Team LiB ◂ PREVIOUS NEXT ▸

Part III: Software Requirements Management

Chapter List

Team LiB ◂ PREVIOUS NEXT ▸
Team LiB ◂ PREVIOUS NEXT ▸

Chapter 18: Requirements Management Principles and Practices

Overview

Chapter 1, "The Essential Software Requirement," divided the discipline of requirements engineering into requirements development and requirements management. Requirements development involves eliciting, analyzing, specifying, and validating a software project's requirements. Typical requirements development deliverables include a vision and scope document, use-case documents, a software requirements specification, a data dictionary, and associated analysis models. Once reviewed and approved, these artifacts define the requirements baseline for the development effort, an agreement between the development group and its customers. The project likely will have additional agreements regarding deliverables, constraints, schedules, budgets, or contractual commitments; those topics lie beyond the scope of this book.

The requirements agreement is the bridge between requirements development and requirements management. Team members must have ready access to the current requirements throughout the project's duration, perhaps through a Web-based gateway to the contents of a requirements management tool. Requirements management includes all activities that maintain the integrity, accuracy, and currency of the requirements agreement as the project progresses. As Figure 18-1 shows, requirements management involves

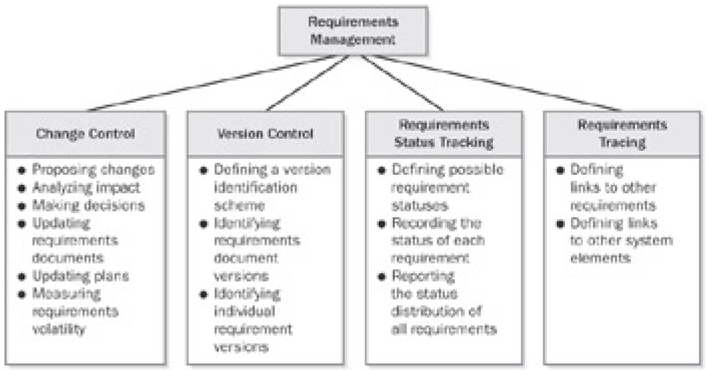

Requirements Management
Change Control
• Proposing changes
• Analyzing impact
• Making decisions
• Updating requirements documents
• Updating plans
• Measuring requirements volatility
Version Control
• Defining a version identification scheme
• Identifying requirements document versions
• Identifying individual requirement versions
Requirements Status Tracking
• Defining possible requirement statuses
• Recording the status of each requirement
• Reporting the status distribution of all requirements
Requirements Tracing
• Defining links to other requirements
• Defining links to other system elements

Figure 18-1: Major requirements management activities.

- Controlling changes to the requirements baseline
- Keeping project plans current with the requirements
- Controlling versions of both individual requirements and requirements documents
- Tracking the status of the requirements in the baseline
- Managing the logical links between individual requirements and other project work products

A development team that accepts newly proposed requirements changes might not be able to fulfill its existing schedule and quality commitments. The project manager must negotiate changes to those commitments with affected managers, customers, and other stakeholders. The project can respond to new or changed requirements in various ways:

- Defer lower-priority requirements
- Obtain additional staff
- Mandate overtime work, preferably with pay, for a short time
- Slip the schedule to accommodate the new functionality
- Let quality suffer in the press to ship by the original date (Too often, this is the default reaction.)

No one approach is universally correct because projects differ in their flexibility of features, staff, budget, schedule, and quality (Wiegers 1996a). Base your choice on the priorities that the key stakeholders established during project planning. No matter how you respond to changing requirements, accept the reality of adjusting expectations and commitments when necessary. This is better than imagining that somehow all the new features will be incorporated by the original delivery date without consequences such as budget overruns or team member burnout.

This chapter addresses basic principles of requirements management. The other chapters in Part III describe certain requirements management practices in more detail, including change control (Chapter 19, "Change Happens"), change-impact analysis (also Chapter 19), and requirements tracing (Chapter 20, "Links in the Requirements Chain"). Part III concludes with a discussion of commercial tools that can help a project team manage its requirements (Chapter 21, aptly titled "Tools for Requirements Management").

The Requirements Baseline

The requirements *baseline* is the set of functional and nonfunctional requirements that the development team has committed to implement in a specific release. Defining a baseline gives the project stakeholders a shared understanding of the capabilities and properties they can expect to see in the release. At the time the requirements are baselined—typically following formal review and approval—they are placed under configuration management. Subsequent changes can be made only through the project's defined change-control process. Prior to baselining, the requirements are still evolving, so there's no point in imposing unnecessary process overhead on those modifications. However, begin practicing version control—uniquely identifying different versions of each requirements document—as soon as you create a preliminary draft of a document.

From a practical perspective, the requirements in the baseline must be distinguished from others that have been proposed but not accepted. The baselined SRS document should contain only those requirements that are planned for a specific release. It should be clearly identified as a baseline version to distinguish it from a series of prior draft versions that had not yet been approved. Storing requirements in a requirements management tool facilitates identifying the requirements that belong to a specific baseline and managing changes to that baseline. This chapter describes some ways to manage the requirements in a baseline.

Team LiB ◀ PREVIOUS NEXT ▶
Team LiB ◀ PREVIOUS NEXT ▶

Requirements Management Procedures

Your organization should define the activities that project teams are expected to perform to manage their requirements. Documenting these activities and training practitioners in their effective application enables the members of the organization to perform them consistently and effectively. Consider addressing the following topics:

- Tools, techniques, and conventions for controlling versions of the various requirements documents and of individual requirements
- How the requirements are baselined
- The requirement statuses that you will use and who may change them
- Requirement status-tracking procedures
- The ways that new requirements and changes to existing ones are proposed, processed, negotiated, and communicated to all affected stakeholders
- How to analyze the impact of a proposed change
- How the project's plans and commitments will reflect requirements changes

You can include all this information in a single requirements management process description. Alternatively, you might prefer to write separate change-control, impact-analysis, and status-tracking procedures. These procedures should apply across your organization because they represent common functions that should be performed by every project team.

More Info Chapter 22, "Improving Your Requirements Processes," describes several useful process assets for requirements management.

Someone must own the requirements management activities, so your process descriptions should also identify the team role that's responsible for performing each task. The project's requirements analyst typically has the lead responsibility for requirements management. The analyst will set up the requirements storage mechanisms (such as a requirements management tool), define requirement attributes, coordinate requirement status and traceability data updates, and generate reports of change activity.

Trap If no one on the project has responsibility for performing requirements management activities, don't expect them to get done.

Team LiB ◂ PREVIOUS NEXT ▸
Team LiB ◂ PREVIOUS NEXT ▸

Requirements Version Control

"I finally finished implementing the multivendor catalog query feature," Shari reported at the Chemical Tracking System's weekly project status meeting. "Man, that was a lot of work!"

"Oh, the customers canceled that feature two weeks ago," the project manager, Dave, replied. "Didn't you get the revised SRS?"

Shari was confused. "What do you mean, it was canceled? Those requirements are at the top of page 6 of my latest SRS."

Dave said, "Hmmm, they're not in my copy. I've got version 1.5 of the SRS. What version are you looking at?"

"Mine says version 1.5 also," said Shari disgustedly. "These documents should be identical, but obviously they're not. So, is this feature still in the baseline or did I just waste 40 hours of my life?"

If you've ever heard a conversation like this one, you know how frustrating it is when people waste time working from obsolete or inconsistent requirements. Version control is an essential aspect of managing requirements specifications and other project documents. Every version of the requirements documents must be uniquely identified. Every team member must be able to access the current version of the requirements, and changes must be clearly documented and communicated to everyone affected. To minimize confusion, conflicts, and miscommunication, permit only designated individuals to update the requirements and make sure that the version identifier changes whenever a requirement changes.

It's Not a Bug; It's a Feature!

True Stories A contract development team received a flood of defect reports from the people testing the latest release of a system they had just delivered to a client. The contract team was perplexed—the system had passed all their own tests. After considerable investigation, it turned out that the client had been testing the new software against an obsolete version of the software requirements specification. What the testers were reporting as bugs truly were features. (Normally, this is just a little joke that software people like to make.) Much of the testing effort was wasted because the testers were looking for the wrong system behaviors. The testers spent considerable time rewriting the tests against the correct version of the SRS and retesting the software, all because of a version control problem.

Each circulated version of the requirements documents should include a revision history that identifies the changes made, the date of each change, the individual who made the change, and the reason for each change. Consider appending a version number to each individual requirement label, which you can increment whenever the requirement is modified, such as Print.ConfirmCopies-1.

The simplest version control mechanism is to manually label each revision of the SRS according to a standard convention. Schemes that try to differentiate document versions based on revision date or the date the document was printed are prone to error and confusion; I don't recommend them. Several of my teams have successfully used a manual convention that labels the first version of any new document "Version 1.0 draft 1." The next draft is "Version 1.0 draft 2." The author increments the draft number with each iteration until the document is approved and baselined. At that time, the label is changed to "Version 1.0 approved." The next version is designated either "Version 1.1 draft 1" for a minor revision or "Version 2.0 draft 1" for a major change. (Of course, "major" and "minor" are subjective or, at least, context-driven). This scheme clearly distinguishes between draft and baselined document versions, but it does require manual discipline.

If you're storing requirements in a word-processing document, you can track changes by using the word processor's revision marks feature. This feature visually highlights changes made in the text with notations such as strikethrough highlighting for deletions, underscores for additions, and vertical revision bars in the margin to show the location of each change. Because these notations visually clutter the document, the word processor lets you view and print either the marked-up document or its final image with the changes applied. When you baseline a document marked up in this way, first archive a marked-up version, then accept all the revisions, and then store the now-clean version as the new baseline file, ready for the next round of changes.

True Stories A more sophisticated technique stores the requirements documents in a version control tool, such as those used for controlling source code through check-in and check-out procedures. Many commercial configuration management tools are available for this purpose. I know of one project that stored several hundred use-case documents written in Microsoft Word in such a version control tool. The tool let the team members access all previous versions of every use case, and it logged the history of changes made to each one. The project's requirements analyst and her back-up person had read-write access to the documents stored in the tool; the other team members had read-only access.

The most robust approach to version control is to store the requirements in the database of a commercial requirements management tool, as described in Chapter 21. These tools track the complete history of changes made to every requirement, which is valuable when you need to revert to an earlier version of a requirement. Such a tool allows for comments describing the rationale behind a decision to add, modify, or delete a requirement; these comments are helpful if the requirement becomes a topic for discussion again in the future.

Team LiB ◀ PREVIOUS NEXT ▶
Team LiB ◀ PREVIOUS NEXT ▶

Requirement Attributes

Think of each requirement as an object with properties that distinguish it from other requirements. In addition to its textual description, each functional requirement should have several supporting pieces of information or *attributes* associated with it. These attributes establish a context and background for each requirement that goes well beyond the description of intended functionality. You can store attribute values in a spreadsheet, a database, or, most effectively, a requirements management tool. Such tools provide several system-generated attributes in addition to letting you define others. The tools let you filter, sort, and query the database to view selected subsets of requirements based on their attribute values. For instance, you could list all high-priority requirements that were assigned to Shari for implementation in release 2.3 and that have a status of Approved.

A rich set of attributes is especially important on large, complex projects. Consider specifying attributes such as the following for each requirement:

- Date the requirement was created
- Its current version number
- Author who wrote the requirement
- Person who is responsible for ensuring that the requirement is satisfied
- Owner of the requirement or a list of stakeholders (to make decisions about proposed changes)
- Requirement status
- Origin or source of the requirement
- The rationale behind the requirement
- Subsystem (or subsystems) to which the requirement is allocated
- Product release number to which the requirement is allocated
- Verification method to be used or acceptance test criteria
- Implementation priority
- Stability (an indicator of how likely it is that the requirement will change in the future; unstable requirements might reflect ill-defined or volatile business processes or business rules)

Wherefore This Requirement?

True Stories The product manager at a company that makes electronic measurement devices wanted to keep a record of which requirements were included simply because a competitor's product had the same capability. A good way to make note of such features is with a Rationale attribute, which indicates why a specific requirement is included in the product.

Select the smallest set of attributes that will help you manage your project effectively. For example, you might not need both the name of the person responsible for the requirement and the subsystem to which it is allocated. If any of this attribute information is stored elsewhere—for example, in an overall development-tracking system—don't duplicate it in the requirements database.

