Software requirements

Wednesday, January 15, 2020 7:04 PM

I'm reading given below books only because I want to learn about **use case modeling** and **software requirements**.

Modeling tools

Requirements modeling tools help the BA create diagrams like those described in Chapter 5 Chapter 12 and Chapter 13. These tools support the use of standard shapes, notations, and syntax for drawing diagrams according to established conventions. They might provide templates as starting points and examples to help the BA learn more about each model. Often these tools automatically connect shapes in diagrams to accelerate the drawing process and to help ensure that the diagrams are drawn correctly. **They also enable you to create diagrams that look cleaner and more consistent than if you draw them manually**. Specialized software modeling tools facilitate iteration by dragging along connected arrows and labels whenever you move a symbol in the diagram; general-purpose drawing tools might not provide that capability.

Many requirements management tools also provide some modeling capability. The most sophisticated tools allow you to trace individual requirements to models or even to specific elements of models. For example, analysts can create swimlane diagrams in the tool, and then after they write requirements, they can trace those requirements back to specific steps in the diagrams.
Keep in mind that no tool will be able to tell you if a requirement or a model element is missing, logically incorrect, or unnecessary. These tools enable BAs to represent information in multiple ways and to spot certain types of errors and omissions, but they don't eliminate the need for thinking and peer review.

People are often resistant to change things that they're familiar with, and they usually have a comfort level with working on requirements in documents. They might have a perception—even if incorrect—that using a requirements tool will be harder for them. Also, don't forget that most of the tool users are already busy. Time must be allocated to let them get used to using the tool in their daily jobs. Eventually, the tool probably won't actually require more time from users, but they first need to get over the learning curve and develop new work habits using the tool. Following are some suggestions to help you deal with issues regarding user adoption and culture change:

- Identify a tool advocate, a local enthusiast who learns the tool's ins and outs, mentors other users, and sees that it gets employed as intended. This person should be an experienced business analyst who can be the single owner for ensuring tool adoption. This initial tool advocate will work with other users on their projects to ingrain the tool into their daily activities. Then he'll train and mentor others to support the tool as other projects adopt it.
- One of the biggest adoption challenges to overcome is that users don't believe the tool will actually add any value. Perhaps they haven't recognized the pain from limitations of their existing manual approaches. Share stories with them about where the lack of a tool caused a negative impact and ask them to think of their own examples.
- Your team members are smart, but it's better to train them than to expect them to figure out how best to use the tool on their own. They can undoubtedly deduce the basic operations, but they won't learn about the full set of tool capabilities and how to exploit them efficiently.
- Because you can't expect instantaneous results, don't base a project's success on a tool you're using for the first time. Begin with a pilot application of the tool on a noncritical project. This will help the organization learn how much effort it takes to administer and support the tool. Chapter 31 describes the learning curve associated with adopting new tools and techniques.

The proliferation and increased usage of tools to assist with requirements development and management represents a significant trend in software engineering that will undoubtedly continue. Too many organizations, though, fail to reap the benefits of their investment in such tools. They do not adequately consider their organization's culture and processes and the effort needed to shift from a document-based requirements paradigm to a tool-based approach. The guidance in this chapter will help you choose appropriate tools and use them effectively. Just remember, a tool cannot replace a solid requirements process or team members with suitable skills and knowledge. A fool with a tool is an amplified fool.

Next steps

- Analyze shortcomings in your current requirements process to see whether a requirements development or requirements management tool is likely to provide sufficient value to justify the investment. Make sure you understand the causes of your current shortcomings; don't simply assume that a tool will magically correct them.
- Before launching a comparative evaluation, assess your organization's readiness for adopting a tool. Reflect on previous attempts to incorporate new tools into your development process. Understand why they succeeded or failed so that you can position yourselves for success this time.

Suppose a tester encounters unexpected functionality with no corresponding written requirement. This code could indicate that a developer implemented a legitimate implied or verbally communicated requirement that the business analyst can now add to the requirements set. Alternatively, it might be "orphan code," an instance of gold-plating that doesn't belong in the product. Trace links can help you sort out these kinds of situations and build a more complete picture of how the pieces of your system fit together. Conversely, tests that are derived from—and traced back to—individual requirements provide a mechanism for detecting unimplemented requirements, because the expected functionality will be missing from the system being tested. Trace links also help you keep track of parentage, interconnections, and dependencies among individual requirements. This information reveals the propagation of change that can result when a particular requirement is deleted or modified.

Figure 29-2 illustrates many kinds of traceability relationships that can be defined on a project. Of course, you don't need to define and manage all these trace link types. On many projects, you can gain most of the traceability benefits you want for just a fraction of the potential effort. Maybe you only need to trace system tests back to functional requirements

or user requirements. Perform a cost-benefit analysis to decide which links will contribute to the success of your project, both in terms of development and long-term maintenance effort. Don't ask team members to spend time recording information unless you know how they can use it.

