SOFTWARE REQUIREMENTS SPECIFICATIONS: THE MISSING LINK

by

Boniface C. Nwugwo

June 8, 1990
Introduction

Have you ever found yourself first trying to identify individual functional requirements from a "Requirements Document" in order to write test cases? Do you sometimes develop the wrong functions? Do you sometimes develop the wrong User Interface? Have you ever identified requirements that you couldn't devise tests for? Do you spend a lot of time on rework? If the answer is "Yes" to any of these questions, then there is probably something wrong or missing in your requirement definition process.

We all like to think of ourselves as rational professionals. However, when you really think about it, our usual process of designing software appears irrational. Often times, programmers start without a clear statement of desired behavior and implementation constraints. Requirement decisions and interpretations are often left open to the programmers. They make a long sequence of design decisions with no clear statement of why they do things the way they do. Their rationale is rarely explained. If we are to be rational designers, and if we are to produce software in a cost-effective manner, we must begin by knowing what we must do to succeed. That information should be documented in a product of work popularly known as software requirements specification document. Completion of this document before we start would allow us to design with all the
requirements in front of us. This statement does not preclude rapid prototyping development process. What it means is that, even in a prototyping development environment, what should come out of your prototype activities is a documentation of what you learned, which should help in refining your requirement specification. This paper presents a message about the good, the bad and the ugly news of our software development process, specifically, the missing step in our Requirements definition process.

The Current State of Things

The ugly news is that we will never find a process that will allow us design software in a perfectly rational manner. Some of the reasons are:

1. In most cases, the customers (users) who commission a software system to be built do not know exactly what they want, and when they do, they are unable to correctly express their needs the first time.

2. Even if all the requirements were available, there are many other unknown facts we need to know to design the software. Many of the details only become available to us as the implementation progresses. Some of the new facts we learn will invalidate our design and we must backtrack.

3. Even if we knew all the details, all but the most trivial projects are subject to change for external reasons.

4. Because we are human beings and fallible, we must make mistakes. Human errors can only be avoided if one can avoid humans. But since this is impossible, even after the concerns are separated, errors will still be made.
For these reasons and many others, it will be quite unrealistic to expect software designers to build software in a rational, error free way from a statement of requirements. The bad news is that we are still building software in an ad hoc and abbreviated format.

Our Requirements definition process is reverse-engineering at best. It lacks an important level of abstraction. Contrary to popular belief, quality is not free. Engineering quality software requires investment - investment in the early stages of requirements and design to avoid later problems, and investment in design and code reviews to shake out errors that could plague the testing effort. You can't achieve quality unless you specify it. If we can stop all the hand waving for a minute and concentrate on defining in precise, measurable terms exactly what we mean when we say "quality", then and only then, will we be able to achieve it. Achieving quality means implementing the quality requirements, no more and no less.

Unfortunately, we're not doing that. What we fail to understand is that requirements are defined in two stages, from general to specific:

1. Requirements Expression, which is a complete and accurate description of the user needs. This is what we refer to as the "Requirements Document," and that's what most people are currently getting.

2. Requirement Specifications - which describes a system that satisfies the user's needs. Under Requirement
Specifications, requirements are defined to the first testable level and clarified. Clarification is an iteration process and no design should begin until requirement specifications have been defined and clarified. More often than not, this is the step that is missing in our requirement definition process. Not only is this step missing, but our requirements expressions tend to come very late in the development process, long after the programmers have made the decision of what they think is best for the user, and have designed what they feel is the solution to the problem. This is counter-productive. If you have to define requirements based on the work you've already done, then all you are doing is legitimizing the design, although it may not be the ideal solution for the problem. Nobody ever goes to a grocery store, buys all the groceries he needs, then goes home to determine what he wants based on the things he bought. No law says one cannot do that, but you have to pay the extra price. That is exactly what we're currently doing, paying extra for our software projects. Each time a Shipping Approval is delayed, or Customer Availability is delayed, we are paying a price. All we have to do is look back, and most likely, the things we overlooked or failed to do in the early phases of the project cause the problem.

