ISSN: 0976-3104
SUPPLEMENT ISSUE

ARTICLE

A FRAMEWORK FOR REMOVING AMBIGUITY FROM SOFTWARE REQUIREMENTS

Sophia Segal*
Director, Business Analyst Solution, Toronto, CANADA

ABSTRACT

Un-ambiguous, transparent software requirements written by accomplished Systems analysts analysts are a rarity. Too frequently, software requirements are vague and open to interpretation, leaving development teams unintentionally deviating from the project requirements at considerable expense and delay. The implementation of a single software requirement is replicated in a number of modules, so rework will not only affect a single module but a number of modules and there is a huge possibility that the budget will increase from the initial estimate [1]. Seventy percentage of software projects fail due to poor requirements with an associated rework spend just north of 45billion USD annually. (Source: Leveraging Business Architecture to Improve Business Requirements Analysis). The strategies for project recovery report by PM solutions, is based on 163 respondents. It states that 74million USD invested in projects annually are at risk of failure. The main cause of troubled projects are challenges that can be addressed by functional requirements that are vague and not transparent.

KEY WORDS
Software Testing, Removing Ambiguity, Critical Analysis

INTRODUCTION

Project success is contingent on transparent requirements. Transparent functional requirements are critical in determining the success of software projects. Clearly documented functional requirements provide transparency and are critical to the project team, so all stakeholders are working towards the same project goals. There is no wasted effort on non-value requirements that do not contribute to the business needs. "If you don't have transparent requirements allowing you to comprehend the project goals and business needs, you cannot decipher whether the decisions you or your project team make are correct." To reduce vague, unrealistic and unachievable requirements being relinquished to the Developers team, functional requirements need to be formulated and documented in such a way which makes them transparent, testable and unambiguous to the project team. Capturing the right level of granularity of a functional requirement is key in avoiding ambiguity in software engineering projects and eliciting a concise, testable functional requirement. This paper discusses different approaches which can be adopted to remove ambiguity from a Software requirement, and how each approach lends itself to validation in the testing phases.

Received: 15 Jan 2017
Revised: 17 Feb 2017
Published: 12 March 2017

WHY IS AN UN-AMBIGUOUS REQUIREMENT CRITICAL FOR PROJECT SUCCESS?

Concise, software requirements written by accomplished Systems analysts are scarce. Too frequently, requirements are poorly written and open to interpretation, coupled with lack of collaboration between Analysts and Testers to conceive and trace, testable requirements from customer need, through strategy, down to testing and implementation is a big enabler of software project failure The definition of a functional requirement is 'A requirement which specifies what the system should do. 'Software developers don't implement business requirements, they implement Functional requirements, specific bits of system behaviour that allow users to execute use cases and achieve their goals'. Ambiguous requirements often leads to testers and developers to unknowingly misinterpret them, which causes requirement errors to slip through UAT and morph rapidly into defects. These defects surface after implementation, at a time when it costs tenfold to pursue and resolve the same defect. It is therefore important to capture and define these shifting, ambiguous requirements and flesh out meticulous acceptance criteria so that the intended results are visible to all. Ambiguity can be one of a range of critters. These include

1. Ambiguous terms: subjective or vague terms that cannot be measured
2. Conflicting requirements: Two or requirements that conflict each other
3. Incomplete requirements: Missing values, business rules, etc.
4. Missing requirements: Possible missing requirements that have not been defined

5. Unclear requirements: Requirements that can be interpreted in multiple ways.
6. Glossary: Term is not found in the glossary reference document
7. Grammar, spelling and wording: spelling mistake, grammar rewording suggestion.

