Jurnal Teknologi

Full Paper

APPLICABILITY AND USABILITY OF PREDEFINED NATURAL LANGUAGE BOILERPLATES IN DOCUMENTING REQUIREMENTS

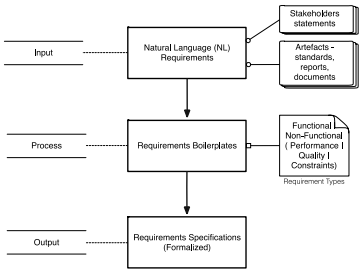
Noraini Ibrahim*, Wan M. N. Wan Kadir, Safaai Deris, Shahliza Abd Halim

Department of Software Engineering, Faculty of Computing, Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia

Article history
Received
2 February 2015
Received in revised form
8 October 2015
Accepted
12 October 2015

*Corresponding author
noraini_ib@utm.my

Graphical abstract



Abstract

Natural language is frequently applied to document the stakeholders' statements during requirement elicitation activities. Nevertheless, the use of generic natural language has potential for the issues of unclear and inconsistent requirements. These issues may result from the diverse interpretations by the stakeholders or other various sources of documents and artefacts. The main objective of this paper was to discuss the definition and application of predefined boilerplates to specify the requirements in the form of natural language statements. The proposed boilerplates were defined and classified based on two main types of requirements, namely functional and non-functional (performance, constraints, and specific quality). Two methods have been applied to evaluate the research results; the applicability of the predefined boilerplates was demonstrated using two different case studies, and the usability aspect is evaluated through synthetic environment experimentation using human respondents. As a summary, the predefined boilerplates were found helpful, especially among novice requirement engineers to express and specify their requirements in a consistent manner and a standardized way, relatively able to improve the quality of the natural language statements.

Keywords: Natural language, requirements boilerplates, applicability, usability

Abstrak

Bahasa tabii sering digunakan untuk mendokumenkan keperluan pihak berkepentingan ketika aktiviti pengumpulan keperluan. Maka, penggunaan bahasa tabii umumnya berpotensi mengakibatkan isu-isu seperti keperluan yang tidak jelas dan tidak konsisten. Isu ini berpunca daripada kepelbagaian tafsiran oleh pihak berkepentingan dan menerusi pelbagai sumber dokumen atau artifak perisian yang lain. Objektif utama kertas kerja ini adalah bagi memperincikan definisi dan aplikasi terhadap boilerplates yang dicadangkan untuk menspesifkasikan keperluan dalam penyataan formal bahasa tabii. Boilerplates yang dicadangkan telah diklasifikasikan kepada dua jenis keperluan yang utama iaitu kefungsian dan bukan kefungsian (prestasi, kekangan dan kualiti spesifik). Dua kaedah digunakan untuk menilai hasil kajian iaitu: keberkesanan boilerplates telah dinilai menerusi dua kajian kes yang berbeza, manakala aspek kebergunaan dinilai berdasarkan eksperimen persekitaran sintetik menggunakan subjek responden. Kesimpulannya, didapati boilerplates yang dicadangkan berupaya untuk membantu jurutera keperluan dalam pengumpulan dan spesifikasi keperluan dengan kaedah yang lebih konsisten dan seragam, yang akhirnya meningkatkan kualiti kenyataan keperluan dalam bahasa tabii.

Kata kunci: Bahasa tabii, keperluan boilerplates, keberkesanan, kebergunaan

1.0 INTRODUCTION

Requirements engineering is an early and critical phase in software development life cycles. Generally, there are four main activities during the requirements engineering phases, namely, elicitation, documentation, testing, and validation management [1]. The systematic process and activities during the requirements engineering phase allow the developer team to appropriately communicate with the related stakeholders in order to capture the needs of stakeholders in solving their business problems and achieving the organisation aims [2].

Requirements are often derived from the stakeholders' statements mentioning the problems that should be addressed by the system. All the relevant requirements from three main sources namely, i) stakeholders (i.e. survey, questionnaires, workshops, etc.), ii) related information of the predecessor systems (i.e. legacy and competitors), iii) other documentation sources (i.e. policies, company reports, documents, etc.) must be further transcribed properly into formal documentation of requirements specifications [1].

It has been reported that natural language is frequently applied to document the stakeholders' statements and their needs during requirements elicitation activities [1, 3-7]. Using natural language in documenting and authoring requirements is applicable in the situation where the stakeholders do not have prior knowledge on notations [1]. Similarly, natural language is generic and comprehensive in describing the various purposes and needs of stakeholders. In addition to that, natural language is generic enough to express the different types of requirements [6].

Other highlight on the natural language usage can be discovered from the survey by Neill and Laplante [7] that was conducted to gather state-of-the-art practices in requirements engineering activities. One remarkable finding concluded from the survey is the substantial use of natural language as an informal representation during the requirements elicitation activity. More than half of the respondents who were professionals in industries and application domains agreed that the use of non-formal representations, such as natural language, did not severely influence the quality aspect in terms of product suitability and usability of the developed software.

Carrillo de Gea *et al*. [4] provide an intensive study regarding the use of RE tools and its capabilities in the RE processes. This review revealed that the use of natural language statements has achieved the highest percentage compared to other semi or formal methods in facilitating specification languages and modelling. However, there is still a lack of tools that provide templates and checklists during the elicitation requirements activities. This study also suggests that it is an added advantage if the tools have a feature that allows elicited requirements to be documented in a persistent format.

