Volume 7, Issue 1, January 2017 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Reducing Missed Requirements Issues: Complete, Unambiguous and Necessary Requirements Elicitation

*Mohamed Hagal, Fatma Kandemili
Material Science and Engineering Department, Kastamonu University, Turkey

DOI: 10.23956/ijarcsse/V7I1/01110

***Abstract*— Eliciting excellent requirements that lead a software system to be free from errors require deep and precise understanding of user requirements where eliciting missing requirements is one of the major factor of software failure, additionally identifying the situations that may cause early system failure are required. To reduce the effect of eliciting missing requirements, an organized conceptual approach is needed. In this paper we propose a systematic approach for eliciting user requirements in a simplified manner that will encourage software developers not only to elicit user requirements, but also to improve the capturing and understanding of the potential requirements that users would not be able to articulate. This proposed approach will be helpful especially for the non-expert developers to elicit excellent requirement with little effort.**

***Keywords*— Requirement elicitation; excellent requirements; software errors; UML; prototyping; exception.**

I. INTRODUCTION

Eliciting precise requirements for any software system is considered the most important activity in order to avoid their failures [1]. Developing a software system which is free from errors is based on the correct understanding of user needs, where one of the major causes of poor understanding and specification of those needs are communication, negotiation and collaboration issues between systems users and system development team [2].

Requirements elicitation is not an easy task and deserve more effort because errors propagated from this stage will be expensive to repair. Problems of requirements such as incompleteness, ambiguity and selecting unnecessary requirements will be critical of developing error free software system. These problems came from the ill-defined software system scope, missing or misunderstanding of user requirements. Detecting and correcting software problems after deployment is more expensive than detecting and correcting them during the requirement analysis stage [3-7]. In addition to difficulties in requirements selection and their priorities; it requires high degree of skills of both; developers and stakeholders [8-10].

The development of reliable software that meets its requirements and being free of errors, is becoming more complex with the increasing of the undealt exceptions. When an exception occurs, the normal flow of the program terminates abnormally. These exceptions can occur for many different reasons, missing requirements, user errors, and programmer's errors or physical errors such as Hardware errors and resource unavailability [11].

The difference between errors and exception is that exceptions can be handled, whereas errors cannot be. Exceptions can be classified into two types: Checked and Unchecked exceptions. Checked exceptions are the exceptions that a compiler can catch, while the unchecked exceptions cannot be caught by a compiler (i.e. runtime exceptions) and they are propagate implicitly (i.e. unhandled exception) [12]. However, exceptions traceability should be included from the early stages of the software development life cycle. For example, exception can be considered as cancer, and if it is detected earlier; it can be handled and its effects will be reduced [13, 14].

The purpose of this paper is to present a simple approach, to enhance the process of requirements elicitation using some structured activities and making eliciting missing or incomplete requirements improbable.

II. METHOD OVERVIEW

Empirical studies in software development lifecycle show that one of the most important reasons for systems failures result from the misunderstandings of the systems requirements, and the late correction of these requirements that would be expensive. There are many issues that lead to a wrong or ambiguous specification: such as scope creeping and when customers cannot articulate their needs at an early stage. The problems arising from the systems' developments show that sometimes there is a gap between the customers and the developers. To overcome this problem, a novel approach for clarifying software requirements is proposed. It consists of several activities, and some documents are generated throughout these activities. By tracing these activities, the requirements are verified, and the customer's feedback is well managed. The conceptual overview of the proposed approach is shown in Fig. 1.

