull et al. 2011).

On the other hand, although being advantageous method in requirement specification, Pohl and Rupp (Pohl & Rupp 2015) also warned that natural language requirements of different types and perspectives could be easily mixed up during documentation. They also added that isolation of information according to perspective is also difficult although requirements are specified in natural language. In essence, natural language requirements can be ambiguous.

As a result of having ambiguous requirement, the issue of volatility of requirement will arise (Yang et al. 2011). In order to reduce the negative impact of using natural language, one of the method we can attempt to look into the usage of boilerplate, which will be discussed further in next section.

2.4 Requirement Boilerplate

2.4.1. Background

In order to reduce problems of natural language especially the ambiguity and inconsistency of requirement, we need to introduce certain structure which restricts the structure of requirement in order to improve the quality of it. In consequence, the idea of creating template of requirement is brought in and formed the natural language requirement boilerplate or simply known as requirement boilerplate.

The idea of boilerplate was first introduced in Requirement Engineering (Hull et al. 2011) in Section 4.8. They described boilerplate as a language of requirement which comes in a format of sentence, but with angle bracket surrounded placeholders. They stated that using boilerplate is a good way to standardize language used for requirement and boilerplate could be collected and reused from project to project. Other than that, they also added that using boilerplate has three main advantages:

1. Allow global change in style of requirement which means by changing boilerplate solely, all requirement based on the boilerplate will be able to be updated to latest format.
2. Allow system information to be processed easily by extracting information from placeholders
3. Protecting confidential information by filtering out confidential information based on placeholders.

Boilerplate is considered a type of structured natural language and semi-formal representation of requirement. Hence, boilerplate is capable of increasing the quality of requirements by using simple sentence structure which reduces the ambiguity of requirement and expressing requirements in consistent manner (Arora et al. 2014).

Boilerplate appears to be solution to the problems incurred due to usage of natural language in requirement specification. Boilerplate acts as a template to express requirement, which makes them consistent. It limits the structure of the sentence, giving requirement a simple yet descriptive expression. Requirement's ambiguity can be avoided as each of the elements are structured accordingly to the template, making it impossible to misinterpret the original meaning implied.

In the next section, we will look into practical perspective of requirement boilerplate and how it could be used to standardize requirements.

2.4.2. Usage of Requirement Boilerplate

Requirement boilerplate are like normal sentences but consisting of placeholders that are wrapped with angle brackets ('<' and '>'). These placeholders can be replaced with other words to become a requirement. It works like a mound or template for sentences.

For example, given a requirement boilerplate as below:

The *<actor>* ***shall be able to*** *<action><target>*

By filling the placeholder of <*actor*>, <*action*> and <*target*>, different requirements can be generated.

For instance, the following requirement are generated from above boilerplate:

The *<user>* ***shall be able to*** *<save><document>*
The *<firewall system>* ***shall be able to*** *<detect><intruder>*
The *<student>* ***shall be able to*** *<register><subject>*

In another case, boilerplate can also be used to express existing requirement in consistent manner, given that the correct type of boilerplate are chosen.

For example, given requirements as below:

User can login

User will need to register an account

If the password is correct, user can login to the system

And some requirement boilerplates as below:

The *<actor>* ***shall be able to*** *<action>*

The *<actor>* ***shall be able to*** *<action><target>*

The *<actor>* ***shall be able to*** *<action><target>****given that*** *<condition>*

The usage of boilerplate can formalize and express the above requirements in a structured and consistent manner:

The *<user>* ***shall be able to*** *<login>*

The *<user>* ***shall be able to*** *<register><account>*

The *<user>* ***shall be able to*** *<login><to the system>****given that*** *<password is correct>*

In our proposed tool, pre-defined requirement boilerplates will be provided for user to perform the actions as shown above in order to specify requirements in a consistent manner. In the next section, we would also like to share some related works that had been done by others using boilerplate.

2.5 Software Quality Models

In our tool, we will be using a software quality model as reference to design and develop non-functional requirement specification modules and boilerplates. In order to do so, we had done some research and review on existing software quality models. In the following sections, the background of quality model and examples will be shown.

2.5.1. Background

Software quality model was defined as “a set of characteristics and relationships between them, which provides a framework for specifying quality requirement and evaluating quality” in (International Organization For Standardization ISO 2011). Quality model usually consists of few quality characteristic, which each of them may be refined into multiple levels of sub-characteristics (ISO/IEC & IEEE 2010). For each sub-characteristics, quality metrics may be assigned to evaluate and measure the quality requirement.

According to (Miguel et al. 2014), software quality models are acceptable methodology that can be used to support the quality management of a software product. This leads to the question whether which quality model should we choose and use. From a research done by (Thapar et al. 2012), they studied 24 quality models and categorized quality models into two types: (1) Basic quality models, which produced from research in the direction of quality improvement and software evaluation, (2) Tailored quality models, which are improved forms of basic quality models as result of adjustment to the needs of underlying application domain.

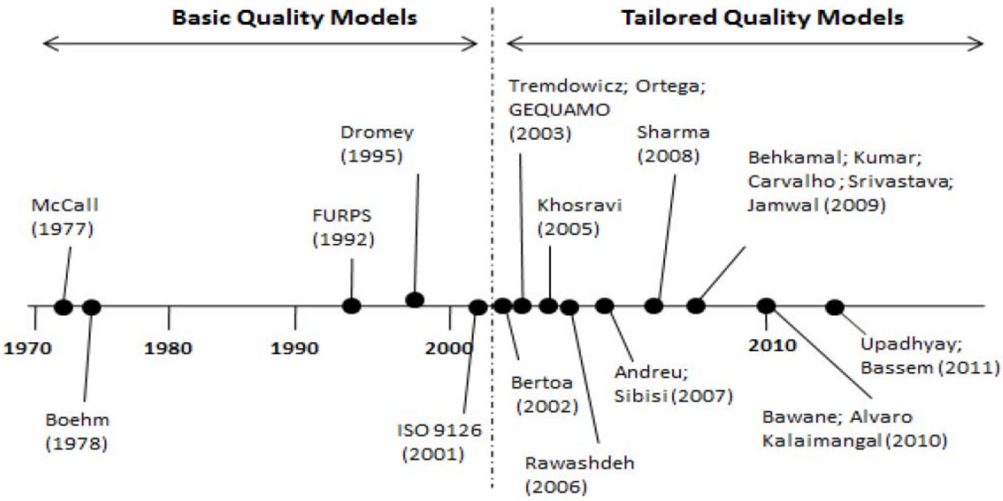


Figure 2.3: Quality models developed before 2011 (Thapar et al. 2012)

The list of major quality models that was introduced before 2011 was shown in Figure 2.3. Among all of these quality models, there are 3 quality models will be discussed in the next section, namely McCall's, Boehm's and ISO 9126 Quality Model.

2.5.2. Basic Quality Model Reviews

To compare between quality models, 3 main quality models will be reviewed and discussed in this section. The reason that these 3 quality models was chosen is because: (1) Both McCall's and Boehm's quality model was the earliest widely recognized quality mode, (2) ISO quality model is latest basic quality models and also recognized globally, (3) There are good comparison that can be made between these models.

First of all, McCall's quality model was introduced earliest back in 1977. McCall identified 3 main perspective to characterize the quality attributes of a software product (McCall et al. 1977), which are: (1) Product revision which based on factor of maintainability, flexibility and testability, (2) Product transition which based on factor of portability, reusability and interoperability, (3) Product operations, which based on correctness, reliability, efficiency, integrity and usability.

In addition, McCall also introduced metrics by measuring quality subjectively. He used the format of yes/no, 1/0 or range of values to consider whether a quality factor is present. McCall covered both viewpoints of developer and user to bridge the gap between them.

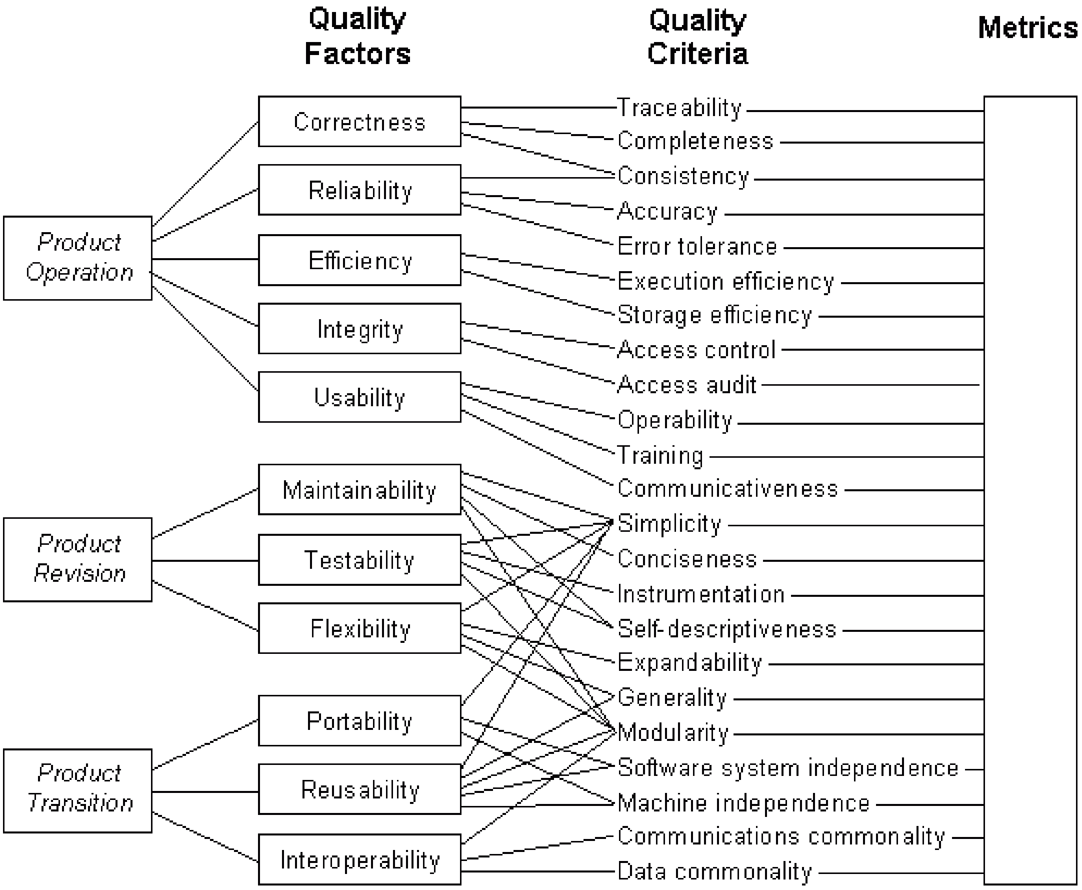


Figure 2.4: McCall's Quality Model

On the other hand, Boehm had introduced a quality model to evaluate quality of software in 1978. As compared to subjective measurement introduced by McCall, Boehm preferred quantitative measurements. Boehm's quality model was based on 3 primary uses at top hierarchy, which are (1) As-is utility, (2) Maintainability, (3) Portability. At the next level, Boehm identified 7 quality factors which are (1) Portability, (2) Reliability, (3) Efficiency, (4) Usability, (5) Testability, (6) Understandability, and (7) Flexibility (Please refer to Figure 2.5).

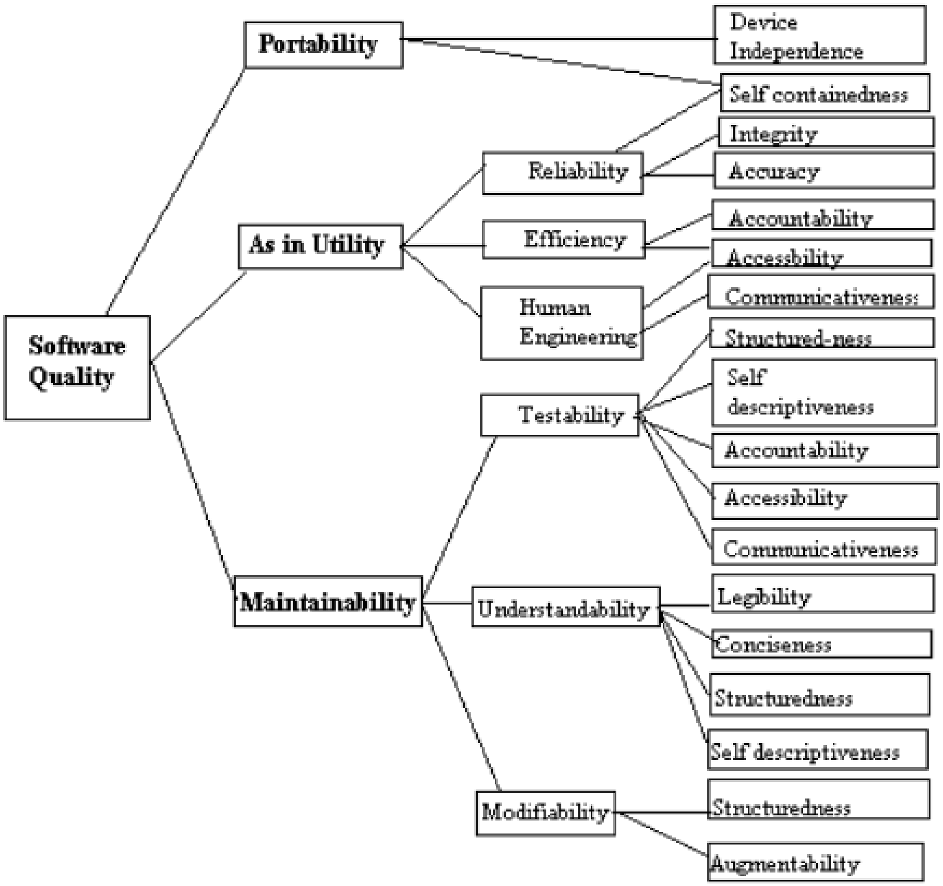


Figure 2.5: Boehm's Quality Model

In 2001, ISO 9126 was introduced to standardize the evaluation of software quality (International Organization For Standardization Iso 2001). The standard address 4 subjects of software quality, which are (1) Quality model, (2) External metrics, (3) Internal metrics and (4) Quality in use metrics. ISO 9126 Part One (ISO 9126-1) extends work done by McCall, Boehm and others in defining quality characteristics.

ISO 9126 focuses on 6 main quality characteristics, which are (1) Functionality, (2) Reliability, (3) Usability, (4) Efficiency, (5) Maintainability, and (6) Portability. Each of these main quality characteristics are further elaborated as sub-characteristics (Please refer to Figure 2.6).

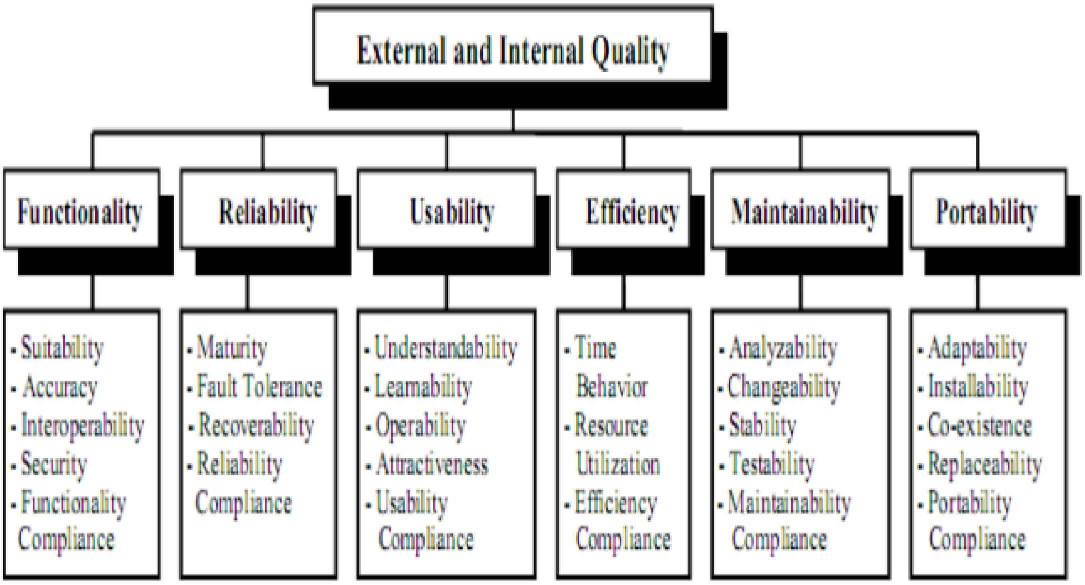


Figure 2.6: ISO 9126 Quality Model

As a comparison, it can be noticed that all of the above 3 quality models has similar quality characteristics or sub-characteristics, which mainly includes the aspect of portability, reliability, efficiency, maintainability and testability. McCall's and ISO model are similar in terms of their coverage as compared to Boehm's model. In term of structure, McCall's model grouped their main characteristics into 3 different group, which is very good to distinguish whether a requirement is related to operation of the product, the review of the product or the transition of product from one release to another.

However, ISO model is well-structured as it grouped quality characteristics based on their focus aspects and has clear distinguish between each quality characteristics. McCall's quality factor was inclusive of many criteria which has no clear boundary make it harder to group requirements.

Furthermore, ISO 9126 was compiled at later time than McCall's Quality Model (1978 vs 2001). This may hint that some elements in McCall's quality model may be outdated, and that ISO may introduced some important characteristics which overlooked by McCall's model. For example, security aspect was introduced in ISO model but wasn't mentioned in McCall's model. This may be related to the fact that everyone is now connected to Internet, as compared to the time when McCall's was introduced, Internet is non-existence. This makes ISO model more suitable to be used than McCall's model.

Later in 2011, ISO 9126 was superseded by a refined version of ISO quality model, which is ISO 25010 that will be further discussed in the next section.

2.5.3. ISO 25010 Quality Model

In a rapid changing environment like IT domain, the wants of user are changing from time to time. Hence, ISO replaced their ISO 9126 model to become ISO 25010 Quality Model which is more extensive. Compared to ISO 9126, ISO 25010 was developed as a part of SQuaRE (Software Product Quality Requirement and Evaluation) ISO standards. The purpose of SQuaRE is to assist in developing and acquiring software products with specification of quality requirements and evaluation.

The quality characteristics of ISO 25010 are shown in Figure 2.7. As compared to ISO 9126, ISO 25010 are more complete. Changes from ISO 9126 to ISO 25010 includes the more emphasizing of "Security" and "Compatibility" aspect where it became new main quality characteristics. The other changes includes renaming certain characteristics to make the term more accurate, such as "Functionality" to "Functional Suitability" and "Efficiency" to "Performance Efficiency".

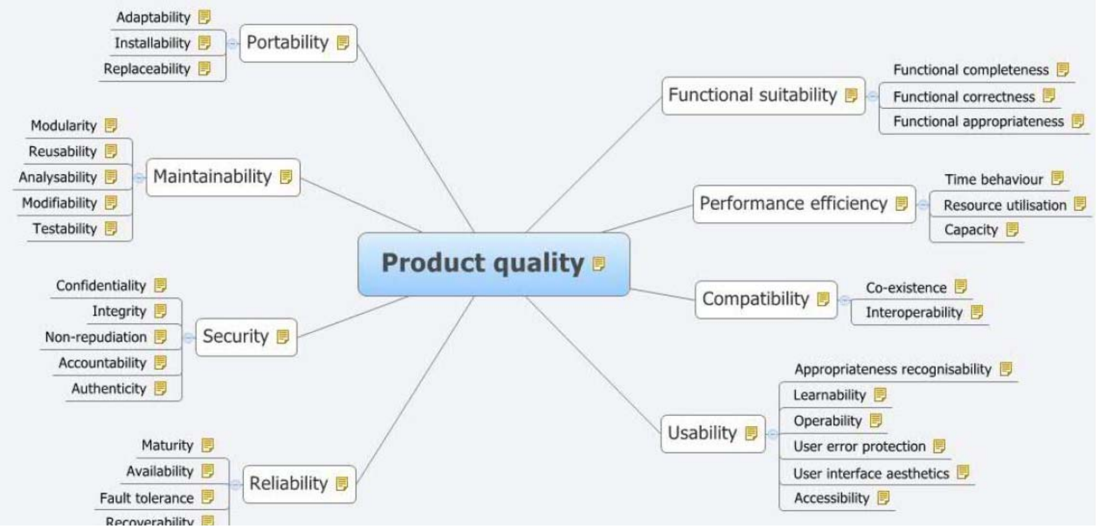


Figure 2.7: ISO 25010 Quality Model

As a summary, ISO 25010 improved ISO 9126 model to tally with the move of trend in industry. In our project, we would refer to ISO 25010 as guideline for non-functional requirement specification.

2.6 Similar Tool Review

In order to compare our proposed tool with other existing tool, we also reviewed and summarized a few existing requirement engineering field related tools. The discussion and comparison are mainly focused on the aspect of requirement elicitation or specification but not requirement prioritization and management due to the scope of our project. However, due to the fact that majority of tool that supports requirement specification are requirement management tools, we cannot avoid the comparison between requirement management tools.

In the following subsections, 3 different tools will be presented, discussed and compared along with our proposed tool.

2.6.1. Enterprise Architect

Enterprise Architect (EA) is an UML modelling tool first released by Sparx Systems in 2000. Despite supporting UML modelling, EA also included some requirement specification and requirement management features (Sparx Systems 2010).

EA allows user to specify both functional and non-functional requirements. These requirements are added manually and user may also specify the status (whether requirements is at proposal stage or implemented), difficulty and priority. The user also may import requirements from CSV file. In addition, user may declare certain terms with their definitions or descriptions in glossary which can be cross referenced within the project.

The main advantages of using EA is the completeness of design models that user can create and the capability of linking between requirement and design models, which allows user to view each related items for a specific requirement. However, EA do have a learning curve where new users will easily get overloaded with significant numbers of functionalities offered.

2.6.2. IBM Rational DOORS

IBM Rational DOORS (Dynamic Object Oriented Requirement Management System) (will be referenced as DOORS in the following) is a requirement management tool offered by IBM. DOORS was first released by Quality Systems and Software (QSS), which then bought over by Telelogic in 2000. Later, Telelogic was acquired by IBM and development of DOORS was continued by IBM Rational in 2008.

DOORS supports importing or exporting between lists of most frequently used software such as Microsoft Word, Microsoft Excel, Microsoft Project and Adobe FrameMaker. DOORS also stores documents in an internal database environment and provide traceability for every changes. DOORS allow tracing from initial requirement till detailed requirements, then to design and test cases. Other than that, DOORS also allows linking between documents and baselining (store current state of document). Last but not least, DOORS allow viewing, filtering, searching and sorting on documents.

DOORS is a multi-user, version controlled, and highly traceable requirement management tool. DOORS requires manual work to specify requirements or can simply import existing requirement from documents and saved into DOORS' database system. The integrated document system provided by DOORS ensures all files are documented, versioned and all changes were traced.

2.6.3. ElicitO Framework

ElicitO is a quality ontology driven non-functional requirement elicitation tool created by (Hazeem et al. 2007) from University of Manchester. ElicitO uses functional ontology as domain model and quality ontology derived from quality models to support the requirement elicitation process (Al Balushi et al. 2013).

ElicitO uses database to store sessions and requirements specified in each sessions. Requirements are added by (1) Selecting a functionality defined in functional ontology, (2) Selecting a quality metrics defined in quality ontology, and (3) Specifying the measurement and value of the metric. For example, user may select "Frequently Asked Question (FAQ)" as functionality, then select "Page download speed" as quality metric, then specifies "15 seconds" as measurement value. This indicates a non-functional requirement which requires "page download speed" of "FAQ" to be "less than 15 seconds".

ElicitO also comes with feature to identify conflicting requirements based on relation defined in ontology and allows discussion on the conflict. All requirements are stored in tabular format in the database and information can be easily extracted from this format. ElicitO provides non-functional requirement elicitation which is quite lacking in many others tools, as well as requirement prioritization based on discussions. The side product of ElicitO, which is functional ontology and non-functional ontology can be reused in other tools, which promotes reusability of knowledge.

2.6.4. Comparison and Discussion

After reviewing these 3 tools, we made a simple comparison between them as well as our proposed tool (please refer to Table 2.1).

Table 2.1 Comparison of Tools

Aspect	Tool			
	EA	**DOORS**	**ElicitO**	**SrsTool**
Supports requirement elicitation (FR)	No	No	Yes (Functional ontology)	Yes (Domain model)
Supports requirement elicitation (NFR)	No	No	Yes (Quality ontology)	Yes (Quality model)
Supports requirement specification (FR)	Yes	Yes	Yes	Yes
Supports requirement specification (NFR)	Yes	Yes	Yes	Yes
Requirement specification approach	Natural language	Natural language	Tabular	Boilerplate
Supports requirement prioritization	Yes	Yes	Yes	No
Supports requirement management	Yes	Yes	No	No
Data storage type	File based	Database	Database	Database
Collaborative	No	Yes	No	Yes
Web based accessibility	No	Yes	No	Yes

Import requirements	Yes	Yes	No	Yes
Export requirements	Yes	Yes	No	Yes
Special feature	UML	Traceability	Ontology	Boilerplate

From the above comparison, we could easily noticed that almost all requirement engineering tools supports requirement specification but in different approach. Requirement management focused tool such as EA and DOORS do not support requirement elicitation and mainly uses natural language for requirement specification. However, they provided complete features to prioritize and manage requirements as well as good traceability for requirements.

In contrast, ElicitO focused more on requirement elicitation and specification using ontology and even offered prioritization using discussion approach. ElicitO also allows user to identify possibly conflicting requirements based on relation defined in ontology. ElicitO constraints all requirements must be based on defined functional and quality ontologies, which produces correct and complete requirement if their ontologies were validated.

Meanwhile, our tool (SrsTool) focused to implement boilerplate in requirement elicitation and specification phases. Our tool supports both functional and non-functional requirement specification but do not support prioritization and management of requirements.

Since requirement phases are more likely to be handle by more than solely a requirement engineer, the aspect of web based accessibility and collaborative features were also looked into comparison. As a result, we noticed that EA and ElicitO is less appealing than DOORS and our tool in this aspect. EA requires user to share project file, while ElicitO depends on the setup of database, whether it is local or web based database server. Other than that, features to import or export requirements is almost a must for a requirement engineering tool.