Trap Selecting too many requirements attributes can overwhelm a team such that they never supply all attribute values for all requirements and don't use the attribute information effectively. Start with perhaps three or four key attributes. Add others when you know how they will add value to your project.

Requirements baselines are dynamic. The set of requirements planned for a specific release will change as new requirements are added and existing ones are deleted or deferred to a later release. The team might be juggling separate requirements documents for the current project and for future releases. Managing these moving baselines with a document-based requirements specification is tedious. If deferred or rejected requirements remain in the SRS, readers can be confused about whether those requirements are included in that baseline. However, you don't want to spend a lot of time dragging requirements from one SRS to another. One way to handle this situation is to store the requirements in a requirements management tool and define a Release Number attribute. Deferring a requirement means changing its planned release, so simply updating the release number shifts the requirement into a different baseline. Rejected requirements are best handled using a requirement status attribute, as described in the next section.

True Stories Defining and updating these attribute values is part of the cost of requirements management, but that investment can yield a significant payback. One company periodically generated a requirements report that showed which of the 750 requirements from three related specifications were assigned to each designer. One designer discovered several requirements that she didn't realize were her responsibility. She estimated that she saved one to two months of engineering design rework that would have been required had she not found out about those requirements until later in the project.

Team LiB ◀ PREVIOUS NEXT ▶
Team LiB ◀ PREVIOUS NEXT ▶

Tracking Requirements Status

"*How are you coming on that subsystem, Jackie?" asked Dave.*

"Pretty good, Dave. I'm about 90 percent done."

Dave was puzzled. "Weren't you 90 percent done a couple of weeks ago?" he asked.

Jackie replied, "I thought I was, but now I'm really 90 percent done."

Software developers are sometimes overly optimistic when they report how much of a task is complete. They often give themselves credit for activities they've started but haven't entirely finished. They might be 90 percent done with the originally identified work, only to encounter additional unanticipated tasks. This tendency to overestimate progress leads to the common situation of software projects or major tasks being reported as 90 percent done for a long time. Tracking the status of each functional requirement throughout development provides a more accurate gauge of project progress. Track status against the expectation of what "complete" means for this product iteration. For example, you might have planned to implement only a part of a use case in the next release, leaving final implementation for a future iteration. In this case, just monitor the status of those functional requirements that were committed for the upcoming release, because that's the set that's supposed to be 100 percent done before shipping the release.

Trap There's an old joke that the first half of a software project consumes the first 90 percent of the resources and the second half consumes the other 90 percent of the resources. Overoptimistic estimation and overgenerous status tracking constitute a reliable formula for project overruns.

Table 18-1 lists several possible requirement statuses. (For an alternative scheme, see Caputo 1998.) Some practitioners add the statuses Designed (that is, the design elements that address the functional requirement have been created and reviewed) and Delivered (the software containing the requirement is in the hands of the users, as for a beta test). It's valuable to keep a record of rejected requirements and the reasons they were rejected, because dismissed requirements have a way of resurfacing during development. The Rejected status lets you keep a proposed requirement available for possible future reference without cluttering up a specific release's set of committed requirements.

Classifying requirements into several status categories is more meaningful than trying to monitor the percent completion of each requirement or of the complete baseline. Define who is permitted to change a status, and update a requirement's status only when specified transition conditions are satisfied. Certain status changes also lead to updating the requirements traceability data to indicate which design, code, and test elements addressed the requirement, as shown in Table 18-1.

Table 18-1: Suggested Requirement Statuses

Status	**Definition**
Proposed	The requirement has been requested by an authorized source.
Approved	The requirement has been analyzed, its impact on the project has been estimated, and it has been allocated to the baseline for a specific release. The key stakeholders have agreed to incorporate the requirement, and the software development group has committed to implement it.
Implemented	The code that implements the requirement has been designed, written, and unit tested. The requirement has been traced to the pertinent design and code elements.
Verified	The correct functioning of the implemented requirement has been confirmed in the integrated product. The requirement has been traced to pertinent test cases. The requirement is now considered complete.
Deleted	An approved requirement has been removed from the baseline. Include an explanation of why and by whom the decision was made to delete it.
Rejected	The requirement was proposed but is not planned for implementation in any upcoming release. Include an explanation of why and by whom the decision was made to reject it.

Figure 18-2 illustrates the tracking of requirements status throughout a hypothetical 10-month project. It shows the percentage of all the system's requirements having each status value at the end of each month. Note that tracking the distribution by percentages doesn't show whether the number of requirements in the baseline is changing over time. The curves illustrate how the project is approaching its goal of complete verification of all approved requirements. A body of work is done when all the allocated requirements have a status of either Verified (implemented as intended) or Deleted (removed from the baseline).

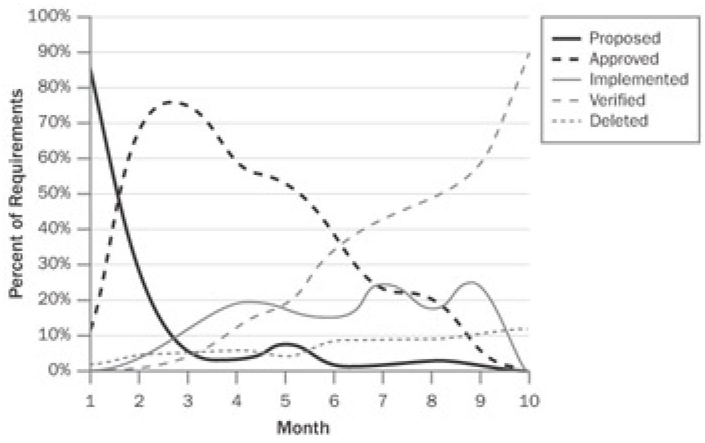


Figure 18-2: Tracking the distribution of requirements status throughout a project's development cycle.

Team LiB ◀ PREVIOUS NEXT ▶
Team LiB ◀ PREVIOUS NEXT ▶

Measuring Requirements Management Effort

As with requirements development, your project plan should include tasks and resources for the requirements management activities described in this chapter. If you track how much effort you spend on requirements management, you can evaluate whether that was too little, about right, or too much and adjust your processes and future planning accordingly.

An organization that has never measured any aspect of its work usually finds it difficult to begin recording how team members spend their time. Measuring actual development and project management effort requires a culture change, and the individual discipline to record daily work activities. Effort tracking isn't as time-consuming as people fear. The team will gain valuable insights from knowing how it has actually devoted its effort to various project tasks (Wiegers 1996a). Note that work effort is not the same as elapsed calendar time. Tasks can be interrupted or they might require interactions with other people that lead to delays. The total effort for a task, in units of labor hours, doesn't necessarily change because of such factors, but the duration increases.

Tracking effort also indicates whether the team is indeed performing the intended actions to manage its requirements. Failing to manage requirements increases the project's risk from uncontrolled changes and from requirements that are inadvertently overlooked during implementation. Count the effort devoted to the following activities as requirements management effort:

- Submitting requirements changes and proposing new requirements
- Evaluating proposed changes, including performing impact analysis
- Change control board activities
- Updating the requirements documents or database
- Communicating requirements changes to affected groups and individuals
- Tracking and reporting requirements status

- Collecting requirements traceability information

Requirements management helps to ensure that the effort you invest in gathering, analyzing, and documenting requirements isn't squandered. For example, failing to keep the requirements set current as changes are made makes it hard for stakeholders to know exactly what they can expect to have delivered in each release. Effective requirements management can reduce the expectation gap by keeping all project stakeholders informed about the current state of the requirements throughout the development process.

Next Steps

- Define a version control scheme to identify your requirements documents. Document the scheme as part of your requirements management process.
- Select the statuses that you want to use to describe the life cycle of your functional requirements. Draw a state-transition diagram to show the conditions that trigger a change from one status to another.
- Define the current status for each functional requirement in your baseline. Keep status current as development progresses.
- Write a description of the processes your organization will follow to manage the requirements on each project. Engage several analysts to draft, review, pilot, and approve the process activities and deliverables. The process steps you define must be practical and realistic, and they must add value to the project.

Team LiB ◂ PREVIOUS NEXT ▸
Team LiB ◂ PREVIOUS NEXT ▸

Chapter 19: Change Happens

Overview

"*How's your development work coming, Glenn?" asked Dave, the Chemical Tracking System project manager, during a status meeting.*

"I'm not as far along as I'd planned to be," Glenn admitted. "I'm adding a new catalog query function for Sharon, and it's taking a lot longer than I expected."

Dave was puzzled. "I don't remember discussing a new catalog query function at a change control board meeting recently. Did Sharon submit that request through the change process?"

"No, she approached me directly with the suggestion," said Glenn. "I should have asked her to submit it as a formal change request, but it seemed pretty simple so I told her I'd work it in. It turned out not to be simple at all. Every time I think I'm done, I realize I missed a change needed in another file, so I have to fix that, rebuild the component, and test it again. I thought this would take about six hours, but I've spent almost four days on it so far. That's why I'm not done with my other scheduled tasks. I know I'm holding up the next build. Should I finish adding this query function or go back to what I was working on

before?"

Most developers have encountered an apparently simple change that turned out to be far more complicated than expected. Developers sometimes don't—or can't—produce realistic estimates of the cost and other ramifications of a proposed software change. And, when developers who want to be accommodating agree to add enhancements that a user requests, requirements changes slip in through the back door instead of being approved by the right stakeholders. Such uncontrolled change is a common source of project chaos, schedule slips, and quality problems, especially on multisite and outsourced development projects. An organization that's serious about managing its software projects must ensure that

- Proposed requirements changes are carefully evaluated before being committed to.
- The appropriate individuals make informed business decisions about requested changes.
- Approved changes are communicated to all affected participants.
- The project incorporates requirements changes in a consistent fashion.

Software change isn't a bad thing. It's virtually impossible to define all of a product's requirements up front, and the world changes as development progresses. An effective software team can nimbly respond to necessary changes so that the product they build provides timely customer value.

But change always has a price. Revising a simple Web page might be quick and easy; making a change in an integrated chip design can cost tens of thousands of dollars. Even a rejected change request consumes the resources needed to submit, evaluate, and decide to reject it. Unless project stakeholders manage changes during development, they won't really know what will be delivered, which ultimately leads to an expectation gap. The closer you get to the release date, the more you should resist changing that release because the consequences of making changes become more severe.

The requirements analyst should incorporate approved changes in the project's requirements documentation. Your philosophy should be that the requirements documentation will accurately describe the delivered product. If you don't keep the SRS current as the product evolves, its value will decrease and the team might function as though it doesn't even have an SRS.

When you need to make a change, start at the highest level of abstraction that the change touches and cascade the impact of the change through related system components. For example, a proposed change might affect a use case and its functional requirements but not any business requirements. A modified high-level system requirement could have an impact on multiple software requirements. Some changes pertain only to system internals, such as the way a communications layer is implemented. These are not user-visible requirements changes but rather design or code changes.

True Stories Several problems can arise if a developer implements a requirement change directly in the code without flowing it through the requirements and design descriptions. The description of what the product does, as embodied in the requirements specification, becomes less accurate because the code is the ultimate software reality. The code can become brittle and fragile if changes are made without respecting the program's architecture and design structure. On one project, developers introduced new and modified functionality that the rest of the team didn't discover until system testing. This required unplanned rework of test procedures and user documentation. Consistent change-control practices help prevent such problems and the associated frustration, development rework, and wasted testing time.

Team LiB ◀ PREVIOUS NEXT ▶
Team LiB ◀ PREVIOUS NEXT ▶

Managing Scope Creep

Capers Jones (1994) reports that creeping requirements pose a major risk to

- 80 percent of management information systems projects.
- 70 percent of military software projects.
- 45 percent of outsourced software projects.

Creeping requirements include new functionality and significant modifications that are presented after the project requirements have been baselined. The longer that projects go on, the more growth in scope they experience. The requirements for management information systems typically grow about 1 percent per month (Jones 1996b). The growth rate ranges up to 3.5 percent per month for commercial software, with other kinds of projects in between. The problem is not that requirements change but that late changes have a big impact on work already performed. If every proposed change is approved, it might appear to project sponsors, participants, and customers that the project will never be completed—and indeed, it might not.

Some requirements evolution is legitimate, unavoidable, and even advantageous. Business processes, market opportunities, competing products, and technologies can change during the time it takes to develop a product, and management might determine that redirecting the project in response is necessary. Uncontrolled scope creep, in which the project continuously incorporates new functionality without adjusting resources, schedules, or quality goals, is more insidious. A small modification here, an unexpected enhancement there, and soon the project has no hope of delivering what the customers expect on schedule and with acceptable quality.