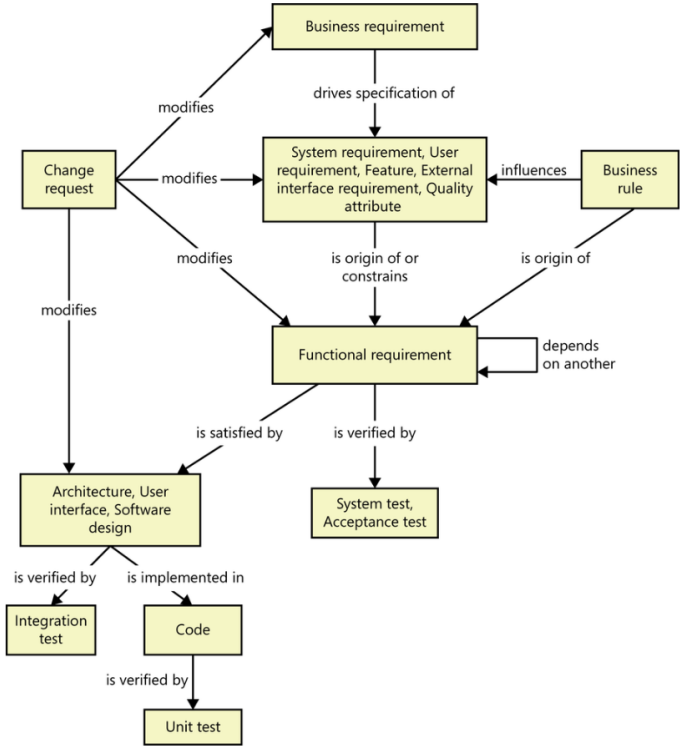

Figure 29-2. Some possible requirements trace links.

Motivations for tracing requirements

I've had the embarrassing experience of writing a program and then realizing that I had inadvertently overlooked a requirement. It was in the SRS—I simply missed it. I had to go back and write additional code after I thought I was done programming. Overlooking a requirement is more than an embarrassment if it means a customer isn't satisfied or a product is missing a critical function. Requirements tracing provides a way to demonstrate compliance with a specification, contract, or regulation. At an organization level, implementing requirements tracing can improve the quality of your products, reduce maintenance costs, and facilitate reuse.

Keeping the link information current as the system undergoes development and maintenance takes discipline and time. If the trace information becomes obsolete, you'll probably never reconstruct it. Obsolete or inaccurate trace data wastes time by sending developers and maintainers down the wrong path, destroying any trust the developers might have had in the information. Because of these realities, you should adopt requirements tracing for the right reasons ([ref196]). Following are some potential benefits of implementing requirements tracing:

- **Finding missing requirements**. Look for business requirements that don't trace to any user requirements, and user requirements that don't trace to any functional requirements.
- **Finding unnecessary requirements**. Look for any functional requirements that don't trace back to user or business requirements and therefore might not be needed.
- **Certification and compliance**. You can use trace information when certifying a safety-critical product, to demonstrate that all requirements were implemented—although that doesn't confirm that they were implemented correctly! Trace information demonstrates that requirements demanded for regulatory compliance have been included and addressed, as is often needed for applications for health care and financial services companies.
- **Change impact analysis**. Without trace information, there's a good chance that you'll overlook a system element that would be affected if you add, delete, or modify a particular requirement.
- **Maintenance**. Reliable trace information facilitates your ability to make changes correctly and completely during maintenance. When corporate policies or government regulations change, software systems often must be updated. A table that shows where each applicable business rule was addressed in the functional requirements, designs, and code makes it easier to make the necessary changes properly.

- **Project tracking**. If you record the trace data during development, you'll have an accurate record of the implementation status of planned functionality. Absent links indicate work products that have not yet been created.
- **Reengineering**. You can list the functions in an existing system you're replacing and trace them to where they are addressed in the new system's requirements and software components.
- **Reuse**. Trace information facilitates the reuse of product components by identifying packages of related requirements, designs, code, and tests.
- **Testing**. When a test fails, the links between tests, requirements, and code point developers toward likely areas to examine for the defect.

Many of these are long-term benefits, reducing overall product life-cycle costs but increasing the development cost by the effort expended to accumulate and manage the trace information. View requirements tracing as an investment that increases your chances of delivering a maintainable product that satisfies all the stated customer requirements. This investment will pay dividends anytime you have to modify, extend, or replace the product. Establishing traces is not much work if you collect the information as development proceeds, but it's tedious and expensive to do on a completed system.

> **Important**
>
> Listing the test cases for each requirement does *not* indicate that the software has passed those tests. It simply indicates that certain tests have been written to verify the requirement at the appropriate time. Tracking testing status is a separate matter.
>
> > **Trap**
> >
> > Gathering and managing requirements trace data must be made the explicit responsibility of certain individuals or it won't happen. Typically, a business analyst or a quality assurance engineer collects, stores, and reports on the trace information.

Next steps

- Set up a trace matrix for 15 or 20 requirements from an important portion of the system you're currently developing. Try the approaches shown in both Tables 29-1 and 29-2. Populate the matrix as the project progresses for a few weeks. Evaluate which method seems most effective and what procedures for collecting and storing traceability information will work for your team.
- The next time you perform maintenance on a poorly documented system, record what you learn from reverse engineering the part of the product you're modifying. Build a fragment of a requirements traceability matrix for the piece of the puzzle you're manipulating so that the next time someone has to work on it they have a head start. Grow the matrix as your team continues to maintain the product.
- Trace your functional requirements back to user requirements, and trace your user requirements to business requirements. Count the requirements that you could cut because they don't link back to a business requirement. Count the requirements that were missing until the trace matrix revealed their absence. Estimate the costs had you not discovered these requirements errors until much later in the project. This analysis will help you judge whether requirements tracing will pay off in your environment.