

With the *credit crunch* affecting IT budgets, **Experimentus**, a leading UK software quality management solutions consultancy, recently highlighted ten ways to reduce software development lifecycle costs within IT departments. Today we look in more depth at the first of these tips, “Focus on well documented Functional Requirements”.

Every software product, no matter what its type or application, has **Requirements**. A Requirement is literally “that which is required; a thing demanded or obligatory”<sup>1</sup>. Every software product created has a purpose – that which it is constructed to achieve – and thus will contain at least one Requirement. After all, if there is no purpose to a product, then it is not constructed – for to do so expends time, money and energy for no return.

Therefore, every software product has an *aim*. From this, it could easily be assumed that success of the product depends entirely upon the product achieving its preconceived aim. However, the real world is a lot more complex.

Indeed, when it comes to measuring the success of a software product, there are various ways of doing so – some of which do not directly relate to satisfaction of the initial aim (or Requirement) of the software. This could be for reasons such as the initial aim changing during the course of the product being created – or even cases where the initial aim has not been met, but enough work which is considered to be *of value* has been achieved that the product can still be released (this can often be the case in incremental, evolutionary-style software projects).

Despite these reasons, most software projects will be viewed as successful only if they achieve what they originally set out to do – if they *achieve their Requirement(s)*.

It should follow that, in order to construct a successful software product, the development team has merely to adhere to the defined Requirements. After all, if the software follows the Requirements with a high degree of accuracy, surely success must result. Once again, the real world is significantly more complex than this.

(**Note:** For reasons of simplicity and brevity, this paper specifically excludes “Agile” development methods, where the Requirements management process is significantly different to that described here.)

One example would be where problems occur in misunderstanding the Requirements.

Depending upon the roles used within the software project, the person writing down the Requirements may or may not be the eventual user of the product. The development team *might* develop the product directly from the written Requirements, or – more commonly – an intermediate step of translating the Requirements into a technical specification may be taken.

---

<sup>1</sup> Definition from Dictionary.Com: <http://dictionary.reference.com/browse/requirement>

The technical specification step itself might be split into two parts – the “Functional Specification” (and, sometimes, the “Non-Functional Specification” to go with it), and the “Technical Specification”. This division (between the Functional Specification and the Technical Specification) can occur when it is viewed that there is a significant translation to be made between the “Real English” of the original Requirements, and the “Technical English” of the Technical Specification (which is unambiguous and which developers can work with more easily) – and so the extra translation step of the Functional Specification, between the Requirements and the Technical Specification, is inserted.

This process can, of course, lead to a game of ‘Chinese Whispers’, especially if different members of the software project are writing each of these documents. The author of the Functional Specification may or may not fully understand the Requirements that he is using as a guide for his document. This should not be a problem if the author *knowingly* does not understand the Requirements – as in this situation, clarification can be asked for and hopefully received from the relevant people. However, if the author believes that they understand the Requirements but in fact have misunderstood – even in a subtle way – then errors can creep into this documentation trail. Obviously, these errors can then be reproduced (and potentially magnified) at the next documentation step (the Technical Specification), and, if not caught, work their way into the product itself.

In an ideal world, the document review cycle should eliminate such errata from being propagated. However, in the real world, it can be the case that the relevant people do not have the time and/or ability to successfully review all documentation sent in their direction. (In these days of “email overload”, receiving several documents in one’s inbox to review whilst attempting to deal with other work will frequently lead to the reviews being passed over for other, higher-priority work.)

Therefore it is certainly true that errors arising on the path from Requirements to Product can survive, grow, and indeed multiply. This can lead to the development team producing software which does not do what the user (or whoever constructed the original Requirements) actually wanted it to do. Naturally, this can often lead to recriminations and a breakdown of the relationship between teams in the project in question!

The several documentation steps detailed above mean that even if the authors of the Requirements write down exactly what they want, and the software developers create exactly what the Technical Specification tells them to, then there can still be a mismatch.

This illustrates the importance of a successfully-managed review cycle – where reviewers perform their reviews promptly, and check not only pedantic spelling and grammar mistakes (as some do) but also check that the document logically follows that which came beforehand – for example, that the Functional Specification actually complies with the requirements (some don’t), and that nothing ‘extra’ has crept into the Functional Specification which was not even mentioned in the Requirements (which can be a favourite way of Specification authors attempting to get the changes that *they* want made to the software, whether or not there’s a Requirement to make those changes).

Even assuming that the “translators” (from Requirements to Technical Specification, and any intermediate steps if they exist) perform an entirely accurate job (or that the review cycle removes all errors), and also assuming that the developers create the product entirely in line with the Technical Specification (or that the software testing cycle removes all errors), the completed software can still not perform as the users originally envisioned it would. In this

case, this can be due to a lack of training and/or ability to write high quality Requirements at the start of a software project.

So, what can happen if the completed software is found to be unsatisfactory to the level of not being fit for purpose? Either the project will be declared a failure as the project board cuts their losses (cue several recent Government IT projects), or some level of development rework will have to take place, leading to potentially major delays and cost overruns (cue several recent Government IT projects). Whichever of these scenarios occurs, it can lead to a breakdown of the relationship between the eventual software users and the development team.

It seems that there is a plethora of ways in which the software development process can fail linked to Requirements. What is required for this process to be a success?

Firstly, Requirements **must** be clearly defined. This can be achieved having the definition performed by people who are linked to the people who will be using the software when it is completed. Requirements definition processes such as UML **could** be used. Furthermore, the people defining the Requirements **must** have had relevant training (for example, IREB Requirement Engineering certifications) – so that all Requirements produced are clear, unambiguous, and represent a true picture of what the completed software needs to achieve. These finalised Requirements **must** be reviewed and agreed by all relevant stakeholders.

Secondly, Requirements **must** be accurately translated into the Technical Specifications that the development team utilise to actually create the software. This can be achieved through use of skilled Requirements Analysts and Software Designers, as well as a robust and well-used review process. The review process **must** involve all relevant stakeholders, including representatives from the development team and the test team. Additionally, all specification documents **must** provide traceability to the original Requirements.

Thirdly, the development team **must** create the software according to the Technical Specification provided. This can be achieved through use of experienced developers and, sometimes, strong management! Additionally, the code **should** be annotated (where possible) with the Requirement references themselves (which should be available within the Technical Specification). This helps to provide complete traceability.

Fourthly, the test and/or quality assurance team **must** check the produced software against the agreed Technical Specifications, and, if possible, **should also** check the produced software against the agreed original Requirements. Any issues raised should be linked to the relevant part of the Technical Specification **and** the relevant Requirement.

Following these steps outlined above will logically lead to a software product which:

- Was discussed and agreed between all relevant parties at the start of the project;

- Fulfils all of the Requirements agreed at the start of the project; and
- Can prove that it fulfils all of the Requirements agreed at the start of the project by complete traceability to those Requirements from the final product.

Such a product would not only be what users wanted, but would also have an easier-to-manage development path.

Naturally, time and financial constraints can stymie attempts to completely follow this process, leaving it perhaps as an ideal to aim for, rather than a complete solution. However, this article has hopefully helped to illuminate some of the pitfalls that may occur in the stages from the conception of a software product to its ultimate construction.

**Nathan Weller, Senior Consultant, Experimentus.**