The first step in managing scope creep is to document the vision, scope, and limitations of the new system as part of the business requirements, as described in Chapter 5. Evaluate every proposed requirement or feature against the business objectives, product vision, and project scope. Engaging customers in elicitation reduces the number of requirements that are overlooked, only to be added to the team's workload after commitments are made and resources allocated (Jones 1996a). Prototyping is another effective technique for controlling scope creep (Jones 1994). A prototype provides a preview of a possible implementation, which helps developers and users reach a shared understanding of user needs and prospective solutions. Using short development cycles to release a system incrementally provides frequent opportunities for adjustments when requirements are highly uncertain or rapidly changing (Beck 2000).

True Stories The most effective technique for controlling scope creep is being able to say no (Weinberg 1995). People don't like to say no, and development teams can receive intense pressure to incorporate every proposed requirement. Philosophies such as "the customer is always right" or "we will achieve total customer satisfaction" are fine in the abstract, but you pay a price for them. Ignoring the price doesn't alter the fact that change is not free. The president of one software tool vendor is accustomed to hearing the development manager say "not now" when he suggests a new feature. "Not now" is more palatable than a simple rejection. It holds the promise of including the feature in a subsequent release. Including every feature that customers, marketing, product management, and developers request leads to missed commitments, slipshod quality, burned-out developers, and bloatware.

Customers aren't always right, but they do always have a point, so capture their ideas for possible inclusion in later development cycles.

In an ideal world, you would collect all of a new system's requirements before beginning construction and they'd remain stable throughout the development effort. This is the premise behind the sequential or waterfall software development life cycle model, but it doesn't work well in practice. At some point you must freeze the requirements for a specific release or you'll never get it out the door. However, stifling change prematurely ignores the realities that customers aren't always sure what they need, customer needs change, and developers want to respond to those changes. Every project needs to incorporate the most appropriate changes into the project in a controlled way.

Trap Freezing the requirements for a new system after performing some initial elicitation activities is unwise and unrealistic. Instead, define a baseline when you think the requirements are well enough defined for design and construction to begin, and then manage the inevitable changes to minimize their adverse impact on the project.

Team LiB ◀ PREVIOUS NEXT ▶
Team LiB ◀ PREVIOUS NEXT ▶

The Change-Control Process

True Stories While performing a software process assessment once, I asked the project team how they incorporated changes in the product's requirements. After an awkward silence, one person said, "Whenever the marketing rep wants to make a change, he asks Bruce or Sandy because they always say 'yes' and the rest of us give marketing a hard time about changes." This didn't strike me as a great change process.

A change-control process lets the project's leaders make informed business decisions that will provide the greatest customer and business value while controlling the product's life cycle costs. The process lets you track the status of all proposed changes, and it helps ensure that suggested changes aren't lost or overlooked. Once you've baselined a set of requirements, follow this process for all proposed changes to that baseline.

Customers and other stakeholders often balk at being asked to follow a new process, but a change-control process is not an obstacle to making necessary modifications. It's a funneling and filtering mechanism to ensure that the project incorporates the most appropriate changes. If a proposed change isn't important enough for a stakeholder to take just a couple of minutes to submit it through a standard, simple channel, then it's not worth considering for inclusion. Your change process should be well documented, as simple as possible, and—above all—effective.

Trap If you ask your stakeholders to follow a new change-control process that's ineffective, cumbersome, or too complicated, people will find ways to bypass the process—and perhaps they should.

Managing requirements changes is similar to the process for collecting and making decisions about defect reports. The same tools can support both activities. Remember, though: a tool is not a substitute for a process. Using a commercial problem-tracking tool to manage proposed modifications to requirements doesn't replace a written process that describes the contents and processing of a change request.

Change-Control Policy

Management should clearly communicate a policy that states its expectations of how project teams will handle proposed requirements changes. Policies are meaningful only if they are realistic, add value, and are enforced. I've found the following change-control policy statements to be helpful:

- All requirements changes shall follow the process. If a change request is not submitted in accordance with this process, it won't be considered.
- No design or implementation work other than feasibility exploration shall be performed on unapproved changes.
- Simply requesting a change doesn't guarantee that it will be made. The project's change control board (CCB) will decide which changes to implement. (We'll discuss change control boards later in this chapter.)
- The contents of the change database shall be visible to all project stakeholders.
- The original text of a change request shall not be modified or deleted.
- Impact analysis shall be performed for every change.
- Every incorporated requirement change shall be traceable to an approved change request.
- The rationale behind every approval or rejection of a change request shall be recorded.

Of course, tiny changes will hardly affect the project and big changes will have a significant impact. In principle, you'll handle all of these through your change-control process. In practice, you might elect to leave certain detailed requirements decisions to the developers' discretion, but no change affecting more than one individual's work should bypass your change-control process. However, your process should include a "fast path" to expedite low-risk, low-investment change requests in a compressed decision cycle.

Trap Development shouldn't use the change-control process as a barrier to halt any changes. Change is inevitable—deal with it.

Change-Control Process Description

Figure 19-1 illustrates a template for a change-control process description to handle requirements modifications and other project changes. You can download a sample change-control process description from *http://www.processimpact.com/goodies.shtml*. The following discussion pertains primarily to how the process would handle requirements changes. I find it helpful to include the following four components in all procedures and process descriptions:

1. **Introduction**

   1.1 Purpose

   1.2 Scope

   1.3 Definitions

2. **Roles and Responsibilities**
3. **Change-Request Status**
4. **Entry Criteria**
5. **Tasks**

   5.1 Evaluate Request

   5.2 Make Decision

   5.3 Make Change

   5.4 Notify All Affected Parties
6. **Veriflcation**

   6.1 Verify Change

   6.2 Install Product
7. **Exit Criteria**
8. **Change-Control Status Reporting**

Appendix: Data Items Stored for Each Request

Figure 19-1: Sample template for a change-control process description.

- Entry criteria—the conditions that must be satisfied before executing the process or procedure
- The various tasks involved in the process or procedure, the project role responsible for each task, and other participants in the task
- Steps to verify that the tasks were completed correctly
- Exit criteria—the conditions that indicate when the process or procedure is successfully completed

1. Introduction

The introduction describes the purpose of this process and identifies the organizational scope to which it applies. If this process covers changes only in certain work products, identify them here. Also indicate whether any specific kinds of changes are exempted, such as changes in interim or temporary work products created during the course of a project. Define any terms that are necessary for understanding the rest of the document in section 1.3.

2. Roles and Responsibilities

List the project team members—by role, not by name—who participate in the change-control activities and describe their responsibilities. Table 19-1 suggests some pertinent roles; adapt these to each project situation. Different individuals need not fill each role. For example, the CCB Chair might also receive submitted change requests. Several—perhaps all—roles can be filled by the same person on a small project.

Table 19-1: Possible Project Roles in Change-Management Activities

Role	**Description and Responsibilities**
CCB Chair	Chairperson of the change control board; generally has final decision-making authority if the CCB does not reach agreement; selects the Evaluator and the Modifier for each change request
CCB	The group that decides to approve or reject proposed changes for a specific project
Evaluator	The person whom the CCB Chair asks to analyze the impact of a proposed change; could be a technical person, a customer, a marketing person, or a combination
Modifier	The person responsible for making changes in a work product in response to an approved change request
Originator	Someone who submits a new change request
Request Receiver	The person to whom new change requests are submitted
Verifier	The person who determines whether the change was made correctly

3. Change-Request Status

A change request passes through a defined life cycle, having a different status at each stage in its life. You can represent these status changes by using a state-transition diagram, as illustrated in Figure 19-2. Update a request's status only when the specified criteria are met.

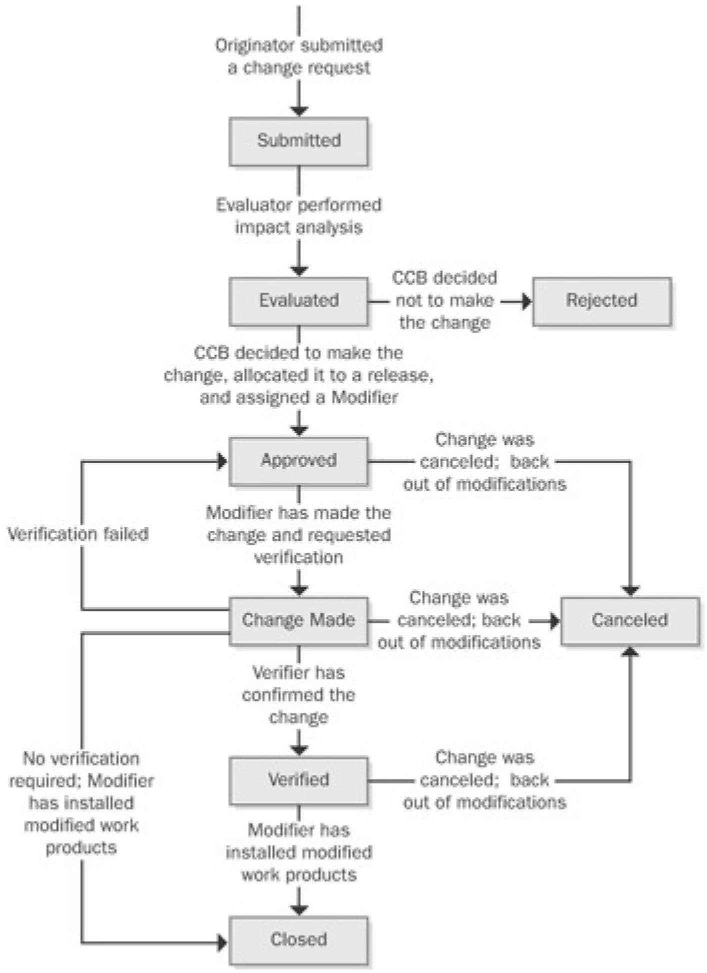

Originator submitted a change request
Submitted
Evaluator performed impact analysis
Evaluated
CCB decided not to make the change
Rejected
CCB decided to make the change, allocated it to a release, and assigned a Modifier
Approved
Change was canceled; back out of modifications
Verification failed
Modifier has made the change and requested verification
Change Made
Change was canceled; back out of modifications
Canceled
Verifier has confirmed the change
No verification required; Modifier has installed modified work products
Verified
Change was canceled; back out of modifications
Modifier has installed modified work products
Closed

Figure 19-2: State-transition diagram for a change request.

4. Entry Criteria

The basic entry criterion for your change-control process is

- A valid change request has been received through an approved channel.

All potential originators should know how to submit a change request, whether it's by completing a paper or Web-based form, sending an e-mail message, or using a change-control tool. Assign a unique identification tag to every change request, and route them all to a single point of contact, the Request Receiver.

5. Tasks

The next step is to evaluate the request for technical feasibility, cost, and alignment with the project's business requirements and resource constraints. The CCB Chair might assign an evaluator to perform an impact analysis, risk analysis, hazard analysis, or other assessments. This analysis ensures that the potential consequences of accepting the change are well understood. The evaluator and the CCB should also consider the business and technical implications of rejecting the change.

The appropriate decision makers, chartered as the CCB, then elect whether to approve or reject the requested change. The CCB gives each approved change a priority level or target implementation date, or it allocates the change to a specific build or release number. The CCB communicates the decision by updating the request's status and notifying all team members who might have to modify work products. Affected work products could include the requirements documentation, design descriptions and models, user interface components, code, test documentation, help screens, and user manuals. The modifier(s) update the affected work products as necessary.

6. Verification

Requirements changes are typically verified through a peer review to ensure that modified specifications, use cases, and models correctly reflect all aspects of the change. Use traceability information to find all the parts of the system that the change touched and that therefore must be verified. Multiple team members might verify the changes made in downstream work products through testing or review. Following verification, the modifier installs the updated work products to make them available to the rest of the team and redefines the baseline to reflect the changes.

7. Exit Criteria

All of the following exit criteria must be satisfied to properly complete an execution of your change-control process:

- The status of the request is Rejected, Closed, or Canceled.
- All modified work products are installed into the correct locations.
- The originator, CCB Chair, project manager, and other relevant project participants have been notified of the change details and the current status of the change request.
- The requirements traceability matrix has been updated. (Chapter 20, "Links in the Requirements Chain," will discuss requirements traceability in greater detail.)