Nevertheless, the use of generic natural language might lead to the issues such as ambiguous, incomplete, and inconsistent requirements [3, 5]. These issues may result from diverse interpretations by stakeholders or other various sources of documents and artefacts [1]. Additionally, issues of ambiguity and incompleteness lead to the volatility problems to the elicited requirements [8], and cause more complex situations if the software has been deployed in the client's site [2].

Meantime, Zowghi and Coulin [8] also mentioned that requirements elicitation activities should be supported by generic applications, such as template-driven documentation generation and assistive groupware. In turn, proper written requirements should facilitate readable specification documents, more understandable stakeholder statements [9-11], and at the same time, they can be analysed, realized, and verified in the next software development phases, namely: design, development, and testing.

In this paper, the main aim was to describe our works on the development of the predefined natural language requirements boilerplates templates, which facilitated a better way of documenting the requirements statements from the stakeholders. The proposed natural language boilerplates were defined based on two fundamental types of requirements: functional and non-functional. Our early findings remarked that the predefined natural language requirements boilerplates were found helpful, especially for the novice requirement engineers to express and specify the requirements in a consistent manner and a standardized way. Besides, this finding relatively proved that the application of the proposed boilerplates in specifying the different types of requirements was able to reduce the ambiguities and the incompleteness of the natural language statements.

The structure of this paper is as follows: Section 2 presents the related works to our study, ranging from the requirements elicitation perspective, the existing works that used natural language, and boilerplates in documenting the requirements specifications. Next in Section 3, the proposed natural language boilerplate templates are discussed in detail. Subsequently, the evaluation to the predefined natural language boilerplates template is explained in Section 4. Finally, the summary and the future works are presented in Section 5 to conclude the overall remarks and discussions on the proposed natural language boilerplates and their validation results.

2.0 RELATED WORKS

This section elaborates several works that are relevant and intertwined to the RE activities (elicitation), related software artefacts (natural language requirement statements), as well as the process (boilerplates), which focus to improve the quality of the documented

requirements specifications based on the natural language approach.

'Boilerplates' word was first coined by Hull *et al*. [6]. Boilerplates represent a collection of sentence patterns or templates that have limited vocabulary and keywords with specific placeholders to be completed. Similarly, Ortel *et al*. [12] defines boilerplate as "requirements specification documentation that consists of a set of pre-defined templates", and the categories of the boilerplates are based on three types of requirements: capability, functional, and constraint.

Meanwhile, Figure 1 portrays the typical input-process-output flows during the requirements documentation activity, which transform the elicited requirements into formalised requirements specifications.

Normally, the elicited or gathered requirements from stakeholders and other sources, such as standards, reports, and related documents, are in the form of natural language statements. All input from the natural language requirements are further articulated and processed based on the proposed requirements boilerplates. During this process, the requirements can be written within controlled vocabulary in the templates with regard to the types of requirements, namely functional and non-functional. As a result, the formalised requirements specifications with standard expressions are appropriately written in the documents.

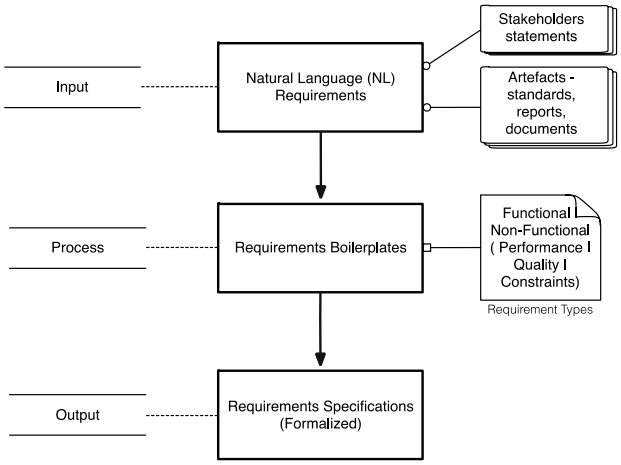


Figure 1 Requirement documentation process using natural language requirements boilerplates

An earlier approach, the BROOD model developed by Loucopoulos and Wan Kadir [13] utilizes formal sentence patterns of business rules to build the prescribed rule templates. The defined rule templates are categorised into five types, namely: attribute constraint, relationship constraint, action assertion, computation, and derivation. The prescribed rule templates provide lists of phrases with suitable variables and keywords that guide novice users to express the BR statements consistently. In addition to that, BROOD rule templates allow the defined BR statements to be associated to the related software design elements. Thus, any changes made to the business rules statements during the later software life cycle phase should as well give impact to the related artefacts, such as structural and behavioural design models.

Similarly, Farfeleder *et al*. [5] use domain ontologies for their boilerplates requirements to transform natural language requirements to become formalised requirements specifications for embedded system. The transformation process is semi-automatic using the developed DODT tool, which still requires the requirements engineer to choose the sets of requirements boilerplate and at certain level, validate the output of the documented requirements manually.

Another relevant work by Pohl and Rupp [1] propose requirements templates based on three main types of system activities: autonomous, user interaction, and interface. Autonomous or independent activities are the system activity that executes the process independently. User interaction activities are the system activity that provides services to the users. Interface activities are the system activity that executes process depending on other parts of the system, or waits for external events to occur. Yet, this template has one limitation, it can only be used to specify functional type of requirements. Some examples of the requirements templates are depicted in Table 1.