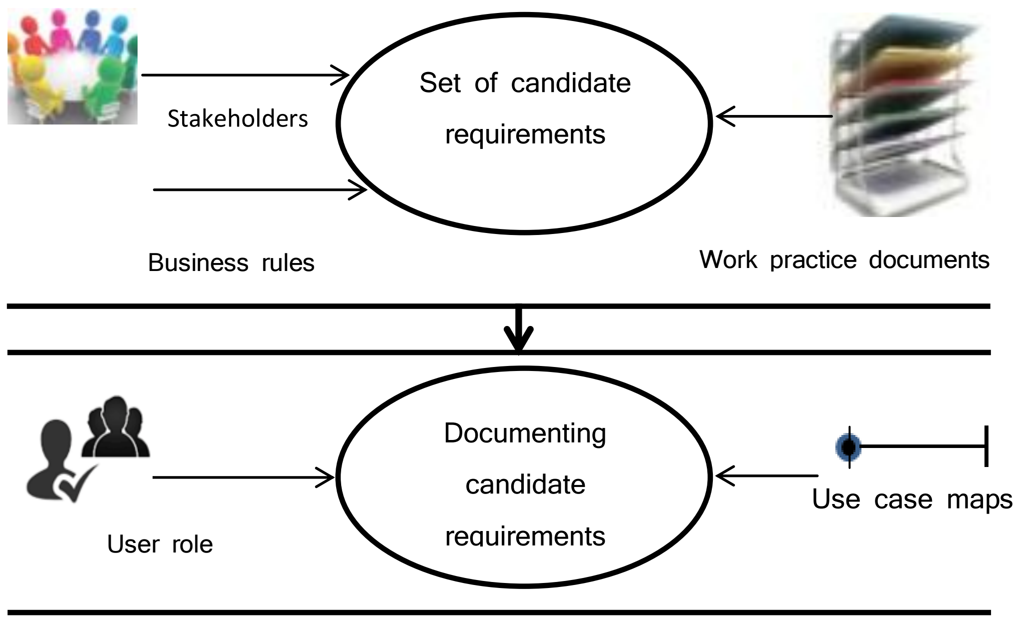

Fig. 1 Conceptual overview of the elicitation process

A. Set of candidate requirements

The objective of this activity is to understand the system to be developed and familiarize the developer to the problem domain and to both gather and record the candidate requirements from stakeholders.

An interview with the potential stakeholders and observations of the work documents flow can be used to elicit these requirements. Interviews may include one or more stakeholders. It may also involve a question and answer session used to discover other potential stakeholders and any contradictions between these requirements.

Interviews facilitate obtaining the approval from stakeholders on their requirements and any changes to them.

Before eliciting the stakeholders' requirements, the developer should know the business rules and conditions that the decisions are relying on it, and the procedures that control and define the work execution of the system and consider their impacts on the work being developed.

Based on the information obtained during this activity, the developer creates some documents which can be used as a base for further activities.

- ***Stakeholders' identification***

The first step towards discovering all the requirements is to understand who the stakeholders are, and what roles they are expected to play i.e. understanding the project's sociology (an organizational structure can be used to discover the other related stakeholders). System's stakeholders are the people who are closely related to the system, they can be direct users, managers of the users, other systems that interact with the developed system, and the developers who work on them [1]. This ensures that all the stakeholders have a better understanding of the requirements process, and as a consequence, the task of obtaining a stable and complete set of requirements becomes easier. Furthermore, an incomplete understanding of stakeholders needs may result into missing or erroneous requirements which consequently develop a wrong software solution.

Next, the developer should write the required information about the stakeholders of the system such as, their names, positions, and descriptions of their relation with the system.

A simple example of stakeholders in the health system is shown in table I. Doctor has a direct interaction with the system. S/he focuses on treating the patients and follows them, while the Pharmacy focuses on giving advice to the patients on the medicines as prescribed by the doctor.

Table I: An example of stakeholder's definition record

Stakeholder name	Position	Relation with system
Doctor	Medical staff	Responsible on medical treatment to patients
Pharmacist	Medical staff in a hospital or on private pharmacy	Advice and give the medicine as prescribe by the doctor

- ***Inspecting work documents***

Requirements elicitation is an essential process to capture the stakeholders needs (user requirements). In the beginning, the customers might not have an enough ability to determine their needs precisely. To simplify that, we identify the user's tasks (responsibilities) that currently perform according to the roles they play, and through the inspection of their work practice documents. This will be considered as a starting point of the negotiation with their software developers. Also, it helps in taking the customer's feedback about their initial requirements. Fig. 2 illustrates an example of work responsibilities document for a doctor.

User role name	Doctor
Role Responsibilities	Business rule(s) and constraints
• Checkup patient	Check the patients history
• Prepare medical report	Write diagnosis and progress report
• Write prescriptions	Check history and diagnosis reports

Fig. 2 An example of work responsibilities record

At the end of this activity, the developer has known the business rules and work constraints, identified the stakeholders, procedures and the descriptions of responsibilities.