8. Change-Control Status Reporting

Identify the charts and reports you'll use to summarize the contents of the change-control database. These charts typically show the number of change requests in each status category as a function of time. Describe the procedures for producing the charts and reports. The project manager uses these reports when tracking the project's status.

Appendix: Data Items Stored for Each Request

Table 19-2 lists some data items to consider storing for each change request. When you define your own list, indicate which items are required and which are optional. Also, indicate whether the value for each item is set automatically by your change-control tool or manually by one of the change-management participants.

Table 19-2: Suggested Change-Request Data Items

Item	Description
Change Origin	Functional area that requested the change; possible groups include marketing, management, customer, software engineering, hardware engineering, and testing
Change-Request ID	Identification tag or sequence number assigned to the request

Change Type	Type of change request, such as a requirement change, a proposed enhancement, or a defect report
Date Submitted	Date the originator submitted the change request
Date Updated	Date the change request was most recently modified
Description	Free-form text description of the change being requested
Implementation Priority	The relative importance of making the change as determined by the CCB: low, medium, or high
Modifier	Name of the person who is primarily responsible for implementing the change
Originator	Name of the person who submitted this change request; consider including the originator's contact information
Originator Priority	The relative importance of making the change from the originator's point of view: low, medium, or high
Planned Release	Product release or build number for which an approved change is scheduled
Project	Name of the project in which a change is being requested
Response	Free-form text of responses made to the change request; multiple responses can be made over time; do not change existing responses when entering a new one
Status	The current status of the change request, selected from the options in Figure 19-2
Title	One-line summary of the proposed change
Verifier	Name of the person who is responsible for determining whether the change was made correctly

Team LiB ◄ PREVIOUS NEXT ►
Team LiB ◄ PREVIOUS NEXT ►

The Change Control Board

The change control board (sometimes known as the *configuration control board*) has been identified as a best practice for software development (McConnell 1996). The CCB is the body of people, be it one individual or a diverse group, who decides which proposed requirement changes and newly suggested features to accept for inclusion in the product. The CCB also decides which reported defects to correct and when to correct them. Many projects already have some de facto group that makes change decisions; establishing a CCB formalizes this group's composition and authority and defines its operating procedures.

A CCB reviews and approves changes to any baselined work product on a project, of which the requirements documents are only one example. Some CCBs are empowered to make decisions and simply inform management about them, whereas others can only make recommendations for management decision. On a small project it makes sense to have only one or two people make the change decisions. Large projects or programs might have several levels of CCBs. Some are responsible for business decisions, such as requirements changes, and some for technical decisions (Sorensen 1999). A higher-level CCB has authority to approve changes that have a greater impact on the project. For instance, a large program that encompasses multiple projects would establish a program-level CCB and an individual CCB for each project. Each project CCB resolves issues and changes that affect only that project. Issues that affect other projects and changes that exceed a specified cost or schedule impact are escalated to the program-level CCB.

To some people, the term "change control board" conjures an image of wasteful bureaucratic overhead. Instead, think of the CCB as providing a valuable structure to help manage even a small project. This structure doesn't have to be time-consuming or cumbersome—just effective. An effective CCB will consider all proposed changes promptly and will make timely decisions based on analysis of the potential impacts and benefits of each proposal. The CCB should be no larger and no more formal than necessary to ensure that the right people make good business decisions about every requested modification.

CCB Composition

The CCB membership should represent all groups who need to participate in making decisions within the scope of that CCB's authority. Consider selecting representatives from the following areas:

- Project or program management
- Product management or requirements analyst
- Development
- Testing or quality assurance
- Marketing or customer representatives
- User documentation
- Technical support or help desk
- Configuration management

Only a subset of these people need to make the decisions, although all must be informed of decisions that affect their work.

The same handful of individuals will fill several of these roles on small projects, and other roles won't need to contemplate every change request. The CCB for a project that has both software and hardware components might also include representatives from hardware engineering, system engineering, manufacturing, or perhaps hardware quality assurance and configuration management. Keep the CCB as small as possible so that the group can respond promptly and efficiently to change requests. As most of us have discovered, large groups have difficulty scheduling meetings and reaching decisions. Make sure that the CCB members understand their responsibilities and take them seriously. To ensure that the CCB has adequate technical and business information, invite other individuals to a CCB meeting when specific proposals are being discussed that relate to those individuals' expertise.

CCB Charter

A charter describes the CCB's purpose, scope of authority, membership, operating procedures, and decision-making process (Sorensen 1999). A template for a CCB charter is available from *http://www.processimpact.com/goodies.shtml*. The charter should state the frequency of regularly scheduled CCB meetings and the conditions that trigger a special meeting. The scope of the CCB's authority indicates which decisions it can make and which ones it must pass on to a higher-level CCB or a manager.

Making Decisions

As with all decision-making bodies, each CCB needs to select an appropriate decision rule and process (Gottesdiener 2002). The decision-making process description should indicate the following:

- The number of CCB members or the key roles that constitutes a quorum for making decisions
- Whether voting, consensus, consultative decision making, or some other decision rule is used
- Whether the CCB Chair may overrule the CCB's collective decision
- Whether a higher level of CCB or someone else must ratify the decision

The CCB balances the anticipated benefits against the estimated impact of accepting a proposed change. Benefits from improving the product include financial savings, increased revenue, higher customer satisfaction, and competitive advantage. The impact indicates the adverse effects that accepting the proposal could have on the product or project. Possible impacts include increased development and support costs, delayed delivery, degraded product quality, reduced functionality, and user dissatisfaction. If the estimated cost or schedule impact exceeds the established thresholds for this level of CCB, refer the change to management or to a higher-level CCB. Otherwise, use the CCB's decision-making process to approve or reject the proposed change.

Communicating Status

Once the CCB makes its decision, a designated individual updates the request's status in the change database. Some tools automatically generate e-mail messages to communicate the new status to the originator who proposed the change and to others affected by the change. If e-mail is not generated automatically, inform the affected people manually so that they can properly process the change.

Renegotiating Commitments

It's not realistic to assume that stakeholders can stuff more and more functionality into a project that has schedule, staff, budget, and quality constraints and still succeed. Before accepting a significant requirement change, renegotiate commitments with management and customers to accommodate the change (Humphrey 1997). You might negotiate for more time or staff or ask to defer pending requirements of lower priority. If you don't obtain some commitment adjustments, document the threats to success in your project's risk list so that people aren't surprised if the project doesn't fully achieve the desired outcomes.

Team LiB ◀ PREVIOUS NEXT ▶
Team LiB ◀ PREVIOUS NEXT ▶

Change-Control Tools

Automated tools can help your change-control process operate more efficiently (Wiegers 1996a). Many teams use commercial problem- or issue-tracking tools to collect, store, and manage requirements changes. A list of recently submitted change proposals generated from the tool can serve as the agenda for a CCB meeting. Problem-tracking tools can also report the number of requests in each status category at any given time.

Because the available tools, vendors, and features frequently change, I won't provide specific tool recommendations here. Look for the following features in a tool to support your requirements change process:

- Lets you define the data items included in a change request
- Lets you define a state-transition model for the change-request life cycle
- Enforces the state-transition model so that only authorized users can make the permitted status changes
- Records the date of every status change and the identity of the person who made it
- Lets you define who receives automatic e-mail notification when an originator submits a new request or when a request's status is updated
- Produces both standard and custom reports and charts

Some commercial requirements management tools—such tools are discussed in Chapter 21—have a simple change-proposal system built in. These systems can link a proposed change to a specific requirement so that the individual responsible for each requirement is notified by e-mail whenever someone submits a pertinent change request.

Tooling Up a Process

True Stories When I worked in a Web development team, one of our first process improvements was to implement a change-control process to manage our huge backlog of change requests (Wiegers 1999b). We began with a process similar to the one described in this chapter. We piloted it for a few weeks by using paper forms while I evaluated several issue-tracking tools. During the pilot process we found several ways to improve the process and discovered several additional data items we needed in the change requests. We selected a highly configurable issue-tracking tool and tailored it to match our process. The team used this process and tool to handle requirements changes in systems under development, defect reports and suggested enhancements in production systems, updates to Web site content, and requests for new development projects. Change control was one of our most successful process improvement initiatives.

Team LiB ◂ PREVIOUS NEXT ▸
Team LiB ◂ PREVIOUS NEXT ▸

Measuring Change Activity

Software measurements provide insights into your projects, products, and processes that are more accurate than subjective impressions or vague recollections of what happened in the past. The measurements you select should be motivated by the questions you or your managers are trying to answer and the goals you're trying to achieve. Measuring change activity is a way to assess the stability of the requirements. It also reveals opportunities for process improvement that might lead to fewer change requests in the future. Consider tracking the following aspects of your requirements change activity (Paulk et al. 1995):

- The number of change requests received, currently open, and closed
- The cumulative number of changes made, including added, deleted, and modified requirements (You can also express this as a percentage of the total number of requirements in the baseline.)
- The number of change requests that originated from each source
- The number of changes proposed and made in each requirement since it was baselined
- The total effort devoted to processing and implementing change requests
- The number of cycles through the change process that it took to correctly implement each approved change (Sometimes changes are implemented improperly or cause other errors that need to be corrected.)

You don't necessarily need to monitor your requirements change activities to this degree. As with all software metrics, understand your goals and how you'll use the data before you decide what to measure (Wiegers 1999a). Start with simple measurements to begin establishing a measurement culture in your organization and to collect the key data you need to manage your projects effectively. As you gain experience, make your measurements as sophisticated as necessary to help your projects succeed.

Figure 19-3 illustrates a way to track the number of requirements changes your project experiences during development. This chart tracks the rate at which proposals for new requirements changes arrive. You can track the number of approved change requests similarly. Don't count changes made before baselining; you know the requirements are still evolving prior to establishing the baseline. Once you've baselined the requirements, though, all proposed changes should follow your change-control process. You should also begin to track the frequency of change (requirements volatility). This change-frequency chart should trend toward zero as the project approaches its ship date. A sustained high frequency of changes implies a risk of failing to meet your schedule commitments. It probably also indicates that the original baselined requirements were incomplete, suggesting that improving your requirements elicitation practices might be a good idea.

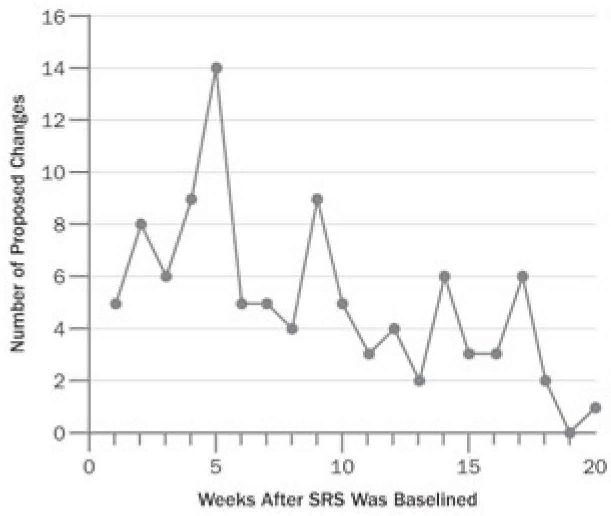


Figure 19-3: Sample chart of requirements change activity.

A project manager who's concerned that frequent changes might prevent the project from finishing on schedule can gain further insight by tracking the requirements change origins. Figure 19-4 shows a way to represent the number of change requests that came from different sources. The project manager could discuss a chart like this with the marketing manager and point out that marketing has requested the most requirements changes. This might lead to a fruitful discussion about what actions the team could take to reduce the number of post-baselining changes received from marketing in the future. Using data as a starting point for such discussions is more constructive than holding a confrontational meeting stimulated by frustration.

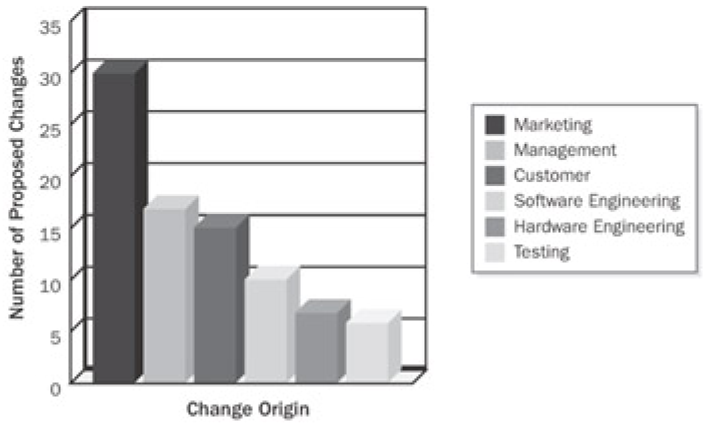


Figure 19-4: Sample chart of requirement change origins.

