Introduction

Requirements engineering is one of the most difficult aspects of software development. It is also arguably the most important aspect. If you don't get the requirements right, it doesn't matter how well you execute the rest of the project work: You will fail. Of course, this doesn't mean you need to—or even can—get *all* the requirements pinned down in exquisite detail before anyone can begin design or construction work. Before you get serious about implementing a specific portion of your product, though, you'd better understand the requirements for that portion, whether it represents 1 percent or 100 percent of the final product.

Expect requirements development to be an iterative and incremental process of exploration, discovery, and invention. It's not a simple matter of "gathering" requirements. Your initial understanding of requirements will change as development progresses. In some cases the business itself is changing rapidly. Many users have difficulty envisioning just how a new product or information system would work. Often they're trapped in the mindset of the current ways of working, be they manual processes or the use of existing applications. It's hard for most users to gain a clear vision of the new product from discussions or written specifications alone. As specifications, designs, and the product itself evolve, the stakeholders will think of more questions, ideas, desires, and pertinent information. Feedback from early versions of the product stimulate new ideas and help clarify the thinking of everyone involved. Iteration is both necessary and valuable.

Many books on software requirements engineering have been published in the last several years including my own *Software Requirements, 2nd Edition* (Wiegers 2003). These books provide solid guidance on the challenges of requirements elicitation, analysis, specification, validation, and management. However, additional requirements-related topics are not covered well by the existing books. Also, some books contain guidance that I believe is simply incorrect.

My objective with this handbook is to address some of these difficult issues in requirements engineering. I've collected input from the many questions I receive from my consulting and training clients and from readers of my books and articles. Recurrent questions indicate points that confuse many practitioners; those are the topics I've chosen to address. I've included numerous true stories from real, personal experiences. These are highlighted with a newspaper icon in the margin.

Organizations of all types are struggling with the same requirements issues. The suggestions I propose in this handbook augment the "good practices" approach I took in *Software Requirements, 2nd Edition*. As with all such advice, you need to think about how best to apply these suggestions to your specific situation. Organizations are different, projects are different, and cultures are different, so techniques that work in one situation might not be just right for another. I hope you find the topics covered in this handbook helpful in your own quest for quality requirements.

I should stress that all the practices I recommend assume that you are dealing with reasonable people. Sometimes an unreasonable customer will insist on a specific solution that isn't a good fit for the problem. Sometimes unreasonable funding sponsors impose their own inappropriate preferences, overriding the thoughtful decisions made by actual user representatives. If you face such a situation, try educating the difficult people to help them understand the risks posed by the approaches they demand and the value of a better approach. People who appear unreasonable often are just uninformed. Sometimes, though, they truly are unreasonable. I can't help you much with that.

To Duplicate or Not to Duplicate

It’s not unusual to discover that you need to replicate a specific requirement or piece of requirements-related information in multiple places in your project documentation. This poses a problem, and I’m not aware of any perfect solution to the problem.

On the one hand, it’s convenient to be able to view and print all the information associated with a specific issue, topic, function, feature, or use case in a single location. This lets you treat that block of information as a self-contained package. You can give this package to a developer to implement and that developer will have all the necessary information grouped together.

On the other hand, duplicating information is risky. Even if the same requirement or definition logically fits in multiple locations, duplicating it creates a maintenance hazard. Someday you might need to modify that replicated item. There is a very good chance that you *won’t* make the same change in every location where the information appears. This introduces an inconsistency that someone else will have to detect and resolve in the future. In addition, replicating information increases the volume of your requirements documentation; big specifications frighten some people.

I encountered exactly this situation recently. A client sent me a set of use cases to review. Four of those use cases were influenced by the same business rule. The use case template the client was using included a space for business rules. Each of these four use cases stated the business rule in its entirety in that template field. Two of the business rule statements were worded exactly the same. The other two were also identical but they differed slightly from the phrasing used in the first pair. This subtle wording difference immediately raised the question: Are these business rules really the same, or are they in fact different in a small but significant way? They looked to have the same meaning to me, but as a reader of the use-case specifications, I really need to find someone who can confirm this interpretation.

As an alternative to duplicating business rules or other bits of requirements knowledge, consider incorporating cross-references to the original source of the information rather than actually replicating the text wherever it logically belongs. Let me describe several ways to do this.

Cross-Referencing

If you’re storing the information in word processing documents, such as Microsoft Word files, exploit the bookmark and cross-reference functions. In Word, first define a bookmark for the master instance of the replicated information and give it a unique name. Anywhere you want the same information to reappear, use the Insert Cross-reference function to insert the bookmarked text wherever you like. The document looks as though exactly the same information appears in multiple places, but the cross-referenced appearances are merely copies. If you modify the master instance of the bookmarked text and then refresh the display or print the document, the change will immediately be echoed in all the other places where you inserted those cross-references.

This approach provides the advantage of having the cross-referenced text appear in all the relevant locations. This gives the reader everything he needs in a single package. It permits unlimited reuse of discrete chunks of information of any size, too. Suppose you have a block of five requirements that appear in the description of multiple use cases. You can insert cross-references to visually echo that block of requirements wherever appropriate.

Using cross-references in this fashion does require that the author or maintainer knows where the master copy of the information resides so he can modify only that occurrence. It also doesn't reduce the size of the requirements documentation, as the full text of the reused information appears in every pertinent location. Of course, modifying the master instance—the bookmarked text—won't update any hardcopies that already contain an echo of the old version!