In summary, our tool being a web application elevated the collaborative and web based accessibility aspect of the tool. Our tool allows user to export requirements which is an added advantage for our tool. As future improvement, our tool may opt to include requirement prioritization or management functionalities as how other tools provided.

2.7 Project Approach Review

2.7.1. Rational Unified Process

To identify which software process model to be implemented in this project, we made a brief comparison between the traditional waterfall and agile development methodology. The following table will summarized some criteria of both methodology.

Table 2.2 Comparison of Waterfall and Agile development model

Waterfall Model	Agile Model
Linear/sequential flow, where there is no return to previous phase	**Iterative**, where it will go back to previous phase every iteration
One shot, which product are delivered directly as a whole	**Incremental**, which product features are delivered module by module
Poor visibility, as product is only visible at end of development	**Good visibility**, as prototype are visible at early stage of development
High risk, as only at the end of testing phase problems are surfaced	**Lower risk**, as during each iteration problems are found
Well documented and recorded	Dependent on type, mostly **less documented**
High cost of requirement **changing**	**Lower cost** of requirement **changing** but requires requirement management
Suitable for **complex** and **reliable** system such as embedded system and banking system	Suitable for **light** and **fast changing requirement** project such as web application and mobile application

As our project is a relative small project and prone to requirement changes, it is best to employ an agile model that is iterative. This will allow more room for requirement changes and allow early prototype to recognize problems. In our project, we chose to employ RUP as reference for the flow of our software process.

Rational Unified Process (RUP) is a software process introduced by Philip Kruchten (Kruchten 2004). RUP is an iterative software development process that derived from Unified Process (UP), where UP itself is derived from the usage of Unified Modelling Language (UML). RUP attempts to employ best features and characteristics of traditional waterfall mode and implement them in an iterative and incremental approach (Pressman 2009).

In RUP, there are three perspective views (Sommerville 2011), which are **dynamic** perspective which shows phases of model, **static** perspective which shows process activities, and **practice** perspective which suggests best practises.

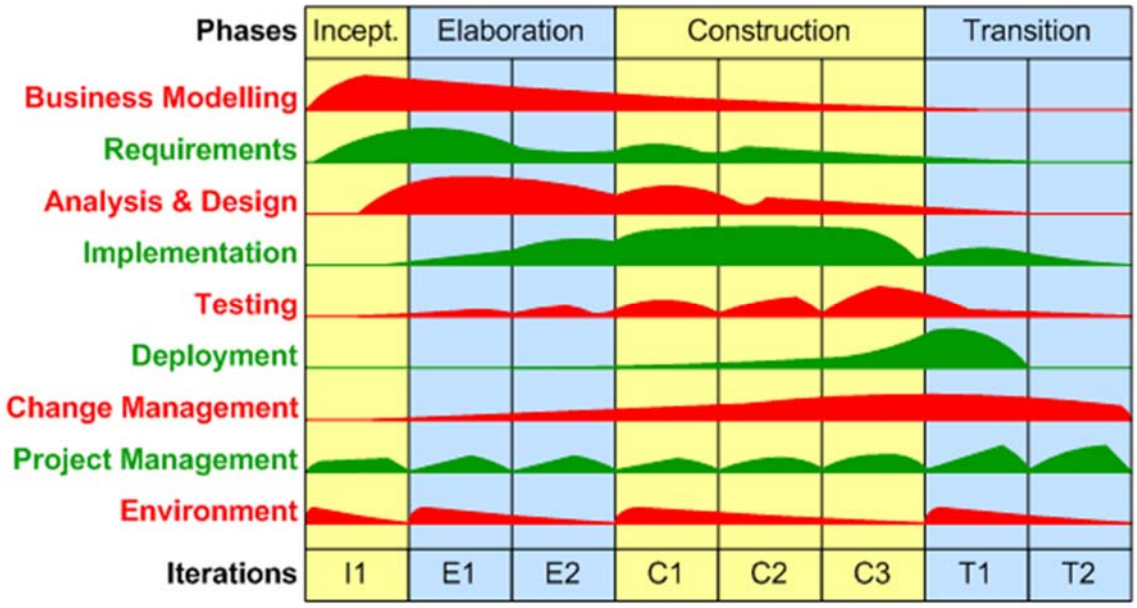


Figure 2.8: Dynamic and static perspective of RUP

In Figure 2.8., both dynamic and static perspective was presented together. The **phases** (inception, elaboration, construction and transition) refers to the **dynamic** perspective, while the **static** perspective refers to activities or **workflows** in RUP (business modelling, requirement, analysis and design, implementation, testing, deployment, change management, project management, environment). All phases are iterative and not bind to all workflows in RUP, which makes each workflow iterative in nature and thus allow changes and flow back to previous phases.

Lastly, the practice perspective of RUP introduces 6 best practises in software development:

1. **Develop software iteratively** by incrementally delivering software components
2. **Manage requirements** to keep track of changes and improve traceability
3. **Use component-based architecture** to structure the system
4. **Visually model software** by using UML models
5. **Continuously verify software quality** to reduce bug and risk
6. **Control changes to software** using change management system and configuration management tool

2.7.2. MongoDB NoSQL Database

In considering the type of database we will be using in this project, there are mainly two types of database we can use:

1. SQL database, which mainly stores normalized data in rows and columns called tables
2. NoSQL database, which supports storing de-normalized data in form of key-value pairs, documents, and many other forms

In our project, the form of data we would be using are JavaScript Objects (as we are using JavaScript mainly). Considering this aspect, using SQL database would be tedious as we need to normalize data into their own tables. In contrast, NoSQL database format such as document oriented database could easily store the JavaScript Object directly into database.

MongoDB is a free, open source, cross platform NoSQL database. MongoDB uses JSON-like document which is called BSON and it is a document oriented database. MongoDB is the most widely supported NoSQL database as compared to other NoSQL databases. MongoDB is also well known as core component for the trending MEAN (MongoDB, ExpressJS, AngularJS, NodeJS) website and web application development stack.

2.7.3. ExpressJS and NodeJS RESTful API

Since we are developing a website, we need to consider both client (frontend) and server (backend) side of the website. For the development of backend, we had looked into either using PHP frameworks or JavaScript based on technical skills available.

When deciding on the software architecture of our backend side, we decided to develop a **RE**presentation **S**tate **T**ransfer (REST) based web service which also known as RESTful service. RESTful service utilizes Hypertext Transfer Protocol (HTTP) headers and verbs such as GET, POST, PUT and DELETE to represent the "state" of the HTTP packet. RESTful service provides a simple and uniform interface for web clients to consume. In considering our server side language to be used, we analyse mainly on the speed and effort required for the particular language to develop a RESTful web service.

Excluding ASP.NET and other language which we are not proficient in, we left with PHP and JavaScript (NodeJS). PHP is a server side object-oriented scripting language. To develop a RESTful server, we had looked into few PHP frameworks which could support easier development of RESTful API such as Yii2 and Slim.

On the other hand, NodeJS is a JavaScript runtime environment that implements an event-driven architecture. NodeJS supports concurrency better than PHP as it runs by looping cycles to wait and handle requests by user. NodeJS also comes with a package manager, which is called NPM (Node Package Manager). NPM allows user to automatically install dependencies by specifying them, and NPM registry hosts a lot of useful JavaScript packages shared by other programmer.

After our considerations, we decided to go with NodeJS which has better community support, documentation and had a lot of community developed packages. ExpressJS is one of the most popular packages that hosted in NPM, which frequently used to build RESTful API. ExpressJS provides a framework for our NodeJS server side to serve the RESTful API for user easily. The combination of ExpressJS and NodeJS allows us to do backend development quickly with some help by using packages provided in NPM.

2.7.4. AngularJS

On the frontend side, there are far more choices that we have. With the technical skills we have, we left to pick between two JavaScript frameworks, which is jQuery and AngularJS. AngularJS is an open source JavaScript framework for web application development. AngularJS is powerful for single page app development, which refers to web application that provides similar experience as desktop app. Since our tool is a web application rather than a website, AngularJS is very suitable to be used for frontend development of our web application.

In addition, AngularJS do not have conflict with jQuery library, which means that it could be used together with jQuery. AngularJS supports client-side routing and two-way binding with HTML components, which could swift up the development of our website. Other than that, AngularJS is also built with Model-View-Controller architecture. This allow separation of business logic and user interface, such that we can easily change to frontend views without altering the logic of the website. Hence, AngularJS was chosen to be build the frontend of our website.

CHAPTER 3

METHODOLOGY

3.1. Chosen Development Methodology

With references from Section 2.7.1, the development methodology we will be implementing is the rational unified process (RUP). Figure 3.1 shows a basic iterative lifecycle of RUP process.

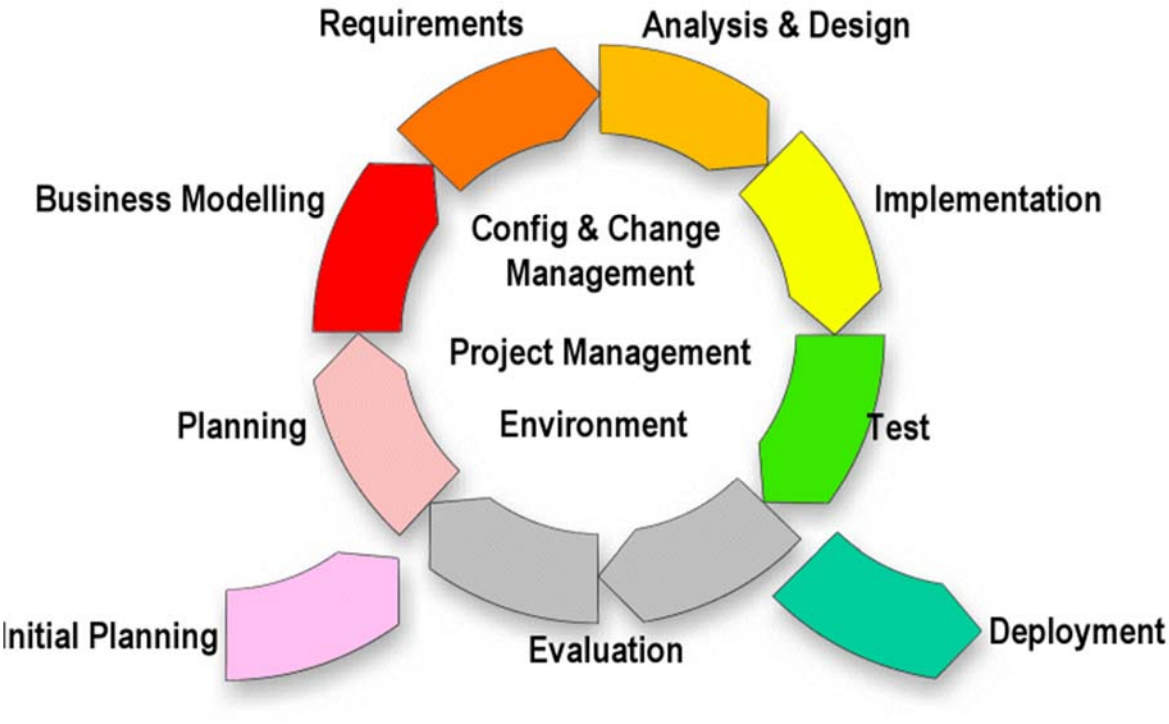


Figure 3.1: Rational Unified Process Cycle

We chose RUP because the concept and workflow of RUP are tally with our project. For instance, we will be implementing 8 requirement specification modules. As such, the implementation process of the modules are likely to be iterative as each module can be considered an iteration from planning, modelling till implementation.

Based on RUP's model (Please refer to Figure 2.8 in Section 2.7.1), the implementation of dynamic perspective of RUP in our project is as following:

Table 3.1 Dynamic Perspective – Project Phases

Phase	Main Activities
Inception	- Project proposal - Literature review - Project methodology review and proposal - Requirement specification - Project plan
Elaboration	- Project methodology finalization - Software architecture design - Software component design - Database design - RESTful route design - Activity diagrams - Sequence diagrams
Construction	- Product development - REST API development - Version control management - Deployment - User acceptance test
Transition	- User feedback survey - Report finalization

As of the practice perspective of RUP, we will implement most of the best practices emphasized by the model (Please refer to Table 3.2).

Table 3.2 Practice Perspective – Best practices

#	Practice	Implementation
1	Develop software iteratively	Yes. We will implement the software using 3 iterations which will be described later.
2	Manage requirements	No. Because the requirement is very unlikely to change.
3	Use component-based architecture	Yes. We will design the software using component based client-server architecture
4	Visually model software	Yes. We will model our software using use case models, activity diagrams and sequence diagrams
5	Continuously verify software quality	Yes. We will test our software with user periodically throughout the development and get user to give feedback to our tool
6	Control changes to software	Yes. We will be using GitHub to host our source code repository so that we can revert any unwanted modification at any time and keep track of our changes.

In our case, we propose to perform 3 iterations of implementation according to the process cycle presented in Figure 3.1. Each of iterations will perform the following implementations:

Iteration 1: User Interface Constructs and Backend Setup

1. Convert User Interface Design into HTML and CSS
2. Setup backend RESTful API and convert document schema into Mongoose schema
3. Link up backend and frontend so that they can communicate via HTTP
4. Setup basic routes for each UI

Iteration 2: Requirement Specification with Boilerplate

1. Implement each requirement specification modules (total of 8 modules)
2. Implement the usage of boilerplate to generate requirement
3. Implement the functionality of save and load project requirements
4. Continuous development of backend REST API

Iteration 3: User Authentication and Feature development

1. Implement the functionality of modifying boilerplates
2. Implement user authentication modules such as Facebook Login
3. Implement the export feature of the tool to export requirements
4. Clean up code

3.2. Chosen Development Tools

3.2.1. JetBrains WebStorm IDE

JetBrains WebStorm is a smart IDE which provides functionality such as code completion and able to interpret and link up between HTML and JavaScript. WebStorm also allows installation of plugins, such as NodeJS and AngularJS plugin which supports the development of our website.

3.2.2. NodeJS packages

To speed up our development, we will be integrating few packages from NPM so that we do not need to code for those packages which already created by other user. The following are main packages will be used in our project:

1. **ExpressJS**, a package which configured to route user requests according to HTTP request method and URL. This package will be used to build RESTful API on our server side.

2. **UnderscoreJS**, a package which defined many functional programming methods such as array manipulation methods, map-reduce methods and etc. This package will be used to assist both client and server side.

3. **Mongoose**, a package which allows MongoDB schema definition, validation and query. This package will be used to defined the database schema and perform database related operations on server side.

4. **Browserify**, a package which allows NodeJS server side code (backend) to be compiled into browser code (frontend). Browserify allows us to do frontend code with NodeJS and AngularJS before compiling them into one final file in frontend.

5. **Passport**, a package which automatically serialize or deserialize user from cookie data to store the user ID that indicates whether user is logged in. Passport will be used with Facebook-passport plugin to support OAuth 2.0 provided by Facebook to log in to the system.

3.2.3. CSS framework

In order to develop our user interface, we decided to use CSS framework to speed up and make development easier. We started with Bootstrap CSS for early stage of development. Bootstrap CSS provides skeleton CSS of user interface and uses grid system to build user interface. This allows user to build responsive websites easily.

However, after considering that Bootstrap CSS are quite boring and not aesthetic, we decided to change to Materialize CSS. Materialize CSS is a CSS theme developed based on material design concept. Materialize CSS also uses grid system and comes with even more functionality such as light box to show images and accordion which act like expandable list.

Since AngularJS is MVC architecture based, we moved our user interface design from Bootstrap to Materialize without changing any logic in the code, solely changes on HTML and CSS.

3.3. Requirement Specification Strategy

In order to capture requirements from our user, we need to provide a platform for user to state what they want. In contrast to normal approach of asking user to pick requirement boilerplate and fill in the details, we are attempting another approach: filling forms.

To do so, we first try to list down possible requirements that user may specify. This process is guided by ISO 25010 Quality Model where we attempt to group requirements based on each quality characteristic. Then, we extract the requirement boilerplate from the requirement by looking at the common sentence structure.

For example, for "Compatibility" quality characteristic, we may want to specify the following requirements:

1. The system shall be able to run in <Microsoft Windows> <7> and above with <some unsupported colour scheme>
2. The system shall be able to run in <Microsoft Windows> <8.1> and above with no compatibility issue
3. The system shall be able to run in <Linux Ubuntu > <12.0> and above with no compatibility issue

4. The system shall be able to run in <Apple MacOS > <10> and above with no compatibility issue

From the above requirements, we can observe a similar sentence structure and extract a requirement boilerplate like the following:

1. The system shall be able to run in <Operating System> <Version> and above with <Compatibility Issue>
2. The system shall be able to run in <Operating System> <Version> and above with no compatibility issue

Based on above approach, we identified the requirement boilerplate for compatibility of operating system. The next step is to create a form to request user to fill in values for <Operating System>, <Version> and <Compatibility Issue>. By using this method, we can let user have more interactive usage with the tool.

Later in Section 6.3 Requirement Specification Strategy Implementation, the real implementation based on this strategy will be shown.

3.4. User Interface Design

In order to implement our tool, we sketched the user interface design using Draw.IO tool before mapping it into real website.

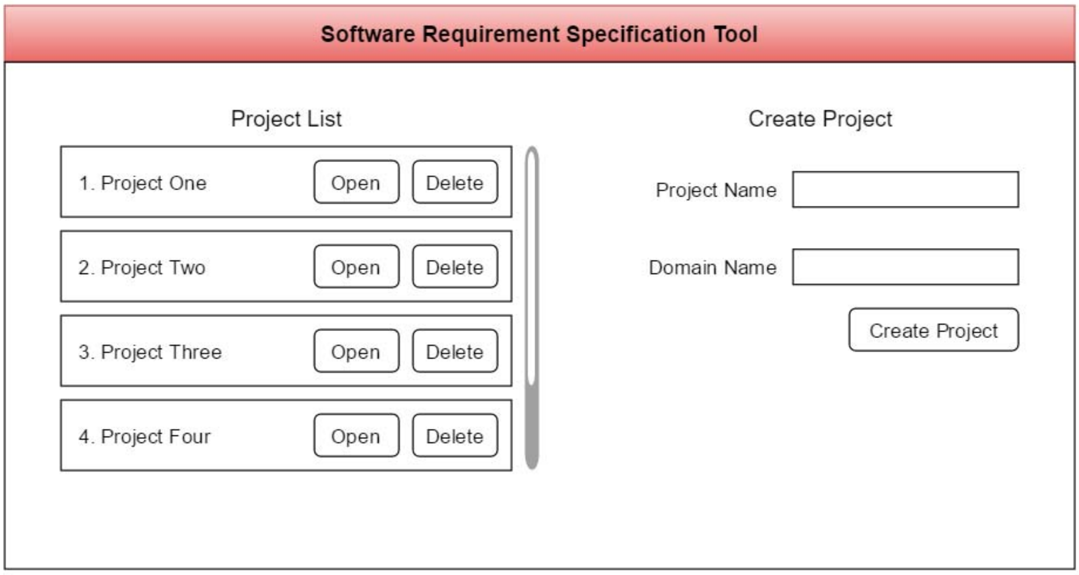


Figure 3.2: UI – Show all project and create project

Figure 3.2 shows the UI where all projects of the user are shown. User may also create new projects in this UI. After creating project, user may open the project to specify requirement or delete the project if no longer needed.

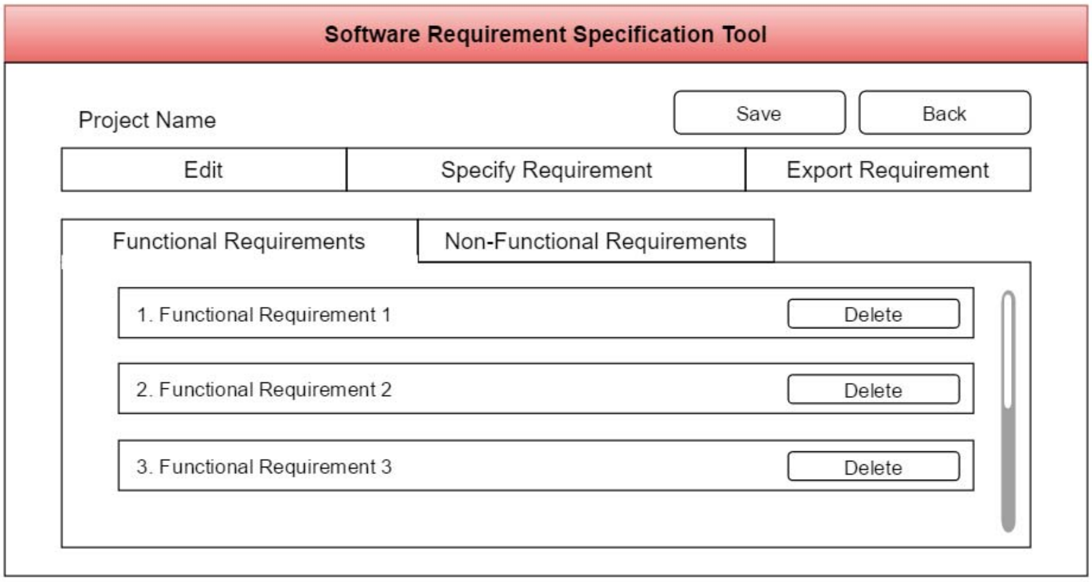
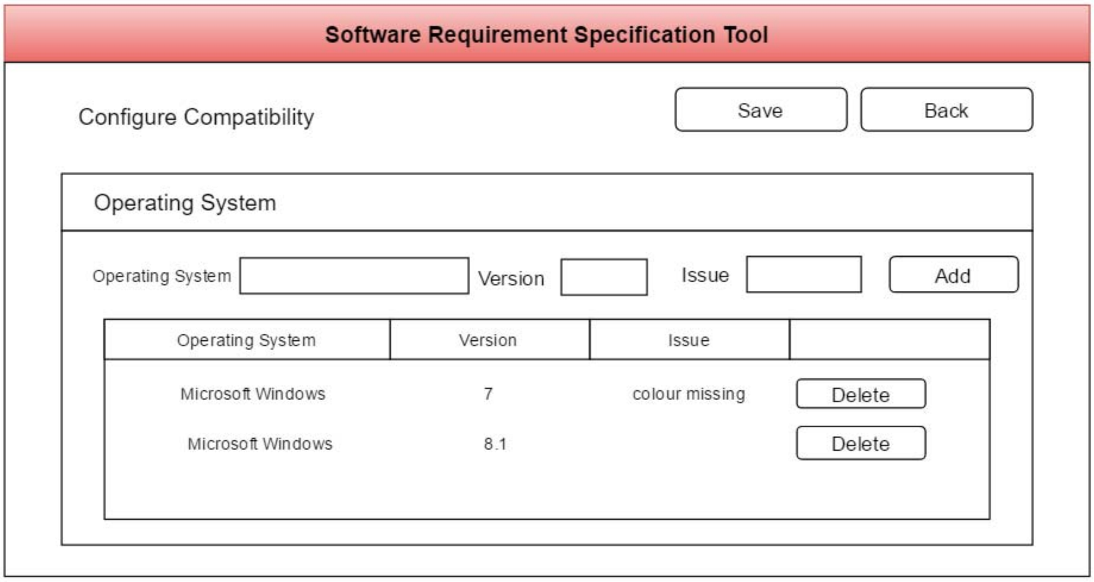

Software Requirement Specification Tool
Project Name
Save
Back
Edit
Specify Requirement
Export Requirement
Functional Requirements
Non-Functional Requirements
1. Functional Requirement 1
Delete
2. Functional Requirement 2
Delete
3. Functional Requirement 3
Delete

Figure 3.3: UI – Show project requirements

Figure 3.3 shows the user interface where all the specified requirements of the project are listed. User may delete any specified requirement and save them to server.

Other than that, user may click on “Edit” button which they can choose to “Edit Project”, “Edit Domain” or “Edit Boilerplate”. These modules will allow user to modify project related data. User may also want to click on “Specify Requirement” which allows them to specify functional requirements or non-functional requirements, and generate requirements accordingly. Lastly, user can click on “Export Requirement” to export all specified requirements to their desired format.



Operating System	Version	Issue	
Microsoft Windows	7	colour missing	Delete
Microsoft Windows	8.1		Delete

Figure 3.4: UI – Specify Requirements

Figure 3.4 shows the UI which user specify their requirements. For instance, the UI shown was the form for user to specify non-functional requirement. The module being used is “Compatibility” module and the user specified that the system will be able to run in Windows 7 but colour may be missing. User can choose to add or delete any specified operating system data.