Table 1 Requirements templates based on system activity [1]

System Activity	Requirements templates
Autonomous	The ***<system name>*** shall/ should/ will ***<process>***
	The ***<system name>*** shall/ should/ will ***<process> <object> <object information>***
	[<When? Under what conditions>] the ***<system name>*** shall/ should/ will ***<process> <object> <object information>***
User interaction	*The* ***<system name>*** *shall/ should/ will provide the* ***<process> to <whom>***
	The ***<system name>*** *shall/ should/ will provide the* ***<process>*** *to* ***<whom> <object> <object information>***
	[<When? Under what conditions>] *the* ***<system name>*** *shall/ should/ will provide the* ***<process>*** *to* ***<whom> <object> <object information>***
Interface	The ***<system name>*** shall/ should/ will able to ***<process>***
	The ***<system name>*** shall/ should/ will able to ***<process> <object> <object information>***
	[<When? Under what conditions>] the ***<system name>*** shall/ should/ will able to ***<process> <object> <object information>***

Apart from the works mentioned earlier, there is a well-known boilerplates proposed by Hull *et al*. [6] that considered requirements based on problem-domain

needs (stakeholders/users requirements) or solution-domain needs (systems requirements). Likewise, both stakeholders and systems requirements categories have identical basis, which are capability (functional) and constraints on capability (non-functional). Table 2 shows some typical boilerplates templates based on requirements types and their subsequent related categories.

Table 2 User and system requirements boilerplates [6]

Type	Category	Typical boilerplates
Stakeholder (User)	**Capability**	The ***<stakeholder type>*** shall be able to ***<capability>***
	Constraint - Performance	The ***<stakeholder type>*** shall be able to ***<capability>*** within ***<performance>*** of ***<event>*** while ***<operational condition>***
System	**Capability**	The ***<system>*** shall be able to ***<function>***
	Constraint – Capacity & Performance	The ***<system>*** shall be able to ***<function>*** not less than ***<quantity> <object>*** while ***<operational condition>***
	Constraint – performance (periodicity)	The ***<system>*** shall be able to ***<function>*** every ***<performance> <units>***

3.0 PREDEFINED REQUIREMENTS BOILERPLATES

In this study, the proposed natural language requirements boilerplates were developed and classified into two basic types, namely, functional and non-functional requirements (performance, specific quality, and constraint) that are originally from a concern-based taxonomy of requirement proposed by Glinz [14].

Figure 2 portrays the further details on the requirements' taxonomy. All the functionalities and their behaviours, such as data, stimuli, and reactions, were categorized into the functional type. Requirements that were related to specific time and space bounds (timing, speed, volume, and throughput) fell into performance type. The specific quality type of requirements defined all the "-ilities" requirements, such as reliability, usability, security, availability, portability, and others. The constraint type requirements were related to physical, legal, cultural, environmental, design, and implementation, as well as interfacing requirements for the developed software systems.

The predefined natural language requirements boilerplates were motivated from similar works by Pohl and Rupp [1], as well as Hull *et al*. [6]. Firstly, the foundation of the requirements templates proposed by Pohl and Rupp [1] are based on the types of system activities namely autonomous, user interaction, and interface, as depicted in Table 1. The proposed requirements templates are only suitable to specify functional type of requirements.

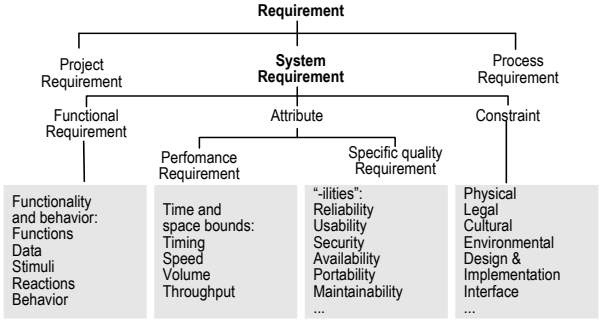


Figure 2 Basic classification of requirements [15]

Secondly, the boilerplates by Hull *et al*. [6] helps to express the requirements based on the needs of problem-domain (stakeholders) or solution-domain (system) and further categorised into the capability (functional) and the constraints on capability (non-functional) requirements, as shown in Table 2.

In this study, the proposed boilerplates by Hull *et al*. [6] are slightly similar with the pre-defined boilerplates. However, the number of boilerplates was expanded, and the proposed boilerplates were adapted and categorised based on the requirements types; functional and non-functional (performance, specific quality, and constraints). Additionally, the categorization helps to represent the generic specifications for both types of users and system requirements.

Table 3 presents the types of requirements and their corresponding boilerplates or templates in the form of natural language requirements. The boilerplates have limited vocabulary, whereby each of it has templates clause that describes the language used to express the requirement, and the "< >" cell is the placeholder, where the gaps were filled in by appropriate keywords or terms that expressed the requirement.

4.0 RESEARCH METHODOLOGY

This section presents the research methodology adopted, which describes on the two methods that have been applied to practically evaluate the research results, namely: case studies applications and synthetic environment experimentation. The main purpose of the case study evaluation is to demonstrate the applicability of the predefined requirements boilerplates in specifying the elicited requirements from the stakeholders. On the other hand, the usability aspect is evaluated through synthetic environment experimentation using human subjects.

