A systematic approach to generate and clarify consistent requirements

Mohamed A. Hagal
Software engineering Department
Benghazi University
Benghazi, Libya
Mohamed.hagal@Benghazi.edu.ly

Sohil F. Alshareef
Software engineering Department
Benghazi University
Benghazi, Libya

Abstract— **The development of software systems inevitably involves the detection and handling of inconsistencies. These inconsistencies can arise in system requirements, design specification and quite often in the descriptions that form the final implemented software product. Now days we see requirement engineers focusing their attention to achieve a solid system and user requirements. This involves requirement quality attributes such as completeness, accuracy and consistency of requirements. Basically the starting point of the development of any software system is the requirement engineering phase, projects with inconsistent requirements face problems project containment, the reason for this risk is the difficulty to keep track on requirements during the development, and of course the evolution of descriptions and software requirement specifications without monitoring the changes in the development. Therefore in this paper, we present an approach to increase the degree of requirement consistency in a systematic steps guide the requirements engineer towards generating precise requirements.**

Keywords— ***Inconsistency management, software requirement specification,prototyping, requirement management.***

I. INTRODUCTION

Software requirement can be considered as specification of particular stakeholder's concern. It can contain design constraints and it should not include low level information such that, design details, project planning, implementation details and testing information. For that reason, the phase of requirement engineering focus on understanding what teams intend to build[1]. A good requirement is a requirement that is, verifiable, clear and concise, complete, consistent, traceable, necessary and implementation free. These characteristics represents a filter, that a good requirement should satisfy.

Through the process of Requirement Engineering, there are a large number of documents used by different stakeholders to document system requirement; these include a Statement of Work (SOW), Concept of Operation Document (CONOPS), Operational Requirements Document (ORD) and Software Requirement Specification (SRS). These documents are written and revised by various stakeholders at different times during the RE process[1]. But, the inconsistency among and within these documents may arise because the difficulty to check a large set of documents for consistency. As the number of requirements grows, it soon becomes infeasible to maintain their consistency. Inconsistency may arise in early stages of software development, and failure to identify it early may cause system failure[2,3]. In general, when inconsistencies are identified in a specification, this mean that this specification contradicts itself, or that a logical contradiction can be derived directly from this specification. If there are contradictions among the descriptions of different requirement or within the same requirements, these also inconsistencies.

Inconsistency in [4] is defined as "any situations in which two descriptions do not obey some relationship that is prescribed to hold between them", while the relationship between descriptions can be expressed as consistency rule.

As the requirement engineers fix the inconsistencies in descriptions or (SRS), they might face some of the axioms of the inconsistency such as[4]:

- Some inconsistencies are never fixed: if the cost of their outweighs the risk of ignoring them.
- Living with inconsistency: monitoring each unresolved inconsistency and their affect.
- Inconsistency is deniable: checking if all reported inconsistencies are really inconsistencies.

The choice of an inconsistency handling depends on their impact on other aspects of the development process. Resolving it may be updating/modifying information from a software description. Table 1 illustrates the summary of inconsistency handling strategy[4] [5] [6].

In addition to, detecting inconsistencies are one of the critical tasks in validating requirements. Bashar Nuseibeh and Steve Easterbrook [6] developed a framework based on well-defined consistency rules. These rules must be expressed precisely before they can checked against the software artifacts in question. They used these rules as input to the inconsistency management framework, by monitoring the evolving descriptions whenever the inconsistency is detected the location, identification and classification of that inconsistency that broke the rule must diagnosed. When diagnosis is performed, the process of handling is considered with wither tolerate its risk and impact or resolve it immediately if the risk is critical to the development. Bashar Nuseibeh, Steve Easterbrook and Alessandra Russo [4] stated some problems of keeping the descriptions maintained, how consistency rules are

978-1-4799-2845-3/13/$31.00 ©2013 IEEE

established and extracted from those descriptions, also the reasons that may lead to the arise of consistency rules. Determining whether a set of descriptions is inconsistent depends on knowing what relationships should hold between them. The authors stated that, in fact there is no escape from inconsistency, but in this context few handling strategies were established, ignoring the inconsistency, ameliorating the inconsistency, deferring the inconsistency and circumventing the inconsistency.