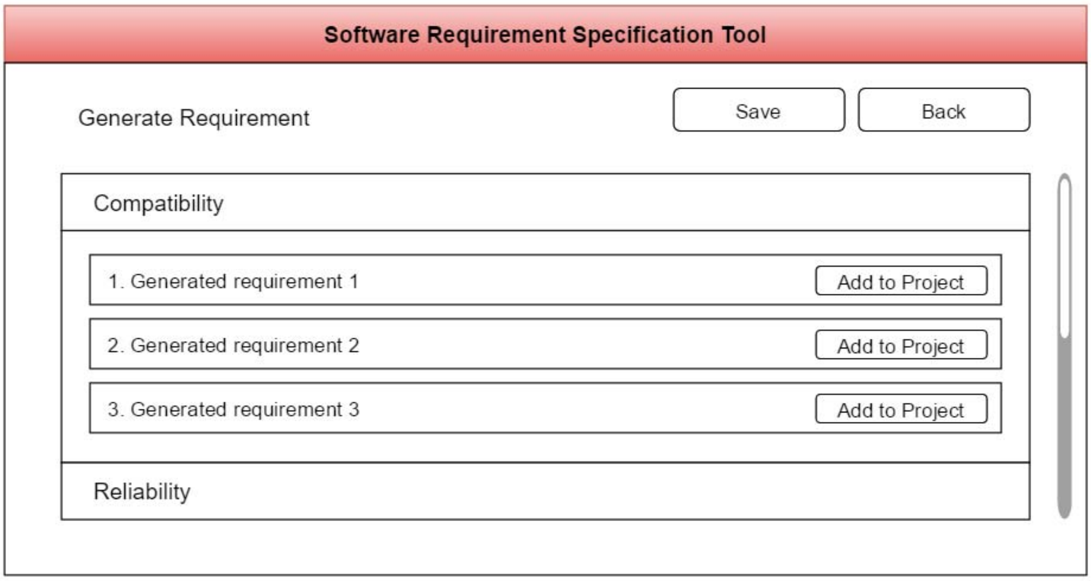
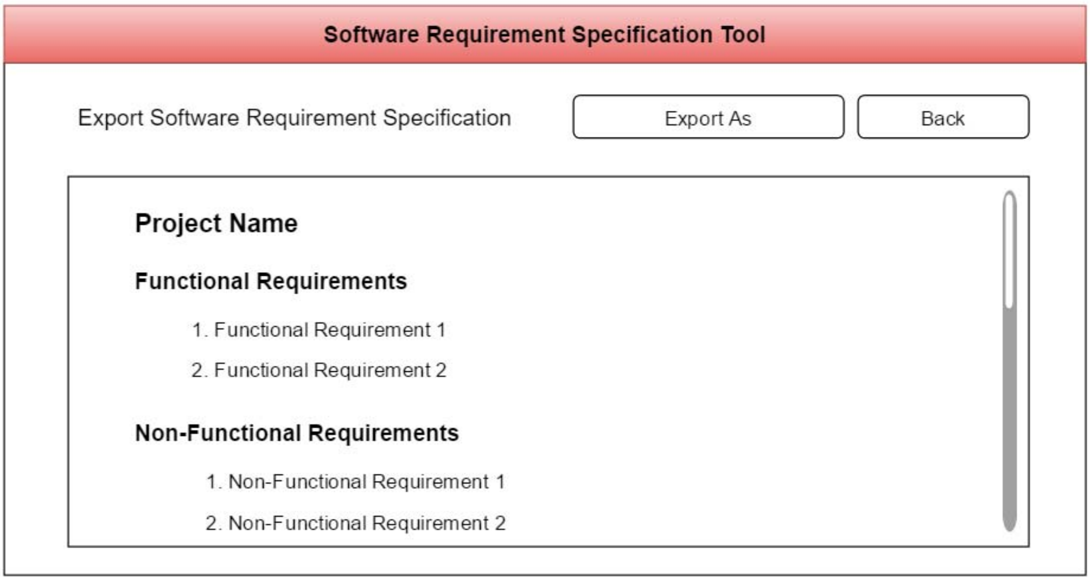

Software Requirement Specification Tool
Generate Requirement
Save
Back
Compatibility
1. Generated requirement 1
Add to Project
2. Generated requirement 2
Add to Project
3. Generated requirement 3
Add to Project
Reliability

Figure 3.5: UI – Generate Requirements

Figure 3.5 shows the UI where user adds generated requirements into their project. These requirements are generated based on user's input in each module and defined boilerplates for the module accordingly.



Software Requirement Specification Tool

Export Software Requirement Specification

Project Name

Functional Requirements

1. Functional Requirement 1
2. Functional Requirement 2

Non-Functional Requirements

1. Non-Functional Requirement 1
2. Non-Functional Requirement 2

Figure 3.6: UI – Export Requirements

Lastly in Figure 3.6, the UI for user to export requirements was shown. All requirements specified, generated and added to the project will be gathered and shown in document-like format. User may export the document to other formats that provided by the system to do further refinement or to be used in their project.

3.5. Project Plan

Please refer to "Appendix A: Work breakdown structure and Gantt chart".

CHAPTER 4

REQUIREMENT SPECIFICATION

4.1. Software Requirements Specification

The following section will describe the initial software requirements specifications that will be achieved by our proposed tool.

4.1.1. Functional Requirements

The term “user” refers to user of our proposed tool, while the term “system” refers to our proposed tool. The term “requirement” refers to both functional and non-functional requirements if not specified.

1. **Boilerplate template maintenance**
    a. The user shall be able to modify boilerplate templates
    b. The user shall be able to restore boilerplate templates to pre-defined boilerplate templates
    c. The user shall not be able to remove pre-defined boilerplate templates

2. **Requirement generation**
   a. The user shall be able to specify functional requirements using defined module.
   b. The user shall be able to specify non-functional requirements using defined module.
   c. The user shall be able to generate requirements using pre-defined boilerplate templates
   d. The user shall be able to generate requirements using user-defined boilerplate templates
   e. The user shall be able to remove generated requirements

3. **Pre-defined requirement boilerplate**
   a. The system shall provide pre-defined functional requirement boilerplates
   b. The system shall provide pre-defined non-functional boilerplate templates grouped by quality characteristics in accordance to ISO 25010 Quality Model (International Organization For Standardization ISO 2011)

4. **Project maintenance**
   a. The user shall be able to create project
   b. The user shall be able to open project
   c. The user shall be able to remove project

5. **Export**
   a. The system shall be able to export software requirement specification (SRS) as plain HTML file (.html)
   b. The system shall be able to export software requirement specification (SRS) as Microsoft Word Document file (.doc)

4.1.2. Non Functional Requirements

1. The total file size of the website shall not exceed 5MB.
2. The system shall provide REST API.
3. The private project shall only be accessible by owner if owner is logged in.
4. The public project shall be accessible by any user.
5. The system shall use OAuth 2.0 with Facebook as provider to login to the system.
6. The user interface of the system shall use material design.
7. The user interface of the system shall consistent through all modules so that will not confuse user.
8. The system shall validate user input for required input to prevent empty data.
9. The system shall prevent user from opening other people's private project.
10. The system shall prevent user from modifying other people's private project.
11. The system shall prevent user from deleting other people's private project.
12. The deletion time of project shall be less than 2 seconds.
13. The time taken to generate requirement shall be less than 5 seconds.
14. The time take to export requirements shall be less than 2 seconds.

4.2. Use Case Modelling

In order to describe the functionalities of our proposed tool, we performed a use case modelling with our proposed tool.

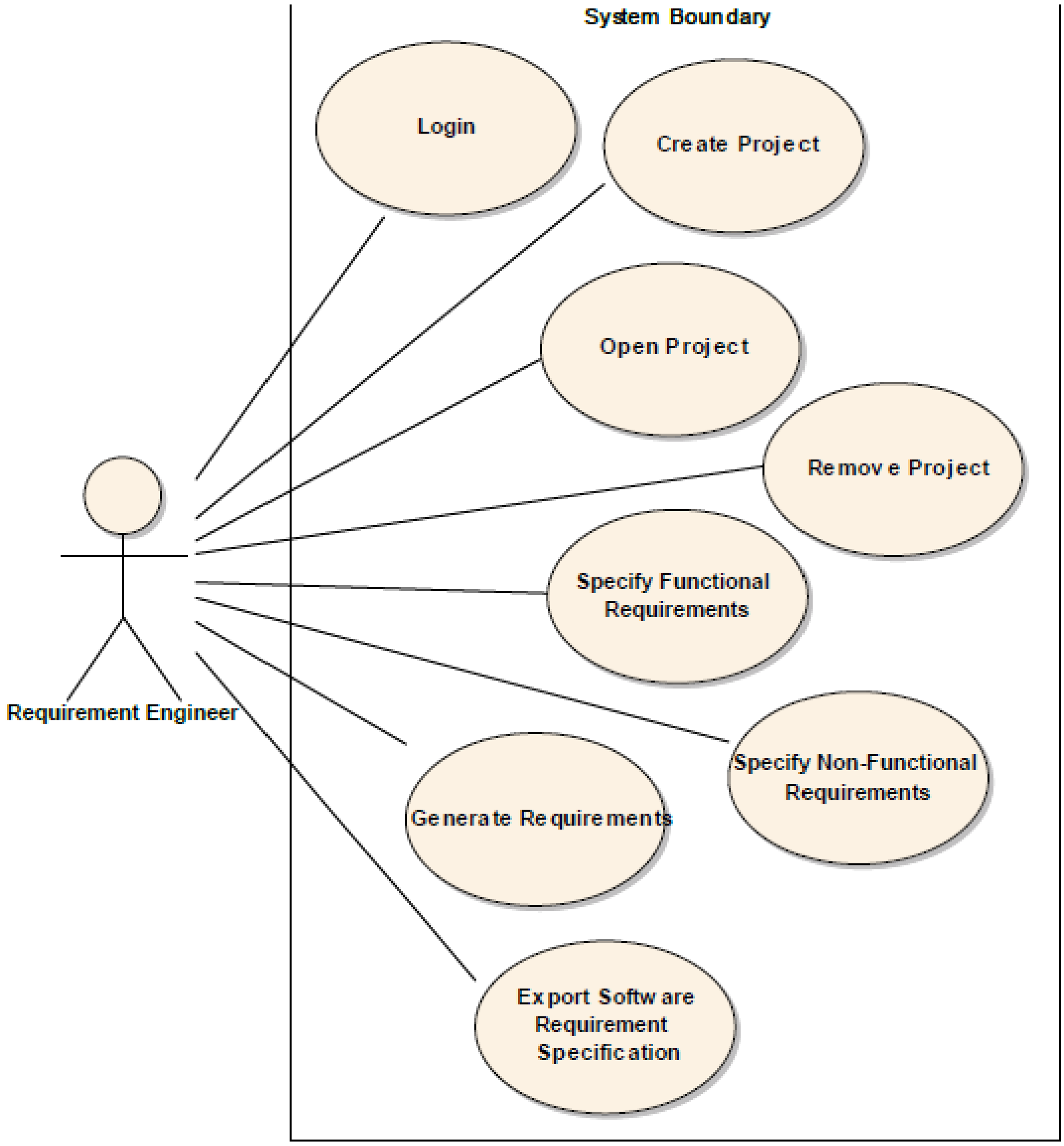

System Boundary
Login
Create Project
Open Project
Remove Project
Specify Functional Requirements
Specify Non-Functional Requirements
Generate Requirements
Export Software Requirement Specification
Requirement Engineer

Figure 4.1: Use Case Diagram

Figure 4.1 describes what actions can be done by requirement engineer when using our proposed tool. These actions are directly related to the functional requirement of our proposed tool.

Please refer to Appendix B: Use Case Descriptions for the description of each use cases stated in Figure 4.1

CHAPTER 5

DESIGN

5.1. Software Architecture Design

The software architecture we implemented is a **client-server** architecture design. In this case, we refer the browser web application built on top of AngularJS as **client**, and the server built on top of NodeJS as **server**. The distribution of software components shown in Figure 5.1.

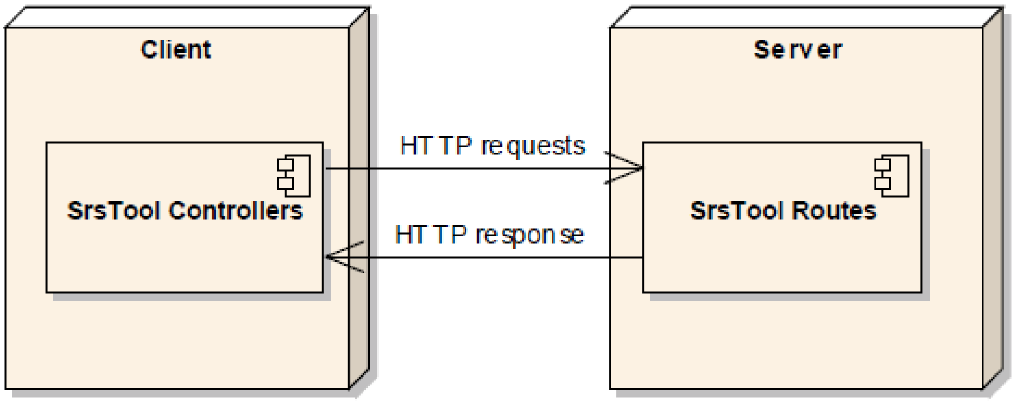


Figure 5.1: Deployment Diagram

Figure 5.1 contains two components from component diagrams, which are SrsTool Controller from Figure 5.2 and SrsTool Routes from Figure 5.3. The reason is because only these two components are actually communicating to each other using HTTP request and response. SrsTool Controllers will perform HTTP request from client side, which will be routed accordingly by SrsTool Routes in server and returns HTTP responses. For further description of each components, please refer to section 5.2.

5.2. Software Component Design

There are two component diagram that we constructed to model the components of both client and server side. The two diagrams will be shown in the following two sections.

5.2.1. Client Component Diagram

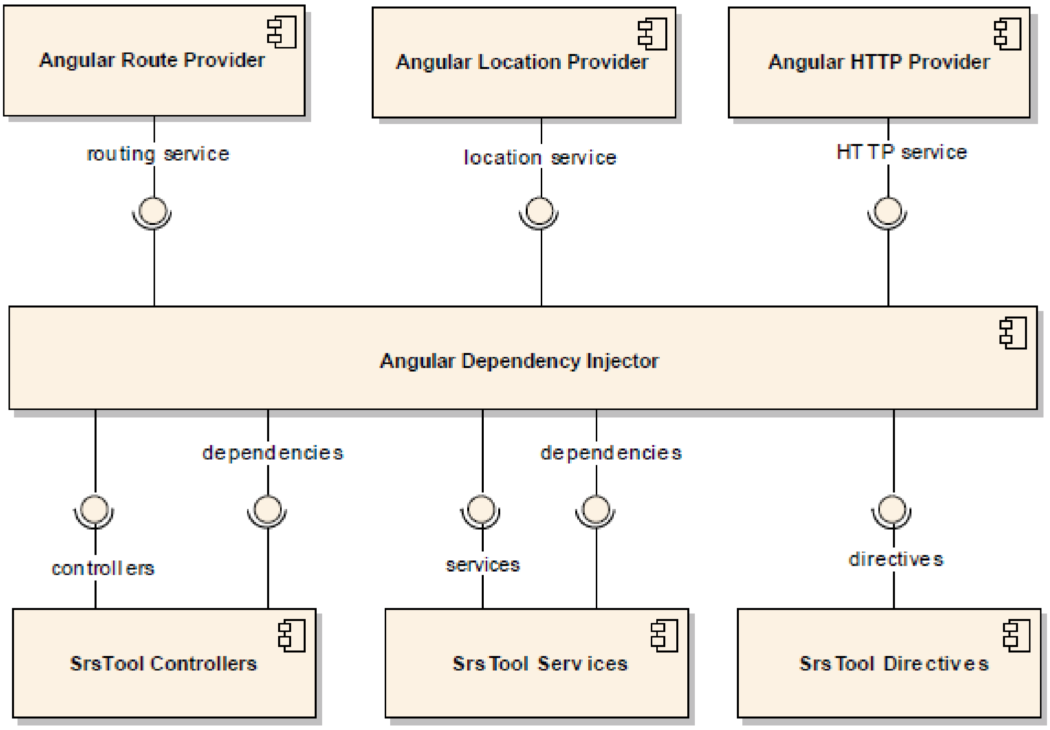


Figure 5.2: Client Component Diagram

Figure 5.2 shows the component diagram of client side. The components with prefix "Angular" are provided by AngularJS itself, while those with prefix "SrsTool" refers to user defined components. The description of each component is as below:

1. **Angular Dependency Injector**

   Angular Dependency Injector provides an interface for all dependency to be declared before usage, and injected into required components when needed. For instance, from the diagram shown, SrsTool controllers and services are the main consumer, while all of the other components are supplier of services. This allows a uniform interface to inject dependencies and reduce the interdependency between components.

2. **Angular Route Provider**

   Angular Route Provider is used to route within different HTML pages on the browser without needing to refresh the page. This is done internally by Angular where it fetches the HTML file via Asynchronous JavaScript and XML request (AJAX) and updates the user interface. This gives user experience as if the web application is loading instantaneously and behaves like a desktop application.

3. **Angular Location Provider**

   Angular Location Provider encapsulates the location path (URL) of web page and converts location into Angular routes accordingly. This will allow us to switch between routes and load different pages without needing to refresh to page.

4. **Angular HTTP Provider**

   Angular HTTP Provider provides HTTP services by encapsulating all HTTP request header type into functions. This allow us to do HTTP request easily without needing to construct AJAX objects as in JavaScript.

5. **SrsTool Directives**

   SrsTool Directives binds custom HTML tags with their associated controllers. This allow our web application to be broken down into subcomponents and allow us to develop modules independently. SrsTool Directives will be compiled and provided as directive for controllers and user interface.

6. **SrsTool Services**

SrsTool Services provides services such as user authentication service on the client side. SrsTool Services can be used in all other component and controllers, which they will provide functionalities that are commonly used by all other components.

7. **SrsTool Controllers**

SrsTool Controllers are controllers for each user interface that we defined. The controller contains all the logic of the application and will perform actions such as saving, loading, routing and generating requirements. SrsTool Controllers are linked to HTML files according to how it was defined in SrsTool Directives.

5.2.2. Server Component Diagram

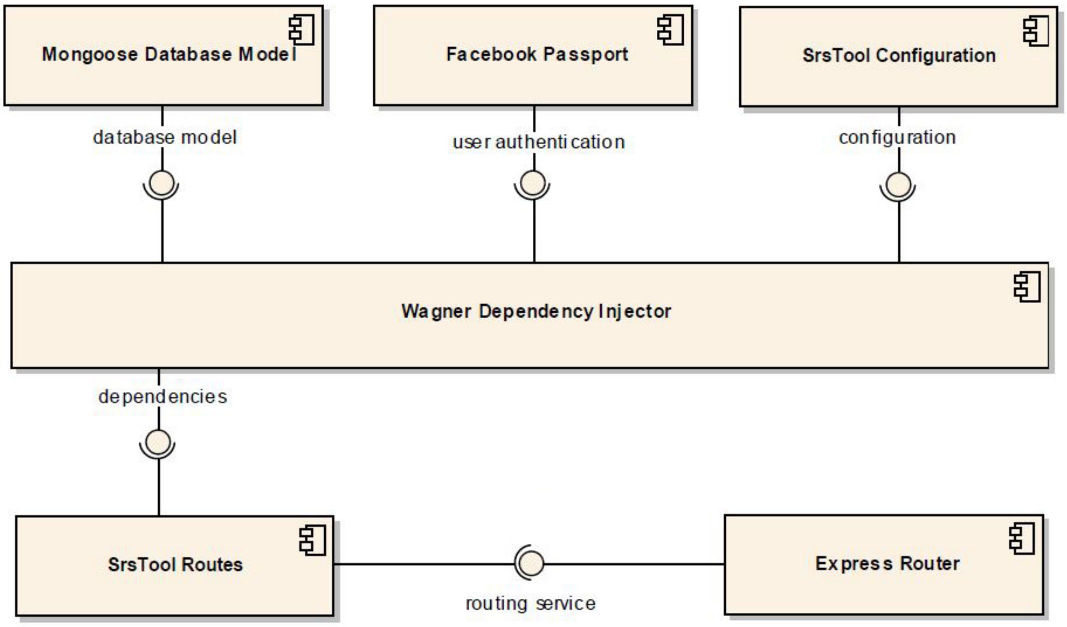


Figure 5.3: Server Component Diagram

Figure 5.3 shows the component diagram of server side. The description of each component is as below:

1. **Wagner Dependency Injector**

   Wagner Dependency Injector provides the similar functionality of Angular Dependency Injector.

2. **Mongoose Database Model**

   Mongoose Database Model allows user to define database schema for MongoDB and provides functionalities such as validating, and basic CRUD (Create, Read, Update, and Delete) actions. Mongoose Database Model also allow user to directly make queries on the collection directly and provides creates Database Access Object for user to do directly data manipulation and save changes.

3. **Facebook Passport**

   Facebook Passport provides OAuth 2.0 features using Facebook as authentication provider. This allows user to sign in to the system using their Facebook account. This allows us to trace private projects based on user's Facebook account and has better and secured login platform.

4. **SrsTool Configuration**

   SrsTool Configuration stores some configuration data which may be used globally.

5. **Express Router**

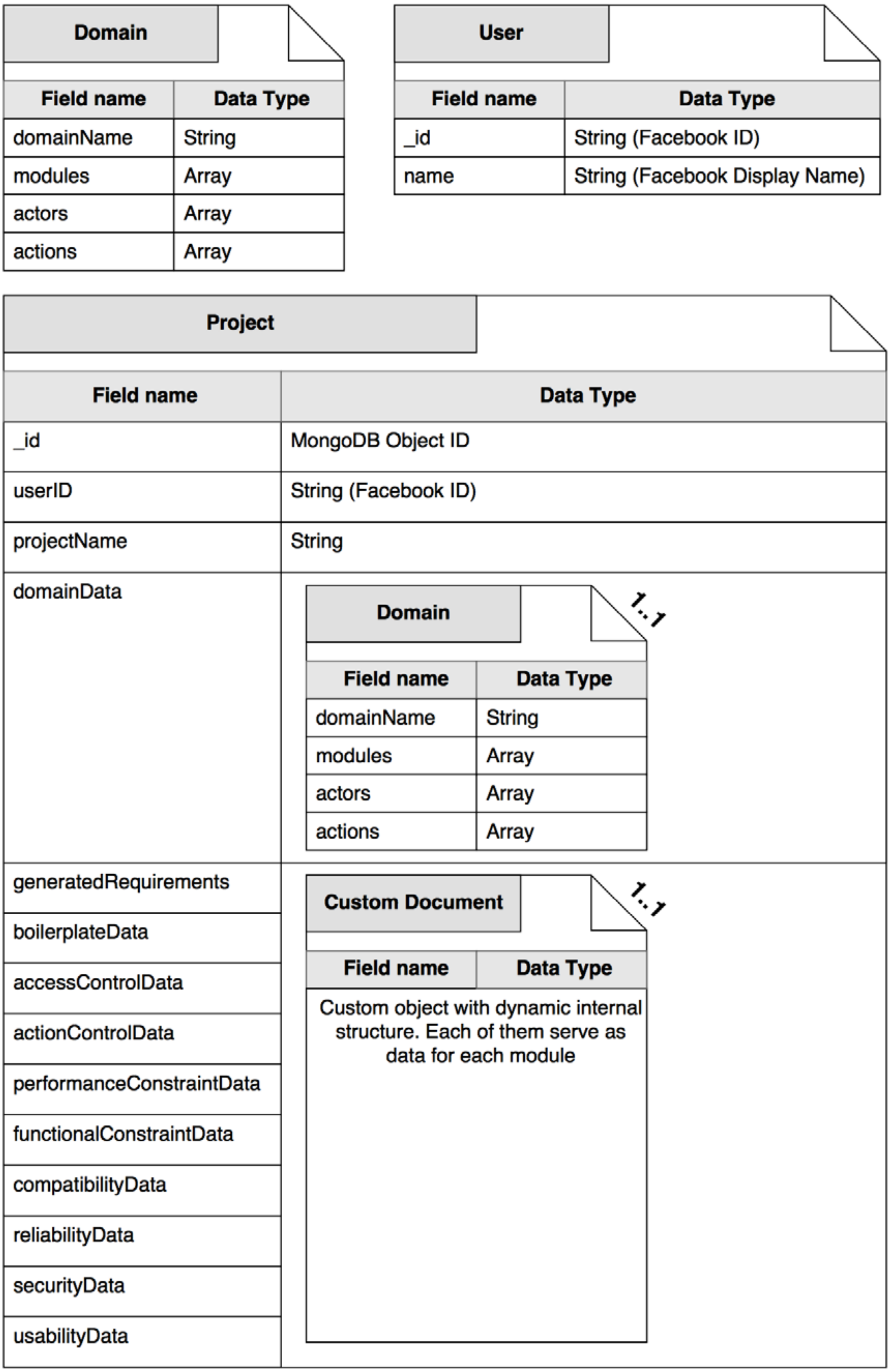
   Express Router is a module that provided by ExpressJS that encapsulate routing functionalities and allow us to just define the name and method of the route. The routing actions will be done internally and the developer will only need to concern about the logic of each route.

6. **SrsTool Routes**

SrsTool Routes are route endpoints defined by developer that will handle HTTP request accordingly. SrsTool routes are built on top of Express Router and will be the only consumer for dependencies such as using Mongoose Database Models to store or retrieve data from MongoDB.

5.3. Database Design

For the database design of our tool, we will be using a modelling technique of embedding document as described by (Vera et al. 2015) in their journal "Data modelling for NoSQL document-oriented databases". This technique de-normalizes the data and store them in a document to allow data manipulation in a single database transaction.



Domain

Field name	Data Type
domainName	String
modules	Array
actors	Array
actions	Array

User

Field name	Data Type
_id	String (Facebook ID)
name	String (Facebook Display Name)

Project

Field name	Data Type
_id	MongoDB Object ID
userID	String (Facebook ID)
projectName	String
domainData	
generatedRequirements	
boilerplateData	
accessControlData	
actionControlData	
performanceConstraintData	
functionalConstraintData	
compatibilityData	
reliabilityData	
securityData	
usabilityData	

Domain 1..1

Field name	Data Type
domainName	String
modules	Array
actors	Array
actions	Array

Custom Document 1..1

Field name	Data Type
Custom object with dynamic internal structure. Each of them serve as data for each module	

Figure 5.4: NoSQL Document Design Diagram

In our database there are 3 collections, which are:

1. **Domain**

   Stores each domain along with all existing modules, actors and actions. New domain attributes will be added to its existing domain document and suggested back to user.

2. **User**

   Stores user's Facebook ID and names. This collection will be used to trace private project's owner by Facebook ID.

3. **Project**

   Stores data of each project. The projects are identified uniquely by ID which is generated by MongoDB upon insertion. The *userID* field refers to the user of project and which empty user ID refers to public project. The *domainData* field stores a sub-document which is de-normalized from Domain collection. The document only contains a subset of all modules, actors and action. The rest of the fields are embedded documents for each modules of the system. For instance, *boilerplateData* stores the boilerplate of each module, and *accessControlData* stores the data for Access Control Module. These embedded documents structure are dynamic and complex, and hence are not described in the ERD.

5.4. RESTful Route Design

Based on the collections that we have in database, routes will be created for related actions for each route. The prefix of route is "*HOST_URL/api/v1/*". For instance, the route "*domain*" will be mapped to become "*HOST_URL/api/v1/domain*", where HOST_URL refers to the URL of the server, like "*www.srs-tool.com/api/v1/domain*". The route part with colon (:) in front (route/:id) refers to the query parameter (route/123 means ID is 123).