Table 3 The predefined natural language requirements boilerplates

Type of Requirements	Natural Language Requirements Boilerplates
Functional	The ***<entity1>*** shall be able to ***<action>***
	The ***<entity1>*** shall be able to be in ***<state>***
	The ***<entity1>*** shall be able to be in ***<effect>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>***
	The ***<entity1>*** shall be able to ***<action>*** in ***<entity2>***
	The ***<entity1>*** shall be able to ***<action>*** to ***<entity2>***
	The ***<entity1>*** shall allow ***<entity2>*** to be in ***<state>***
	The ***<entity1>*** shall allow ***<entity2>*** to be able to ***<action>***
	The ***<entity1>*** shall allow ***<entity2>*** to be in ***<effect>***
Performance	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** not less than ***<quantity>*** times per ***<units>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** at least ***<quantity>*** times per ***<units>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** within ***<quantity>*** times per ***<units>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** at minimum rate of ***<quantity>*** times per ***<units>***
	The ***<entity1>*** shall be able to ***<action>*** not less than ***<quantity>*** ***<entity2>***
	The ***<entity1>*** shall be able to ***<action>*** at least ***<quantity>*** ***<entity2>***
	The ***<entity1>*** shall be able to ***<action>*** within ***<quantity>*** ***<entity2>***
	The ***<entity1>*** shall be able to ***<action>*** at minimum rate of ***<quantity>*** ***<entity2>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** not less than ***<quantity>*** ***<units>*** from ***<event>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** at least ***<quantity>*** ***<units>*** from ***<event>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** within ***<quantity>*** ***<units>*** from ***<event>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** at minimum rate of ***<quantity>*** ***<units>*** from ***<event>***
Specific Quality	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** for a sustained period of ***<units1>*** every ***<quantity>*** ***<units2>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** composed of not less than ***<quantity>*** ***<units>*** with ***<external entity>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** composed of at least ***<quantity>*** ***<units>*** with ***<external entity>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** composed of within ***<quantity>*** ***<units>*** with ***<external entity>***
	The ***<entity1>*** shall be able to ***<action>*** ***<entity2>*** composed of at minimum rate of ***<quantity>*** ***<units>*** with ***<external entity>***
	WHILE ***<operational condition>*** ... The ***<entity1>*** shall be able to ***<action>***
	WHILE ***<operational condition>*** ... The ***<entity1>*** shall be able to ***<action>*** ***<entity2>***
	IF ***<operational condition>*** THEN ... The ***<entity>*** shall <action>
Constraints	The ***<entity1>*** shall not allow ***<entity2>*** to ***<action>***
	The ***<entity1>*** must be ***<action>*** not less than ***<quantity>*** times per ***<units>***
	The ***<entity1>*** must be ***<action>*** at least ***<quantity>*** times per ***<units>***
	The ***<entity1>*** must be ***<action>*** within ***<quantity>*** times per ***<units>***
	The ***<entity1>*** must be ***<action>*** at minimum rate of ***<quantity>*** times per ***<units>***
	WHILE ***<operational condition>*** ... The ***<entity1>*** shall not ***<action>***
	WHILE ***<operational condition>***... The ***<entity1>*** may be ***<state>***
	WHILE ***<operational condition>***... The ***<entity1>*** shall not ***<action>*** except for other ***<action>***
	WHILE ***<operational condition>***... The ***<entity1>*** may be ***<state>*** without ***<effect>***
	WHILE ***<operational condition>***... The ***<entity1>*** may be ***<state>*** without ***<effect>*** other ***<action>***

4.1 Case Study Applications

A study by Zelkowitz and Wallace [15] classified case study as one of observational study types that provides a data collection method to represent the current situation or phenomenon of the organisation or application domain that is currently under study. It is a suitable research methodology in software engineering area because it helps to detail the study context by involving the controlling factors that are related to the case study environment as mentioned by Runeson and Höst [16].

The four basic processes in the case study evaluation, namely: design and planning, data collection, data analysis and reporting, are captured in Figure 3. It is suggested by Robson [17] that case study designs should consider the aim and purpose to be achieved, the case to be studied, theory to be applied, research questions to be answered, data collection methods and strategies, and result analysis to be reported.

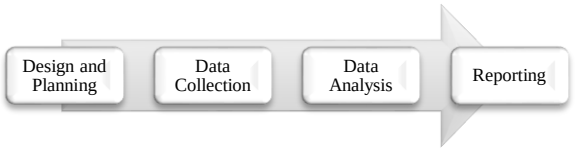


Figure 3 Case study process [16-17]

During design and planning processes, the case study elements i.e. the chosen cases and the its related subjects were defined. Two different context of real environment industrial applications systems, namely System-A and System-B were chosen. Generally, System-A is a medium-scale healthcare application that provides services among the healthcare industry players, while System-B is web-based system that offers services in assets and facilities management.

In conducting the case studies, it is essential to gather all related requirements for the both systems during the data collection phase. Due to that, features and functionalities of the systems were analysed, and all reports or manual documents obtained from the systems' stakeholders were also observed closely.

All elicited requirements statements were then classified based on three main business processes of the System-A, namely: (i) registration, (ii) billing, (iii) invoicing; and four business processes of the System-B system, namely: (i) assets registration, (ii) assets tracking, (iii) assets maintenance and (iv) assets complaints.

In the data analysis stage, the classified requirements statements were further categorised into functional or non-functional requirements types (performance, specific quality, and constraint). The requirements types are described in previous Section 3.

Next, the most critical task, namely to rephrase and specify the requirements statements appropriately using the pre-defined natural language requirements boilerplates based on their types was performed.

4.2 Synthetic Environment Experimentation

Synthetic environment experimentation is a classical scientific method that can be used to evaluate empirical studies in the software engineering research and practices [16-17]. According to Zelkowitz and Wallace [15], synthetic environment experimentation is defined as: *"A replicated experiment is conducted in a smaller artificial environment, but in a realistic settings compared to the real projects."*

Basically, usefulness is a criterion for the usability factor that helps user to solve the experimental task in an acceptable way using the provided handy features and functionalities provided by the process. [18]. In this experiment, the usefulness of the predefined requirements boilerplates is verified by checking whether the subjects agreed that the pre-defined templates helped to solve the required experimental tasks in an acceptable way.