A common denominator in software engineering projects is the techniques or approaches applied to requirements elicitation, requirements analysis and validation [2,3,4]. These approaches have been adopted to consistently remove ambiguity in requirements and are more testable, which leads to long-term visibility of the solution need. The two main approaches are: 1) Use Case 2) Acceptance Criteria

***Corresponding Author**
Email:
sarahdorabrown@ymail.com
Tel.: +1 647 986 3181

USING THE USE CASE APPROACH

Use case diagrams emphasize the functionality requirements of a system and have become a frequent practice in software development projects. They are best utilized to provide a high-level description of what an existing or proposed system should be able to do and who or what will interact with it. It is a highly regarded technique for specifying and documenting functional requirements in software projects and are often used as supplementary documentation to the business requirements document which encompasses functional and non-functional requirements. I have also adapted them for storyboard tools for agile user meetings. They define the requirements of the system being modelled and hence are leveraged to write test cases for the modelled system. Imagine travelling to an unexplored exotic country where your plans involve renting mode of transport and touring the local sights. Most travellers would not contemplate this trip without a travel sightseeing book. Despite the significance of a travel sightseeing book, or map, the development team or Quality Assurance team, too often plunge into the Solution phase without them. As a result, development teams and QA teams can unintentionally, deviate from the project objectives and software requirements at considerable expense and delay. The Use Case also provides an added layer of functional requirement details and depicts low level behaviour and usability on how users would interact with the software whereas the business requirements document is more high level and independent of Solution implementation. Use Case are a valuable approach to remove ambiguity in what the solution is delivering by eliciting conversation and collaboration with stakeholders so all functional requirements are clear and visible Formulate Use case descriptions providing the sequence of steps an actor has to go through to achieve a goal and also provided details on exception and alternative flows. Each Use Case's purpose is to capture common user functionality requirements. Use cases can be supported by textual descriptions known as use case descriptions to provide greater level of detail on system functionality. Use cases focus on the "what" and not the "how". Below is a simple example and basic rules of a use case description.

An actor can be a person, company, software or external entity that interacts with the system. For example, a customer, a bank, a database can be referred to as actors. Actors are modelled as being external to the system boundary. A use case is an action or functionality that is performed within a system. It is represented as a combination of a verb and a noun, e.g. Print Report, Prepare Invoice; Display Amount. Main Flow is the regular flow of action in an application. The main flow of events describes a single path through the system that results in a user completing their goal. For example, in the login page after entering user name and password the standard next screen should appear is considered the main flow. An exception flow is the unintended path through the system.They are the flow of events which occur in case of the errors or exceptions. Alternate flow are any alternative actions that can be performed or variations from the main flow, if not selected. The alternate flow formulates a scenario other than the main flow that results in a user completing their goal. A precondition of a use case explains the state that the system must be in for the use case to be able to start. A prerequisite that needs to be met before the use case is triggered. A post-condition of a use case lists possible states that the system can be in after the use case is at its end state. The post-event that is actioned after the Use Case end state.

According to BABOK V2, the action "extends" allows an analyst to articulate additional behaviour of a parent use case. The parent use case is entirely functional on its own and is independent of the extending child use case. An extension is functionally similar to an 'alternate flow, but is captured in a separate use case to avoid confusion.

Use Case: A user need to enter his/her ID and password to get signed in. The login page consists of two fields and both field need to be filled in order to proceed. Once the login is successful, the user is redirected to his/her account.
Test Cases: There can be a number of test cases which will help us determine if the above functional requirement is met or not. These are listed below

Table 1. Test Case vs Expected Outcome

Test case	Expected outcome
1. Clicking the login button by keeping both the ID and passwords fields empty	Error messages against both field prompting the User to enter ID and password. Login attempt unsuccessful
2. Clicking the login button by keeping the ID filed empty while the password is entered correctly	Error message prompting the user to enter ID login attempt unsuccessful
3. Clicking the login button by keeping the password field empty while the ID is entered incorrectly	Two different error messages. One error message prompts the user to enter the correct ID while the other one prompts the user to enter password. Login attempt unsuccessful
4. Clicking the login button by keeping the password field empty while ID is entered incorrectly	Two different error messages.one error message prompts the user to enter the correct password while the other one prompts the user to enter password. Login attempt

		unsuccessful
5.	Clicking the login button by keeping the ID filed empty while the password is entered incorrectly	Two different error messages. One error message prompts the user to enter the correct password while the other one prompts the user to enter ID. Login attempt unsuccessful
6.	Clicking the login button by entering incorrect ID and password	Error messages against both fields prompting the user to enter the correct ID and password. Login attempt unsuccessful
7.	Clicking login button by entering the correct ID and password	Validating user ID and password login attempt successful