1. Domain

Table 5.1: Route Design for Domain

Route	Method	Description
domains/names	GET	Returns a list of domain names
domains/:id	GET	Returns Domain document based on ID, or null if not found

2. User

Table 5.2: Route Design for User

Route	Method	Description
me	GET	Returns current logged in user's data, or null if not logged in
auth/facebook	GET	Redirects user to login with Facebook
auth/facebook/callback	GET	Redirects user to login with Facebook, then redirects user to their Projects page
auth/logout	GET	Log out user and redirect them to homepage

3. Project

Table 5.3: Route Design for Project

Route	Method	Description
projects/public	GET	Returns a list of public projects with project names and project ID
projects/private	GET	Returns a list of private projects with project names and project ID. Returns empty array if user is not logged in
projects/	POST	Creates new project and return result to indicate whether creation of project is successful
projects/:id	GET	Returns Project document based on ID of the project, or null if not found
projects/:id	DELETE	Returns result to indicate whether deletion of project is successful
projects/:id/:subroute	GET	Returns Project document with only certain selected fields indicated in server, or null if project is not found or sub-route doesn’t exist
projects/:id/:subroute	PATCH	Returns result to indicate whether updating project’s data is successful
projects/:id/project-data	PATCH	Returns result to indicate whether updating project’s data is successful
projects/:id/domain-data	PATCH	Returns result to indicate whether updating project’s data is successful

5.5. Activity Diagram

To illustrate the interaction between components and process flow in our tool, we prepared activity diagram for each use case in our project.

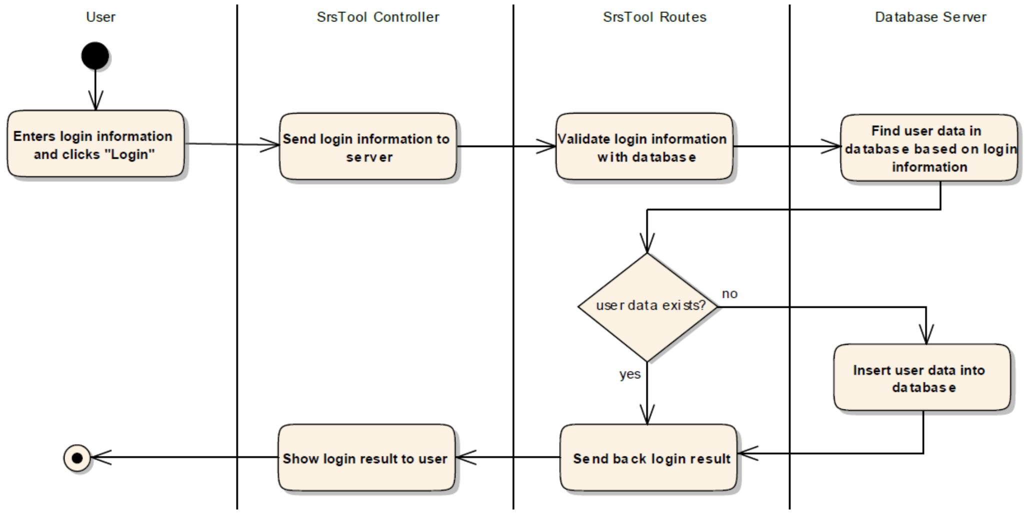

User
SrsTool Controller
SrsTool Routes
Database Server
Enters login information and clicks "Login"
Send login information to server
Validate login information with database
Find user data in database based on login information
user data exists?
no
yes
Insert user data into database
Send back login result
Show login result to user

Figure 5.5: Activity Diagram – Login

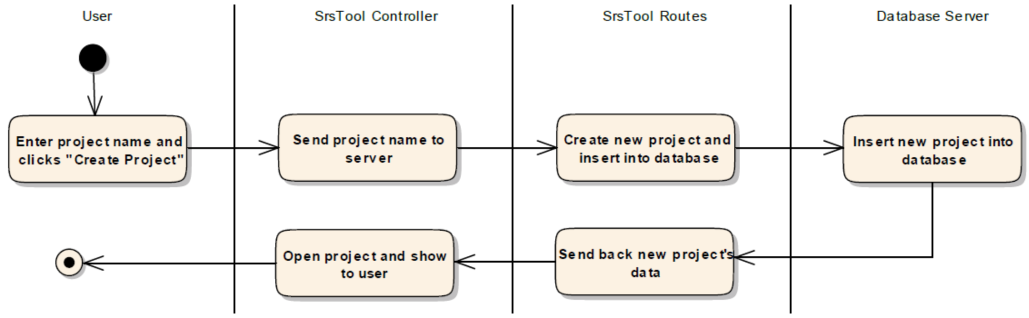

User
SrsTool Controller
SrsTool Routes
Database Server
Enter project name and clicks "Create Project"
Send project name to server
Create new project and insert into database
Insert new project into database
Send back new project's data
Open project and show to user

Figure 5.6: Activity Diagram – Create Project

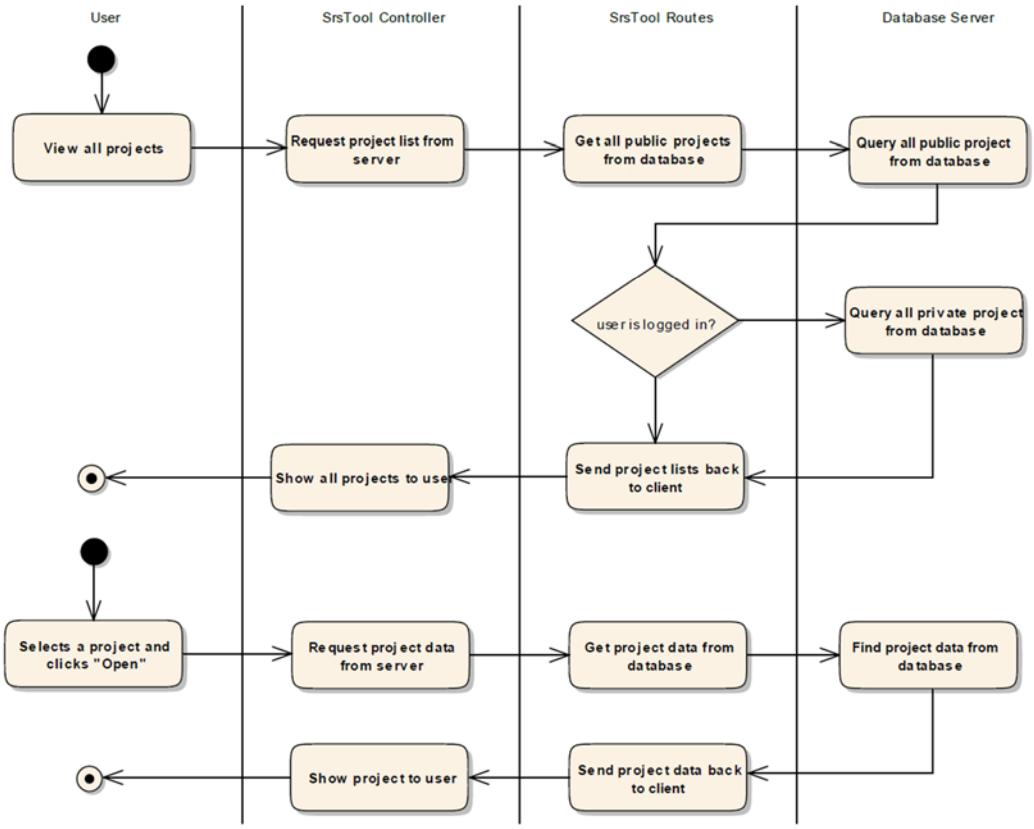

User
SrsTool Controller
SrsTool Routes
Database Server
View all projects
Request project list from server
Get all public projects from database
Query all public project from database
user is logged in?
Query all private project from database
Show all projects to user
Send project lists back to client
Selects a project and clicks "Open"
Request project data from server
Get project data from database
Find project data from database
Show project to user
Send project data back to client

Figure 5.7: Activity Diagram – Open Project

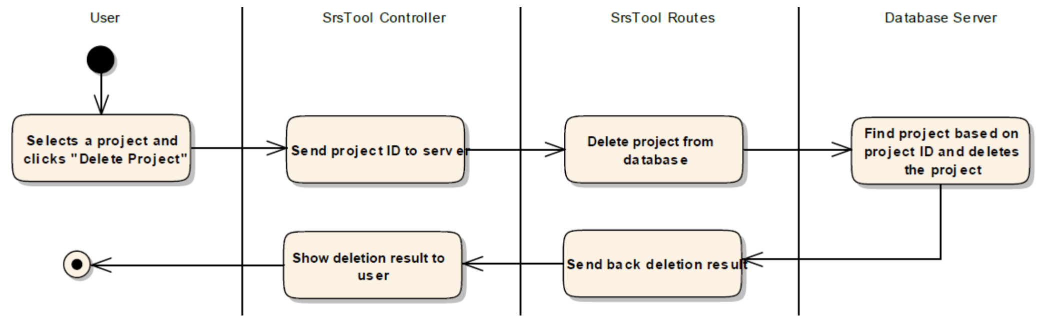

User
SrsTool Controller
SrsTool Routes
Database Server
Selects a project and clicks "Delete Project"
Send project ID to server
Delete project from database
Find project based on project ID and deletes the project
Show deletion result to user
Send back deletion result

Figure 5.8 Activity Diagram – Remove Project

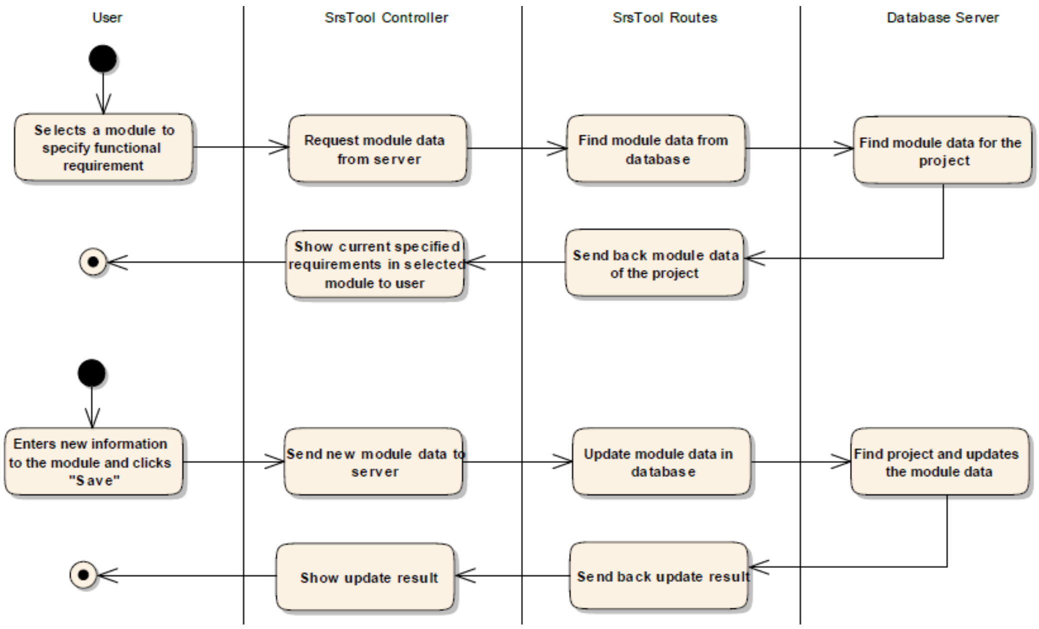

User
SrsTool Controller
SrsTool Routes
Database Server
Selects a module to specify functional requirement
Request module data from server
Find module data from database
Find module data for the project
Send back module data of the project
Show current specified requirements in selected module to user
Enters new information to the module and clicks "Save"
Send new module data to server
Update module data in database
Find project and updates the module data
Send back update result
Show update result

Figure 5.9: Activity Diagram – Specify Functional Requirement

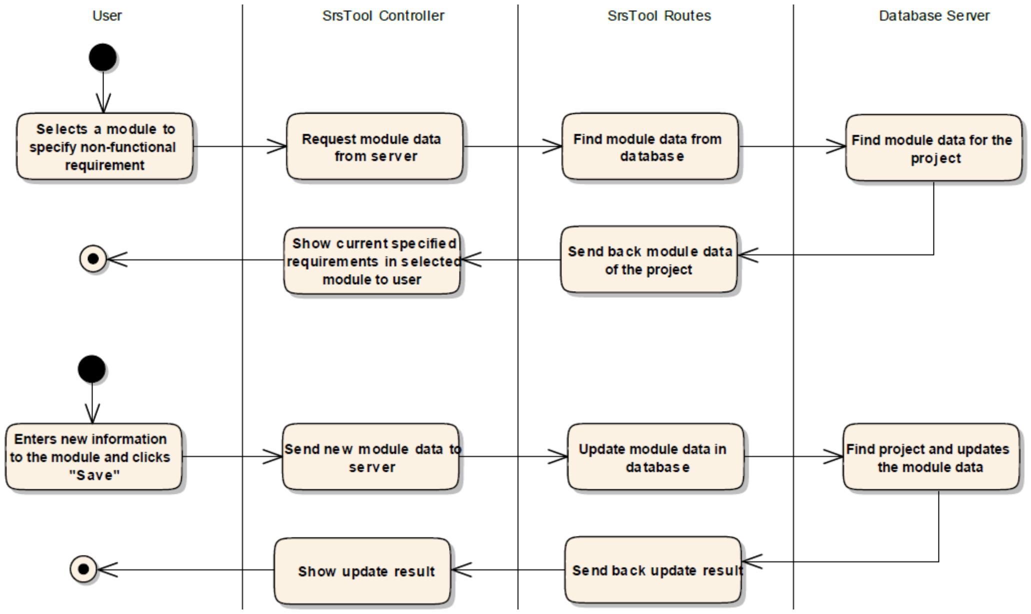

User
SrsTool Controller
SrsTool Routes
Database Server
Selects a module to specify non-functional requirement
Request module data from server
Find module data from database
Find module data for the project
Send back module data of the project
Show current specified requirements in selected module to user
Enters new information to the module and clicks "Save"
Send new module data to server
Update module data in database
Find project and updates the module data
Send back update result
Show update result

Figure 5.10: Activity Diagram – Specify Non-Functional Requirement

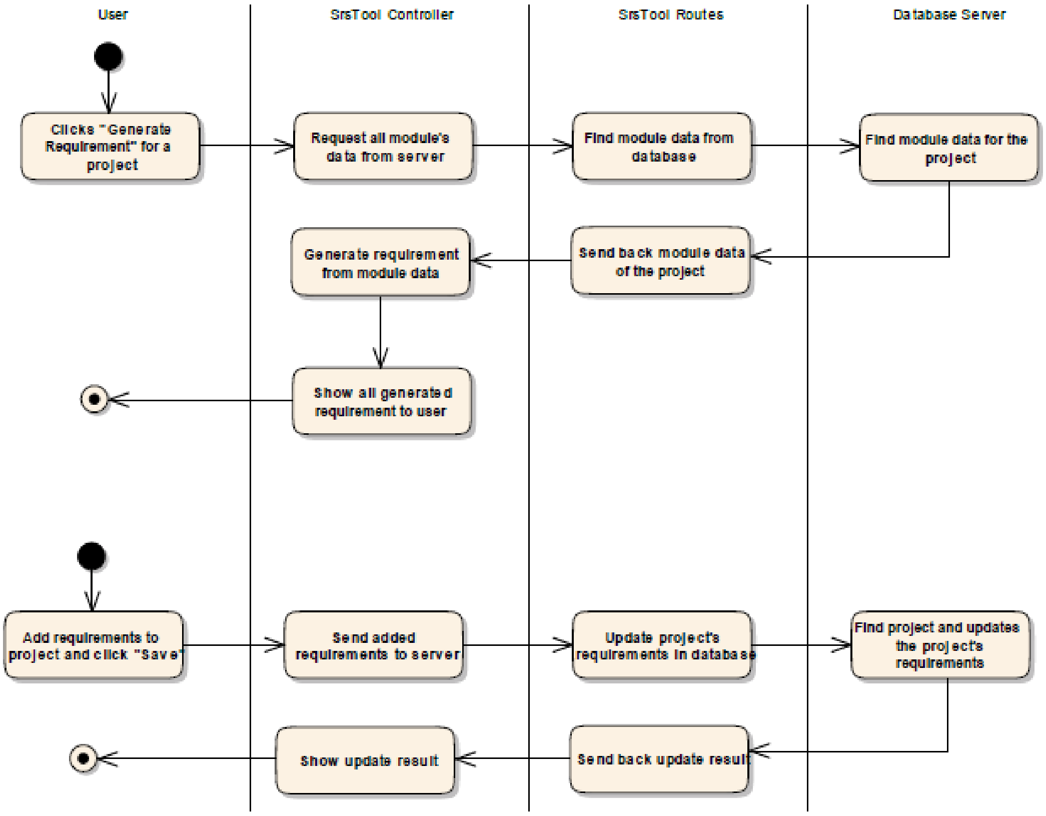

User
SrsTool Controller
SrsTool Routes
Database Server
Clicks "Generate Requirement" for a project
Request all module's data from server
Find module data from database
Find module data for the project
Generate requirement from module data
Send back module data of the project
Show all generated requirement to user
Add requirements to project and click "Save"
Send added requirements to server
Update project's requirements in database
Find project and updates the project's requirements
Show update result
Send back update result

Figure 5.11: Activity Diagram – Generate Requirements

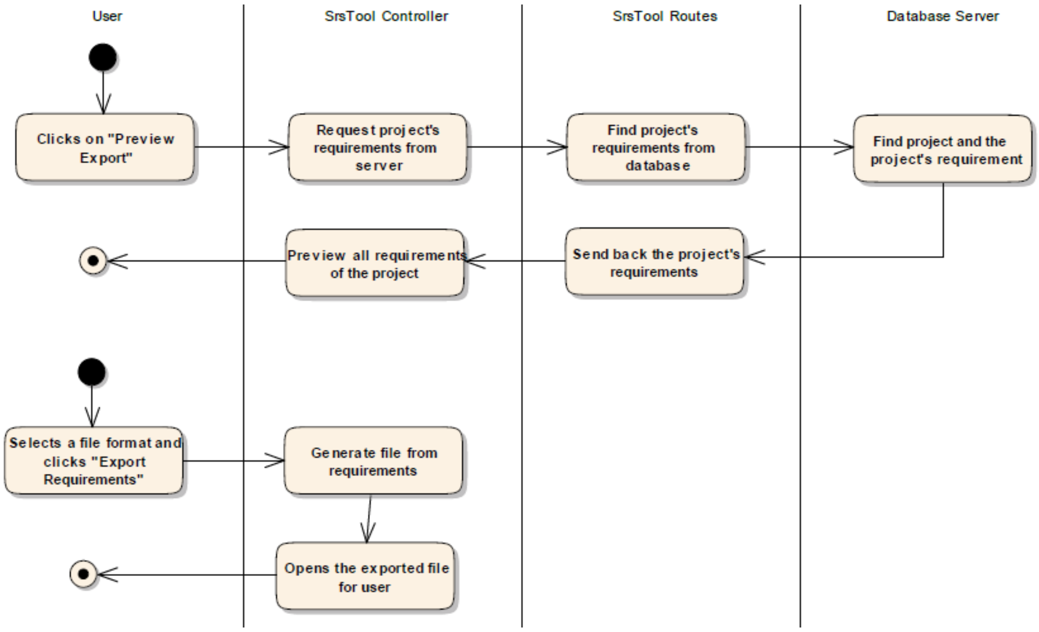

User
SrsTool Controller
SrsTool Routes
Database Server
Clicks on "Preview Export"
Request project's requirements from server
Find project's requirements from database
Find project and the project's requirement
Preview all requirements of the project
Send back the project's requirements
Selects a file format and clicks "Export Requirements"
Generate file from requirements
Opens the exported file for user

Figure 5.12: Activity Diagram – Export Software Requirement Specification

5.6. Sequence Diagrams

To further clarify the process flow, we also prepared sequence diagrams for each use cases in our project. See Appendix D: Sequence Diagrams for all the diagrams.

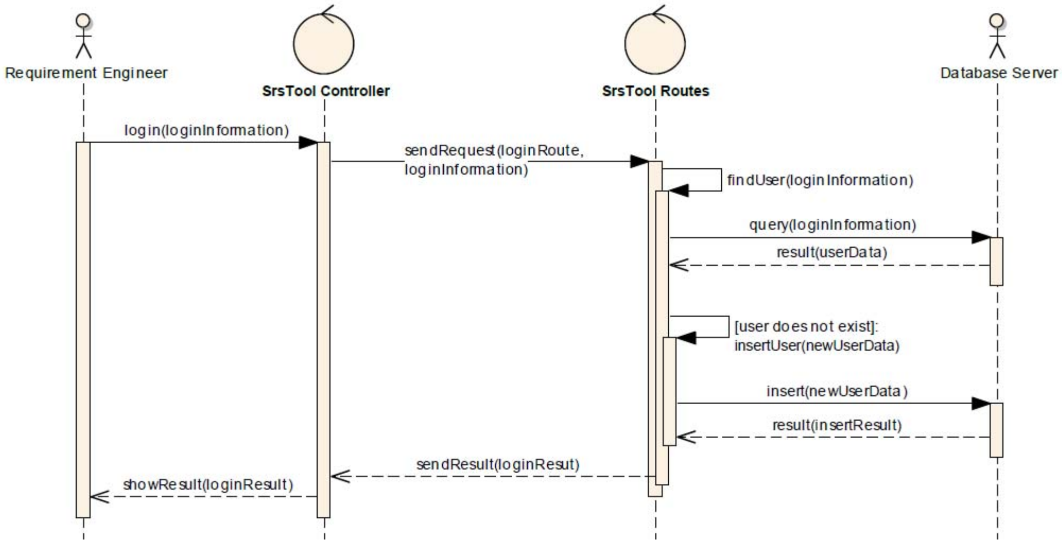


Figure 5.13: Sequence Diagram – Login

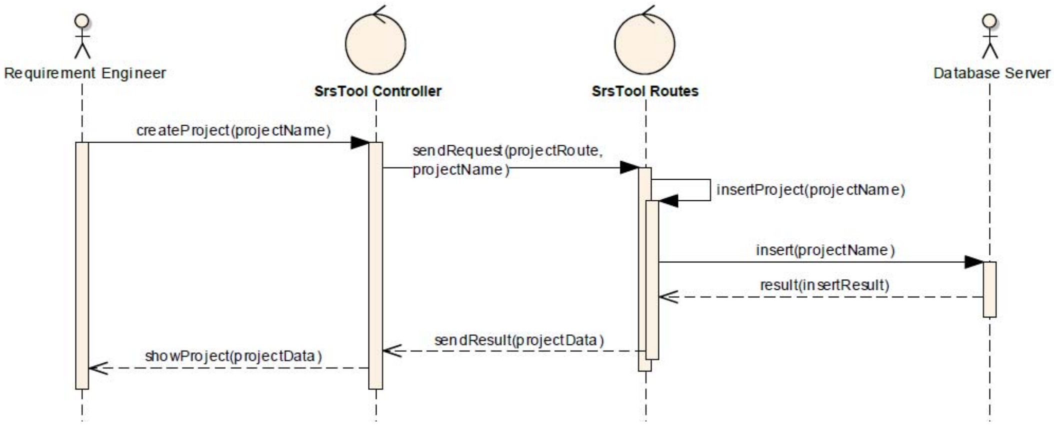


Figure 5.14: Sequence Diagram – Create Project

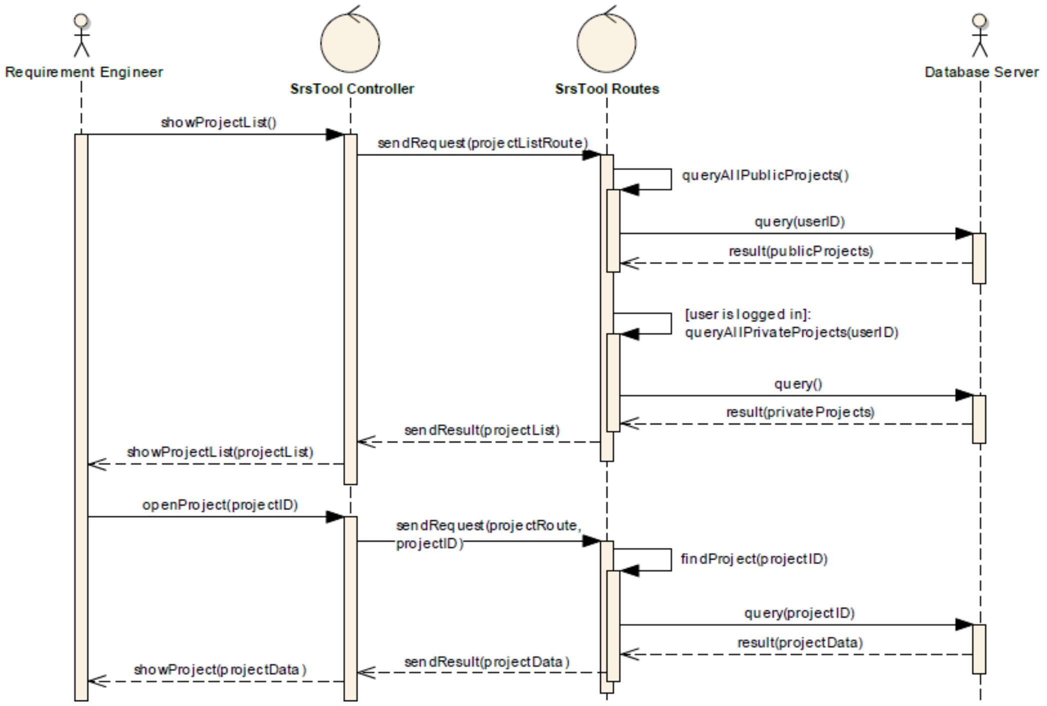


Figure 5.15: Sequence Diagram – Open Project

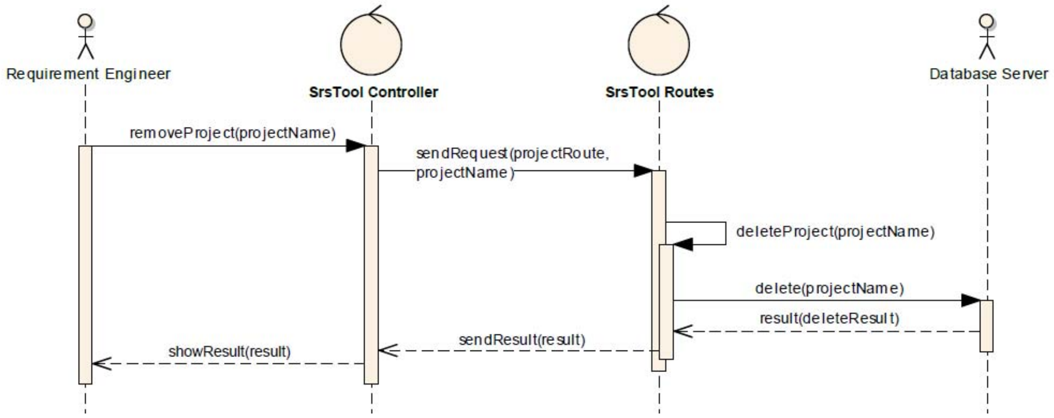


Figure 5.16: Sequence Diagram – Remove Project

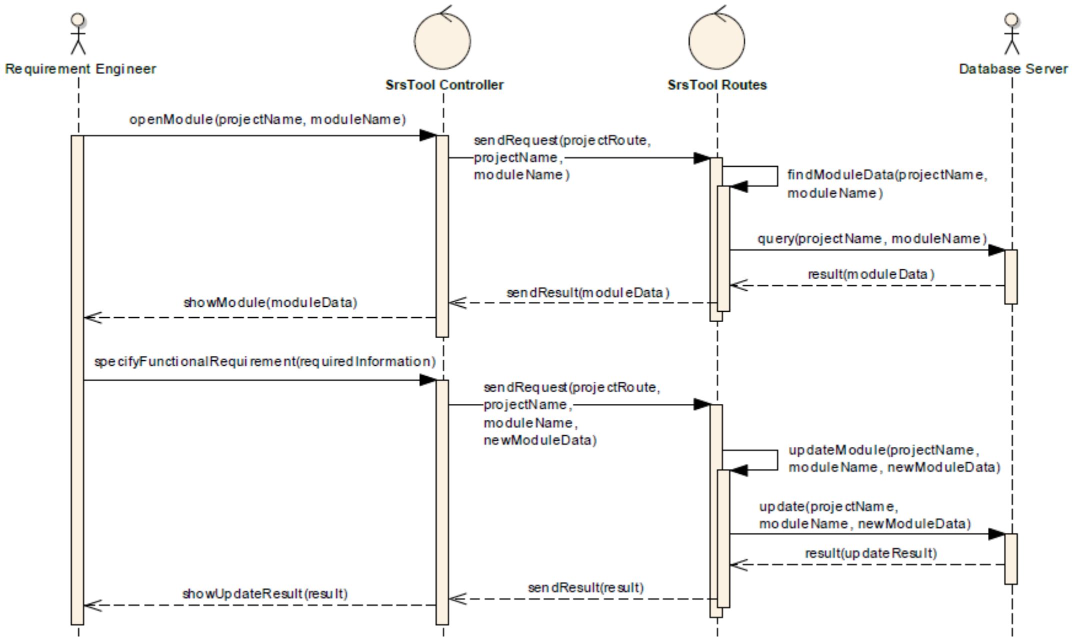

Requirement Engineer
SrsTool Controller
SrsTool Routes
Database Server
openModule(projectName, moduleName)
sendRequest(projectRoute, projectName, moduleName)
findModuleData(projectName, moduleName)
query(projectName, moduleName)
result(moduleData)
sendResult(moduleData)
showModule(moduleData)
specifyFunctionalRequirement(requiredInformation)
sendRequest(projectRoute, projectName, moduleName, newModuleData)
updateModule(projectName, moduleName, newModuleData)
update(projectName, moduleName, newModuleData)
result(updateResult)
sendResult(result)
showUpdateResult(result)

