Requirements Traceability

Neville Turbit

Overview

If a draft report was issued and subject to a number of changes, the author would automatically give it a version number. With requirements, which are also subject to change, the version is often not applied. There may have been the requirements document in January, and one in February, but what exactly has changed? Does the final requirements document represent what is built?

Traceability Definition

Requirements traceability is defined as the ability to describe and follow the life of a requirement, in both a forward and backward direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases)[i]. It can be achieved by using one or more of the following techniques:

- ***Cross referencing.*** This involves embedding phrases like "see section x" throughout the project documentation (e.g., tagging, numbering, or indexing of requirements, and specialized tables or matrices that track the cross references).
- ***Specialized templates and integration or transformation documents.*** These are used to store links between documents created in different phases of development.
- ***Restructuring.*** The documentation is restructured in terms of an underlying network or graph to keep track of requirements changes (e.g., assumption-based truth maintenance networks, chaining mechanisms, constraint networks, and propagation).

An Example of Traceability Problems

One of our clients had a project that dragged on for a few years. It was stopped and started a few times. After two years, much had been built but requirements were patchy. Some had been changed before development began, some had been added or deleted, and some had changed during development. The system was based on some complex legislation which had gone through a change during that period.

When it was decided to finish the development, nobody knew the complex business rules embedded in the software. Nobody knew if it actually met the requirements of the current legislation. Nobody knew if business rules were applied consistently to all situations. For example, if benefits were only available to people in a certain age range, and the age range legislation had changed during the last year, did the application ensure everyone must be in the new range? The problem was exacerbated by the fact that nobody knew where the rules were applied in the code. The only way to find out was to review the relevant parts of the code line by line.

In the end, a team of developers examined every line of code, and sought out business rules. The code was rewritten in many places to remove hard coding of rules. The cost was substantial and there was a delay to the project.

Reasons for Traceability

There are a number of reasons for making requirements traceable

Evolving Requirements

If requirements change, it is important to identify when and why they changed. Will someone remember in 12 months time why we decided to raise the maximum credit days from 90 to 120? Will they have it changed back to 90 days only to find payment cannot be made on a contract that was set at 120 days?

Testing

Unless there is traceability, how will the testers know what to test? The test plan should be a mirror of the requirements. If the requirements say 120 days credit then the testing needs to test 120 days. As the test plan is developed during the building of the application, the testers need to know when things change. They will need to adjust their test plan. If there is a late change to 180 days, unless it can be identified, the testers may well miss it.

System Documentation

Every developer has had to carry out maintenance on a system that had documentation that was out of date. If maintenance needs to be carried out on the credit days section, and the tester is looking to find out how many days, then out of date requirements can lead to an error. The requirements need to become system documentation and be kept up to date. Changes need to be tracked.

I had to sort out a pension payment system in a major international bank. They had a parameter driven system to calculate pension payments based on complex actuarial tables. There was a development environment, a test environment and a production environment. Each had their own sets of reference tables, and each set was different. The first thought was to go back to the system documentation but this had never been kept up to date. We had no idea which actuarial data was current. In the end, we had to have the actuaries re-create all the data and then load it into all three environments. Probably for reasons associated with self preservation, management were not keen to know how many pensions had been paid incorrectly.

Setting up Requirements Traceability

There are many tools available to manage traceability. The Rational Suite of products is a good example. For less complex systems, some basic documentation will suffice. For example, in Word, the use of hidden text in a document is useful. You use Format, Font, Hidden. You can convert the original information to hidden, and add a note. Be sure to enter the date of the change.

Another technique if Excel is being used is to hide the old row or column and add a note to the cell with reasons for the change. If Access is used, a relatively simple database can be developed that holds prior versions of requirements.

Requirements Traceability Matrix

A simple matrix might look like this:

ID	User Requirements	System Reference
UF1	Add new customers	S1, S2
UBR4	Cannot add a user if they already exist	S1, S55
UD5	User surname is mandatory	S1
Etc		

In this example, there are a number of IDs

- UF is "User Functionality"
- UBR is User "Business Rule"
- UD is "User Data"

The actual IDs will be determined by the technique used to document the requirements. For example you may refer to Use Cases by number.

The system reference will be determined by how the system is structured. If the requirement is used in more than one place, it should be noted. In this way, the testers know which parts of the system to test for particular transactions, and if a change is made, it needs to be made in both places – e.g. ability to add a user if they already exist.

Backward Compatibility

We can take the requirements and look forward but we also want to look backward from the system, and determine what rules impact each part of the system. A backward looking matrix might look like this.

System Reference	Functional Requirement	ID
S1	The system shall enable customers to be added	UF1, UBR4, UD5

In this way, a developer can view the rules and requirements around each unit of code. The tester also knows which functions to test when focusing on a particular code fix.

Summary

Traceability is important for a number of reasons. These include:

- Knowing what is a current requirement
- Knowing why a requirement was changed
- Documentation to form the basis of testing

- Understanding where requirements have been built into the system
- Forming the basis for ongoing system documentation

Whilst it is ideal to have a tool to manage requirements, not having a tool is not a reason for not tracing requirements. A simple manual system is easily created to make sure you know what is being specified.

The Author

Neville Turbit has had over 15 years experience as an IT consultant and almost an equal time working in Business. He is the principal of Project Perfect. Project Perfect is a project management software consulting and training organisation based in Sydney Australia. Their focus is to provide creative yet pragmatic solutions to Project Management issues.

Project Perfect sell "Project Administrator" software, which is a tool to assist organisations better manage project risks, issues, budgets, scope, documentation planning and scheduling. They also created a technique for gathering requirements called "Method H"™, and sell software to support the technique. For more information on Project tools or Project Management visit www.projectperfect.com.au

References

[i] Gotel, Orlena. *Contribution Structures for Requirements Traceability.* London, England: Imperial College, Department of Computing, 1995.