How Use Case Requirements support testability

A Use Case plays an integral part in defining a Test Case and functional requirements modelled in the form of a use case can serve as a valuable source for test cases. Capturing functional requirements in the use case illustrates exactly how functionality works in a current system, ensuring that critical functionality requirements are not overlooked. Test cases are drawn from functional specification documents, business requirement documents, design decisions or prototypes. If all the possible control flows of a system are captured, then accordingly, all scenarios of control flows can be captured and tested to verify and validate the desired output. While use cases significantly differ from test cases, they guide the QA team to elaborate Use Cases into Test Cases. "Pre-Conditions", "Post-Condition", "Main Flow", "Alternative Paths", "Exception Paths", and "Business Rules" are all source material for creating complete test scripts and establishing both the Use Case and Test Case are aligned. I have addressed below, an example, illustrating simple functionality which requires a user to enter their ID and Password in order to login to their account. While the Use Case defines the mechanism through which this functionality will be achieved, Test Cases helps capture all the possible scenarios of this particular Use Case functionality. Use Case: A user needs to enter his/her ID and Password to get signed in. The login page consists of two fields and both fields need to be filled in order to proceed. Once the login is successful, the user is redirected to their account. Test Cases: There can be a number of test cases which will help us determine if the above functional requirement is met or not. These are listed below:

What is Acceptance Criteria and how does it support Requirements Testability

Acceptance criteria are a set of statements, expressed in clear, structured English language, each resulting in a pass or a fail that specify both functional and non-functional requirements which examines if the software requirement has been met or not. There are specific boundaries to the acceptance criteria which provides clarity to the completeness of the software requirement.

Defining concise acceptance criteria is key to a complete Test Case. Not only does it clearly illustrate what the user expects from a scenario, how the requirement should be met, but also verifies the quality and scope of a test case scenario and exit criteria. This is calculated by counting all the acceptance criteria, including scenarios, and dividing the number by how many acceptance criteria have been completed with the expected results.

CONCLUSION

While quality use cases capturing functional requirements may seem time consuming and tedious, the result is a foundation for work by the analysis team, couples with collaboration with the development team, and the testing team. Good Use case documentation provide a valuable return on the analysis team's investment in time and resources. It is good business practice that the Use Case and Test Cases are aligned. The Use Case approach abundantly illustrates the functional requirements that a user will perform with a system. Methodically thinking through the tasks that are involved between user and system fleshes out any requirements which are ill-defined earlier in the software projects, as does generating test cases from use cases. 'If the use Cases for a system are complete, accurate and clear, the process of deriving the test cases is straightforward. '[karl wiegers] Project success is contingent on systems analysts delivering transparent requirements.

Project success is contingent on good requirements and collaboration. Requirements that are testable are critical in determining the success of software projects.

Acceptance criteria for requirements provide transparency to the project team, so they are working towards the same goals. There is no wasted effort on non-value requirements that do not contribute to the business needs and there is a definite boundary to when a requirement has been met. "If you don't have transparent, testable requirements allowing you to comprehend the project goals and business needs, you cannot decipher whether the decisions you or your project team make are correct."

CONFLICT OF INTEREST
All the authors declare no conflict of interest with in this research.

ACKNOWLEDGEMENTS
None

FINANCIAL DISCLOSURE
None

REFRENCES

[1] Engineering and Managing Software Requirements Editors: Aurum, Aybüke, Wohlin, Claes (Eds.), Participation of QA Testers in the requirement definition process
[2] http://searchsoftwarequality.techtarget.com/answer/How-QA-testers-participate-in-the-requirements-definition-process
[3] Karl wiegers, software requirements.Chennai soils. Geotechnical and Geological Engineering, 28(2):119-137.
[4] Raptis, G. E., Katsini, C. P., & Payne, S. J. (2013, September). VirDO: A Virtual Workspace for Research Documents. In International Conference on Theory and Practice of Digital Libraries (pp. 470-473). Springer Berlin Heidelberg. doi:10.1007/978-3-642-40501-3_63