

Requirements Quality Assurance in a SDLC

[Square brackets contain comments for the reader (and myself) and are not intended to add content to this article.]

1 Introduction

The contents of this article are drawn, in part from my experience, in part from ideas with which I have experimented, the rest I fabricated for the purpose of furthering the ideas in this article.

This paper was written in response to a comment I made on the RQNG website about wishing to "read more about quality assurance in working with requirements." At the request of RQNG, I am going to expand on a section in my book that deals with this subject.

[BTW; the book 'Analysis Through Pictures' is available for free download from www.lulu.com – and that is the only plug I intend to include in this paper.]

Back in 2009, when I was compiling chapters for the book, I posted several drafts to RQNG for comment. Since then I have released version 3 for download. Chapter 1 of the book gives an overview of my personal experience working with requirements, and what I consider some of the best practices I have learned from this experience.

This paper expounds upon some of the best practices described in this chapter which concern requirements quality, by comparing requirements quality assurance with that of software development. This paper starts by quoting directly from Chapter1; the Introduction section. I then go on to further examine some of the recommendations from the same chapter.

1.1 Requirements

From Chapter1; Section 1 of Analysis Through Pictures:

My experience of working with requirements spans more than 20 years. For me, a major enlightenment about the true purpose of requirements came about when tasked with documenting a presentation that described the characteristics of a 'good' requirement. The characteristics with which our team came up included the following:

- *Completeness – A functional requirement should describe all observable inputs, all observable outputs, when they can occur, who (actor which) is allowed to initiate the requirement and the maximum time in which the requirement is allowed to complete.*
- *Consistency – The requirement should not be in conflict with any other requirements for the project. For example, if one requirement states that all user inputs will be*

processed within 5 seconds of entry and another states that the system shall respond to a particular user input within 10 seconds, there is a conflict in the requirements.

- *Correctness – Are the requirements specifying a solution which the business wants implemented? For example, stating that a "data constant will be configurable" – does the business need this capability?*
- *Design independence – If there is another way of specifying the requirement which will result in a different solution to the problem, then it probably contains design information and should be abstracted to allow for all possible design decisions. For example, stating that when the main window is closed that the system will logout the user etc, is too detailed, suppose we decide not to build a windowed application? A better way to abstract the requirement would be to say that when the user exits the application that they will be logged out, etc.*

<end of edit of italics>

- *Feasibility – Is it possible to implement the requirement within the specified time and budget? (And, we might want to ask if there is there a positive Return On Investment (ROI) for implementing this requirement?)*
- *No negativity – Stating the system ‘shall not’ do something is either untestable or irrelevant. The requirements should state what the system will do; and anything else be considered out of scope. For example, instead of stating that something shall not be ‘red’, state the colors that it may take. Negativity example; Stating that the system ‘shall not’ allow a user to view user information, is already covered by a requirement that an administrator shall be able to view user information. The fact that it has been stated that administrators may view this information, excludes all other actors, unless explicitly stated. A rule of thumb is that the system does nothing unless it is stated as a function of the requirements.*
- *Relevance – Is the requirement in scope for the current effort. For example, do you need to be specifying an interface to another system, if there is no requirement to automate communication with that system for this release?*
- *Testability – Is it possible to define specific ranges of inputs and outputs for the requirement such that when the implementation of the requirement is executed all inputs cause the system to produce the specified outputs, otherwise the implementation of the requirement fails? For example, when declaring that a function will be available during ‘business working hours’, have those hours been defined?*
- *Non-Ambiguity – The requirement should not contain any words that are open to interpretation. This includes all adjectives and adverbs, unless they are clearly defined in a project glossary. For example, the system shall allow ‘several’ users to change their profile. Unless the meaning of the word ‘several’ is explicitly defined*

this requirement has little meaning; and it can be satisfied by allowing exactly 2 users to change their profile and no more.