Hyperlinks

An alternative strategy is to employ hyperlinks in word processing documents, PDF, HTML, or other types of files. In this scheme, you store every unique piece of information in a logically sensible location, typically grouped with other information of the same type. You might collect all business rules in a business rules catalog, all data definitions in a data dictionary, and definitions of key terms in a glossary. Then, anywhere you need to refer to one of these items in another document, you insert a hyperlink to the place where the information resides.

In the use case example I described earlier, the requirements analyst could have entered just the identifying label and name of the common business rule in the business rules field for all four use cases. It isn't necessary—or desirable—to include the full statement of the rule itself in each place. The analyst could make each occurrence of the business rule name or label into a hyperlink to the appropriate entry in the business rules catalog. This gives the reader easy access to that information. Any time the rule changes, only one location needs to be modified. The hyperlinks will always point to the current and correct instance of that rule.

Using hyperlinks like this increases the maintainability of the requirements data and increases information accuracy. It also reduces the volume of requirements information because each unique piece of information is stored only once. Collecting information of the same type together in one location, as in a business rules catalog, makes it easier to review the information in context. This helps you to see relationships between the items, duplicates (or worse, near-duplicates), conflicts, errors, and gaps.

The big downside to this scheme is that the reader has to follow the links to assemble a full picture of the use case or other package of requirements information. This can be tedious if, say, a use case points to multiple business rules or other external pieces of data. Hyperlinks work well for online navigation but aren't so convenient if you want to print out a portion of the requirements documentation.

Traceability Links

A third option is to store your requirements information in a database, such as that associated with a commercial requirements management tool. There are many such tools available. You can find descriptions and product feature comparisons of many of the commercial RM tools at http://www.paper-review.com/tools/rms/read.php and http://www.volere.co.uk/tools.htm. Requirements management tools allow you to create and populate various requirement classes or types. You can define distinct sets of attributes or metadata for each requirement type. Every object stored in the database receives a unique identifier. You can establish logical connections, called traceability links or traces, between two objects of the same type or of different types.

Again returning to our use case example, you might create one requirement type of "use case" and another of "business rule" in the requirements management tool. You store all the pertinent business rule statements and use case descriptions in the database. You then create a traceability link between each use case and each business rule that affects that use case, as illustrated in Figure 2. This traceability approach facilitates reusing requirement objects not only within a single project but even across multiple projects. The tool user can traverse the traceability links to access all of the information that is logically connected to some entity, such as a specific use case.

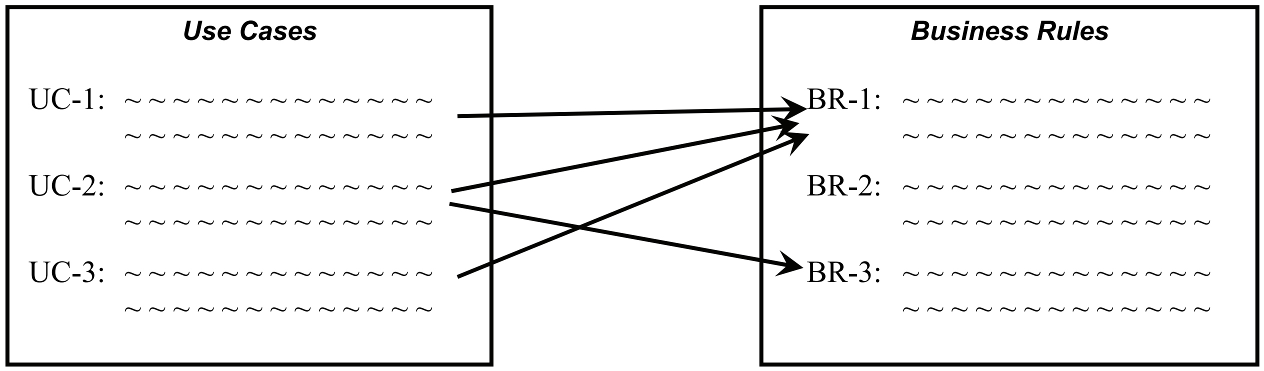

Figure 2: Traceability links connect use cases and business rules.

Because each discrete piece of requirement information is only stored in a single location, only that one instance needs to be modified when a change is necessary. The more capable requirements management tools will provide a visual indicator whenever an object on either end of a traceability link changes. This alerts the user to check the object on the other end of the link to see if an update is needed there. For example, if business rule BR-3 is changed, then any use case that links to BR-3 might also have to be modified. You might also need to update any functional requirements or test cases that trace back to that same use case to keep them consistent with the altered business rule. This multilink tracing is virtually impossible to do by hand; you really need a tool.

Unfortunately, storing the requirements information in a database still makes it hard to pull together all the items that belong to a logical package of requirements information. The ideal RM tool would follow the traceability links and insert the information from the other end of each link into reports generated from the database contents.

Recommendation

There is no perfect solution to the problem of dealing with requirements information that logically fits in more than one place. On balance, I believe the inconvenience of following hyperlinks or traceability links to access connected information beats the maintenance and communication problems caused by duplicating information. Therefore, I recommend that you store each piece of unique requirements information just once. Use one of the mechanisms described in this chapter to point to that master source whenever you need to. As with all of my recommendations, you need to look at each specific situation and do what makes the most sense for that context. There may be times in which it is more convenient or clearer to actually include redundant information. This is analogous to selectively denormalizing a relational database for some legitimate reason. However, you do need to recognize the risk of having one instance of that information get out of sync with its fellows.