Figure 5.17: Sequence Diagram – Specify Functional Requirement

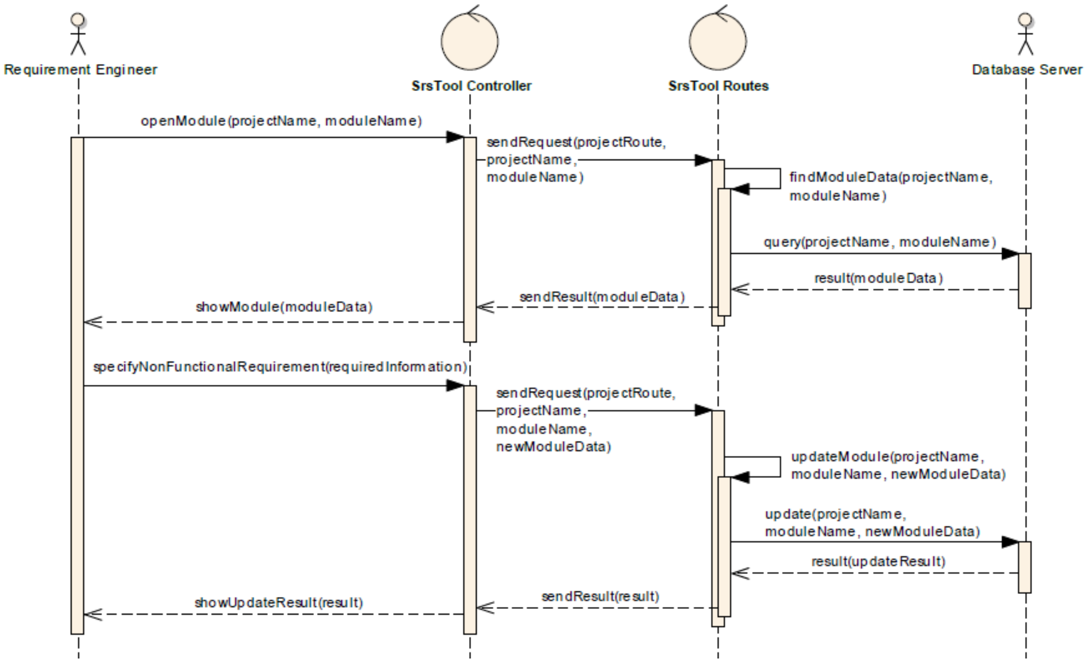

Requirement Engineer
SrsTool Controller
SrsTool Routes
Database Server
openModule(projectName, moduleName)
sendRequest(projectRoute, projectName, moduleName)
findModuleData(projectName, moduleName)
query(projectName, moduleName)
result(moduleData)
sendResult(moduleData)
showModule(moduleData)
specifyNonFunctionalRequirement(requiredInformation)
sendRequest(projectRoute, projectName, moduleName, newModuleData)
updateModule(projectName, moduleName, newModuleData)
update(projectName, moduleName, newModuleData)
result(updateResult)
sendResult(result)
showUpdateResult(result)

Figure 5.18: Sequence Diagram – Specify Non-Functional Requirement

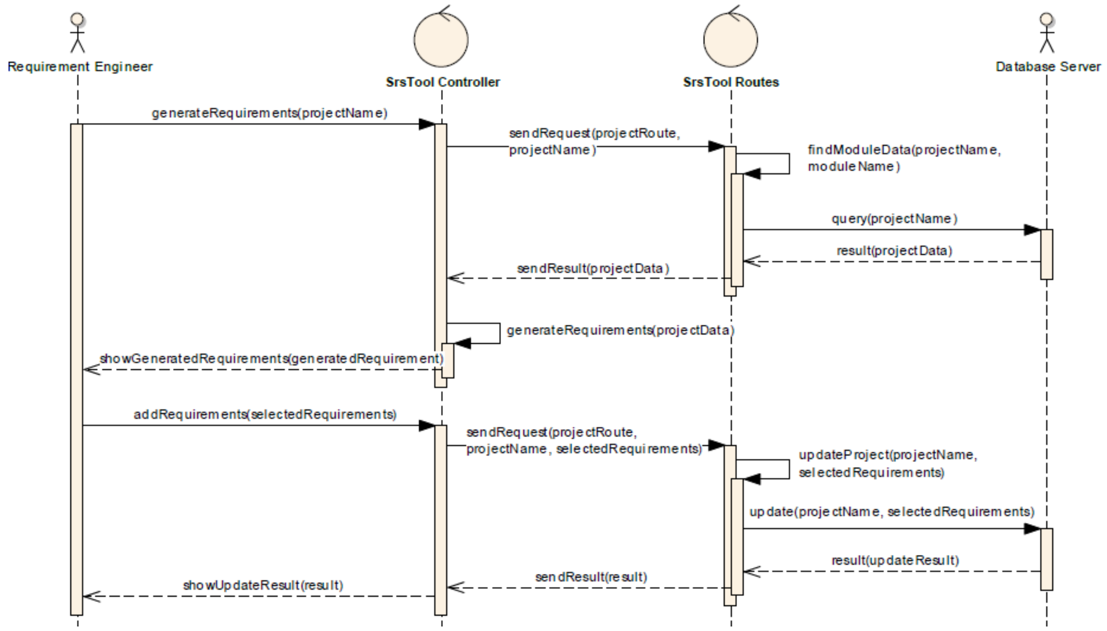

Requirement Engineer
SrsTool Controller
SrsTool Routes
Database Server
generateRequirements(projectName)
sendRequest(projectRoute, projectName)
findModuleData(projectName, moduleName)
query(projectName)
result(projectData)
sendResult(projectData)
generateRequirements(projectData)
showGeneratedRequirements(generatedRequirement)
addRequirements(selectedRequirements)
sendRequest(projectRoute, projectName, selectedRequirements)
updateProject(projectName, selectedRequirements)
update(projectName, selectedRequirements)
result(updateResult)
sendResult(result)
showUpdateResult(result)

Figure 5.19: Sequence Diagram – Generate Requirements

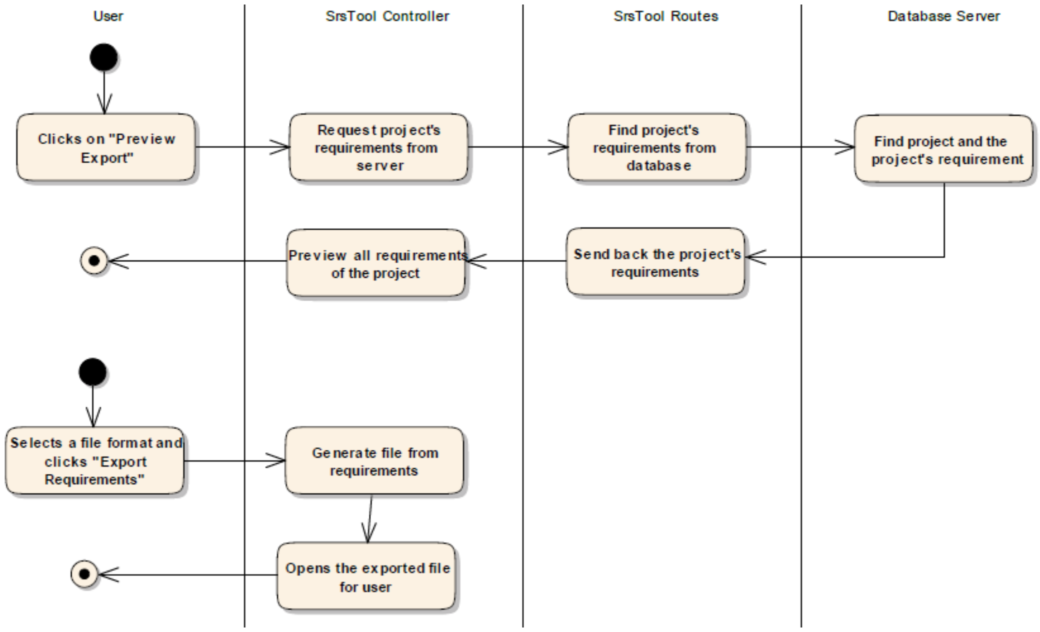

User
SrsTool Controller
SrsTool Routes
Database Server
Clicks on "Preview Export"
Request project's requirements from server
Find project's requirements from database
Find project and the project's requirement
Send back the project's requirements
Preview all requirements of the project
Selects a file format and clicks "Export Requirements"
Generate file from requirements
Opens the exported file for user

Figure 5.20: Sequence Diagram – Export Software Requirement Specification

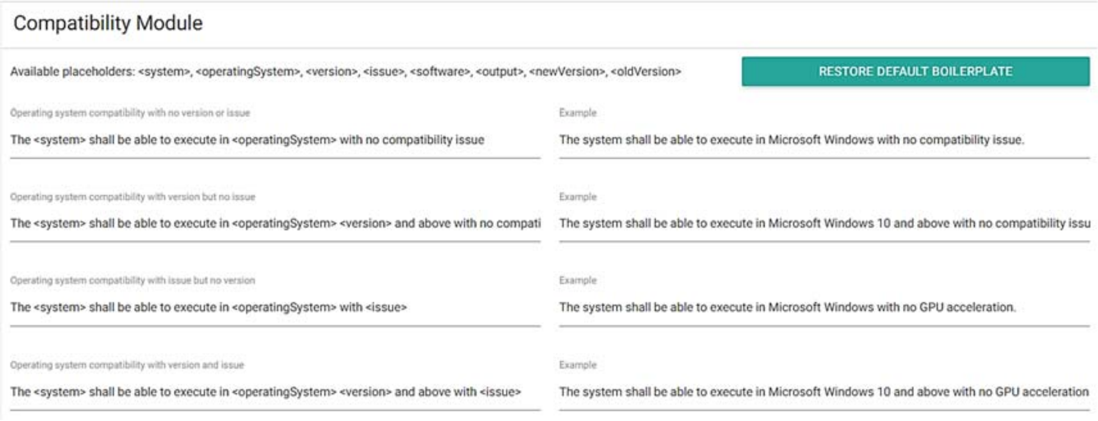
CHAPTER 6

CODING AND IMPLEMENTATION

6.1. Requirement Specification Strategy Implementation

As per discussed in Section 3.3 Requirement Specification Strategy, we implemented the requirement specification strategy into our tool. In this section, the coding aspect of the implementation will be discussed. We will use the "Reliability" module as example.

Based on the discussion, we extracted the boilerplate and saved them in the system so that it can be used to generate requirement later on. However, we also gave the user option to modify the boilerplate to their desired format. As a backup, we also prepared the option for user to restore predefined boilerplate (Please refer to Figure 6.1).



Compatibility Module

Available placeholders: <system>, <operatingSystem>, <version>, <issue>, <software>, <output>, <newVersion>, <oldVersion>

Operating system compatibility with no version or issue

The <system> shall be able to execute in <operatingSystem> with no compatibility issue

Example

The system shall be able to execute in Microsoft Windows with no compatibility issue.

Operating system compatibility with version but no issue

The <system> shall be able to execute in <operatingSystem> <version> and above with no compati

Example

The system shall be able to execute in Microsoft Windows 10 and above with no compatibility issu

Operating system compatibility with issue but no version

The <system> shall be able to execute in <operatingSystem> with <issue>

Example

The system shall be able to execute in Microsoft Windows with no GPU acceleration.

Operating system compatibility with version and issue

The <system> shall be able to execute in <operatingSystem> <version> and above with <issue>

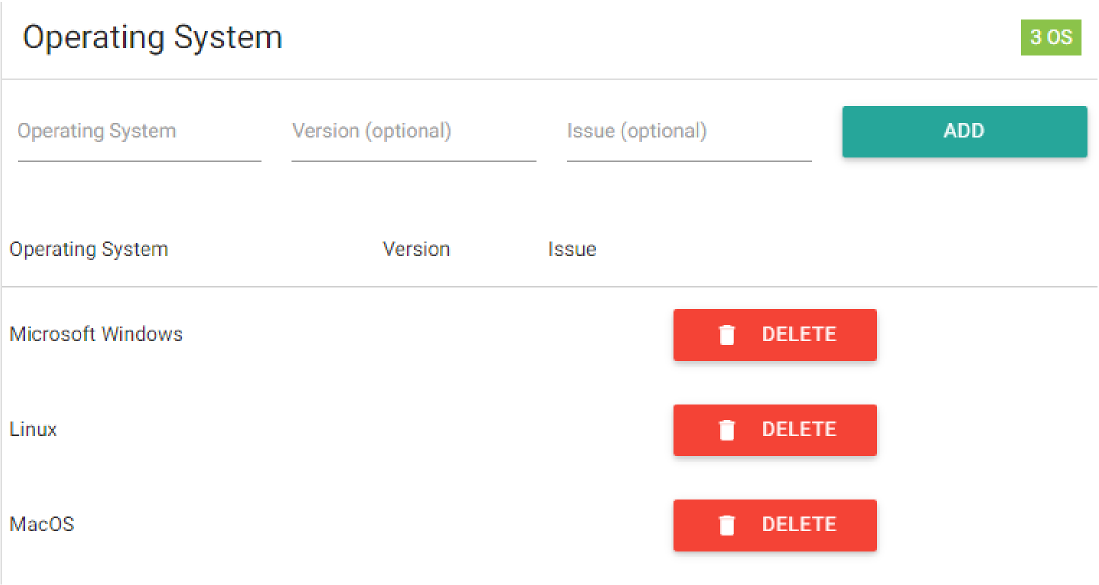
Example

The system shall be able to execute in Microsoft Windows 10 and above with no GPU acceleration

Figure 6.1: Boilerplate for Compatibility Module

In order to gather inputs from user for placeholder's values, such as <operatingSystem>, <version> and <issue> as shown in Figure 6.1, we designed an interactive user interface which involves user filling up input textboxes and add them to the module's data (Please refer to Figure 6.2).

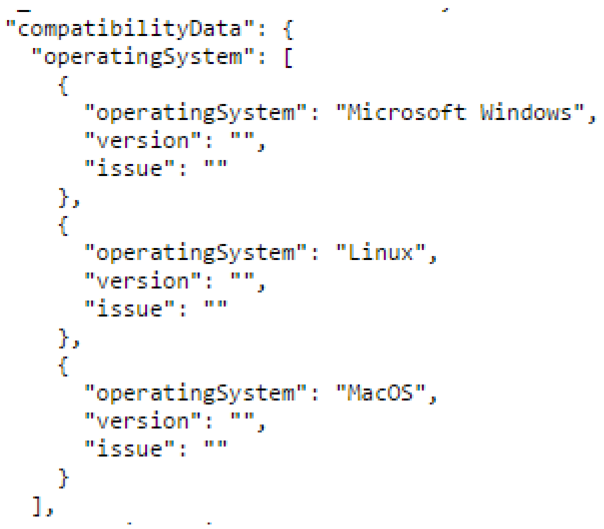
The form will request user to enter all the required inputs (in this case, <operatingSystem>) and optional inputs (<version> and <issue>). Based on user's input, the system will determine which boilerplate to use, such as using boilerplate without <version> if user did not specify version of the operating system.



Operating System	Version	Issue	
Microsoft Windows			DELETE
Linux			DELETE
MacOS			DELETE

Figure 6.2: Input Form for Compatibility Module

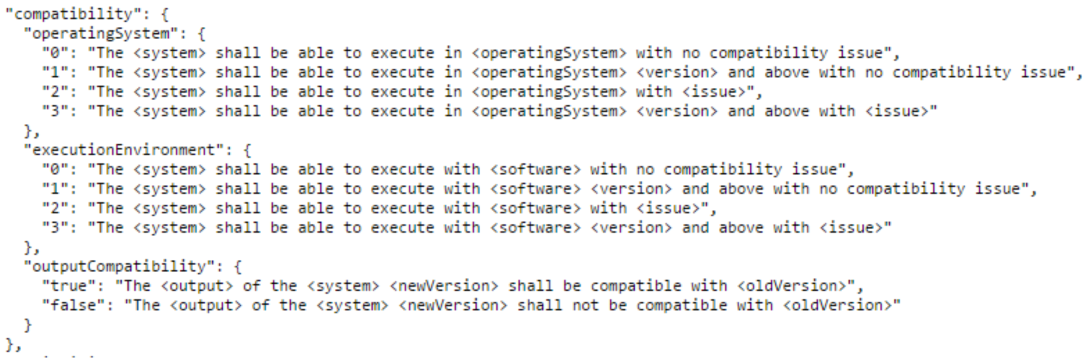
After specifying the operating system's compatibility of their system, the user can now generate requirement based on defined boilerplate and data for "Compatibility" module.



```
"compatibilityData": {
  "operatingSystem": [
    {
      "operatingSystem": "Microsoft Windows",
      "version": "",
      "issue": ""
    },
    {
      "operatingSystem": "Linux",
      "version": "",
      "issue": ""
    },
    {
      "operatingSystem": "MacOS",
      "version": "",
      "issue": ""
    }
  ],
```

Figure 6.3: JSON data for Compatibility Module

In Figure 6.3, the extracted JSON data "Compatibility" module are shown. The data indicates that there are 3 specified operating system, which are "Microsoft Windows", "Linux" and "MacOS" and each of them does not have any version or issues specified.

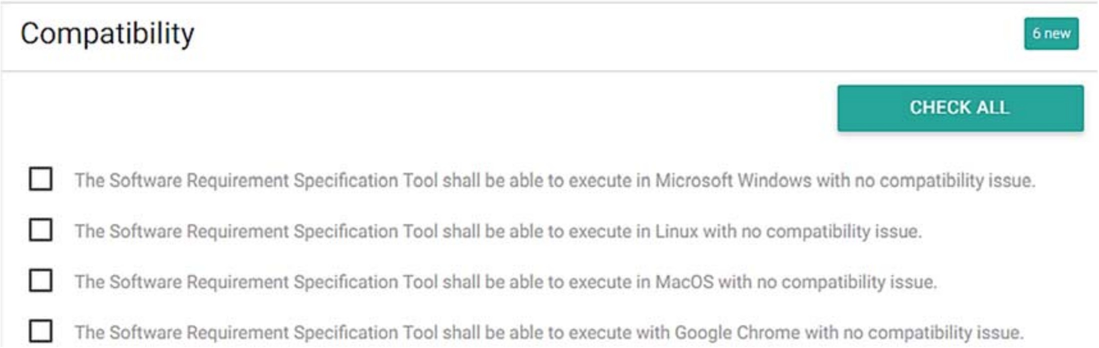


```
"compatibility": {
  "operatingSystem": {
    "0": "The <system> shall be able to execute in <operatingSystem> with no compatibility issue",
    "1": "The <system> shall be able to execute in <operatingSystem> <version> and above with no compatibility issue",
    "2": "The <system> shall be able to execute in <operatingSystem> with <issue>",
    "3": "The <system> shall be able to execute in <operatingSystem> <version> and above with <issue>"
  },
  "executionEnvironment": {
    "0": "The <system> shall be able to execute with <software> with no compatibility issue",
    "1": "The <system> shall be able to execute with <software> <version> and above with no compatibility issue",
    "2": "The <system> shall be able to execute with <software> with <issue>",
    "3": "The <system> shall be able to execute with <software> <version> and above with <issue>"
  },
  "outputCompatibility": {
    "true": "The <output> of the <system> <newVersion> shall be compatible with <oldVersion>",
    "false": "The <output> of the <system> <newVersion> shall not be compatible with <oldVersion>"
  }
},
```

Figure 6.4: JSON data for Compatibility Module boilerplate

In Figure 6.4, the boilerplate data for Compatibility module is shown. These boilerplate will be used along with JSON data from Figure 6.3 to generate requirements for user.

As a result, the requirement generated based on boilerplate and module data are shown in Figure 6.5.



□ The Software Requirement Specification Tool shall be able to execute in Microsoft Windows with no compatibility issue.

□ The Software Requirement Specification Tool shall be able to execute in Linux with no compatibility issue.

□ The Software Requirement Specification Tool shall be able to execute in MacOS with no compatibility issue.

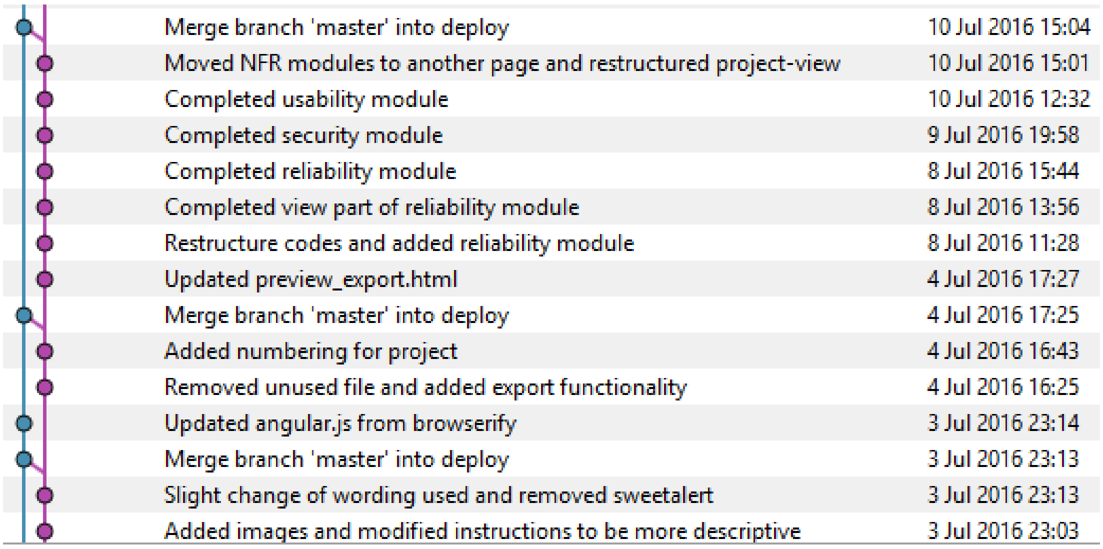
□ The Software Requirement Specification Tool shall be able to execute with Google Chrome with no compatibility issue.

Figure 6.5: Generated requirement for Compatibility Module

6.2. Version Control System

In order to keep trace of changes made to source code, we used Git version control system. We created a private repository which only accessible by ourselves and pushed any changes to the repository. This will ensure that we always have a backup on the server and can freely to make any changes as long as we committed latest code to the server.

In our repository there are two branches, which are master branch and deploy branch. As the name suggests, master branch contains the master copy of our source code. Any latest change will uploaded instantly to master branch. In contrast, deploy branch is only updated when changes made in master branch is stabled. Deploy branch's source code must be ensured of minimum bug free and stable as it will be uploaded to the web hosting server.