I then continue to list some 'good practices':

- Textual Requirements – Describes how to ensure that your written requirements are complete and consistent.
- Duplication – Discourages copying and pasting work (text or diagrams), from one place (or document) to another.
- Ambiguity – Discusses the problems with using the same term(s) with more than 1 meaning on the same project.
- Inconsistent Documentation – Concerns documents that (supposedly) describe the same information (or have similar purposes), but have different content, deviating from the approved document template.
- Unnecessary Documentation – Discusses the creation of documentation, without first identifying an audience.
- Document Content Organization - Relates to unnecessary documentation; this practice is concerned with putting the right information in the right place in the document.
- Ad Hoc Procedures - Concerns having a standard set of procedures for handling files and making sure everyone on the project understands how to use them.

The parts upon which I wish to expand, involve writing quality requirements.

1.2 Requirement Types

Business needs, stakeholder requests, business rules, supplementary requirements, non-functional requirements .. the list of requirement types includes too many to list them all here.

This document however, concerns Functional Requirements which may be considered the most complicated type of 'system requirement' in terms of management and control, in that they are likely to change and have an associated risk and impact on the architecture. The quality controls for other requirement types may include a subset of what is described herein.

Definition of a Functional Requirement – concerning software systems, describes activities which the system will perform, from a 'black box' point of view.

Black Box – meaning the activity performed by the requirement may be observed (or tested), through the interfaces to the system. This means that we do not know what is happening inside the 'black box', nor should the requirement make assumptions about

that activity because it is not observable through the system interfaces. [IME: One of the most important pieces of documentation that is overlooked when specifying a system is its interfaces between systems.]

2 Software Quality and Lifecycle

When discussing a requirements lifecycle, I find it helps to think of a functional requirement like a software function. [Probably because my background includes writing and designing software systems.] This section discusses quality in a typical software lifecycle.

2.1 Quality Control of Software

Software code generally follows a lifecycle from specification of the software through to deployment of the code.

During the code lifecycle, certain quality controls are involved which may include:

Standards – Ensuring that the software is written according to a consistent set of guidelines.

Version Control – Monitoring the history of the software during its development lifecycle.

Configuration Management - Ensuring that the correct software components are developed for a specific deployment (or build), in such a way that changing software does not impact the success of a previous build.

Code Reviews and Walkthroughs – Used to verify that the quality of the code meets the standards and requirements for the project.

2.2 Software Lifecycle

As software is developed it follows a lifecycle. (Note that there is no development method implied in the following lifecycle.)

The stages that a software module transition through are:

- Requirements – The requirements for the software are developed. The requirements are then solicited, analyzed and approved for development by the software team.
- Design – In this stage of the lifecycle the software is being organized into subsystems and modules. If you are using an Object-Oriented Design method, you will be organizing software into layers, packages and classes; classes which contain attributes and methods (amongst other characteristics).
- Implementation – During implementation the design of the methods are being detailed as lines of code and attributes are being defined as data.

- Testing – After implementation the code is tested to ensure that it performs according to its requirements. Once testing has passed, the code is baselined and placed under version control, to ensure that unexpected changes do not occur.
- Deployment – A code module is assigned to one (or more) software builds and a build is released into a production environment.

3 Requirements Quality and Lifecycle

Just as software has certain quality assurance controls imposed on it during its development so should the requirements.

3.1 Quality Controls

Functional requirements may be developed using similar quality practices to that of software code.

3.1.1 Standards

Using a standard format for the functional requirement – my preferred format to write textual requirements is:

'When' in a certain state and **'upon'** some externally visible event occurring, containing certain interface data, from a specific actor, **'the system shall'**, produce some externally visible output, **'within'** a certain time frame.

(Briefly: Since the functional requirement describes an activity, one assumes that the activity must have a precondition, a trigger and a postcondition. The start point is **'when'** the system is in a predefined state. The stop point is the visible output at the completion of the **'within'** time frame. The trigger is **'upon'** something happening to the system event.)

Since the functional requirement is a 'black box' type artifact all input and output data must be externally visible.

Text is by no means the only way to specify a functional requirement. You may use a spreadsheet to capture the information, use a formal language such as OCL, or even specify the information graphically using UML for example.