B. Documenting user candidate requirements

In this activity, the candidate requirements will be conducted to their relative user roles. Each requirement will be demonstrated in the use case map notations to show the responsibilities of the system and the user to execute the requirement. Stakeholders' feedback should be documented and managed in the form of update, change in requirements or even adding new requirements, and each validated task will be documented.

- ***User role model***

Building an initial prototype for the candidate requirements are useful for the customer's satisfaction and for maintaining their feedback about analyzing of the current work. This process helps in many ways for updating, modifying or even getting new requirements that the customer needs.

For simplifying this process, we propose to start by identifying the candidate requirements that are conducted by each user role through A UML use case diagram (Fig. 3). This diagram can be used to represent the behavior of user systems [14]. We adopt to use this diagram to show the interactions between the proposed users in the system according to their tasks that they are responsible off (i.e. each role responsible for one or more tasks). This will lead to validate and detect other user requirements that might be missed. Additionally, the use of cases association relations (e.g. "extends" and "includes") will improve the concept of task (user requirement) completion; because the incompleteness of a user requirements can cause errors or exceptions later which may be expensive to repair.

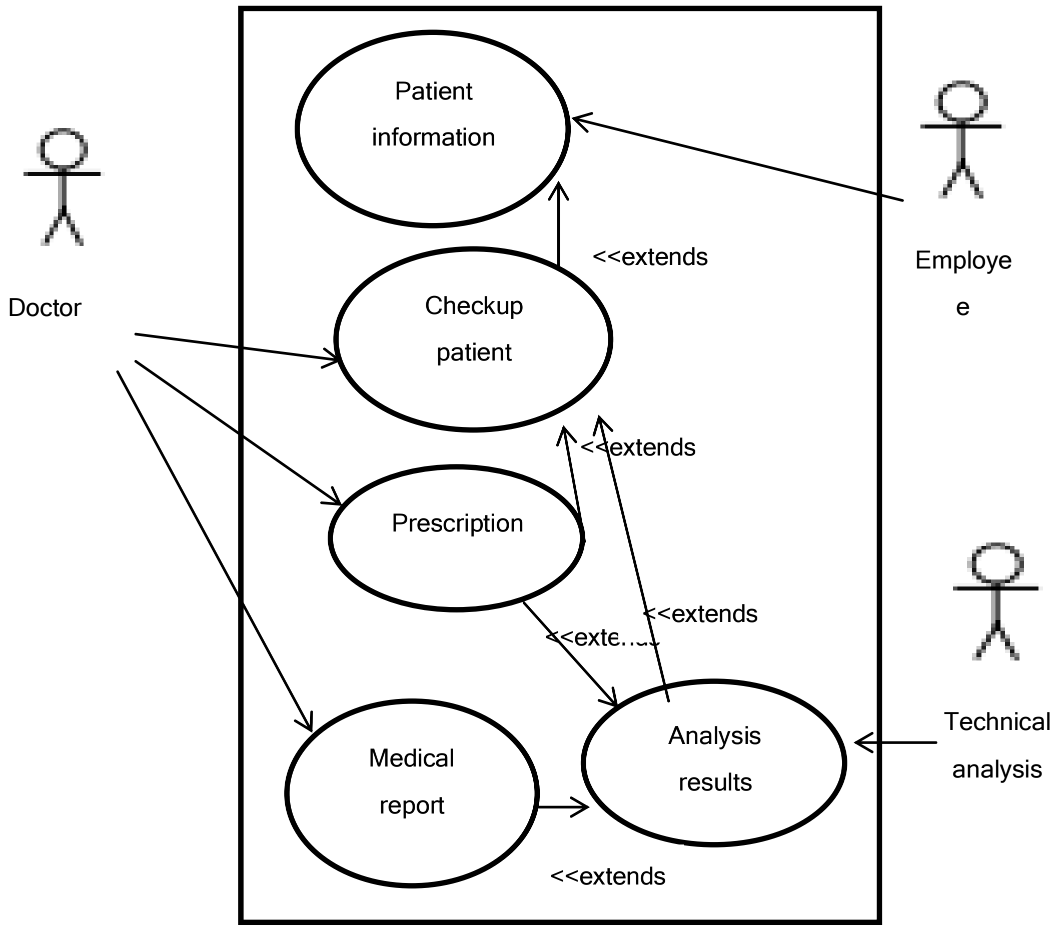

Fig. 3 User role model for a simple clinic system

As shown in the figure, Doctor is responsible on four user requirements, Employee is responsible on one user requirement and the Technical analysis is responsible on one user requirements. Association relations show that there are dependencies between some user requirements.

- ***Visualizing user requirements***

Clarifying user requirements is important for generating complete and precise requirements. It is well known that customers usually cannot articulate their needs in a complete form without the help of developers. To clarify the definition of user requirements, a use case map (UCM) notation to clarify and validate the user requirements in a visual manner at a system level to get clear customers' feedback.