5.0 RESULTS AND DISCUSSION

This section presents the applicability evaluation results of the predefined requirements boilerplates based on two case studies, namely System-A and System-B, as well as the results of the synthetic environment experimentation using human subjects to evaluate the usability of the predefined requirements boilerplates.

5.1 Case Study Evaluation Results

Generally, the System-A community consists of healthcare providers (HCPs), paymasters, and suppliers. HCPs are the users in System-A that manages patients' records, billings, and paymaster invoicing. HCPs are also the parties who distribute medications and provide treatments, such as: hospitals, general practitioners (GPs) and dentists. The paymasters are the parties that hire and pay the medications services. Insurers, employers, and any managed care companies are some examples of paymasters. Finally, supplier is the System-A owner that fundamentally administers the System-A.

Basically, there are three main features or functionalities provided by the System-A, namely: i) registration, ii) billing, and iii) invoicing. Figure 4 portrays the basic business workflow of the System-A.

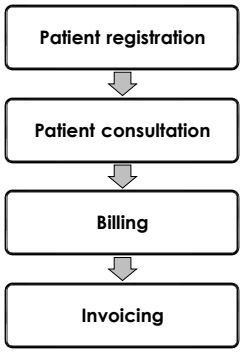


Figure 4 Workflow in System-A

All cash and panel patients must first be registered into the system. The system will automatically assign a unique registration ID for the new patient. For every visit to the clinic of a System-A healthcare provider, the patient will be registered by the HCP clinic staff for consultation and will be inserted into patient queue list. Once consultation completed, the HCP clinic staff will issue a bill to the patient according to the prescription by the on-duty doctor. Cash patients then pay the bills and provided with their prescriptions. As for panel patients, their bills are paid by their paymaster and included into the invoice of their employer (paymaster).

As for System-B, the offered main service is to manage all assets and facilities in. In specific, the System-B helps to allocate and record the assets/facilities, track the recent location of the assets, manage the maintenance of the assets/facilities and its history, and record the information of the disposed assets. Figure 5 illustrates in brief the business workflow of assets life cycle in System-B as aforementioned.

Meantime, there are three main categories of System-B users: i) OAD staff, ii) internal user and iii) public user. The different categories have various types of users and subsequently their accessibility to the System-B, as shown in Table 4.

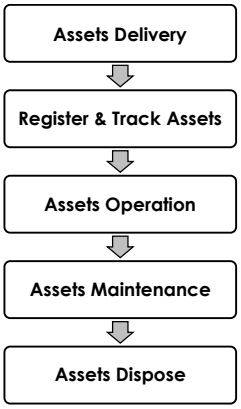


Figure 5 Workflow in System-B

Table 4 Categories of System-B user

Category	User	Access to System-B
OAD staff	Supervisor	Register asset, Update assets information, View assets location and movement information, View complaint, Validate progress for work assignment, View and create maintenance schedule
	Clerk	View complaint, View and assign work assignment, validate work assignment
	Worker	View complaint, View work assignment, Perform job for work assignment
Internal	All UTM staff (other than OAD staff)	Create complaint
Public	Contractor, asset supplier, agent	Create complaint

The following Table 5 and 6 shows the results of requirements statements elicited of the System-A and System-B based on its relevant business process during data collection activities.

Table 7 and Table 8 present some examples of requirements statements gathered from the System-A and System-B case studies that have been rephrased to more formalized requirements specifications using the predefined requirements boilerplates. Based on Table 7 and 8 results, this finding relatively proved that the requirements for System-A and System-B systems could be successfully specified based on the proposed natural language boilerplates.

From the practical point of view, the suggested requirements boilerplates provide a support instrument for the requirements analyst to specify the requirements specifications in the standard form of language expressions. The boilerplates have limited vocabulary, whereby each of it has templates clause that describes the language used to express the requirements, and the "< >" cell is the placeholder, where the gaps were filled in by appropriate keywords or terms that expressed the requirements. Thus, it is a good method of standardising the language used for expressing the specific types of requirements [6]. It also offers a minimum set of attributes in writing and expressing requirements in a standard way. It assists software analyst to specify the requirements using a consistent language by choosing a suitable pre-defined templates and filling in the gaps (placeholders).

Table 5 Elicited requirements for System-A

Business Process	Requirements Statements
Registration	A new patient must be registered with unique ID
	A registered patient may have more than one paymaster
	The status of the patient is set as 'banned' if they have an outstanding balance invoice
	The system will insert the patient into consultation queue list once he/she completed the consultation registration
	A patient is treat as an emergency case if the condition of the patient is critical
	A patient can be terminated from the list of payees by their paymaster
	A panel patient is allowed to register for his panel clinics based on the maximum number of clinics set by the paymaster
Billing	The bill amount for the panel patient must not exceed the total amount limit set by their paymaster
	The bill is created once the patient complete their consultation
	The Chief Clinic Assistant is allow to modify the created bill by the clinic assistant
	The HCP shift leader is allows to make correction to incorrect or mistakes in any bill that are issued by the HCP clinic staff.
	Patient can pay the bill by cash, cheque or credit card
	A bill contains patient information, total amount, consultation descriptions – prescriptions and medical services, issue date, issue staff and paymaster code (for panel patient only).
	Once the patient pay the bill, the bill is set to 'fully paid' or 'partly paid' from 'unpaid' status, according to the paid amount. This feature is applied to cash or partially sponsored panel patient.
	As for panel patient, the bill is set to 'invoiced' status when the invoice is created after the panel patient receives their prescriptions.
Invoicing	The Account Clerk is allow to create reminders for the past due invoices
	The Account Clerk will issue the first reminder if a payment is not received within 30 days from the invoice date.
	The HCP clinic staff will verify all the created bills for the panel patients before it is listed as an invoice. Once verified, the status for the particular bill is changed to 'invoiced'
	The invoice date is set to the end of month for the panel with monthly-basis interval invoice.
	All the status of the invoice should be fixed to be end by 12:00 am on the next day of the invoice date
	The amount of invoice for paymaster System-A usage is calculated based on the total number of the payees.