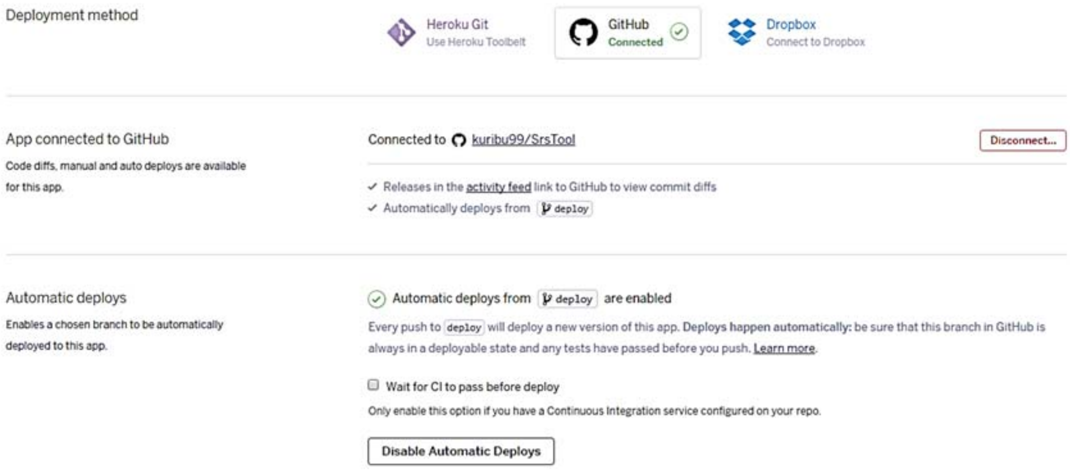
Merge branch 'master' into deploy	10 Jul 2016 15:04
Moved NFR modules to another page and restructured project-view	10 Jul 2016 15:01
Completed usability module	10 Jul 2016 12:32
Completed security module	9 Jul 2016 19:58
Completed reliability module	8 Jul 2016 15:44
Completed view part of reliability module	8 Jul 2016 13:56
Restructure codes and added reliability module	8 Jul 2016 11:28
Updated preview_export.html	4 Jul 2016 17:27
Merge branch 'master' into deploy	4 Jul 2016 17:25
Added numbering for project	4 Jul 2016 16:43
Removed unused file and added export functionality	4 Jul 2016 16:25
Updated angular.js from browserify	3 Jul 2016 23:14
Merge branch 'master' into deploy	3 Jul 2016 23:13
Slight change of wording used and removed sweetalert	3 Jul 2016 23:13
Added images and modified instructions to be more descriptive	3 Jul 2016 23:03

Figure 6.6: Example of Git commits and merges

As shown in Figure 6.6, the line on the left represents Deploy branch, while the branch on the right represents Master branch. The first few top commits shows that development of usability, security and reliability module was done in Master branch before merged into Deploy branch.

6.3. Automated Deployment

In addition to version control system, we also utilized cloud platform and services and to perform automated deployment of our website. This is done by connecting our GitHub repository to Heroku, a cloud application platform.



Deployment method

Heroku Git
Use Heroku Toolbelt

GitHub
Connected ✓

Dropbox
Connect to Dropbox

App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to kuribu99/SrsTool

✓ Releases in the activity feed link to GitHub to view commit diffs
✓ Automatically deploys from `deploy`

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

✓ Automatic deploys from `deploy` are enabled

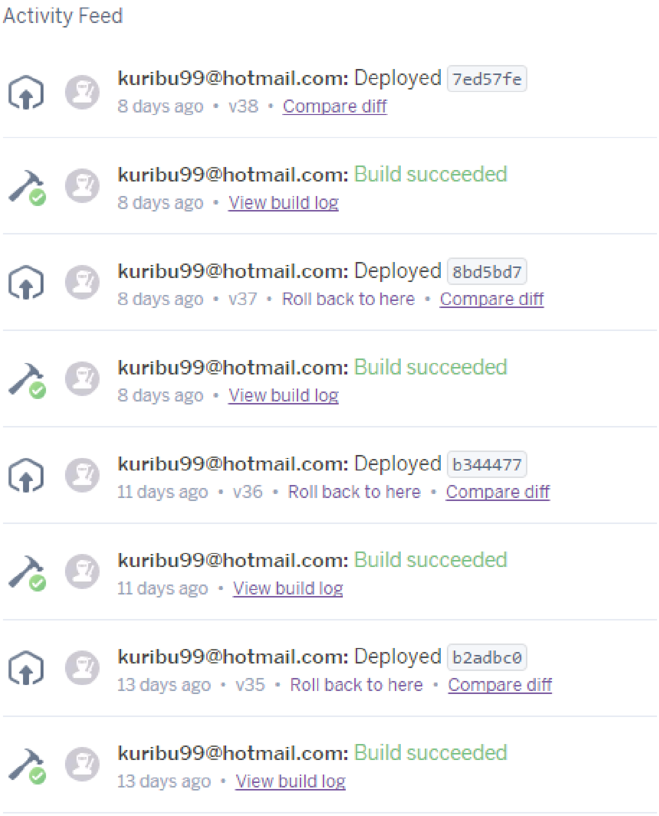
Every push to `deploy` will deploy a new version of this app. Deploys happen automatically: be sure that this branch in GitHub is always in a deployable state and any tests have passed before you push. Learn more.

□ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Figure 6.7: Deployment configuration in Heroku

As shown in Figure 6.7, we configured Heroku to automatically update and deploy from Deploy branch of our repository every time we made changes to it. For example, Figure 6.8 shows some activities that triggered by changes in deploy branch.



Activity Feed

kuribu99@hotmail.com: Deployed `7ed57fe`
8 days ago • v38 • Compare diff

kuribu99@hotmail.com: Build succeeded
8 days ago • View build log

kuribu99@hotmail.com: Deployed `8bd5bd7`
8 days ago • v37 • Roll back to here • Compare diff

kuribu99@hotmail.com: Build succeeded
8 days ago • View build log

kuribu99@hotmail.com: Deployed `b344477`
11 days ago • v36 • Roll back to here • Compare diff

kuribu99@hotmail.com: Build succeeded
11 days ago • View build log

kuribu99@hotmail.com: Deployed `b2adbc0`
13 days ago • v35 • Roll back to here • Compare diff

kuribu99@hotmail.com: Build succeeded
13 days ago • View build log

Figure 6.8: Example of Deployment activities

CHAPTER 7

TESTING AND EVALUATION

7.1. Testing and Evaluation Strategy

To evaluate our completed tool, we conducted survey on some undergraduates and graduates. In order to collect the most representative result from undergraduate, we restrict that the participant must be conducting or had conducted at least one project, in which the project includes a proper requirement phase. This will ensure that our participant has experience of specifying requirement in order to make a contrast between requirement specification with or without a tool.

Before conducting the survey, the participant is required to use our tool to specify one of their project's requirements. Guidance and instructions will be provided when necessary to the participant so that they can understand how to use the tool within limited allocated time. Lastly, the participant will fill in a survey form which evaluates the tool from 5 aspects:

1. User's personal experience in conducting software projects
2. User's feedback on the functionality aspect of the tool
3. User's feedback on the usability and user interface aspect of the tool
4. Measurements based on user's project's requirement specification
5. User's personal opinion regarding the tool

From the result of this survey, we will expect to be able to answer the following questions:

1. What are the popular methods used by user to specify requirements?
2. Do user know generally knows what is requirement boilerplate?
3. How do user feel about using requirement boilerplate to specify requirements?
4. Are the functionalities of the tool complete and suitable?
5. Are the requirement boilerplates of the tool appropriate?
6. Does the tool provide good user interface and experience?
7. What are the approximate figure for new functional and non-functional requirements specified using the tool?
8. How much coverage does the tool provided to specify existing requirements?
9. How efficient is the tool as compared to original method of requirement specification?
10. Does the tool triggers user to specify more non-functional requirement?

Please refer to Appendix C: Feedback Survey Form for the printed copy of the survey.

7.2. Testing and Evaluation Result

Over a period of 18 days, we had conducted our evaluation on our tool on 14 participants, consisting of 13 undergraduates and 1 graduate. In this section, we will be discussing and analysing the result of the survey.

For all data presented in this section, a summary of all data can be referred from Appendix D: Feedback Survey Result. In Section D: Measurement, we requested participant to deduce the number of requirements based on the ratio of how many requirement they specified and how many requirements actually is in the project.

For example, if the participant has 100 requirements in their project and they used the tool to specify only 20, and they successfully specified 15 out of the 20 requirements, they will deduce that they will successfully specify 75 requirements and fail to specify 25 requirements.

On average, each participant took 30 minutes to attempt to specify part of the requirements of one of their project using our tool and another 10 minute to complete the survey. Participant will attempt to specify approximately 20% of their project's requirement due to time constraint. Final year project participants were more enthusiastic when specifying their requirements as compared to other participants.

Table 7.1 shows the actual figure from the collected feedbacks. Experienced user are participants who conducted at least 5 projects while normal user are those who conducted between 1 to 5 projects. All participants had at least conducted 1 project.

The value of Table 7.1 are represented in few formats: (1) Plain numbers, representing actual figure, (2) Percentage, representing percentage of user, (3) [Number/Number], representing [Score/Maximum score].

Table 7.1: Feedback Summary

Aspect	User	
	Normal	Experienced
Section A: Experience		
Number of participants	10	4
Mainly uses natural language sentences to specify requirements	100%	75%
Mainly uses Microsoft Word to specify requirements	100%	50%
Uses collaborative tool to specify requirements	0%	50%
Have prior knowledge about boilerplate	0%	25%

Section B: Functionality		
The tool provides sufficient feature	8.1/10.0	7.5/10.0
The tool provides sufficient module to specify requirements	9.0/10.0	7.5/10.0
The modules are appropriate and suitable	7.8/10.0	7.8/10.0
The predefined boilerplates are appropriate	7.5/10.0	6.3/10.0
Section C: User Interface and Experience		
The UI is consistent	4.1/5.0	4.3/5.0
The UI is well designed	3.2/5.0	3.8/5.0
The UI shows overall process flow of using the tool	3.7/5.0	3.8/5.0
The tool is easy to learn	3.1/5.0	3.3/5.0
The tool is interactive and fun	6.9/10.0	6.8/10.0
Section D: Measurements		
Average number of FR before using tool	18	10
Average number of NFR before using tool	8	10
Average number of FR after using tool	25	10
Average number of NFR after using tool	16	11
Average number of requirement failed to specify with tool	2	1
Average number of new requirement specified	14	2
Average time used to specify requirement without tool	57 minutes	33 minutes
Average time used to specify requirement with tool	28 minutes	16 minutes
The tool helps speed up the requirement specification process	8.4/10.0	7.5/10.0
Section E: Personal Opinion		
The description and instruction is sufficient	7.7/10.0	5.5/10.0
The tool helps to specify more NFR	8.9/10.0	8.0/10.0
Boilerplate helps user in requirement specification	7.7/10.0	6.8/10.0
Will use this tool to specify requirement in future	90%	100%

As a comparison, we found that most participants are normal users. Normal users mainly uses natural language sentences to specify requirements. They do not use collaborative tools such as Google Docs or Trello to specify requirements but only uses Microsoft Words to do so. Normal users do not know the existence of requirement boilerplate.

In contrast, experienced user are exposed to some other requirement specification techniques such as formal notation and use cases. Some of them uses collaborative tool due to working environment in a team and some had experience dealing with requirement boilerplate.

On average, both types of user think that the functionality of the tool is quite appropriate and sufficient. Normal users rates the functionality of the tool slightly better than experienced user, while experienced user liked more on the user interface and user experience of the tool. However, both types of user remains neutral about the learnability of the tool. In general, they think the tool is quite interactive and fun than their current method of requirement specification but not easy to learn.

In terms of measurement, normal users specifies about 2 times of the number of requirements than an experienced user. Only about 10% of existing requirements were failed to be specified using the tool. Users specifies up to 50% more new requirements when using the tool and only used about half of the original amount of time needed to specify requirements. All users agreed that the tool speeds up the requirement specification process.

Lastly, experienced user thinks that the description and instructions given by the tool are just enough as compared to normal user who thinks that they are quite sufficient. All users agreed that the tool helps them to specify non-functional requirements and boilerplate are quite useful to assist requirement specification phases. Almost all users are keen to reuse the tool to specify requirement in future.

To summarize the whole evaluation result, we constructed Table 7.2 and calculated the average score of each section to represent user’s satisfaction. We also evaluated the effectiveness and efficiency of our tool by using measurements given by users in their feedback. The term "$f(x)$" refers to the formula of calculation.

Table 7.2: Evaluation Summary

Aspect	**User**	
	Normal	**Experienced**
User satisfaction level on **Functionality** of the system $f(x) = Avg.score\ of\ Section\ B\ x\ 100\%$	81%	73%
User satisfaction level on **User Interface Design and Experience** of the system $f(x) = Avg.score\ of\ Section\ C\ x\ 100\%$	70%	74%
Overall user satisfaction level $f(x) = Average\ of\ Section\ B\ and\ C$	76%	74%
Effectiveness of the tool (% of improvement) $f(x) = \left(\frac{Avg.Req.with\ tool}{Avg.Req.without\ tool} - 1\right) x\ 100\%$	+58%	+5%
Efficiency of the tool (% of improvement) $f(x) = (\frac{Avg.Time\ with\ tool}{Avg.Time\ without\ tool} - 1)\ x\ 100\%$	+104%	+106%

CHAPTER 8

CONCLUSION AND DISCUSSIONS

8.1. Conclusion

As a conclusion, throughout the period of 7 months from 18th January 2016 till 19th August 2016, we had completed a software project titled “Software Requirement Specification Tool”. We had fulfilled each of our project objectives in accordance to each chapter in this report:

1. Proposing to conduct this project by preparing project proposal (Please refer to Chapter 1: Introduction)
2. Reviewing every aspect of this project through literature studies (Please refer to Chapter 2: Literature Review)
3. Planning and deciding the methodology to be used to conduct the project (Please refer to Chapter 3: Methodology)
4. Specifying and modelling requirements that shall be fulfilled in this project (Please refer to Chapter 4: Requirement Specification)
5. Analysing and designing each aspect of the tool of the project (Please refer to Chapter 5: Design)
6. Coding and implementing the project to produce our proposed tool (Please refer to Chapter 6: Coding and Implementation)
7. Testing and evaluating the effectiveness and other aspect of our produced tool (Please refer to Chapter 7: Testing and Evaluation)

8.2. Limitations

Despite successfully fulfilling all our objectives in this project, there are some limitation to the tool that we had completed due to other factors such as time and scope. The following is the list of limitations that our tool have after researching and comparing:

1. We only focused on developing functionalities to assist user in requirement specification and do not include other aspect of requirement engineering, such as requirement prioritization and management

2. Due to constraint of time, we had limited our project scope to support only requirement specification for user requirements or system requirements and focused more to improve quality of requirements by specifying non-functional requirements based on ISO 25010 model. However, a complete software requirement specification (SRS) actually includes more than solely functional and non-functional user requirements and system requirements

3. When proposing the development of this project, we researched and found out that ontology and domain model could enhance and support requirement specification. However, in order to generate a correct and validated ontology, a lot of research, time and effort will need to be done. Within our timeframe, we did not manage to include ontology as the semantic aspect for our tool and we only can rely on the user to validate the requirements and ensure the completeness of requirement based on their own specification and as per defined in ISO model.

8.3. Future Improvement Roadmap

As a closure for this project, we had planned a roadmap for future improvement of this tool. The following features are not included in our project, but it will be very helpful to be added to overcome the limitations of this tool.

1. To improve this tool, we may add in other functionalities such as allow user to prioritize requirements after specifying it, providing requirement traceability matrix for user and etc. This will make our tool a full-stack requirement engineering tool which supports all generic workflow of requirement engineering phases.

2. In our tool, we mainly focused on elicitation and specification of user requirement and system requirement in our tool. However in real life projects, there are much more other types of requirement to be considered in order to produce a complete software requirement specification. Hence, it is suggested to support the specification of other types of requirement in order to make this tool complete.

3. We propose to integrate ontology validated by experts to help user to specify requirements. This will generate a correct and complete software requirement specification.

4. From our survey, we found that experienced user tend to use collaborative tool in their requirement specification process. This suggested that in the industry, requirement specification process are most likely conducted by few persons. Hence, we propose to enhance the collaborative aspect of our tool.

BIBLIOGRAPHY

Anon, 2015. *The Pulse of Profession - Capturing the Value of Project Management*, Pennsylvania. Available at: http://www.pmi.org/~/media/PDF/learning/pulse-of-the-profession-2015.ashx.

Arora, C. et al., 2014. Requirement boilerplates: Transition from manually-enforced to automatically-verifiable natural language patterns. *2014 IEEE 4th International Workshop on Requirements Patterns, RePa 2014 - Proceedings*, pp.1–8.

Azuma, M., 2004. Applying ISO/IEC 9126-1 Quality Model to quality requirements engineering on critical software. *Proceedings of the 3rd IEEE Int. Workshop on Requirements for High Assurance Systems (RHAS).*

Al Balushi, T.H., Sampaio, P.R.F. & Loucopoulos, P., 2013. Eliciting and prioritizing quality requirements supported by ontologies: A case study using the ElicitO framework and tool. *Expert Systems*, 30(2), pp.129–151.

Bourque, P. & Fairley, R.E., 2014. *Guide to the Software Engineering - Body of Knowledge.*, Available at: www.swebok.org.

Bures, T. et al., 2012. Requirement Specifications Using Natural Languages. , (December). Available at: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2012-05.pdf.

Christel, M.G. & Kang, K.C., 1992. *Issues in Requirements Elicitation*, Pennsylvania.

Clancy, T., 1995. *The Standish group: the chaos report*, Dorfman, M. & Thayer, R.H., 1990. *Standards, guidelines, and examples on system and software requirements engineering*, IEEE Computer Society Press. Available at: https://books.google.com.my/books?id=qbNQAAAAMAAJ.

Firesmith, D., 2007. Common requirements problems, their negative consequences, and the industry best practices to help solve them. *Journal of Object Technology*, 6(1), pp.17–33.

Hazeem, T. et al., 2007. ElicitO : A Quality Ontology-Guided NFR Elicitation Tool. , (1), pp.306–319.

Hull, E., Jackson, K. & Dick, J., 2011. *Requirements Engineering* Third., London: Springer London. Available at: http://link.springer.com/10.1007/978-1-84996-405-0.

Ibrahim, N. et al., 2015. Applicability and Usablity of Predefined Natural Language Boilerplates in Documenting Requirements. , pp.127–137.

IEEE, 1998. IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std 830-1998*, pp.1–40.

International Organization For Standardization ISO, 2011. *ISO/IEC 25010: 2011*, Available at: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733.

International Organization For Standardization Iso, 2001. ISO/IEC 9126-1. *Software Process: Improvement and Practice*, 2(1), pp.1–25. Available at: http://ebookbrowse.com/iso-iec-9126-1-2001-pdf-d72715451.

ISO/IEC & IEEE, 2010. ISO/IEC/IEEE 24765:2010 - Systems and software engineering - Vocabulary. *Iso/Iec Ieee*, 2010, p.410. Available at: http://www.iso.org/iso/catalogue_detail.htm?csnumber=50518.

Kotonya, G. & Sommerville, I., 1998. *Requirements Engineering: Processes and Techniques*, J. Wiley. Available at: https://books.google.com.my/books?id=Up1QAAAAMAAJ.

Kruchten, P., 2004. *The Rational Unified Process: An Introduction*, Addison-Wesley. Available at: https://books.google.com.my/books?id=RYCMx6o47pMC.

McCall, J.A., Richards, P.K. & Walters, G.F., 1977. Factors in Software Quality - Volume 1 - Concept and Definitions of Software Quality. *Defense Technical Information Center*, 1, 2 and 3(ADA049014), p.168. Available at: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA049014.

Miguel, J.P., Mauricio, D. & Rodríguez, G., 2014. A Review of Software Quality Models for the Evaluation of Software Products. *International Journal of Software Engineering & Applications (IJSEA)*, 5(6), pp.31–53.

Neill, C.J. & Laplante, P. a., 2003. Requirements engineering: The state of the practice. *IEEE Software*, 20(6), pp.40–45. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1241365.

Nuseibeh, B. & Easterbrook, S., 2000. Requirements engineering: a roadmap. *Proceedings of the conference on The future of Software engineering - ICSE '00*, 1, pp.35–46. Available at: http://portal.acm.org/citation.cfm?doid=336512.336523.

Pohl, K. & Rupp, C., 2015. *Requirements Engineering Fundamentals A Study Guide for the Certified Professional for Requirements Engineering Exam*,

Pressman, R.S., 2009. *Software Engineering A Practitioner's Approach 7th Ed - Roger S. Pressman*, Available at: http://www.amazon.com/s/ref=nb_sb_noss?url=search-alias%3Daps&field-keywords=9780073375977.

Sommerville, I., 2011. *Software Engineering* Ninth Edit., Pearson. Available at: https://books.google.com.my/books?id=l0egcQAACAAJ.

Sparx Systems, 2010. Requirements Management with Enterprise Architect. , pp.1–50. Available at: http://www.sparxsystems.com/uml_tool_guide/modeling_tool_features/requirements_model_pattern.htm\npapers2://publication/uuid/DAB1BC3A-BB0B-4BD8-87C4-864921842E1A.

Thapar, S., Singh, P. & Rani, S., 2012. Challenges to the Development of Standard Software Quality Model. *International Journal of Computer Applications*, 49(10), pp.1–7. Available at: http://www.ijcaonline.org/archives/volume49/number10/7660-0765.

Vera, H. et al., 2015. Data modeling for NoSQL document-oriented databases. *CEUR Workshop Proceedings*, 1478, pp.129–135.

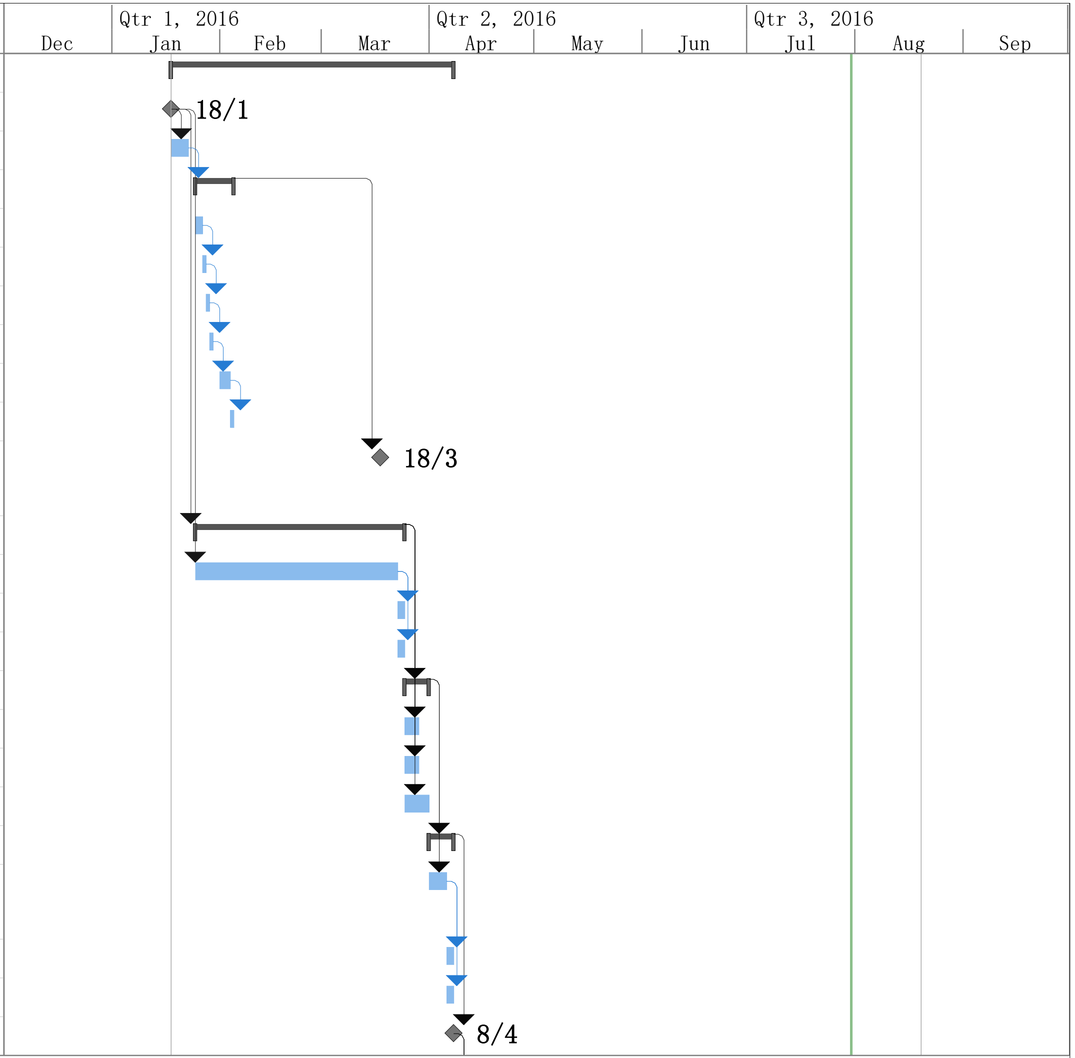
Yang, H. et al., 2011. Analysing anaphoric ambiguity in natural language requirements, in: Requirement Engineering. , pp.163–189.

Zave, P., 1995. Classification of research efforts in requirements engineering. *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)*, 29(4), pp.315–321.