Elements of UCM can be combined in one or more components, and each component represents an entity responsible on the responsibilities inside it. It describes the causal relations between responsibilities on the paths that illustrate the system functionalities. Paths represent the continuation of system scenario, which are represented by wriggling lines. Each path consists of some responsibilities (actions or operations, represented by (Bold "X") that is a component responsible to perform it. The filled circle represents the preconditions of the scenarios and the solid bars represent the ending points of the scenarios (i.e. post conditions), and a plug-in stubs (plain diamond) which can be used to decompose a complex maps [15]. We assume to use the plain diamond when a scenario contains another scenario that complements it (for example, prescription will complement the check-up process, so prescription will be represented as a plain diamond in a check-up scenario). To simply the prototype, we will use two components representations; one for a system and the other for a user to show the interaction between them (Fig. 4).

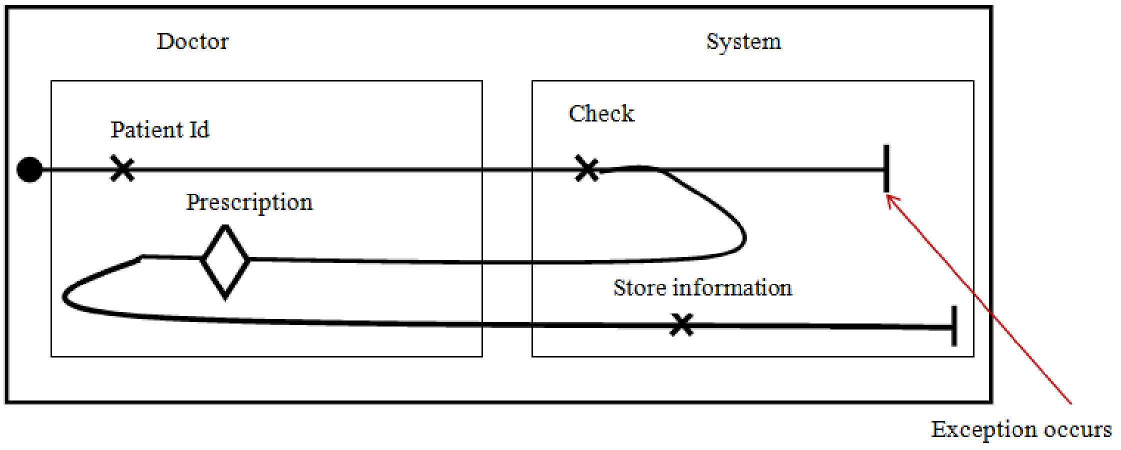

Fig. 4 "Check-up patient" use case example

- ***Goal model***

This model is derived from the user role model. It describes the plan of a complete goal, where each goal consists of one or more user requirements to fulfil (Table I). If a goal is not achieved, it means user requirement(s) has terminated (an exception has occurred). This means the normal sequence of the user requirement is terminated (as shown in Fig. 4), an exception is occurred, the goal is not achieved and the prescription process is not performed; i.e. goal terminated). Uncovered exceptions will lead to an incomplete system specification. This model (Table II) consists of two columns, goal column represents the name of the goal to be achieved, and user requirement column represents the list of user requirements that are required complete the goal.