Table 6 Elicited requirements for System-B

Business Process	Elicited Requirements Statements
Assets registration	The system has to assign a unique ID for all new assets to be registered.
	The system must prevent any unauthorized user or public user to register for new assets.
	Only supervisor can register the new assets into the system.
	The supervisor is allowed to enter and update assets information based on the assets categories.
	The supervisor is allowed to view the assets information by selecting the assets categories types of building, space, equipment or infrastructure.
Assets tracking & allocation	The system must prevent any access by unauthorized user, public user, students and staff to view or track the assets location.
	The supervisor is allowed to allocate the asset to the building or space location.
	The supervisor is allowed to track specific assets by selecting the category types of building, space, equipment or infrastructure.
	The system is able to track the service respond by duration time of week, month or year.
Assets maintenance	The system must prevent any access from unauthorized user, public user or staff to distribute the job tasks to the assigned workers.
	The supervisor is allowed to set-up the maintenance schedule for the assets.
	The supervisor is allowed to select the specific lists of contractor for every set-up maintenance schedule of the assets.
	The supervisor is allowed to select the specific duration time for every set-up maintenance schedule of the assets.
	The equipment must be disposed at maximum rate of six years.
	The equipment must be disposed if the warranty is five years.
	The equipment must be serviced at least two times per year on every 15th January and June months.
	The equipment must be serviced while it still under warranty.
	The equipment must be serviced after five years it has been used.
	The system must able to compute the interference time by total up the response time + checking time + service time.
	The system must able to compute the total of distribution time for every job task by complaint time minus (–) the time that the contractor receives the assigned job task from the supervisor.
Assets complaint	The supervisor is allowed to update the progress of the assigned job tasks either new or in process while the status of the job tasks is not finish.
	The system is able to notify the supervisor if the interference time is more than 30 hours
	The contractor user is allowed to view the schedule of the assigned job task for the specific selected assets.
	Any types of user are allowed to enter the new complaints regarding the assets and facilities.
	Any unauthorized and unregistered users are allowed to enter new complaints through hotline complaint section only.
	The supervisor is allowed to view the details of the complaints based on the location of the assets.
	The system is able to distribute the job tasks to the staff based on the type of the job for every new complaint.

Table 7 Rephrased requirements specifications for System-B

Business Process: Asset Registration

- Requirement statement: The system has to assign a unique ID for all new assets to be registered.
- Type: **Functional requirement** *(behaviour)*

Re-phrased Requirement

- Boilerplate: The **<entity1>** *shall be able* to **<action> <entity2>**
- Specification: The <**system**> shall be able to <**assign**> a unique ID for every **<new registered asset>**

Business Process: Asset Complaint

- Requirement statement: The system is able to notify the supervisor if the interference time is more than 30 hours
- Type: **Performance requirement** *(timing)*

Re-phrased Requirement

- Boilerplate: The <**entity1**> *shall be able* to <**action**> <**entity2**> not less than <**quantity**> <**units**>
- Specification: The <**system**> shall be able to <**notify**> the <**supervisor**> if the <**interference time**> is <**more than 30 hours**>

Business Process: Assets tracking & allocation

- Requirement statement: The system must prevent any access by unauthorized user, public user, students and staff to view or track the assets location..
- Type: **Constraint requirement** *(legal)*

Re-phrased Requirement

- Boilerplate: The <**entity1**> shall not allow **<entity2>** to <**action**>
- Specification: The **<system>** shall not allow any <**unauthorized access | public user | student | staff]**> to <**view**> the assets location

Business Process: Asset maintenance

- Requirement statement: The equipment must be serviced at least two times per year on every 15th January and June months.
- Type: **Specific quality requirement** *(maintainability)*

Re-phrased Requirement

- Boilerplate: **The <entity1>** *shall be able to* **<action>** *for every* **<quantity> <units2>**
- Specification: The <equipment> shall be able to be <serviced>… on every <15th January and June> <month>

Table 8 Rephrased requirements specifications for System-A

Business Process: Registration (Patient record maintenance) • Requirement statement: Paymaster HR Officer may terminate any patient from their list of payees • Type: **Functional requirement** *(user function)* **Re-phrased Requirement** • Boilerplate: The **<entity1>** *shall be able to* **<action> <entity2>** • Specification: The **<Paymaster-HROffice>** *shall be able to* **<terminate> <Patient>** from their list of payees
Business Process: Billing (Bill preparation) • Requirement statement: The amount of a panel patient's bill must not exceed the maximum bill amount set by the paymaster. • Type: **Performance requirement** *(throughput)* **Re-phrased Requirement** • Boilerplate: The **<entity1>** *shall be able to* **<action>** *not less than* **<quantity> <entity2>** Specification: The **<paymaster>** *shall be able to* **<setMaxAmount>** *not less than* **<totalAmount>** of the **<PatientBill>**
Business Process: Registration (Patient consultation) • Requirement statement: Any patient with an outstanding balance should be banned from consultation registration. • Type: **Constraint requirement** *(legal regulation)* **Re-phrased Requirement** • Boilerplate: WHILE **<operational condition>** ... The **<entity1>** *shall not* **<action>** Specification: WHILE **<patient_status = outstanding >** ... The **<Patient>** *shall not* allow to **<register_consultation>**
Business Process: Registration (Patient consultation) • Requirement statement: The Account Clerk will issue the first reminder if a payment is not received within 30 days from the invoice date. • Type: **Specific quality requirement** *(maintainability)* **Re-phrased Requirement** • Boilerplate: **The <entity1>** *shall be able to* **<action> <entity2>** *for a sustained period of* **<units1> every <quantity> <units2>** Specification: The **<account_clerk>** shall be able to **<issue> <reminder>** *for a sustained period of* **<invoice-date>** in every **<30> <days>**