3.1.2 Versioning and Change Control

A functional requirement may take several states during its development. As the requirement passes through a gate to its next state, a version of the requirement is recorded along with attributes which describe the requirement in its current state and why it was changed. Together these versions give us a history of the requirement from its inception.

Working – In this state a requirement is conceived from discussions with the business and created in a requirements repository for the project. Alternatively, the requirement may already exist and is assigned to the project. Also, the requirement is uncontrolled and may be edited without any need for version control.

Under Review - In the state an initial version of the requirement is baselined and distributed for approval by the project stakeholders. Stakeholders will approve the requirement, not just for inclusion of its functionality into the project scope, but also QA will be verifying if the requirement meets the standards, project management will be evaluating the cost of the requirement, IT will be assessing the feasibility and work effort and testing will be assessing the requirement for testability.

Approved – Once the requirement has been approved for inclusion in the development effort it must undergo a change control process in order for it to be changed. At this point the requirement is being developed into software for a system build. All changes to the requirement must be approved and reviewed. The changes to the requirement are recorded and the impact of the change to the project budget and schedule is assessed.

3.1.3 Configuration Management

Ensures that changing requirements do not affect approved requirements using by separation of concerns, such as Object-Oriented Analysis techniques to package requirements.

As part of the approval process, the requirement is assigned to a project release. It is also assigned other attributes such as:

Risk – Describes possible impacts to the success of the project due to changes to the project schedule and cost that may not be fully assessed during specification of the requirement.

Complexity – How much will it cost to implement the requirement due to unknown and new technology.

Subject Matter Expert – The person responsible for clarifying questions about the requirement.

Responsible – The person who is maintaining the requirement.

Release – When is the requirement expected to be deployed?

Priority – How important is it to the business that this requirement is deployed.

These attributes are used to assign the requirement to a particular configuration, with the most risky and complex requirements assessed first. Priority is used to assess which requirements are considered most important to the stakeholders. All attributes are used to some extent to determine the order in which requirements are assigned to software builds.

3.1.4 Reviews and Walkthroughs

Just as code undergoes a walkthrough, requirements too are reviewed to ensure they meet the project standards, that they are within scope, are accurate (meet the needs of the business) and are complete (are not missing any features).

3.2 The Requirement Lifecycle

An equivalent lifecycle to the software code lifecycle might be described as follows, where the equivalent software state is in brackets (..):

Elicitation (Requirements) – The requirements are being solicited from the stakeholders. This is a subset of the software Requirements activity, but for requirements, there is no analysis or approval happening during this stage of the lifecycle.

Analysis (Design) - The requirement is being introduced as part of an analysis effort. If using an Object-Oriented Analysis method to manage and organize your requirements, then as with software, the requirements are assigned to classes along with the data and other attributes that are manipulated by the requirement.

Documentation (Implementation) - The requirement is being documented in detail according to project standards. At this stage in the requirement lifecycle the analyst is preparing the requirement for review and approval. This may be in the form of a set of documentation, with the requirement following the standards laid out in section 3.1.1. Traceability is added to the requirement to justify its existence.

Review (Testing) - The requirement is being reviewed to assess its consistency with other requirements and with the business needs.

Approval (Deployment) - The requirement is being developed into software code as part of a release.

As with software, at each stage in the requirement lifecycle quality controls can be applied to the requirement as deemed necessary.

4 Conclusion

What this paper attempts to address is some aspects of requirements quality management. What I try to demonstrate is that quality in requirements may be analogous to that of writing code during software development. As with code, requirements should be 'developed' to project standards, they should be controlled and configuration managed. Requirements also follow a lifecycle, which although not identical, has similar aspects to that of a software release.

One may read many papers on the attributes of requirements, how to write requirements and how to perform traceability, but Quality control in the requirements lifecycle is a subject about which literature may not be easy to find.

Here I try to touch upon some aspects of quality assurance in requirements management that provide food for thought. These ideas are gained from my experience

with managing (or not managing, as the case may be), requirements and the details of each topic will be different according to individual projects characteristics.

I welcome comments with knowledge gained from other's experiences. This article merely scratches the surface of this subject matter and I would be happy to discuss ideas that expand upon any of these topics.