In [7], the authors defined the step of analyzing and detecting inconsistencies in three tasks: capturing the requirements, requirements definition and requirements validation. The detection of inconsistencies is focused in the last task. The approach focuses on the detection of inconsistent specification of contexts in a goal model, and the detection of conflicting context changes that arise as a consequence of the actions performed by the system.

TABLE1. INCONSISTENCY HANDLING STRATEGY

Strategy	Why, reason or rationale	What should be done
Ignoring	The effort of fixing an inconsistency is too great relative to the low risk that the inconsistency will have any adverse consequences	Dictating that the decision should be revisited as system evolves or project processes.
Deferring	Provides developers with more time to elicit further information to facilitate resolution or to render the inconsistency unimportant	Isolating the inconsistency and flag the parts of the descriptions that are affected, and continue development while the risk is tolerated.
Circumventing	This implies that the rule is wrong or because the inconsistency represents an exception to the rule that had not been captured.	Modifying the rule by turning it off for a specific context.
Ameliorate	It may be cost effective to improve the descriptions that contain inconsistencies without necessarily resolve them all. it includes adding information to the description that alleviates some adverse of an inconsistent, or solves other inconsistencies as a side effect.	Use ameliorate as a tool to continue development, and push the project into desirable direction.

In [8], the authors presented a Multiple Criteria Decision Making method to support aspectual conflict management in aspect oriented requirements treatment and deals only with concern conflicts by applying consistency rules among view points.

In [9], the authors presented an approach for detecting conflicts and inconsistencies in web application requirements, by characterizing the kind of inconsistencies in web application requirements and how to isolate them using a model driven approach.

II. THE PROPOSED APPROACH

The main objective of this approach is to prevent the risk of inconsistency from having a major impact on the software development. Our approach defines a framework for generating requirements in a several systematic steps takes into consideration the degree of consistency in earlier steps. The proposed approach consists of the following steps:

- Requirements elicitation: using a well-known requirements technique such as interviewing. During this activity there will be a several questions that are key-targeted to the problem and the objective of the system that will be developed.
- Studying the work documents: Tracing the work documents, and draw a sketch illustrates their flow. This sketch helps in generating the candidate requirements and to highlight the dependency (priority) between them. Figure 1 bellow illustrates an example of such sketch.

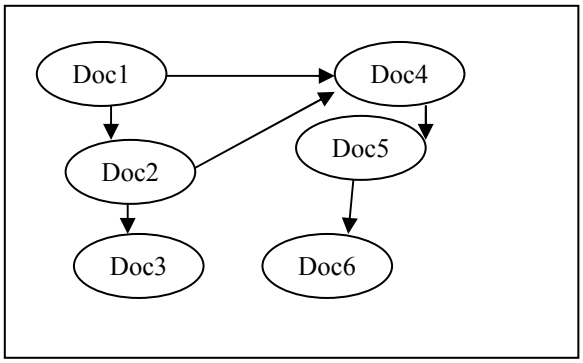

FIGURE 1: AN EXAMPLE OF WORK FLOW DOCUMENTS

- Document the purpose of each work document and its relation with other work documents. Documentation can include such elements: Document number, document purpose, extracted requirements, an indication to other related documents and the relations description.

- Document the requirements viewpoints traceability table which contains the name of the candidate requirement and the stakeholders viewpoints to it. This will lead to a common understanding and sharing ideas between system stakeholders. All viewpoints about each requirements should be documented and then discussed.
  - For example, in a library system two stakeholders have two viewpoints for loan books from the library. the first told that the system should ensure book loaned as long as needed, and the second stakeholder says the system should ensure loaned books returned in one week. So, this will cause inconsistency requirement if not managed. So, in this case all viewpoints should be documented. Table2 bellow illustrates such example.

TABLE2. STAKEHOLDERS VIEW POINTS REQUIREMENTS DOCUMENTATION

Requirement Id	1	*Requirement name*	Loan Book
Stakeholder	Viewpoint description		
A	The system ensures the book loaned as long as need		
B	The system ensures the books loaned returned in one week		

- Build a prototype model for each requirement to be clarified. If a requirements has two inconsistency viewpoints, then the prototype will be build for the first viewpoint, and then negotiated with the other stakeholders how have another viewpoints to reach a common understanding, as well as to get the final specification of the requirements. This task can be done using two steps: (1) using UML activity diagram to illustrate the flow of actions of each requirement. (2) then, An equivalent UI prototype (or paper prototype) in the form of windows navigations to represent the actions presented in the activity diagram in a visual manner which simplifies the understanding of requirements. Figure 2 bellow illustrates an example of a UML activity model and its equivalent prototype.