5.2 Synthetic Environment Experimentation Results

There were 23 subjects have participated in the synthetic environment experiment conducted. The samples of experimental subjects were selected based on the expectation that they must have at least minimal understanding on SE theories and principles.

In addition to that, the library system case study was chosen in this synthetic environment experimentation because it represents the application domain with realistic problems, yet sufficient enough to advocate acceptable and reasonable change request implementation cases. Furthermore, the library system case study is easily understandable for the experimental subjects. The experimental subjects have been exposed to the library system since the first day they registered as a student at the university. Each of them is a user of the library system and they are familiar with the features, functionality and system environments provided by the library system. The library system case study is assumed to provide core services such login, borrowing (loan) available materials items, returning (check-in), renewal, reservation, searching, and checking accounts.

The validity of the conducted experiment is assured so the analysis results should be reliable enough to be trusted. Sample questions sets are referred and adopted from Software Usability Measurement Inventory (SUMI) [19]. SUMI is an industry standard evaluation questionnaire for assessing quality of use of software by end users. In addition to that, the quality ratings for each question are based on ordinal Likert scale of 5 options to be chosen (1: Strongly disagree, 2: Disagree, 3: Undecided, 4: Agree, 5: Strongly agree). Basically, five related questions were designed to evaluate the usefulness criteria from subjects' viewpoints, as follows:

i. It allows for easier selection to classify types of requirements.
ii. It allows for easier way to define the requirement specifications.
iii. It allows for easier way to choose input statement for requirement specifications (i.e. by fill-in the "< >" placeholders).
iv. It allows for easier way to express the requirement in consistent manner.
v.

Figure 6 presents the summary of the descriptive statistics for the conducted exploratory survey. In summary, all respondents are agreed towards the usefulness criteria of the proposed natural language boilerplate templates. Table 4 shows the frequency distribution for the 4 usefulness criteria in details.

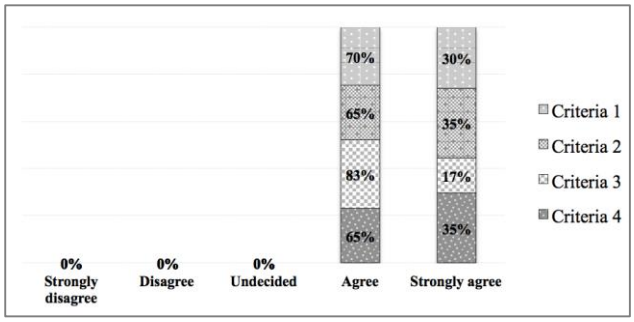


Figure 6 Summary of respondents' perspectives

Table 9 presents the results of 4 usefulness criteria in evaluating the predefined requirements boilerplates. The first usefulness criteria is evaluated to observe the subjects' viewpoint in terms of easier selection in classifying the four types of requirements, namely; functional, performance, constraint and specific quality. In general, it is concluded that all 23 subjects agreed with the first usefulness criteria. The highest frequency of 16 subjects or 69.6% strongly agreed, followed by 30.4% or 7 subject agreed with this criteria. None of the 23 subjects was in opposition to this first usefulness criteria.

Next criteria in evaluating usefulness is by looking further on subjects' agreement whether the boilerplates have provided support and a more easy

way in defining the identified requirements specifications for library system. No one from total 23 subjects was opposing this second criteria of usefulness. In contrast, all subjects agreed with the criteria, with majority of 15 subjects or 65.2%agreed and followed by 34.8% or 8 subject strongly agreed with the second usefulness criteria of the predefined requirements boilerplates in providing more flexible way in defining requirements specifications.

Subsequently, another criteria is to judge subjects' opinion on the usefulness of the predefined requirements boilerplates support in choosing the suitable input statement for requirements specification, by just filling in the value of specific information in each selected "< >" palettes/placeholders. Of 23 subjects, 19 subjects or approximately 82.6% agreed and followed by 4 subjects or around 17.4% distributions that were totally agreed with this criteria. No subjects were found opposing this criteria.

The forth usefulness criteria is to determine subjects' agreement whether they found that the predefined requirements boilerplates have provided easier way in consistently expressing the requirements specification. The highest frequency distribution of 65.2% or 15 subjects agreed, followed by 34.8% or 8 subjects that totally agreed to this criteria.

Table 9 Results of 4 usefulness criteria

Usefulness Criteria	Likert Scales	Frequency (N=23)
1. It allows for easier selection to classify types of requirements.	Strongly disagree	0
	Disagree	0
	Undecided	0
	Agree	16
	Strongly agree	7
2. It allows for easier way to define the requirements specifications.	Strongly disagree	0
	Disagree	0
	Undecided	0
	Agree	15
	Strongly agree	8
3. It allows for easier way to choose input statement for requirements specifications (that is by filling-in the "< >" placeholders).	Strongly disagree	0
	Disagree	0
	Undecided	0
	Agree	19
	Strongly agree	4
4. It allows for easier way to express the requirements in consistent manner.	Strongly disagree	0
	Disagree	0
	Undecided	0
	Agree	15
	Strongly agree	8

6.0 SUMMARY AND FUTURE WORK

In this paper, we have presented our study on the definition and application of the predefined natural language requirements boilerplates. The focal aim of this study was to facilitate a better way of documenting the elicited requirements from the stakeholders; particularly in specifying the requirements based on the predefined natural language boilerplates, which relatively improved the quality of the natural language statements.