The good news is that we are doing something and we can improve our ways. What is said above is quite obvious, known to every careful thinker, and admitted by the honest ones. It is a clear, noble goal to achieve confidence in the statement of requirements before proceeding with the design, implementation, and eventual installation of a software-based system. The quantitative benefits of containing requirements problems to the early phases of the life cycle have been evident for some time, e.g., in the relative-error cost profile provided by Boehm¹. The data show, for example, how the relative cost of an error regarding requirements geometrically escalates as compared to
what it would have been if it had been contained in the
requirements phase of the project.

If we have identified an ideal process, but cannot follow it completely, we can still follow it as closely as possible and we can write the documentation that we would have produced if we had followed the ideal process. Even if we cannot know all the facts necessary to design an ideal system, the effort to find those facts before we start to code will help us to design better and backtrack less.

Why must we have requirement specifications?

1. We want to avoid making requirements decisions accidentally while designing the program. Programmers working on a system are very often, not familiar with the application. Having a complete reference on externally visible behavior relieves them of any need to decide what is best for the user.

2. We want to avoid duplication of effort and inconsistency. Without a requirement specification, designers, programmers, and reviewers alike would ask many of the questions it answered repeatedly throughout the development. This is expensive and often results in inconsistent answers.

3. A requirement specification is a valuable insurance against the costs of personnel turnover. The knowledge that we gain about the requirements will never be lost when someone leaves a project.

4. A complete requirements specification is necessary (but not sufficient) for making good estimates of the amount of work and other resources that it will take to build a system.

5. A complete requirement specification provides a good basis for test plan development. Without it, we do not know what to test for.

6. The requirements document will be used, long after the system is operational to define the constraints for future changes.
7. It can be used to settle arguments among all the parties involved, End User, Developer, and the Buyer.

8. The Requirements document can and should be used as a reference document rather than an introductory narrative about the system to be built.

How can we improve our process?

As a company, we must continually strive to improve the quality of our products. When we talk about software product quality, we mean the attributes of the product resulting from software development process. Product quality would then include, for example, the completeness and clarity of our requirements documents, the clarity of our design documents, the traceability of our designs, the reliability of our code, the coverage of our tests, and so on.

Improvement of our process must be a collective responsibility.

(a) First, management commitment is needed in order to achieve this goal. Engineers can yell and scream about the need for a change until they turn blue, but without management involvement, it will never happen. Project managers, as well as Project leaders must take the initiative of making sure that no single code is written until requirements have been completely defined and verified. I know what you're thinking now --- what if you're in a rapid development environment? Working in a rapid development or prototyping environment does not mean that a requirements specification is not necessary. It means that you fully don't understand what it is you want, so you build prototypes to help you understand your requirements better and to prove that you can do it. The end result of your prototype activity should be a documentation of the things you learned from the exercise. These lessons learned should help you in refining your requirements. Therefore, Project managers, as well as
Project leaders must learn not to rush into premature coding, popularly known as "WISCA" (Why Isn't Sam Coding Anything) syndrome. Where information is not available before development must begin, the areas of incompleteness should be explicitly indicated.

(b) Training is very essential if we are to succeed. We need to learn that better quality requires investment, which results in less rework, which in turn reduces the total cost of development since rework costs approximately 50 percent of the total cost of a project².

(c) We must be willing to change the status quo. In other words, we must be willing to try new methods other than the ones we're used to. Software Engineering is still going through an evolutionary process, and for us to survive in the software development business, we have to change with the trend. We cannot afford the "Don't rock the boat" attitude, if we are to succeed.

Conclusion

It takes considerable effort to produce complete, testable software requirements specification, it saves labor in the long run. However, if a project is worth doing, its requirement specification is worth doing it right.

References