Team LiB
Team LiB
PREVIOUS NEXT
PREVIOUS NEXT

Change Isn't Free: Impact Analysis

True Stories The need for impact analysis is obvious for major enhancements. However, unexpected complications can lurk below the surface of even minor change requests. A company once had to change the text of one error message in its product. What could be simpler? The product was available in both English and German language versions. There were no problems in English, but in German the new message exceeded the maximum character length allocated for error message displays in both the message box and a database. Coping with this apparently simple change request turned out to be much more work than the developer had anticipated when he promised a quick turnaround.

Trap Because people don't like to say "no," it's easy to accumulate a huge backlog of approved change requests. Before accepting a proposed change, make sure you understand the rationale behind it, its alignment with the product vision, and the business value that the change will provide.

Impact analysis is a key aspect of responsible requirements management (Arnold and Bohner 1996). It provides accurate understanding of the implications of a proposed change, which helps the team make informed business decisions about which proposals to approve. The analysis examines the proposed change to identify components that might have to be created, modified, or discarded and to estimate the effort associated with implementing the change. Before a developer says, "Sure, no problem" in response to a change request, he or she should spend a little time on impact analysis.

Impact Analysis Procedure

The CCB Chair will typically ask a knowledgeable developer to perform the impact analysis for a specific change proposal. Impact analysis has three aspects:

1. Understand the possible implications of making the change. Change often produces a large ripple effect. Stuffing too much functionality into a product can reduce its performance to unacceptable levels, as when a system that runs daily requires more than 24 hours to complete a single execution.

2. Identify all the files, models, and documents that might have to be modified if the team incorporates the requested change.

3. Identify the tasks required to implement the change, and estimate the effort needed to complete those tasks.

   Trap Skipping impact analysis doesn't change the size of the task. It just turns the size into a surprise. Software surprises are rarely good news.

Figure 19-5 presents a checklist of questions designed to help the impact analyst understand the implications of accepting a proposed change. The checklist in Figure 19-6 contains prompting questions to help identify all software elements that the change might affect. Traceability data that links the affected requirement to other downstream deliverables helps greatly with impact analysis. As you gain experience using these checklists, modify them to suit your own projects.

- Do any existing requirements in the baseline conflict with the proposed change?
- Do any other pending requirements changes conflict with the proposed change?
- What are the business or technical consequences of not making the change?
- What are possible adverse side effects or other risks of making the proposed change?
- Will the proposed change adversely affect performance requirements or other quality attributes?
- Is the proposed change feasible within known technical constraints and currents staff skills?
- Will the proposed change place unacceptable demands on any computer resources required for

the development, test, or operating environments?

- Must any tools be acquired to implement and test the change?
- How will the proposed change affect the sequence, dependencies, effort, or duration of any tasks currently in the project plan?
- Will prototyping or other user input be required to verify the proposed change?
- How much effort that has already been invested in the project will be lost if this change is accepted?
- Will the proposed change cause an increase in product unit cost, such as by increasing third-party product licensing fees?
- will the change affect any marketing, manufacturing, training, or customer support plans?

Figure 19-5: Checklist of possible implications of a proposed change.

- Identify any user interface changes, additions, or deletions required.
- Identify any changes, additions, or deletions required in reports, databases, or files.
- Identify the design components that must be created, modified, or deleted.
- Identify the source code files that must be created, modified, or deleted.
- Identify any changes required in build files or procedures.
- Identify existing unit, integration, system, and acceptance test cases that must be modified or deleted.
- Estimate the number or new unit, integration, system, and acceptance test cases that will be required.
- Identify any help screens, training materials, or other user documentation that must be created or modified.
- Identify any other applications, libraries, or hardware components affected by the change.
- Identify any third-party software that must be purchased or licensed.
- Identify any impact the proposed change will have on the project's software project management plan, quality assurance plan, configuration management plan, or other plans.

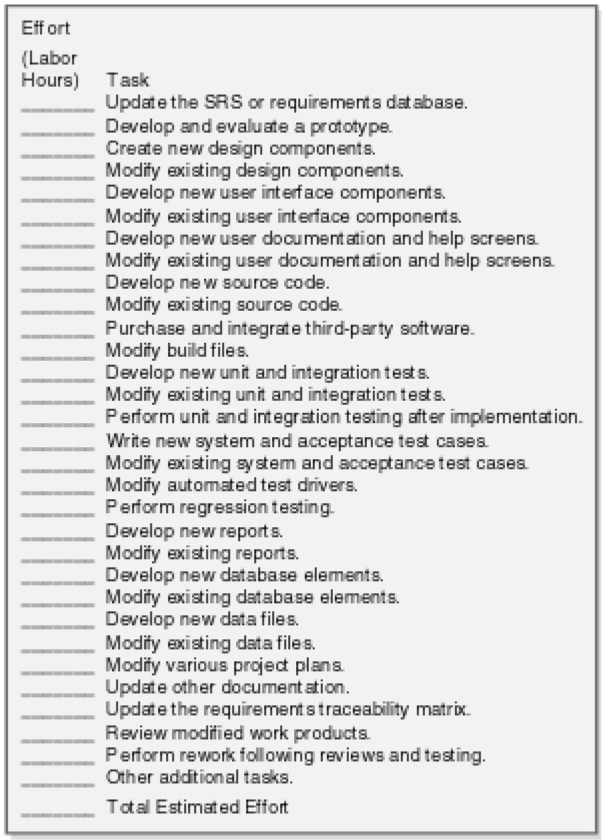
Figure 19-6: Checklist of possible software elements affected by a proposed change.

Following is a simple procedure for evaluating the impact of a proposed requirement change. Many estimation problems arise because the estimator doesn't think of all the work required to complete an activity. Therefore, this impact analysis approach emphasizes comprehensive task identification. For substantial changes, use a small team—not just one developer—to do the analysis and effort estimation to avoid overlooking important tasks.

1. Work through the checklist in Figure 19-5.

2. Work through the checklist in Figure 19-6, using available traceability information. Some requirements management tools include an impact analysis report that follows traceability links and finds the system elements that depend on the requirements affected by a change proposal.

3. Use the worksheet in Figure 19-7 to estimate the effort required for the anticipated tasks. Most change requests will require only a portion of the tasks on the worksheet, but some will involve additional tasks.



Effort (Labor Hours)	Task
________	Update the SRS or requirements database.
________	Develop and evaluate a prototype.
________	Create new design components.
________	Modify existing design components.
________	Develop new user interface components.
________	Modify existing user interface components.
________	Develop new user documentation and help screens.
________	Modify existing user documentation and help screens.
________	Develop new source code.
________	Modify existing source code.
________	Purchase and integrate third-party software.
________	Modify build files.
________	Develop new unit and integration tests.
________	Modify existing unit and integration tests.
________	Perform unit and integration testing after implementation.
________	Write new system and acceptance test cases.
________	Modify existing system and acceptance test cases.
________	Modify automated test drivers.
________	Perform regression testing.
________	Develop new reports.
________	Modify existing reports.
________	Develop new database elements.
________	Modify existing database elements.
________	Develop new data files.
________	Modify existing data files.
________	Modify various project plans.
________	Update other documentation.
________	Update the requirements traceability matrix.
________	Review modified work products.
________	Perform rework following reviews and testing.
________	Other additional tasks.
________	Total Estimated Effort

Figure 19-7: Estimating effort for a requirement change.

4. Total the effort estimates.

5. Identify the sequence in which the tasks must be performed and how they can be interleaved with currently planned tasks.

6. Determine whether the change is on the project's critical path. If a task on the critical path slips, the project's completion date will slip. Every change consumes resources, but if you can plan a change to avoid affecting tasks that are currently on the critical path, the change won't cause the entire project to slip.

7. Estimate the impact of the proposed change on the project's schedule and cost.

8. Evaluate the change's priority by estimating the relative benefit, penalty, cost, and technical risk

compared to other discretionary requirements.

9. Report the impact analysis results to the CCB so that they can use the information to help them decide whether to approve or reject the change request.

More Info Regarding step 8, Chapter 14 ("Setting Requirement Priorities") describes the rating scales for prioritizing requirements based on benefit, penalty, cost, and risk.

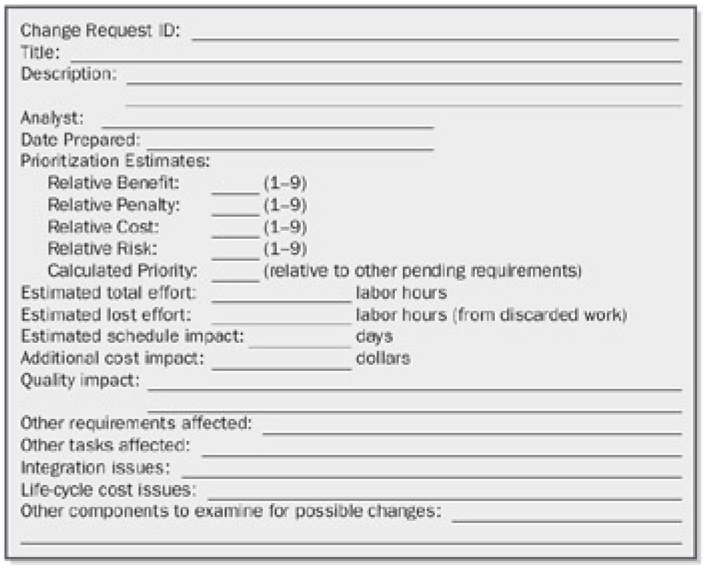
In most cases, this procedure shouldn't take more than a couple of hours to complete. This seems like a lot of time to a busy developer, but it's a small investment in making sure the project wisely invests its limited resources. If you can adequately assess the impact of a change without such a systematic evaluation, go right ahead; just make sure you aren't stepping into quicksand. To improve your ability to estimate the impacts of future changes, compare the actual effort needed to implement each change with the estimated effort. Understand the reasons for any differences, and modify the impact estimation checklists and worksheet accordingly.

Money Down the Drain

True Stories Two developers at the A. Datum Corporation estimated that it would take four weeks to add an enhancement to one of their information systems. The customer approved the estimate, and the developers set to work. After two months, the enhancement was only about half done and the customer lost patience: "If I'd known how long this was really going to take and how much it was going to cost, I wouldn't have approved it. Let's forget the whole thing." In the rush to gain approval and begin implementation, the developers didn't do enough impact analysis to develop a reliable estimate that would let the customer make an appropriate business decision. Consequently, the A. Datum Corporation wasted several hundred hours of work that could have been avoided by spending a few hours on an up-front impact analysis.

Impact Analysis Report Template

Figure 19-8 suggests a template for reporting the results from analyzing the potential impact of each requirement change. Using a standard template makes it easier for the CCB members to find the information they need to make good decisions. The people who will implement the change will need the analysis details and the effort planning worksheet, but the CCB needs only the summary of analysis results. As with all templates, try it and then adjust it to meet your project needs.



Change Request ID: ____
Title: ____
Description: ____

Analyst: ____
Date Prepared: ____
Prioritization Estimates:
Relative Benefit: ____ (1–9)
Relative Penalty: ____ (1–9)
Relative Cost: ____ (1–9)
Relative Risk: ____ (1–9)
Calculated Priority: ____ (relative to other pending requirements)
Estimated total effort: ____ labor hours
Estimated lost effort: ____ labor hours (from discarded work)
Estimated schedule impact: ____ days
Additional cost impact: ____ dollars
Quality impact: ____

Other requirements affected: ____
Other tasks affected: ____
Integration issues: ____
Life-cycle cost issues: ____
Other components to examine for possible changes: ____

Figure 19-8: Impact analysis report template.

Requirements change is a reality for all software projects, but disciplined change-management practices can reduce the disruption that changes can cause. Improved requirements elicitation techniques can reduce the number of requirements changes, and effective requirements management will improve your ability to deliver on project commitments.

Note Figures 19-5 through 19-8 are available at http://www.processimpact.com/goodies.shtml.

Next Steps

- Identify the decision makers on your project, and set them up as a change control board. Have the CCB adopt a charter to make sure everyone understands the board's purpose, composition, and decision-making process.