APPENDICES

Appendix A: Work breakdown structure and Gantt chart

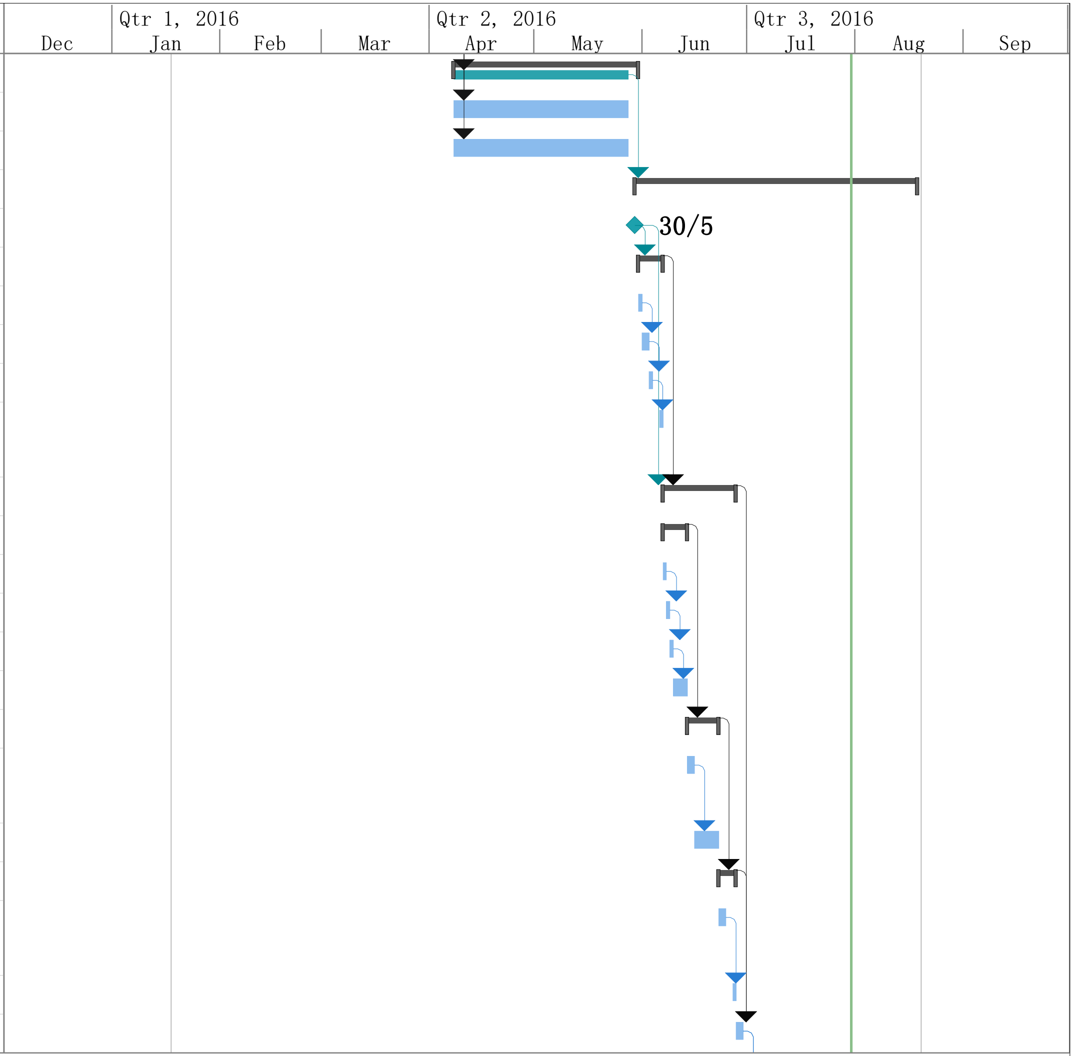
ID	Task Name	Duration	Start	Finish
1	**Project 1**	**59 days**	**Mon 18/1/16**	**Fri 8/4/16**
2	Project 1 start date	0 days	Mon 18/1/16	Mon 18/1/16
3	Title selection	5 days	Mon 18/1/16	Fri 22/1/16
4	**Proposal preparation**	**9 days**	**Mon 25/1/16**	**Thu 4/2/16**
5	Problem statement	2 days	Mon 25/1/16	Tue 26/1/16
6	Project goal	1 day	Wed 27/1/16	Wed 27/1/16
7	Project objectives	1 day	Thu 28/1/16	Thu 28/1/16
8	Proposed solution	1 day	Fri 29/1/16	Fri 29/1/16
9	Project scope	3 days	Mon 1/2/16	Wed 3/2/16
10	Project approach	1 day	Thu 4/2/16	Thu 4/2/16
11	Project 1 preliminary report submission	0 days	Fri 18/3/16	Fri 18/3/16
12	**Literature review**	**44 days**	**Mon 25/1/16**	**Thu 24/3/16**
13	Background review	42 days	Mon 25/1/16	Tue 22/3/16
14	Methodology review	2 days	Wed 23/3/16	Thu 24/3/16
15	Approach review	2 days	Wed 23/3/16	Thu 24/3/16
16	**Methodology**	**5 days**	**Fri 25/3/16**	**Thu 31/3/16**
17	Chosen methodology	2 days	Fri 25/3/16	Mon 28/3/16
18	Chosen tool	2 days	Fri 25/3/16	Mon 28/3/16
19	Project plan	5 days	Fri 25/3/16	Thu 31/3/16
20	**Project Specification**	**5 days**	**Fri 1/4/16**	**Thu 7/4/16**
21	Software requirement specification	3 days	Fri 1/4/16	Tue 5/4/16
22	Use case modelling	2 days	Wed 6/4/16	Thu 7/4/16
23	Preliminary UI design	2 days	Wed 6/4/16	Thu 7/4/16
24	Project 1 report submission	0 days	Fri 8/4/16	Fri 8/4/16



Project: Project Plan
Date: Sun 31/7/16

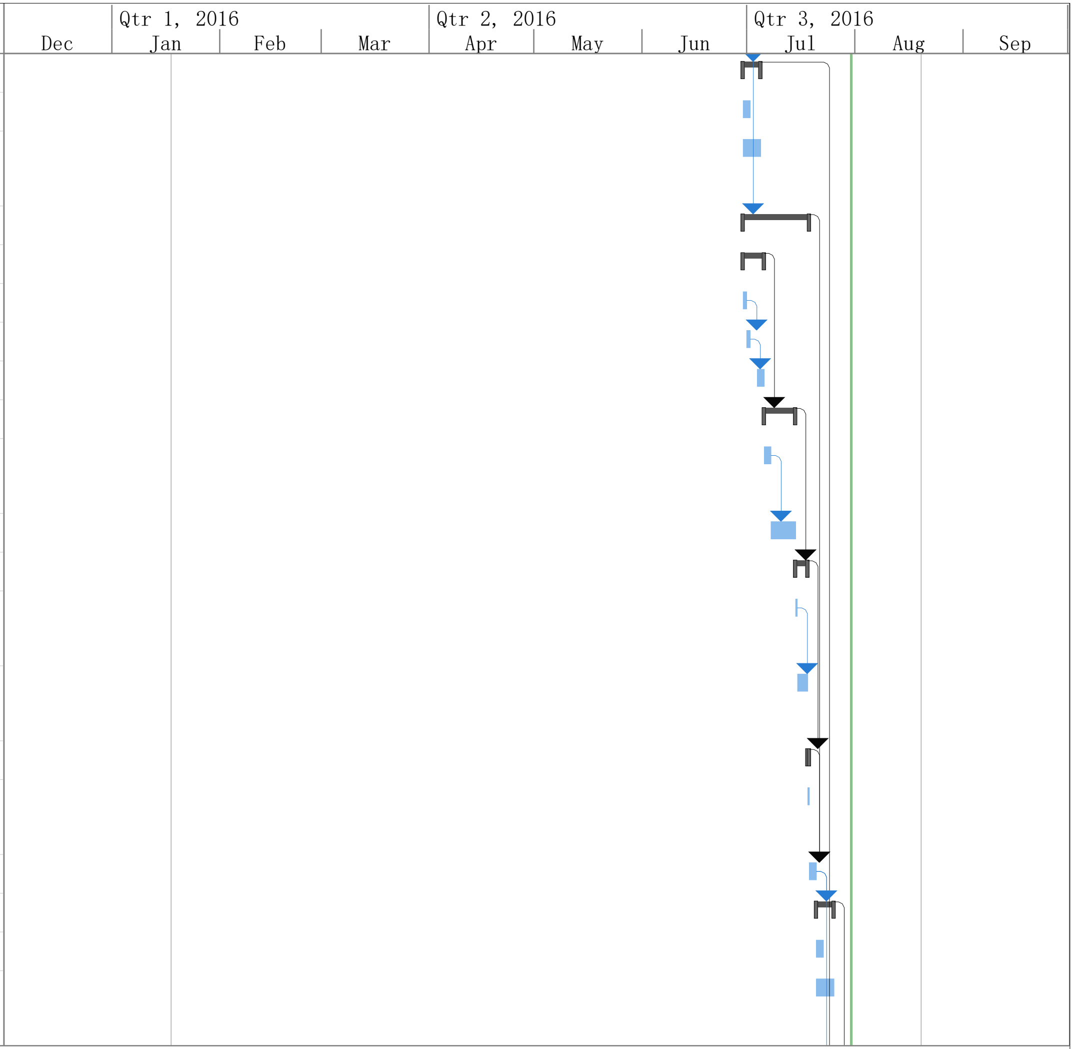
Task	▬	Inactive Summary	▭	External Tasks	▬
Split	┈	Manual Task	▬	External Milestone	◇
Milestone	◆	Duration-only	▬	Deadline	⇩
Summary	┏━┓	Manual Summary Rollup	▬	Progress	━
Project Summary	┏━┓	Manual Summary	┏━┓	Manual Progress	━
Inactive Task	▭	Start-only	[		
Inactive Milestone	◇	Finish-only	]		

ID	Task Name	Duration	Start	Finish
25	**Self exploration**	**37 days**	**Fri 8/4/16**	**Mon 30/5/16**
26	Tool exploraration	36 days	Fri 8/4/16	Fri 27/5/16
27	API exploration	36 days	Fri 8/4/16	Fri 27/5/16
28	**Project 2**	**59 days?**	**Mon 30/5/16**	**Fri 19/8/16**
29	Project 2 start date	0 days	Mon 30/5/16	Mon 30/5/16
30	**Project 1 review**	**5 days**	**Tue 31/5/16**	**Mon 6/6/16**
31	Proposal review	1 day	Tue 31/5/16	Tue 31/5/16
32	Literature review	2 days	Wed 1/6/16	Thu 2/6/16
33	Methodology review	1 day	Fri 3/6/16	Fri 3/6/16
34	Requirement specification finalization	1 day	Mon 6/6/16	Mon 6/6/16
35	**Iteration 1**	**15 days**	**Tue 7/6/16**	**Mon 27/6/16**
36	**Design**	**5 days**	**Tue 7/6/16**	**Mon 13/6/16**
37	Architecture Diagram	1 day	Tue 7/6/16	Tue 7/6/16
38	Component diagram	1 day	Wed 8/6/16	Wed 8/6/16
39	Activity Diagram	1 day	Thu 9/6/16	Thu 9/6/16
40	Sequence diagram	2 days	Fri 10/6/16	Mon 13/6/16
41	**Coding and Implementation**	**7 days**	**Tue 14/6/16**	**Wed 22/6/16**
42	UI design and implementation	2 days	Tue 14/6/16	Wed 15/6/16
43	Logic implementation	5 days	Thu 16/6/16	Wed 22/6/16
44	**Deployment**	**3 days**	**Thu 23/6/16**	**Mon 27/6/16**
45	Deployment method selection	2 days	Thu 23/6/16	Fri 24/6/16
46	Website Deployment	1 day	Mon 27/6/16	Mon 27/6/16
47	Review for Iteration 1	2 days	Tue 28/6/16	Wed 29/6/16



Task	▬	Inactive Summary	▭	External Tasks	▬
Split	┅	Manual Task	▬	External Milestone	◇
Milestone	◆	Duration-only	▬	Deadline	⇩
Summary	▬	Manual Summary Rollup	▬	Progress	─
Project Summary	▬	Manual Summary	▬	Manual Progress	─
Inactive Task	▭	Start-only	[		
Inactive Milestone	◇	Finish-only	]		

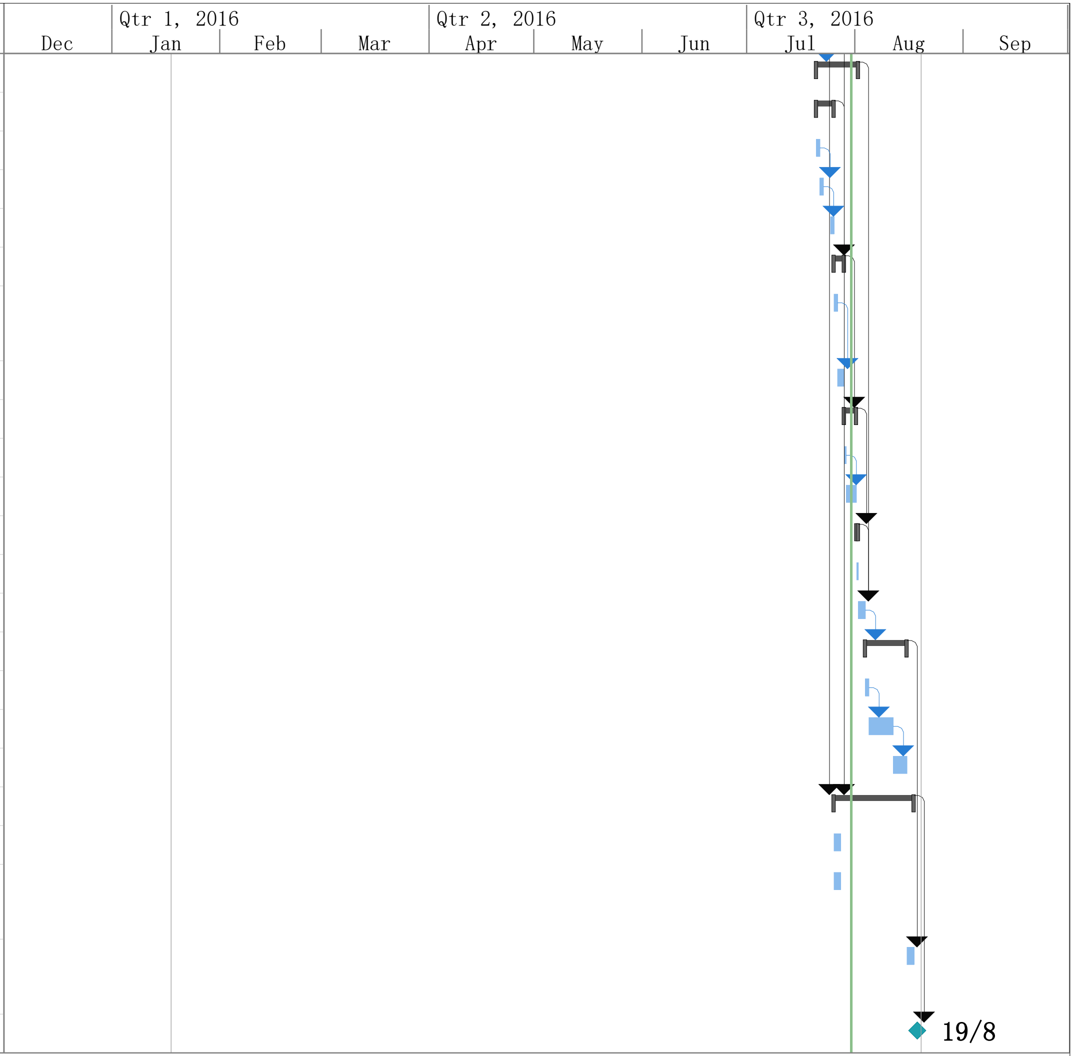
ID	Task Name	Duration	Start	Finish
48	**Report compilation**	**3 days**	**Thu 30/6/16**	**Mon 4/7/16**
49	Prepare Chapter 5 - Design	2 days	Thu 30/6/16	Fri 1/7/16
50	Prepare Chapter 6 - Coding and Implementation	3 days	Thu 30/6/16	Mon 4/7/16
51	**Iteration 2**	**13 days**	**Thu 30/6/16**	**Mon 18/7/16**
52	**Design**	**4 days**	**Thu 30/6/16**	**Tue 5/7/16**
53	Component diagram	1 day	Thu 30/6/16	Thu 30/6/16
54	Activity Diagram	1 day	Fri 1/7/16	Fri 1/7/16
55	Sequence diagram	2 days	Mon 4/7/16	Tue 5/7/16
56	**Coding and Implementation**	**7 days**	**Wed 6/7/16**	**Thu 14/7/16**
57	UI design and implementation	2 days	Wed 6/7/16	Thu 7/7/16
58	Logic implementation	5 days	Fri 8/7/16	Thu 14/7/16
59	**Testing**	**1.5 days**	**Fri 15/7/16**	**Mon 18/7/16**
60	Preliminary User Acceptance Test	0.5 days	Fri 15/7/16	Fri 15/7/16
61	Testing feedback integration	1 day	Fri 15/7/16	Mon 18/7/16
62	**Deployment**	**0.5 days**	**Mon 18/7/16**	**Mon 18/7/16**
63	Update deployed website	0.5 days	Mon 18/7/16	Mon 18/7/16
64	Review for Iteration 2	2 days	Tue 19/7/16	Wed 20/7/16
65	**Report compilation**	**3 days**	**Thu 21/7/16**	**Mon 25/7/16**
66	Prepare Chapter 5 - Design	2 days	Thu 21/7/16	Fri 22/7/16
67	Prepare Chapter 6 - Coding and Implementation	3 days	Thu 21/7/16	Mon 25/7/16



Project: Project Plan
Date: Sun 31/7/16

Task	▬	Inactive Summary	▭	External Tasks	▬
Split	┈┈┈	Manual Task	▬	External Milestone	◇
Milestone	◆	Duration-only	▬	Deadline	⇩
Summary	┌─┐	Manual Summary Rollup	▬	Progress	─
Project Summary	┌─┐	Manual Summary	┌─┐	Manual Progress	─
Inactive Task	▭	Start-only	[		
Inactive Milestone	◇	Finish-only	]		

ID	Task Name	Duration	Start	Finish
68	**Iteration 3**	**8 days**	**Thu 21/7/16**	**Mon 1/8/16**
69	**Design**	**3 days**	**Thu 21/7/16**	**Mon 25/7/16**
70	Component diagram	1 day	Thu 21/7/16	Thu 21/7/16
71	Activity Diagram	1 day	Fri 22/7/16	Fri 22/7/16
72	Sequence diagram	1 day	Mon 25/7/16	Mon 25/7/16
73	**Coding and Implementation**	**3 days**	**Tue 26/7/16**	**Thu 28/7/16**
74	UI design and implementation	1 day	Tue 26/7/16	Tue 26/7/16
75	Logic implementation	2 days	Wed 27/7/16	Thu 28/7/16
76	**Testing**	**1.5 days**	**Fri 29/7/16**	**Mon 1/8/16**
77	Final User Acceptance Test	0.5 days	Fri 29/7/16	Fri 29/7/16
78	Testing feedback integration	1 day	Fri 29/7/16	Mon 1/8/16
79	**Deployment**	**0.5 days**	**Mon 1/8/16**	**Mon 1/8/16**
80	Update deployed website	0.5 days	Mon 1/8/16	Mon 1/8/16
81	Review for Iteration 3	2 days	Tue 2/8/16	Wed 3/8/16
82	**Product Evaluation**	**8 days**	**Thu 4/8/16**	**Mon 15/8/16**
83	Survey form preparation	1 day	Thu 4/8/16	Thu 4/8/16
84	Conduct survey	5 days	Fri 5/8/16	Thu 11/8/16
85	Survey result analysis	2 days	Fri 12/8/16	Mon 15/8/16
86	**Report compilation**	**17 days**	**Tue 26/7/16**	**Wed 17/8/16**
87	Complete Chapter 5 - Design	2 days	Tue 26/7/16	Wed 27/7/16
88	Complete Chapter 6 - Coding and Implementation	2 days	Tue 26/7/16	Wed 27/7/16
89	Complete Chapter 7 - Testing and Evaluation	2 days	Tue 16/8/16	Wed 17/8/16
90	Project 2 submission	0 days	Fri 19/8/16	Fri 19/8/16



Project: Project Plan
Date: Sun 31/7/16

Task	▬	Inactive Summary	▭	External Tasks	▬
Split	┈	Manual Task	▬	External Milestone	◇
Milestone	◆	Duration-only	▬	Deadline	⬇
Summary	▬	Manual Summary Rollup	▬	Progress	▬
Project Summary	▬	Manual Summary	▬	Manual Progress	▬
Inactive Task	▭	Start-only	[		
Inactive Milestone	◇	Finish-only	]		

Appendix B: Use Case Descriptions

Use Case Description – Login

Use Case Name: **Login**	ID: **001**	Importance Level: **High**
Primary Actor: **Requirement engineer**	Use Case Type: **Detail, Essential**	
Stakeholders and Interests: - **Requirement engineer: wants to login to the system to use the system**		
Brief Description: **This use case describes how requirement engineer login to the system**		
Trigger: **Requirement engineer clicks on Login button.** Type: **-**		
Relationships: Association: **Requirement Engineer** Include: Extend: Generalization:		

Normal Flow of Events:

User	System
1. User clicks on Login button	**2. System shows dialog to enter required information to login**
3. User enters required information	**4. System validates the user information and logs user into the system**

Alternate/Exceptional Flows:

User	System
3.1.1 User entered wrong information.	**3.1.2 System prompt user that information was incorrect and request user to try again.**

Use Case Description – Create Project

<table>
<tr><td colspan="2">Use Case Name: **Create Project**</td><td>ID: **002**</td><td>Importance Level: **High**</td></tr>
<tr><td colspan="2">Primary Actor: **Requirement engineer**</td><td colspan="2">Use Case Type: **Detail, Essential**</td></tr>
<tr><td colspan="4">Stakeholders and Interests:
- **Requirement engineer: wants to create a project to specify their requirements**</td></tr>
<tr><td colspan="4">Brief Description: **This use case describes how requirement engineer create a project**</td></tr>
<tr><td colspan="4">Trigger: **Requirement engineer clicks on “Create Project” button**</td></tr>
<tr><td colspan="4">Relationships:
Association: **Requirement engineer**
Include:
Extend:
Generalization:</td></tr>
<tr><td colspan="4">**Normal Flow of Events:**</td></tr>
<tr><td>**User**</td><td colspan="3">**System**</td></tr>
<tr><td>**1. User clicks on “Create Project” button**</td><td colspan="3">**2. System shows dialog to prompt user for project name**</td></tr>
<tr><td>**3. User enters the project name**</td><td colspan="3">**4. System creates the project and opens the project for user**</td></tr>
<tr><td colspan="4">**Alternate/Exceptional Flows:**</td></tr>
<tr><td>**User**</td><td colspan="3">**System**</td></tr>
<tr><td></td><td colspan="3"></td></tr>
</table>

Use Case Description – Open Project

<table>
<tr><td colspan="2">Use Case Name: <b>Open Project</b></td><td>ID: <b>003</b></td><td>Importance Level: <b>High</b></td></tr>
<tr><td colspan="2">Primary Actor: <b>Requirement engineer</b></td><td colspan="2">Use Case Type: <b>Detail, Essential</b></td></tr>
<tr><td colspan="4">Stakeholders and Interests:
- <b>Requirement engineer: wants to open their project</b></td></tr>
<tr><td colspan="4">Brief Description: <b>This use case describes how requirement engineer open their project</b></td></tr>
<tr><td colspan="4">Trigger: <b>Requirement engineer clicks on the “Open Project” button of one of their project from their list of project</b></td></tr>
<tr><td colspan="4">Relationships:
Association: <b>Requirement engineer</b>
Include:
Extend:
Generalization:</td></tr>
<tr><td colspan="4"><b>Normal Flow of Events:</b></td></tr>
<tr><td><b>User</b></td><td colspan="3"><b>System</b></td></tr>
<tr><td><b>1. User clicks on “Open Project” button</b></td><td colspan="3"><b>2. System verifies that the user is logged in and opens the project</b></td></tr>
<tr><td colspan="4"><b>Alternate/Exceptional Flows:</b></td></tr>
<tr><td><b>User</b></td><td colspan="3"><b>System</b></td></tr>
<tr><td></td><td colspan="3"><b>2.1.1 User is not logged in.
System requests user to log in</b></td></tr>
</table>

Use Case Description – Remove Project

Use Case Name: **Remove Project**	ID: **004**	Importance Level: **High**
Primary Actor: **Requirement engineer**	Use Case Type: **Detail, Essential**	
Stakeholders and Interests: - **Requirement engineer: wants to remove project that is no longer needed**		
Brief Description: **This use case describes how requirement engineer remove project**		
Trigger: **Requirement engineer clicks on the “Remove Project” button of one of their project from their list of project**		
Relationships: Association: **Requirement engineer** Include: Extend: Generalization:		

Normal Flow of Events:	
User	**System**
1. User clicks on “Remove Project” button	**2. System prompts user for confirmation**
3. User clicks “Yes” to confirm deletion	**4. System verifies that user is logged in and removes the project from database**

Alternate/Exceptional Flows:	
User	**System**
3.1.1 User presses “Cancel” button	**3.1.2 System closes the dialog**
	3.2.1 User is not logged in. System requests user to log in

Use Case Description – Specify Functional Requirements

<table>
<tr><td colspan="2">Use Case Name: <b>Specify Functional Requirement</b></td><td>ID: <b>005</b></td><td>Importance Level: <b>High</b></td></tr>
<tr><td colspan="2">Primary Actor: <b>Requirement engineer</b></td><td colspan="2">Use Case Type: <b>Detail, Essential</b></td></tr>
<tr><td colspan="4">Stakeholders and Interests:
- <b>Requirement engineer: wants to specify functional requirement of their project</b></td></tr>
<tr><td colspan="4">Brief Description: <b>This use case describes how requirement engineer specify functional requirement</b></td></tr>
<tr><td colspan="4">Trigger: <b>Requirement engineer clicks on “Specify Requirements” menu tab, and then clicks on “Specify Functional Requirements” button</b></td></tr>
<tr><td colspan="4">Relationships:
Association: <b>Requirement engineer</b>
Include:
Extend:
Generalization:</td></tr>
<tr><td colspan="4"><b>Normal Flow of Events:</b></td></tr>
<tr><th>User</th><th colspan="3">System</th></tr>
<tr><td><b>1. User clicks on “Specify Functional Requirements” button</b></td><td colspan="3"><b>2. System shows user list of modules that can be used to specify functional requirement</b></td></tr>
<tr><td><b>3. User chooses a module and click on the module</b></td><td colspan="3"><b>4. System opens the module and request required information from user</b></td></tr>
<tr><td><b>5. User enters required information and press “Save” button</b></td><td colspan="3"><b>6. System saves the information to be used to generate requirement</b></td></tr>
<tr><td><b>7. User clicks “Back” button</b></td><td colspan="3"><b>8. System closes the module and return to previous UI</b></td></tr>
<tr><td colspan="4"><b>Alternate/Exceptional Flows:</b></td></tr>
<tr><th>User</th><th colspan="3">System</th></tr>
<tr><td><b>5.1.1 User presses “Cancel” button</b></td><td colspan="3"><b>5.1.2 System closes the dialog and return to previous UI without saving the entered information</b></td></tr>
</table>