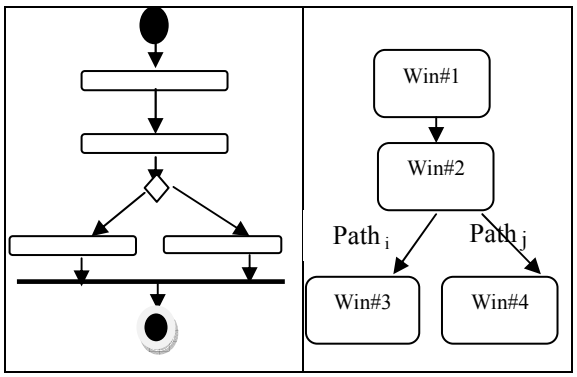

FIGURE2. AN EXAMPLE OF UML ACTIVITY DIAGRAM AND ITS EQUIVALENT PROTOTYPE

Finally, build a consistency traceability table to indicate how requirements are related to one another. This will contributes in assess the impact of change in one requirement may cause a change in another one. For that, if requirements are properly traced it facilitates tracking changes through the different levels of specifications. Table 3 illustrates such traceability table. A character "X" can be used to indicate to the existence of dependency between requirements.

TABLE 3. REQUIREMENTS CONSISTENCY TRACEABILITY TABLE

Req. Id.	R_1	R_2	…..	R_n
R_1				
R_2	X			
:				
R_n				

III. CONCLUSION

Although extracting software requirements is a daunting task, requirements engineers attempt to generate requirements that free from inconsistency and incompleteness. for that, many approaches have been defined to avoid or reduce these problems, but these approaches depends on their experiences. Moreover, generating requirements caused by the adaptation with the changes of software requires more time and effort. In this paper, the proposed approach based on tracking work flow documents have been used to extract functional requirements satisfied stakeholders' needs in a systematic way. presented. The interesting tension in this approach is between extraction of requirements by using the work flow documents and clarifying theses requirements using a UML activity diagram to illustrate the flow of actions of each requirement, and then presenting them in equivalent prototypes in the form of windows navigations.
Further work at this topic will include applying case studies for the purpose of its evaluation.

REFERENCES

[1] Asghar S., Umar, M., Requirement Engineering Challenges in Development of Software Applications and Selection of Customer-off-the-Shelf (COTS) Components, International Journal of Software Engineering (IJSE). Vol 1, Issue: 1, May, 2010.

[2] Khalidi R., A New Technique for Detecting and Locating Inconsistencies in Software Requirements, proceeding 11th international arab conference on information technology (ACIT2010), 2010.

[3] Nuseibeh B., Easterbrook S.; "The Process of Inconsistency Management: A Framework for Understanding , Proceeding 1st International Workshop on the Requirements Engineering Process (REP'99) , Sep. 1999.

[4] Bashar Nuseibah, Steve M. Easterbrook and Alessandra Russo,Making inconsistency respectable in software development, Journal of systems and software, 58(2):171-180, 2001.

[5] Bashar Nuseibeh, Steve M. Easterbrook, and Alessandra Russo, Leveraging Inconsistency in Software Development.
, *IEEE Computer 33(4):24-29*, *2000.*

[6] Bashar Nuseibeh, Steve M. Easterbrook: The Process of Inconsistency Management: A Framework for Understanding. DEXA Workshop, 1999.

[7] Fabiano Dalpiaz, Paolo Giorgini: Reasoning with contextual requirements: Detecting inconsistency and conflicts. Information & Software Technology 55(1): 35-57, 2013.

[8] Vieira, F., Brito, I., Moreira, A., "Using Multi-criteria Analysis to Handle Conflicts During Composition", Workshop on Early Aspects, 5th International Conference on Aspect-Oriented Software Development (AOSD 2006), Bonn, Alemanha, 20 March 2006.

[9] Matias Urbieta, María José Escalona Cuaresma, Esteban Robles Luna, and Gustavo Rossi.ICWE Workshops, volume 7059 of Lecture Notes in Computer Science, page 278-288. Springer, 2011.