The predefined natural language requirements boilerplates were classified based on two main types of requirements, namely functional and non-functional (performance, constraints, and specific quality). Meanwhile, the feasibility and the applicability of the predefined natural language requirements boilerplates were demonstrated using two industrial strength case studies, namely the System-A, a healthcare application and the System-B, an asset maintenance and management system. The findings concluded that the predefined boilerplates were feasible enough to assist in specifying the requirements statements in controlled and limited language with consistent sets of vocabularies.

Apart from the case study evaluation, the synthetic environment experimentation is also performed to explore and quantify the usefulness of the predefined requirements boilerplates from the end-user perspectives. 4 criteria of the usefulness are verified by checking whether the subjects agreed that the predefined requirements boilerplates helped to solve the required experimental tasks in an acceptable way. The element of predefined templates helps to define new requirements specification. As a result, it offers a simpler way, helping to reduce human efforts and producing fewer errors. The suggested templates are also reusable, which will result to higher flexibility of software systems specifications, and at the same time helps to express requirements specifications in a more consistent and standardised way.

In the future, it is expected that the pre-defined natural language requirements boilerplates to be revised, considering the domain-specific ontologies to facilitate the issue of limited constraints and vocabulary of the requirements statements. In addition, initial findings on System-A and System-B systems should be further extended to other case studies that are complex enough for more detailed evaluations and results.

Acknowledgement

The authors would like to express their deepest gratitude to Research Management Center (RMC), Universiti Teknologi Malaysia (UTM) and Ministry of Education Malaysia for their financial support under Fundamental Research Grant Scheme (Vot number R.J130000.7828.4F216).

References

[1] Pohl, K. and Rupp, C. 2011. *Requirements Engineering Fundamentals A Study Guide for the Certified Professional*

for Requirements Engineering Exam. 1 edition. CA, USA: Rocky Nook.
[2] Ibrahim, N., Wan Kadir, W. M. N., and Deris, S. 2009. Propagating Requirement Change into Software Designs to Resilient Software Evolution in *The 16th IEEE Asia Pacific Software Engineering Conference (APSEC'09)*.
[3] Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., Gnaga, R. 2013. RUBRIC: A Flexible Tool for Automated Checking of Conformance to Requirement Boilerplates. In: *2013 9th Joint Meeting European Software Engineering Conference ACM SIGSOFT Symposium Foundation Software Engineering. ESEC/FSE 2013*- Proc. 599-602.
[4] Carrillo de Gea, J. M., Nicolás, J., Fernández Alemán, J. L., Toval, A., Ebert, C., Vizcaíno, A. 2012. Requirements Engineering Tools: Capabilities, Survey and Assessment. *Information Software Technology*. 54(2012): 1142-1157.
[5] Farfeleder, S., Moser, T., Krall, A., Stalhane, T., Zojer, H., Panis, C. 2011. DODT: Increasing Requirements Formalism Using Domain Ontologies for Improved Embedded Systems Development. *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. 271-274.
[6] Hull, E., Jackson, K., Dick, J. 2005. *Requirements Engineering*. Springer, London,
[7] Neill, C. J., Laplante, P. A. 2003. Requirements Engineering: The State of the Practice. *IEEE Software*. 20(2003): 40-45.
[8] Zowghi, D. and Coulin, C. 2005. Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In *Engineering and Managing Software requirements*. Springer. 19-46.
[9] Mavin, A. 2012. Listen, Then Use EARS. *IEEE Softw*. 29(2): 17-18, Mar.
[10] Mavin, A. and Wilkinson, P. 2010. Big Ears (The Return of 'Easy Approach to Requirements Engineering'). *18th IEEE International Requirements Engineering Conference*. 6: 277-282.
[11] Mavin, A., Wilkinson, P., Harwood, A. and Novak, M. 2009. Easy Approach to Requirements Syntax (EARS). *17th IEEE International Requirements Engineering Conference*. 317-322.
[12] Ortel, M., Malot, M., Baumgart, A., Becker, J. S., Bogusch, R., Farfeleder, S. *et al*. 2013. Requirements Engineering. In: A. Rajan, T. Wahl (Eds.). CESAR-Cost-efficient Methods and Processes for Safety-relevant Embedded Systems. Springer, Vienna,
[13] Loucopoulos, P., Wan Kadir, W. M. N. 2008. BROOD: Business Rules- Driven Object Oriented Design. *Journal of Database* Management. 19(2008): 41-73.
[14] Glinz, M. 2007. On Non-Functional Requirements. *15th IEEE International Requirement Engineering Conference*.
[15] Zelkowitz, M.V. and Wallace, D. 1997. Experimental Validation in Software Engineering. *Information and Software Technology*. 39: 735-743.
[16] Runeson, P. and Höst, M. 2009. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering*. 14(2009): 131-164.
[17] Wohlin, P. R., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. 2012. *Experimentation in Software Engineering*. Springer, Berlin Heidelberg,
[18] Robson, C. 2002. *Real World Research*. Second. Oxford, UK: Blackwell Publishing,
[19] Kirakowski, J. 1994. SUMI Questionnaire Homepage, [Online]. Available: http://sumi.ucc.ie/.