- Define a state-transition diagram for the life cycle of proposed requirements changes in your project, starting with the diagram in Figure 19-2. Write a process to describe how your team will handle proposed requirements changes. Use the process manually until you're convinced that it's practical and effective.

- Select a commercial problem- or issue-tracking tool that's compatible with your development environment. Tailor it to align with the change-control process you created in the previous step.

- The next time you evaluate a requirement change request, first estimate the effort by using your old method. Then estimate it again using the impact analysis approach described in this chapter. If you implement the change, compare the two estimates to see which agrees more closely with the actual effort. Modify the impact analysis checklists and worksheet based on your experience.

Team LiB PREVIOUS NEXT
Team LiB PREVIOUS NEXT

Chapter 20: Links in the Requirements Chain

Overview

"*We just found out that the new union contract is changing how overtime pay and shift bonuses are calculated,*" *Justin reported at the weekly team meeting.* "*They're also changing how the seniority rules affect priority for vacation scheduling, shift preferences, and everything else. We have to update the payroll and staff scheduling systems to handle all these changes right away. How long do you think it will take to get this done, Chris?*"

"*Man, that's going to be a lot of work,*" *said Chris.* "*The logic for the seniority rules is sprinkled all through the scheduling system. I can't give you a decent estimate. It's going to take hours just to scan through the code and try to find all the places where those rules show up.*"

Simple requirement changes often have far-reaching impacts, necessitating that many parts of the product be modified. It's hard to find all the system components that might be affected by a requirement modification. Chapter 19 discussed the importance of performing an impact analysis to make sure the team knows what it's getting into before it commits to making a proposed change. Change impact analysis is easier if you have a road map that shows where each requirement or business rule was implemented in the software.

This chapter addresses the subject of requirements tracing (or traceability). Requirements tracing documents the dependencies and logical links between individual requirements and other system elements. These elements include other requirements of various types, business rules, architecture and other design components, source code modules, test cases, and help files. Traceability information facilitates impact analysis by helping you identify all the work products you might have to modify to implement a proposed requirement change.

Team LiB ◂ PREVIOUS NEXT ▸
Team LiB ◂ PREVIOUS NEXT ▸

Tracing Requirements

Traceability links allow you to follow the life of a requirement both forward and backward, from origin through implementation (Gotel and Finkelstein 1994). Chapter 1, "The Essential Software Requirement," identified traceability as one of the characteristics of excellent requirements specifications. To permit traceability, each requirement must be uniquely and persistently labeled so that you can refer to it unambiguously throughout the project. Write the requirements in a fine-grained fashion, rather than creating large paragraphs containing many individual functional requirements that lead to an explosion of design and code elements.

Figure 20-1 illustrates four types of requirements traceability links (Jarke 1998). Customer needs are traced *forward to requirements*, so you can tell which requirements will be affected if those needs change during or after development. This also gives you confidence that the requirements specification has addressed all stated customer needs. Conversely, you can trace *backward from requirements* to customer needs to identify the origin of each software requirement. If you represented customer needs in the form of use cases, the top half of Figure 20-1 illustrates tracing between use cases and functional requirements.

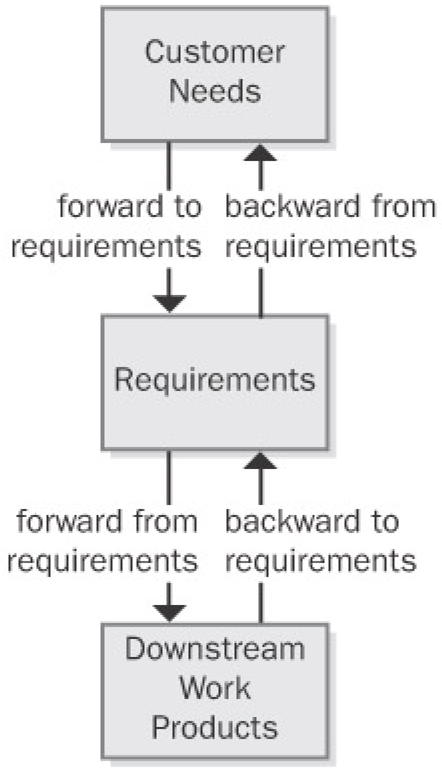


Figure 20-1: Four types of requirements traceability.

The bottom half of Figure 20-1 indicates that, as requirements flow into downstream deliverables during development, you can trace *forward from requirements* by defining links between individual requirements and specific product elements. This type of link assures that you've satisfied every requirement because you know which components address each one. The fourth type of link traces specific product elements *backward to requirements* so that you know why each item was created. Most applications include code that doesn't relate directly to user-specified requirements, but you should know why someone wrote every line of code.

Suppose a tester discovers unexpected functionality with no corresponding written requirement. This code could indicate that a developer implemented a legitimate implied requirement that the analyst can now add to the specification. Alternatively, it might be "orphan" code, an instance of gold plating that doesn't belong in the product. Traceability 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, test cases that are derived from—and traced back to—individual requirements provide a mechanism for detecting unimplemented requirements because the expected functionality will be missing.

Traceability links help you keep track of parentage, interconnections, and dependencies among individual requirements. This information reveals the propagation of change that can result when a specific requirement is deleted or modified. If you've mapped specific requirements to tasks in your project's work-breakdown structure, those tasks will be affected when a requirement is changed or deleted.

Figure 20-2 illustrates many kinds of direct traceability relationships that can be defined on a project. You don't need to define and manage all these traceability link types. On many projects you can gain 80 percent of the desired traceability benefits for perhaps 20 percent of the potential effort. Perhaps you only need to trace system tests back to functional requirements or use cases. Decide which links are pertinent to your project and can contribute the most to successful development and efficient maintenance. Don't ask team members to spend time recording information unless you have a clear idea of how you expect to use it.

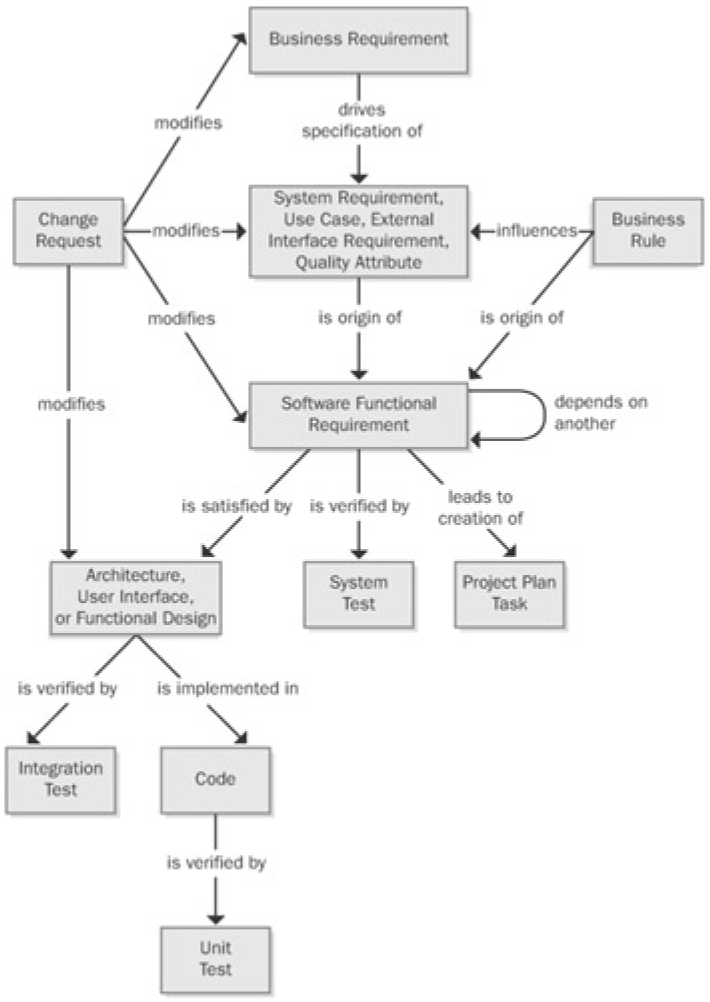


Figure 20-2: Some possible requirements traceability links.

Team LiB ◀ PREVIOUS NEXT ▶
Team LiB ◀ PREVIOUS NEXT ▶

Motivations for Tracing Requirements

True Stories I've had the embarrassing experience of writing a program and then realizing I had inadvertently omitted a requirement. It was in the spec—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 that a customer isn't satisfied or a delivered product is missing a safety-critical function. At one level, requirements tracing provides a way to demonstrate compliance with a contract, specification, or regulation. At a more sophisticated level, requirements tracing can improve the quality of your products, reduce maintenance costs, and facilitate reuse (Ramesh 1998).

Tracing requirements is a manually intensive task that requires organizational commitment. Keeping the link information current as the system undergoes development and maintenance takes discipline. If the traceability information becomes obsolete, you'll probably never reconstruct it. Obsolete traceability data wastes time by sending developers and maintainers down the wrong path. Because of these realities, you should adopt requirements traceability for the right reasons (Ramesh et al. 1995). Following are some potential benefits of implementing requirements traceability:

- **Certification** You can use traceability information when certifying a safety-critical product to demonstrate that all requirements were implemented—although it doesn't confirm that they were implemented correctly or completely! Of course, if the requirements are incorrect or key requirements are absent, even the best traceability data won't help you.

- **Change impact analysis** Without traceability information, there's a high probability of overlooking a system element that would be affected if you add, delete, or modify a particular requirement.
- **Maintenance** Reliable traceability information facilitates making changes correctly and completely during maintenance, which improves your productivity. When corporate policies or government regulations change, software applications often require updating. A table that shows where each applicable business rule was implemented in the functional requirements, designs, and code makes it easier to make the necessary changes properly.
- **Project tracking** If you diligently record the traceability data during development, you'll have an accurate record of the implementation status of planned functionality. Missing links indicate work products that have not yet been created.
- **Reengineering** You can list the functions in a legacy system you're replacing and record where they were addressed in the new system's requirements and software components. Defining traceability links is a way to capture some of what you learn through reverse engineering of an existing system.
- **Reuse** Traceability information facilitates reusing product components by identifying packages of related requirements, designs, code, and tests.
- **Risk reduction** Documenting the component interconnections reduces the risk if a key team member with essential knowledge about the system leaves the project (Ambler 1999).
- **Testing** When a test yields an unexpected result, the links between tests, requirements, and code point you toward likely parts of the code to examine for a defect. Knowing which tests verify which requirements can save time by letting you eliminate redundant tests.

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 traceability information. View requirements tracing as an investment that increases your chances of delivering a maintainable product that satisfies all the stated customer requirements. Although difficult to quantify, this investment will pay dividends anytime you have to modify, extend, or replace the product. Defining traceability links is not much work if you collect the information as development proceeds, but it's tedious and expensive to do on a completed system.

Team LiB ◄ PREVIOUS NEXT ►
Team LiB ◄ PREVIOUS NEXT ►

The Requirements Traceability Matrix

True Stories The most common way to represent the links between requirements and other system elements is in a *requirements traceability matrix*, also called a *requirements trace matrix* or a *traceability table* (Sommerville and Sawyer 1997). Table 20-1 illustrates a portion of one such matrix, drawn from the Chemical Tracking System.When I've set up such matrices in the past, I made a copy of the baselined SRS and deleted everything except the labels for the functional requirements. Then I set up a table formatted like Table 20-1 with only the Functional Requirement column populated. As fellow team members and I worked on the project, we gradually filled in the blank cells in the matrix.

Table 20-1: One Kind of Requirements Traceability Matrix

User Requirement	Functional Requirement	Design Element	Code Module	Test Case
UC-28	catalog.query.sort	Class catalog	catalog.sort()	search.7 search.8

UC-29	catalog.query.import	Class catalog	catalog.import()
			catalog.validate()
				search.12
				search.13
				search.14

Table 20-1 shows how each functional requirement is linked backward to a specific use case and forward to one or more design, code, and test elements. Design elements can be objects in analysis models such as data flow diagrams, tables in a relational data model, or object classes. Code references can be class methods, stored procedures, source code filenames, or procedures or functions within the source file. You can add more columns to extend the links to other work products, such as online help documentation. Including more traceability detail takes more work, but it gives you the precise locations of the related software elements, which can save time during change impact analysis and maintenance.

Fill in the information as the work gets done, not as it gets planned. That is, enter "catalog.sort()" in the Code Module column of the first row in Table 20-1 only when the code in that function has been written, has passed its unit tests, and has been integrated into the source code baseline for the product. This way a reader knows that populated cells in the requirements traceability matrix indicate work that's been completed. Note that 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.