Use Case Description – Specify Non-Functional Requirements

Use Case Name: **Specify Non-Functional Requirements**	ID: **006**	Importance Level: **High**
Primary Actor: **Requirement engineer**	Use Case Type: **Detail, Essential**	
Stakeholders and Interests: - **Requirement engineer: wants to specify non-functional requirement of their project**		
Brief Description: **This use case describes how requirement engineer specify non-functional requirement**		
Trigger: **Requirement engineer clicks on "Specify Requirements" menu tab, and then clicks on "Specify Non-Functional Requirements" button**		
Relationships: Association: **Requirement engineer** Include: Extend: Generalization:		

Normal Flow of Events:

User	System
1. User clicks on "Specify Non-Functional Requirements" button	**2. System shows user list of modules that can be used to specify non-functional requirement**
3. User chooses a module and click on the module	**4. System opens the module and request required information from user**
5. User enters required information and press "Save" button	**6. System saves the information to be used to generate requirement**
7. User clicks "Back" button	**8. System closes the module and return to previous UI**

Alternate/Exceptional Flows:

User	System
5.1.1 User presses "Cancel" button	**5.1.2 System closes the dialog and return to previous UI without saving the entered information**

Use Case Description – Generate Requirements

Use Case Name: **Generate Requirements**	ID: **007**	Importance Level: **High**
Primary Actor: **Requirement engineer**	Use Case Type: **Detail, Essential**	
Stakeholders and Interests: - **Requirement engineer: wants to generate requirements from the information they specified and saved in the system**		
Brief Description: **This use case describes how requirement engineer generate requirements**		
Trigger: **Requirement engineer clicks on "Specify Requirements" menu tab, and then clicks on "Generate Requirements" button**		
Relationships: Association: **Requirement engineer** Include: Extend: Generalization:		

Normal Flow of Events:

User	System
1. User clicks on "Generate Requirements" button	**2. System fetches information that user entered in requirement specification modules from the server.** **3. System uses defined boilerplates to generate requirement based on the information** **4. System shows all generated requirements to user**
5. User selects the requirement that they wanted to add to the Software Requirement Specification and clicks "Save" button	**6. System adds the selected requirement to the database and return to previous UI**

Alternate/Exceptional Flows:

User	System
5.1.1 User presses "Cancel" button	**5.1.2 System closes the dialog and return to previous UI**

Use Case Description – Export Software Requirement Specification

<table>
<tr><td colspan="2">Use Case Name: <b>Export Software Requirement Specification</b></td><td>ID: <b>008</b></td><td>Importance Level: <b>High</b></td></tr>
<tr><td colspan="2">Primary Actor: <b>Requirement engineer</b></td><td colspan="2">Use Case Type: <b>Detail, Essential</b></td></tr>
<tr><td colspan="4">Stakeholders and Interests:
- <b>Requirement engineer: wants to export current project’s Software Requirement Specification to other file format</b></td></tr>
<tr><td colspan="4">Brief Description: <b>This use case describes how requirement engineer export current project’s Software Requirement Specification to other file format</b></td></tr>
<tr><td colspan="4">Trigger: <b>Requirement clicks on “Export Requirements” button</b></td></tr>
<tr><td colspan="4">Relationships:
Association: <b>Requirement engineer</b>
Include:
Extend:
Generalization:</td></tr>
<tr><td colspan="4"><b>Normal Flow of Events:</b></td></tr>
<tr><td><b>User</b></td><td colspan="3"><b>System</b></td></tr>
<tr><td><b>1. User clicks on “Export Requirements” button</b></td><td colspan="3"><b>2. System shows dialog to choose file format to be exported</b></td></tr>
<tr><td><b>3. User chooses a format and clicks “Export”</b></td><td colspan="3"><b>4. System exports the SRS into selected file format and saves the file to user’s PC</b></td></tr>
<tr><td colspan="4"><b>Alternate/Exceptional Flows:</b></td></tr>
<tr><td><b>User</b></td><td colspan="3"><b>System</b></td></tr>
<tr><td><b>3.1.1 User presses “Back” button</b></td><td colspan="3"><b>3.1.2 System closes the dialog and return to previous UI</b></td></tr>
</table>

Appendix C: Feedback Survey Form

Software Requirement Specification Tool - Feedback Survey

Hi.
First of all, thank you for participating in this quick survey to evaluate my final year project product - Software Requirement Specification Tool.

The purpose of this survey is to gather feedback for my product which will be used to evaluate and improve this tool. All respondents must had attempted to specify one of their project's requirement using the tool before completing this survey.

The scope of evaluation includes the following 5 sections:
1. Your experience on conducting software projects
2. Your feedback on the functionality aspect of the tool
3. Your feedback on the usability and user interface aspect of the tool
4. Measurement based on the project's requirement specification
5. Your personal opinion regarding the tool

Your response will be kept confidential and used only to evaluate and to improve this tool. However, the feedback may be used for project submission. Hence, you may optionally disclose your name depending on your preference.

*Required

Section A: Your experience

In this section, we will be gathering information regarding to your experience in software projects and some other technical knowledge.

1. **Your name (Optional)**
   Nickname would do, too

   ....

2. **How many software projects you had conducted? ***
   Inclusive of partial completed projects, final year projects, and any assignment that includes requirement phase
   Mark only one oval.

   ○ None
   ○ 1 - 5 software projects
   ○ > 5 software projects
   ○ Other: ....

3. **What are the methods that you currently applied to specify requirements? ***
   Excluding elicitation or gathering stage (which usually involves questionnaire, interview, observation and etc)
   Tick all that apply.

   □ Use cases
   □ Formal user requirement notation
   □ Natural language (normal sentence)
   □ Structured natural language (boilerplate, template)
   □ Other: ....

4. **Please list down any tool that you used to specify your requirements for any of the software projects** *

Microsoft Word is also considered as a "tool" although it doesn't provide any feature to assist you

....

5. **Do you have any prior knowledge about "boilerplate" before using this tool?** *

Mark only one oval.

○ Yes

○ No

Section B: Functionality

In this section, we will would like to know what you think about the functionality aspect of the tool

6. **Does the tool provide sufficient feature for requirement specification** *

Feature refers to keyword suggestion, requirement specification modules, boilerplate modification, export modules

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
The tool is very lacking of feature	○	○	○	○	○	○	○	○	○	○	The tool covers most of things I could thought of

7. **Does the provide requirement specification modules sufficient to specify all of your requirements** *

Modules refers to action control, access control, and etc

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Insufficient, I can't specify much requirement	○	○	○	○	○	○	○	○	○	○	Sufficient, I can specify many different requirements

8. **Does the provided modules appropriate and suitable to specify requirement** *

Modules refers to action control, access control, and etc

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
The modules are not appropriate and should be redesigned	○	○	○	○	○	○	○	○	○	○	The modules are appropriate and well designed

9. **Are the predefined boilerplates provided appropriate** *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
The boilerplates are not appropriate and not suitable	○	○	○	○	○	○	○	○	○	○	The boilerplates are appropriate and very suitable to specify requirement

Section C: User Interface and Experience

In this section, we would like to know how do you feel when using our tool

10. **What do you think about the following statements?** *

Mark only one oval per row.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
The user interface of tool is consistent	○	○	○	○	○
The user interface of the tool is well designed	○	○	○	○	○
The user interface of the tool shows the overall process flow of using the tool	○	○	○	○	○
The tool is very easy to learn	○	○	○	○	○

11. **How do you feel when using this tool** *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Boring	○	○	○	○	○	○	○	○	○	○	Interactive/Fun

Section D: Measurements

In this section, we would like to have some measurable figures from the project you specified using the tool

12. **How many functional requirement you had specified BEFORE using this tool?** *

13. **How many non-functional requirement you had specified BEFORE using this tool?** *

14. **How many functional requirement you had specified AFTER using this tool?** *

15. **How many non-functional requirement you had specified AFTER using this tool?** *

16. **How many existing requirements that you failed to specify using this tool?** *
Those you had originally but fail to be specified or transferred into this tool

17. **How many new requirements that you had specified using this tool?** *
Those you didn't specified originally but added after using this tool

18. **How many time you spent to specified your original requirement? (In number of minutes)** *
Approximate figure should be sufficient

19. **How many time you spent in this tool to specify your requirement? (In number of minutes)** *
Approximate figure should be sufficient

20. **How fast do you specify your requirement with this tool compared to your original method of requirement specification?** *
Approximate figure should be sufficient
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Slower than original	○	○	○	○	○	○	○	○	○	○	Faster than original

Section E: Personal Opinion

In this section. we would like to have your personal opinion about this tool

21. **Does the description and instruction provided sufficiently teach you how to use this tool?** *
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very insufficient	○	○	○	○	○	○	○	○	○	○	Very sufficient

22. **Does this tool trigger or help you to specify more non-functional requirement?** *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Not at all	○	○	○	○	○	○	○	○	○	○	Very much

23. **Do you think boilerplate helps or limits user in requirement specification?** *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Limits user	○	○	○	○	○	○	○	○	○	○	Helps user

24. **Will you use this tool to specify requirement in future?** *

Mark only one oval.

○ Yes

○ No

○ Other: ____________________

25. **Last but not least, do you have any suggestions or improvement areas that we could look on?**

____

____

____

____

____

Powered by Google Forms

Appendix D: Feedback Survey Result

Summary

Section A: Your experience

Your name (Optional)

Anders Cheow

Ang Zi Xun

Law Teck Chuan

KS

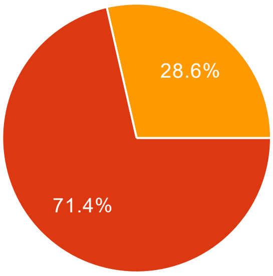
Wai Kei

Yin

Jekkie

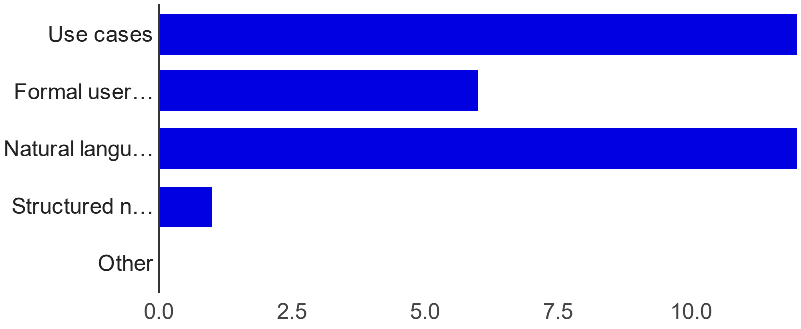
yihui

How many software projects you had conducted?



None	**0**	0%
1 - 5 software projects	**10**	71.4%
> 5 software projects	**4**	28.6%
Other	**0**	0%

What are the methods that you currently applied to specify requirements?



Use cases	**12**	85.7%

Formal user requirement notation	**6**	42.9%
Natural language (normal sentence)	**12**	85.7%
Structured natural language (boilerplate, template)	**1**	7.1%
Other	**0**	0%

Please list down any tool that you used to specify your requirements for any of the software projects

Microsoft Word

Microsoft word

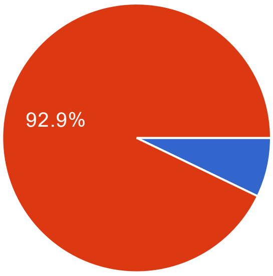
Microsoft Word, Enterprise Architect

Google Doc, Use Case

word editing tool such as microsoft word

trello

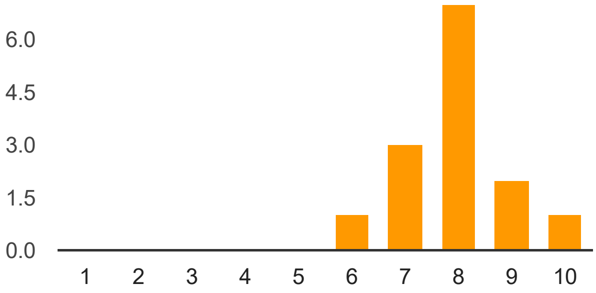
Do you have any prior knowledge about "boilerplate" before using this tool?



Yes	**1**	7.1%
No	**13**	92.9%

Section B: Functionality

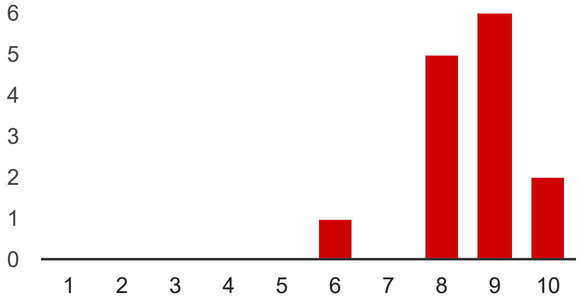
Does the tool provide sufficient feature for requirement specification



The tool is very lacking of feature: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**0**	0%
5	**0**	0%
6	**1**	7.1%

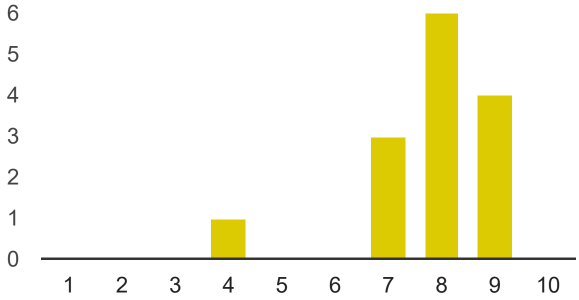
7	**3**	21.4%
8	**7**	50%
9	**2**	14.3%
The tool covers most of things I could thought of: 10	**1**	7.1%

Does the provide requirement specification modules sufficient to specify all of your requirements



Insufficient, I can't specify much requirement: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**0**	0%
5	**0**	0%
6	**1**	7.1%
7	**0**	0%
8	**5**	35.7%
9	**6**	42.9%
Sufficient, I can specify many different requirements: 10	**2**	14.3%

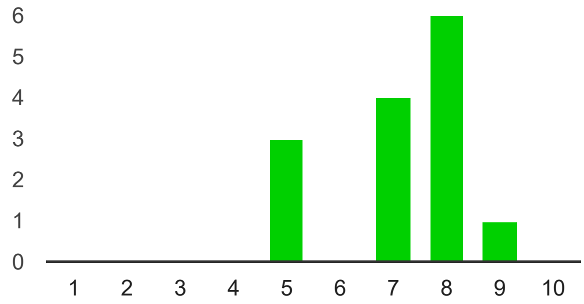
Does the provided modules appropriate and suitable to specify requirement



The modules are not appropriate and should be redesigned: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**1**	7.1%
5	**0**	0%

6	**0**	0%
7	**3**	21.4%
8	**6**	42.9%
9	**4**	28.6%
The modules are appropriate and well designed: 10	**0**	0%

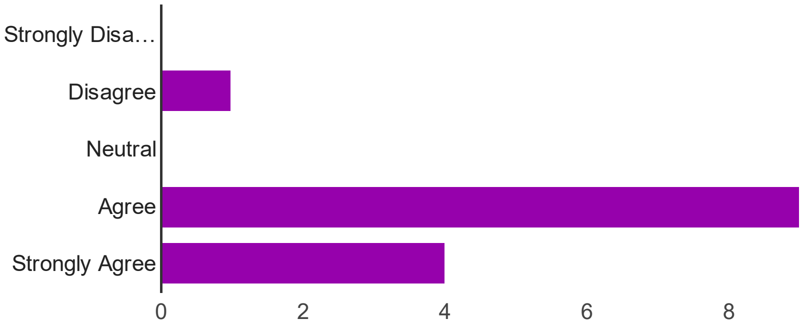
Are the predefined boilerplates provided appropriate



The boilerplates are not appropriate and not suitable: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**0**	0%
5	**3**	21.4%
6	**0**	0%
7	**4**	28.6%
8	**6**	42.9%
9	**1**	7.1%
The boilerplates are appropriate and very suitable to specify requirement: 10	**0**	0%

Section C: User Interface and Experience

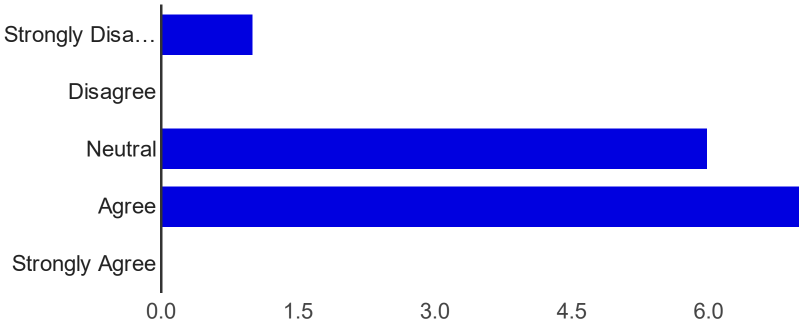
The user interface of tool is consistent [What do you think about the following statements?]



Strongly Disagree	**0**	0%
Disagree	**1**	7.1%

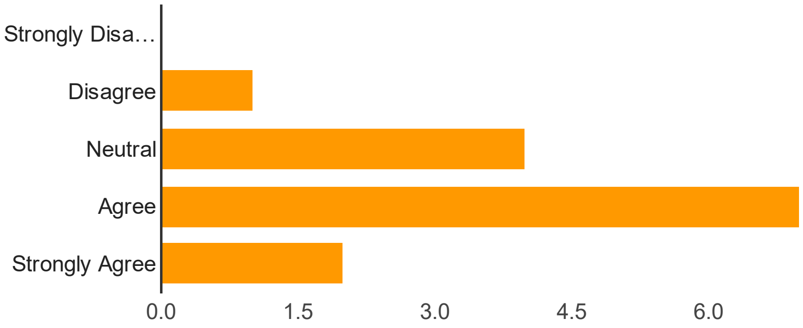
Neutral	**0**	0%
Agree	**9**	64.3%
Strongly Agree	**4**	28.6%

The user interface of the tool is well designed [What do you think about the following statements?]



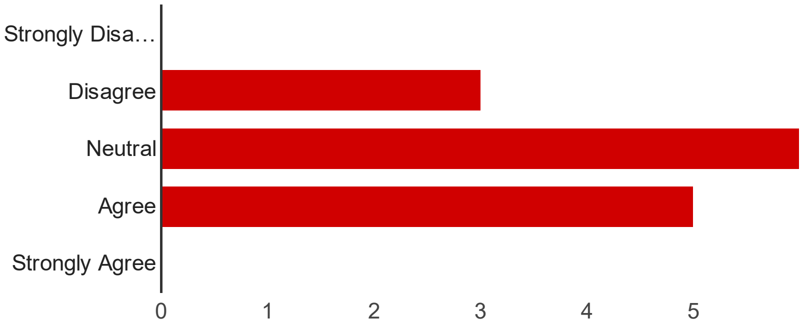
Strongly Disagree	**1**	7.1%
Disagree	**0**	0%
Neutral	**6**	42.9%
Agree	**7**	50%
Strongly Agree	**0**	0%

The user interface of the tool shows the overall process flow of using the tool [What do you think about the following statements?]



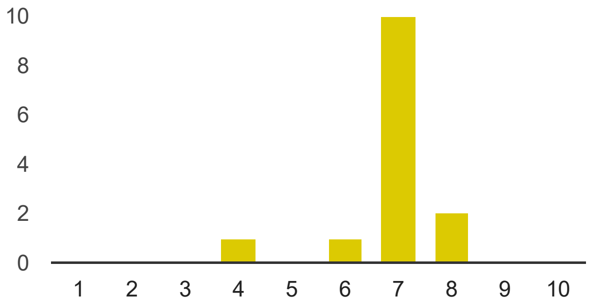
Strongly Disagree	**0**	0%
Disagree	**1**	7.1%
Neutral	**4**	28.6%
Agree	**7**	50%
Strongly Agree	**2**	14.3%

The tool is very easy to learn [What do you think about the following statements?]



Neutral	**6**	42.9%
Agree	**5**	35.7%
Strongly Agree	**0**	0%

How do you feel when using this tool



Boring: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**1**	7.1%
5	**0**	0%
6	**1**	7.1%
7	**10**	71.4%
8	**2**	14.3%
9	**0**	0%
Interactive/Fun: 10	**0**	0%

Section D: Measurements

How many functional requirement you had specified BEFORE using this tool?

35
8
10
5
9

53
20
3
6
15

How many non-functional requirement you had specified BEFORE using this tool?

5
0
20
10
7
15
4
11
6

How many functional requirement you had specified AFTER using this tool?

3
35
12
25
10
18
92
7

How many non-functional requirement you had specified AFTER using this tool?

21
10
9
35
11
33
25
0
7
5
2

How many existing requirements that you failed to specify using this tool?

0
2
1
10
3
4

How many new requirements that you had specified using this tool?

0
5
21
15
6
57
10
20
3
4

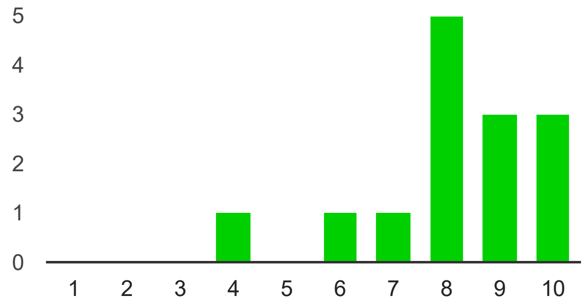
How many time you spent to specified your original requirement? (In number of minutes)

60
10
180
1
90
20
2
30

How many time you spent in this tool to specify your requirement? (In number of minutes)

30
20
10
35
60
0.2
5
45

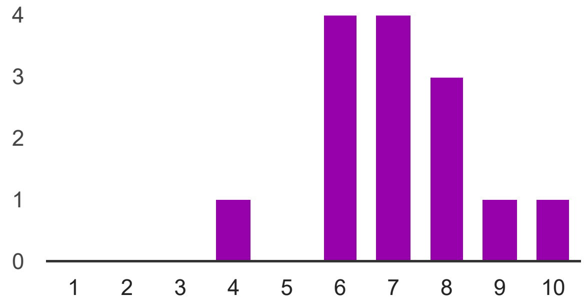
How fast do you specify your requirement with this tool compared to your original method of requirement specification?



Slower than original: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**1**	7.1%
5	**0**	0%
6	**1**	7.1%
7	**1**	7.1%
8	**5**	35.7%
9	**3**	21.4%
Faster than original: 10	**3**	21.4%

Section E: Personal Opinion

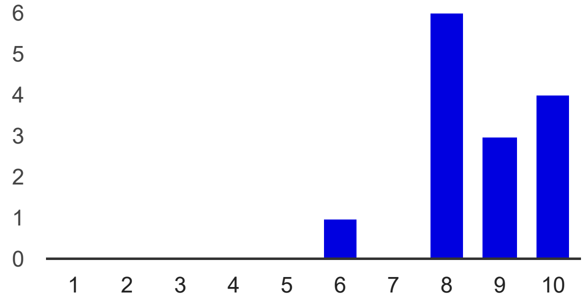
Does the description and instruction provided sufficiently teach you how to use this tool?



Very insufficient: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**1**	7.1%
5	**0**	0%
6	**4**	28.6%
7	**4**	28.6%

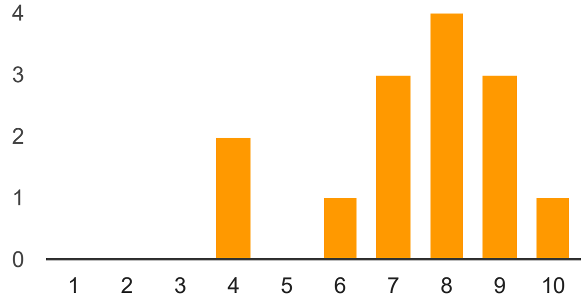
8	**3**	21.4%
9	**1**	7.1%
Very sufficient: 10	**1**	7.1%

Does this tool trigger or help you to specify more non-functional requirement?



Not at all: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**0**	0%
5	**0**	0%
6	**1**	7.1%
7	**0**	0%
8	**6**	42.9%
9	**3**	21.4%
Very much: 10	**4**	28.6%

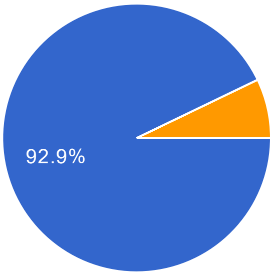
Do you think boilerplate helps or limits user in requirement specification?



Limits user: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**2**	14.3%
5	**0**	0%
6	**1**	7.1%
7	**3**	21.4%

8	**4**	28.6%
9	**3**	21.4%
Helps user: 10	**1**	7.1%

Will you use this tool to specify requirement in future?

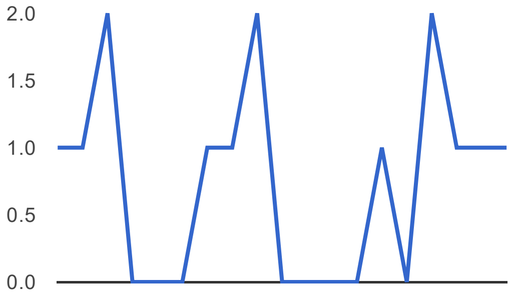


Yes	**13**	92.9%
No	**0**	0%
Other	**1**	7.1%

Last but not least, do you have any suggestions or improvement areas that we could look on?

Provide examples for each attributes
provide more appropriate examples for requirement.
Back to original page after successful save. Provide more details toast. Consider various type of input such as DateTime and so on.
redirect after submit ,use correct icon , provide easy access navigation
The user interface can be improved with better error checking and interaction.
1.need some improvement on interface designs, the interface should let user know what it should do at the first glance rather than trials and errors. 1.1 the tab contains "Instructions, Functional Requirements, Non-Functional Requirements" is hardly to be recognize. can change color on 'Active' tab. 1.2 may add some tool-tips. 1.3 can consider adding some animations. 2. the tutorials can be shown in 'Modal'. 3. the application of Material Design is good especially the cards view.
Provide the function of use enter to add the field in every module

Number of daily responses


2.0
1.5
1.0
0.5
0.0