Table II An example of process goal model

Goal	User requirement
Treat patient	1. Check-up
	2. Prescription
	3. Store information
Record patient information	1. Patient info

As shown in the above table, the example consists of two goals “Treat patient” and “Record patient information”. The "Treat patient" goal will be achieved when all of its sub-goals (Check-up, prescription and store information) are fulfilled, while the "Record patient information" goal requires the patient information to be complete.

Finally, customer's feedback is important, and it may lead to update or capture other user requirements, and also the developer may also propose addition of further requirements which are normally missed. These modifications or additions can be clarified and validated using the proposed approach.

III. CONCLUSIONS

From literature, eliciting complete, unambiguous and necessary requirements are one of the most important challenges that face software developers. These challenges cause systems failures. One of such failures is the abnormal situations that a software system may face (i.e. exception).

In this paper, we presented an approach that is efficient and effective for generating complete and unambiguous user requirements in a visual manner, and through organized steps that can help software developers to overcome the shortcomings of the existing techniques. This approach can be used as a guideline to cover all the necessary requirements in a precise manner.

ACKNOWLEDGE

We would like to express our gratitude to Professor Mohamed El-ammari and Dr. Radwan Abuissa for their editing and comments.

REFERENCES

[1] Nisar, S., M. Nawaz, and M. Sirshar, *Review analysis on requirement elicitation and its issues.* Int. J. Comput. Commun. Syst. Eng.(IJCCSE), 2015. **2**: p. 484-489.

[2] Davis, C.J., et al., *Communication challenges in requirements elicitation and the use of the repertory grid technique.* Journal of Computer Information Systems, 2006. **46**(5): p. 78-86.

[3] Alrajeh, D., et al. *Generating obstacle conditions for requirements completeness*. in *Proceedings of the 34th International Conference on Software Engineering*. 2012. IEEE Press.

[4] Sutcliffe, A. and P. Sawyer. *Requirements elicitation: Towards the unknown unknowns*. in *2013 21st IEEE International Requirements Engineering Conference (RE)*. 2013. IEEE.

[5] Davey, B. and K.R. Parker, *Requirements elicitation problems: a literature analysis.* Issues in Informing Science and Information Technology, 2015. **12**: p. 71-82.

[6] Babar, M.I., M. Ramzan, and S.A. Ghayyur. *Challenges and future trends in software requirements prioritization*. in *Computer Networks and Information Technology (ICCNIT), 2011 International Conference on*. 2011. IEEE.

[7] Ruhe, G., A. Eberlein, and D. Pfahl, *Trade-off analysis for requirements selection.* International Journal of Software Engineering and Knowledge Engineering, 2003. **13**(04): p. 345-366.

[8] Sadiq, M. and M. Shahid, *Elicitation and prioritization of software requirements.* International Journal of Recent Trends in Engineering, 2009. **2**(3): p. 138-142.

[9] Ma, Q., *The effectiveness of requirements prioritization techniques for a medium to large number of requirements: a systematic literature review*. 2009, Auckland University of Technology.

[10] Konaté, J., A.E.K. Sahraoui, and G.L. Kolfschoten, *Collaborative requirements elicitation: A process-centred approach.* Group Decision and Negotiation, 2014. **23**(4): p. 847-877.

[11] Cailliau, A. and A. van Lamsweerde. *Integrating exception handling in goal models*. in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. 2014. IEEE.

[12] Van Dooren, M. and E. Steegmans. *Combining the robustness of checked exceptions with the flexibility of unchecked exceptions using anchored exception declarations*. in *ACM SIGPLAN Notices*. 2005. ACM.

[13] Lennon, A.M., et al., *The early detection of pancreatic cancer: what will it take to diagnose and treat curable pancreatic neoplasia?* Cancer research, 2014. **74**(13): p. 3381-3389.

[14] Kösters, G., H.-W. Six, and M. Winter. *Validation and verification of use cases and class models*. in *7th International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'2001, Proc.)*. 2001. Citeseer.

[15] Amyot, D. and G. Mussbacher. *On the extension of UML with use case maps concepts*. in *International Conference on the Unified Modeling Language*. 2000. Springer.