Nonfunctional requirements such as performance goals and quality attributes don't always trace directly into code. A response-time requirement might dictate the use of certain hardware, algorithms, database structures, or architectural choices. A portability requirement could restrict the language features that the programmer uses but might not result in specific code segments that enable portability. Other quality attributes are indeed implemented in code. Integrity requirements for user authentication lead to derived functional requirements that are implemented through, say, passwords or biometrics functionality. In those cases, trace the corresponding functional requirements backward to their parent nonfunctional requirement and forward into downstream deliverables as usual. Figure 20-3 illustrates a possible traceability chain involving nonfunctional requirements.

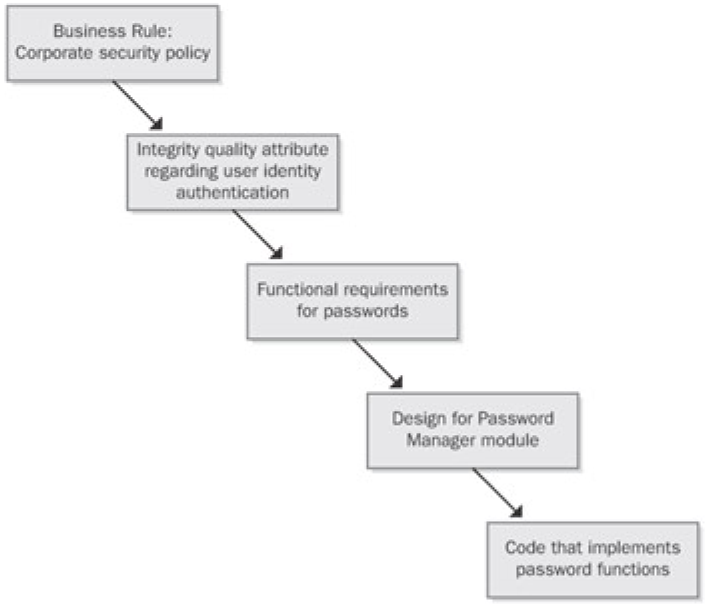

Business Rule: Corporate security policy
Integrity quality attribute regarding user identity authentication
Functional requirements for passwords
Design for Password Manager module
Code that implements password functions

Figure 20-3: Sample traceability chain for requirements dealing with application security.

Traceability links can define one-to-one, one-to-many, or many-to-many relationships between system elements. The format in Table 20-1 accommodates these cardinalities by letting you enter several items in each table cell. Here are some examples of the possible link cardinalities:

- **One-to-one** One design element is implemented in one code module.
- **One-to-many** One functional requirement is verified by multiple test cases.
- **Many-to-many** Each use case leads to multiple functional requirements, and certain functional requirements are common to several use cases. Similarly, a shared or repeated design element might satisfy a number of functional requirements. Ideally, you'll capture all these interconnections, but in practice, many-to-many traceability relationships can become complex and difficult to manage.

Another way to represent traceability information is through a set of matrices that define links between pairs of system elements, such as these:

- One type of requirement to other requirements of that same type
- One type of requirement to requirements of another type
- One type of requirement to test cases

You can use these matrices to define various relationships that are possible between pairs of requirements, such as "specifies/is specified by," "depends on," "is parent of," and "constrains/is constrained by" (Sommerville and Sawyer 1997).

Table 20-2 illustrates a two-way traceability matrix. Most cells in the matrix are empty. Each cell at the intersection of two linked components is marked to indicate the connection. You can use different symbols in the cells to explicitly indicate "traced-to" and "traced-from" or other relationships. Table 20-2 uses an arrow to indicate that a certain functional requirement is traced from a particular use case. These matrices are more amenable to automated tool support than is the single traceability table illustrated in

Table 20-1.

Table 20-2: Requirements Traceability Matrix Showing Links Between Use Cases and Functional Requirements

Functional Requirement	**Use Case**			
	UC-1	**UC-2**	**UC-3**	**UC-4**
FR-1	↵			
FR-2	↵			
FR-3			↵	
FR-4			↵	
FR-5		↵		↵
FR-6			↵	

Traceability links should be defined by whoever has the appropriate information available. Table 20-3 identifies some typical sources of knowledge about links between various types of source and target objects. Determine the roles and individuals who should supply each type of traceability information for your project. Expect some pushback from busy people whom the analyst or project manager asks to provide this data. Those practitioners are entitled to an explanation of what requirements tracing is, why it adds value, and why they're being asked to contribute to the process. Point out that the incremental cost of capturing traceability information at the time the work is done is small; it's primarily a matter of habit and discipline.

Trap Gathering and managing requirements traceability data must be made the explicit responsibility of certain individuals or it won't happen. Typically, a requirements analyst or a quality assurance engineer collects, stores, and reports on the traceability information.

Table 20-3: Likely Sources of Traceability Link Information

Link Source Object Type	**Link Target Object Type**	**Information Source**
System requirement	Software requirement	System engineer
Use case	Functional requirement	Requirements analyst
Functional requirement	Functional requirement	Requirements analyst
Functional requirement	Test case	Test engineer
Functional requirement	Software architecture element	Software architect
Functional requirement	Other design elements	Designer or Developer
Design element	Code	Developer
Business rule	Functional requirement	Requirements analyst

Team LiB PREVIOUS NEXT
Team LiB PREVIOUS NEXT

Tools for Requirements Tracing

Chapter 21, "Tools for Requirements Management," describes several commercial requirements management tools that have strong requirements-tracing capabilities. You can store requirements and other information in a tool's database and define links between the various types of stored objects, including peer links between two requirements of the same kind. Some tools let you differentiate "traced-to" and "traced-from" relationships, automatically defining the complementary links. That is, if you indicate that requirement R is traced to test case T, the tool will also show the symmetrical relationship in which T is traced from R.

Some tools automatically flag a link as *suspect* whenever the object on either end of the link is modified. A suspect link has a visual indicator (such as a red question mark or a diagonal red line) in the corresponding cell in the requirements traceability matrix. For example, if you changed use case 3, the requirements traceability matrix in Table 20-2 might look like Table 20-4 the next time you see it. The suspect link indicators (in this case, question marks) tell you to check whether functional requirements 3, 4, and 6 need to be changed to remain consistent with the modified UC-3. After making any necessary changes, you clear the suspect link indicators manually. This process helps ensure that you've accounted for the known ripple effects of a change.

True Stories The tools also let you define cross-project or cross-subsystem links. I know of one large software product that had 20 major subsystems, with certain high-level product requirements apportioned among multiple subsystems. In some cases, a requirement that was allocated to one subsystem was actually implemented through a service that another subsystem provided. This project used a requirements management tool to successfully track these complex traceability relationships.

Table 20-4: Suspect Links in a Requirements Traceability Matrix

Functional Requirement	**Use Case**			
	UC-1	**UC-2**	**UC-3**	**UC-4**
FR-1	↵			
FR-2	↵			
FR-3			?↵	
FR-4			?↵	
FR-5		↵		↵
FR-6			?↵	

It's impossible to perform requirements tracing manually for any but very small applications. You can use a spreadsheet to maintain traceability data for up to a couple hundred requirements, but larger systems demand a more robust solution. Requirements tracing can't be fully automated because the knowledge of the links originates in the development team members' minds. However, once you've identified the links, tools can help you manage the vast quantity of traceability information.

Team LiB ◀ PREVIOUS NEXT ▶
Team LiB ◀ PREVIOUS NEXT ▶

Requirements Traceability Procedure

Consider following this sequence of steps when you begin to implement requirements traceability on a specific project:

1. Select the link relationships you want to define from the possibilities shown in Figure 20-2.
2. Choose the type of traceability matrix you want to use: the single matrix shown in Table 20-1 or several of the matrices illustrated in Table 20-2. Select a mechanism for storing the data: a table in a text document, a spreadsheet, or a requirements management tool.
3. Identify the parts of the product for which you want to maintain traceability information. Start with the critical core functions, the high-risk portions, or the portions that you expect to undergo the most maintenance and evolution over the product's life.
4. Modify your development procedures and checklists to remind developers to update the links after implementing a requirement or an approved change. The traceability data should be updated as soon as someone completes a task that creates or changes a link in the requirements chain.
5. Define the tagging conventions you will use to uniquely identify all system elements so that they can be linked together (Song et al. 1998). If necessary, write scripts that will parse the system files to construct and update the traceability matrices.
6. Identify the individuals who will supply each type of link information and the person who will coordinate the traceability activities and manage the data.
7. Educate the team about the concepts and importance of requirements tracing, your objectives for this activity, where the traceability data is stored, and the techniques for defining the links (for example, by using the traceability feature of a requirements management tool). Make sure all participants commit to their responsibilities.
8. As development proceeds, have each participant provide the requested traceability information as they complete small bodies of work. Stress the need for ongoing creation of the traceability data, rather than for attempts to reconstruct it at a major milestone or at the end of the project.
9. Audit the traceability information periodically to make sure it's being kept current. If a requirement is reported as implemented and verified yet its traceability data is incomplete or inaccurate, your requirements tracing process isn't working as intended.

I've described this procedure as though you were starting to collect traceability information at the outset of a new project. If you're maintaining a legacy system, the odds are good that you don't have traceability data available, but there's no time like the present to begin accumulating this useful information. The next time you add an enhancement or make a modification, write down what you discover about connections between code, tests, designs, and requirements. Build the recording of traceability data into your procedure for modifying an existing software component. You'll never reconstruct a complete requirements traceability matrix, but this small amount of effort might make it easier the next time someone needs to work on that same part of the system.

Team LiB PREVIOUS NEXT
Team LiB PREVIOUS NEXT

Is Requirements Traceability Feasible? Is It Necessary?

True Stories You might conclude that creating a requirements traceability matrix is more expensive than it's worth or that it's not feasible for your big project, but consider the following counterexample. A conference attendee who worked at an aircraft manufacturer told me that the SRS for his team's part of the company's latest jetliner was a stack of paper six feet thick. They had a complete requirements traceability matrix. I've flown on that very model of airplane, and I was happy to hear that the developers had managed their software requirements so carefully. Managing traceability on a huge product with many interrelated subsystems is a lot of work. This aircraft manufacturer knows it is essential; the Federal Aviation Administration agrees.

True Stories Even if your products won't cause loss to life or limb if they fail, you should take requirements tracing seriously. The CEO of a major corporation who was present when I described traceability at a seminar asked, "Why *wouldn't* you create a requirements traceability matrix for your strategic business systems?" That's an excellent question. You should decide to use any improved requirements engineering practice based on both the costs of applying the technique and the risks of *not* using it. As with all software processes, make an economic decision to invest your valuable time where you expect the greatest payback.

Next Steps

- Set up a traceability matrix for 15 or 20 requirements from an important portion of the system you're currently developing. Try the approaches shown in both Table 20-1 and Table 20-2. Populate the matrix as development 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 your reverse-engineering analysis of the part of the product you are modifying. Build a fragment of a 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 traceability matrix as your team continues to maintain the product.

Team LiB ◄ PREVIOUS NEXT ►
Team LiB ◄ PREVIOUS NEXT ►

Chapter 21: Tools for Requirements Management

Overview

In earlier chapters, I discussed the creation of a natural-language software requirements specification to contain the functional and nonfunctional requirements and the creation of documents that contain the business requirements and use-case descriptions. A document-based approach to storing requirements has numerous limitations, including the following:

- It's difficult to keep the documents current and synchronized.

- Communicating changes to all affected team members is a manual process.
- It's not easy to store supplementary information (attributes) about each requirement.
- It's hard to define links between functional requirements and other system elements.
- Tracking requirements status is cumbersome.
- Concurrently managing sets of requirements that are planned for different releases or for related products is difficult. When a requirement is deferred from one release to another, an analyst needs to move it from one requirements specification to the other.
- Reusing a requirement means that the analyst must copy the text from the original SRS into the SRS for each other system or product where the requirement is to be used.
- It's difficult for multiple project participants to modify the requirements, particularly if the participants are geographically separated.
- There's no convenient place to store proposed requirements that were rejected and requirements that were deleted from a baseline.

A requirements management tool that stores information in a multiuser database provides a robust solution to these restrictions. Small projects can use spreadsheets or simple databases to manage their requirements, storing both the requirements text and several attributes of each requirement. Larger projects will benefit from commercial requirements management tools. Such products let users import requirements from source documents, define attribute values, filter and display the database contents, export requirements in various formats, define traceability links, and connect requirements to items stored in other software development tools.

Trap Avoid the temptation to develop your own requirements management tool or to cobble together general-purpose office automation products in an attempt to mimic the commercial products. This initially looks like an easy solution, but it can quickly overwhelm a team that doesn't have the resources to build the tool it really wants.

Note that I classify these products as requirements *management* tools, not requirements *development* tools. They won't help you identify your prospective users or gather the right requirements for your project. However, they provide a lot of flexibility in managing changes to those requirements and using the requirements as the foundation for design, testing, and project management. These tools don't replace a defined process that your team members follow to elicit and manage its requirements. Use a tool when you already have an approach that works but that requires greater efficiency; don't expect a tool to compensate for a lack of process, discipline, experience, or understanding.

This chapter presents several benefits of using a requirements management tool and identifies some general capabilities you can expect to find in such a product. Table 21-1 lists several of the requirements management tools presently available. This chapter doesn't contain a feature-by-feature tool comparison because these products are still evolving and their capabilities change with each release. The prices, supported platforms, and even vendors of software development tools also change frequently, so use the Web addresses in Table 21-1 to get current information about the products (recognizing that Web addresses themselves are subject to change if, say, one tool vendor acquires another, as happened two weeks prior to this writing). You can find detailed feature comparisons of these and many other tools at the Web site for the International Council on Systems Engineering (*http://www.incose.org/toc.html*), along with guidance on how to select a requirements management tool (Jones et al. 1995).

Table 21-1: Some Commercial Requirements Management Tools

Tool	**Vendor**	**Database- or Document-Centric**
Active! Focus	Xapware Technologies, http://www.xapware.com	Database
CaliberRM	Borland Software Corporation, http://www.borland.com	Database
C.A.R.E.	SOPHIST Group, http://www.sophist.de	Database
DOORS	Telelogic, http://www.telelogic.com	Database
RequisitePro	Rational Software Corporation, http://www.rational.com	Document
RMTrak	RBC, Inc., http://www2.rbccorp.com	Document
RTM Workshop	Integrated Chipware, Inc., http://www.chipware.com	Database
Slate	EDS, http://www.eds.com	Database
Vital Link	Compliance Automation, Inc., http://www.complianceautomation.com	Document

One distinction between the tools is whether they are database-centric or document-centric. Database-centric products store all requirements, attributes, and traceability information in a database. Depending on the product, the database is either commercial or proprietary, relational or object-oriented. Requirements can be imported from various source documents, but they then reside in the database. In most respects, the textual description of a requirement is treated simply as a required attribute. Some products let you link individual requirements to external files (such as Microsoft Word files, Microsoft Excel files, graphics files, and so on) that provide supplementary information to augment the contents of the requirements repository.

The document-centric approach treats a document created using a word-processing program (such as Microsoft Word or Adobe FrameMaker) as the primary container for the requirements. RequisitePro lets you select text strings in a Word document to be stored as discrete requirements in a database. Once the requirements are in the database, you can define attributes and traceability links, just as you can with the database-centric products. Mechanisms are provided to synchronize the database and document contents. RTM Workshop straddles both paradigms by being primarily database-centric but also letting you maintain requirements in a Microsoft Word document.

These tools aren't cheap, but the high cost of requirements-related problems can justify your investment in them. Recognize that the cost of a tool is not simply what you pay for the initial license. The cost also includes the host computer, annual maintenance fees and periodic upgrades, and the costs of installing the software, performing administration, obtaining vendor support and consulting, and training your users. Your cost-benefit analysis should take into account these additional expenses before you make a purchase decision.

Team LiB ◀ PREVIOUS NEXT ▶
Team LiB ◀ PREVIOUS NEXT ▶

Benefits of Using a Requirements Management Tool

Even if you do a magnificent job gathering your project's requirements, automated assistance can help you work with these requirements as development progresses. A requirements management tool becomes most beneficial as time passes and the team's memory of the requirements details fades. The following sections describe some of the tasks such a tool can help you perform.

Manage versions and changes Your project should define a requirements baseline, a specific collection of requirements allocated to a particular release. Some requirements management tools provide flexible baselining functions. The tools also maintain a history of the changes made to every requirement. You can record the rationale behind each change decision and revert to a previous version of a requirement if necessary. Some of the tools, including Active! Focus and DOORS, contain a simple, built-in change-proposal system that links change requests directly to the affected requirements.

Store requirements attributes You should record several descriptive attributes for each requirement, as discussed in Chapter 18. Everyone working on the project must be able to view the attributes, and selected individuals will be permitted to update attribute values. Requirements management tools generate several system-defined attributes, such as the date a requirement was created and its current version number, and they let you define additional attributes of various data types. Thoughtful definition of attributes allows stakeholders to view subsets of the requirements based on specific combinations of attribute values. You might ask to see a list of all the requirements originating from a specific business rule so that you can judge the consequences of a change in that rule. One way to keep track of the requirements that are allocated to the baselines for various releases is by using a Release Number attribute.

Facilitate impact analysis The tools enable requirements tracing by letting you define links between different types of requirements, between requirements in different subsystems, and between individual requirements and related system components (for example, designs, code modules, tests, and user documentation). These links help you analyze the impact that a proposed change will have on a specific requirement by identifying other system elements the change might affect. It's also a good idea to trace each functional requirement back to its origin or parent so that you know where every requirement came from.

More Info Chapter 19 describes impact analysis, and Chapter 20 addresses requirements tracing.

Track requirements status Collecting requirements in a database lets you know how many discrete requirements you've specified for the product. Tracking the status of each requirement during development supports the overall status tracking of the project. A project manager has good insight into project status if he knows that 55 percent of the requirements committed to the next release have been verified, 28 percent have been implemented but not verified, and 17 percent are not yet fully implemented.

Control access The requirements management tools let you define access permissions for individuals or groups of users and share information with a geographically dispersed team through a Web interface to the database. The databases use requirement-level locking to permit multiple users to update the database contents concurrently.

Communicate with stakeholders Some tools permit team members to discuss requirements issues electronically through threaded conversations. Automatically triggered e-mail messages notify affected individuals when a new discussion entry is made or when a specific requirement is modified. Making the requirements accessible on line can save travel costs and reduce document proliferation.

Reuse requirements Storing requirements in a database facilitates reusing them in multiple projects or subprojects. Requirements that logically fit into multiple parts of the product description can be stored once and referenced whenever necessary to avoid duplicating requirements.

Team LiB ◀ PREVIOUS NEXT ▶
Team LiB ◀ PREVIOUS NEXT ▶

Requirements Management Tool Capabilities

Commercial requirements management tools let you define different requirement types (or classes), such as business requirements, use cases, functional requirements, hardware requirements, and constraints. This lets you differentiate individual objects that you want to treat as requirements from other useful information contained in the SRS. All the tools provide strong capabilities for defining attributes for each requirement type, which is a great advantage over the typical document-based SRS approach.

Most requirements management tools integrate with Microsoft Word to some degree, typically adding a tool-specific menu to the Microsoft Word menu bar. Vital Link is based on Adobe FrameMaker, and Slate integrates with both FrameMaker and Word. The higher-end tools support a rich variety of import and export file formats. Several of the tools let you mark text in a Word document to be treated as a discrete requirement. The tool highlights the requirement and inserts Word bookmarks and hidden text into the document. The tools can also parse documents in various fashions to extract individual requirements. The parsing from a word-processed document will be imperfect unless you were diligent about using text styles or keywords such as "shall" when you created the document.

The tools support hierarchical numeric requirement labels, in addition to maintaining a unique internal identifier for each requirement. These identifiers typically consist of a short text prefix that indicates the requirement type—such as UR for a user requirement—followed by a unique integer. Some tools provide efficient Microsoft Windows Explorer–like displays to let you manipulate the hierarchical requirements tree. One view of the requirements display in DOORS looks like a hierarchically structured SRS.

Output capabilities from the tools include the ability to generate a requirements document, either in a user-specified format or as a tabular report. CaliberRM has a powerful "Document Factory" feature that lets you define an SRS template in Word by using simple directives to indicate page layout, boilerplate text, attributes to extract from the database, and the text styles to use. The Document Factory populates this template with information it selects from the database according to user-defined query criteria to produce a customized specification document. An SRS, therefore, is essentially a report generated from selected database contents.

All the tools have robust traceability features. For example, in RTM Workshop, each project defines a class schema resembling an entity-relationship diagram for all the stored object types. Traceability is handled by defining links between objects in two classes (or within the same class), based on the class relationships defined in the schema.

Other features include the ability to set up user groups and define permissions for selected users or groups to create, read, update, and delete projects, requirements, attributes, and attribute values. Several of the products let you incorporate nontextual objects such as graphics and spreadsheets into the requirements repository. The tools also include learning aids, such as tutorials or sample projects, to help users get up to speed.

These products show a trend toward increasing integration with other tools used in application development, as illustrated in Figure 21-1. When you select a requirements management product, determine whether it can exchange data with the other tools you use. Here are just a few examples of the tool interconnections that these products exhibit today:

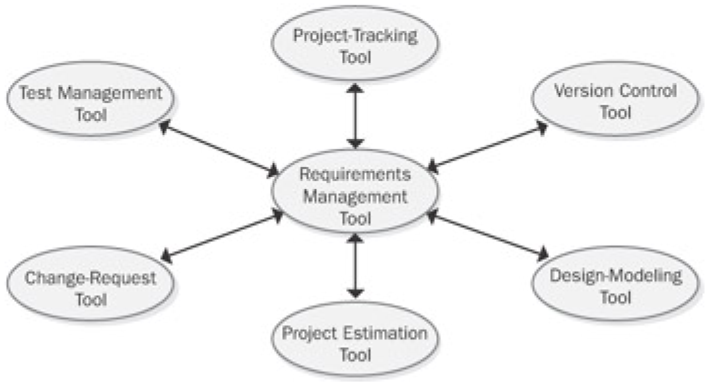


Figure 21-1: Requirements management tools integrate with other kinds of software tools.

- You can link requirements in RequisitePro to use cases modeled in Rational Rose and to test cases stored in Rational TeamTest.
- DOORS lets you trace requirements to individual design elements stored in Rational Rose, Telelogic Tau, and other design modeling tools.
- RequisitePro and DOORS can link individual requirements to project tasks in Microsoft Project.
- CaliberRM has a central communications framework that lets you link requirements to use-case, class, or process design elements stored in TogetherSoft Control Center, to source code stored in Borland's StarTeam, and to test elements stored in Mercury Interactive's TestDirector. You can then access those linked elements directly from the requirements stored in CaliberRM's database.

When evaluating tools, think about how you'll take advantage of these product integrations as you perform your requirements engineering, testing, project tracking, and other processes. For example, consider how you would publish a baselined set of requirements into a version control tool and how you would define traceability links between functional requirements and specific design or code elements.

Team LiB PREVIOUS NEXT
Team LiB PREVIOUS NEXT

Implementing Requirements Management Automation

Any of these products will move your requirements management practices to a higher plane of sophistication and capability. However, the diligence of the tools' users remains a critical success factor. Dedicated, disciplined, and knowledgeable people will make progress even with mediocre tools, whereas the best tools won't pay for themselves in the hands of unmotivated or ill-trained users. Don't write a check for a requirements management tool unless you're willing to respect the learning curve and make the time investment. Because you can't expect instantaneous results, don't base a project's success on a tool you're using for the first time. Gain some experience working with the tool on a pilot project before you employ it on a high-stakes project.

More Info Chapter 22, "Improving Your Requirements Processes," describes the learning curve associated with adopting new tools and techniques.

Selecting a Tool

Select a tool based on the combination of platform, pricing, access modes, and requirements paradigm—document-centric or database-centric—that best fits your development environment and culture. Some companies contract tool evaluations to consultants, who can assess a company's needs comprehensively and make recommendations from the available tool candidates. If you do the evaluation yourself, the following procedure can help you select the right tool:

1. First, define your organization's requirements for a requirements management tool. Identify the capabilities that are most significant to you, the other tools with which you'd like the product to integrate, and whether issues such as remote data access through the Web are important. Decide whether you want to continue using documents to contain some of your requirements information or whether you prefer to store all the information in a database.

2. List 10 to 15 factors that will influence your selection decision. Include subjective categories such as tailorability, as well as the efficiency and effectiveness of the user interface. Cost will be a selection factor, but evaluate the tools initially without considering their cost.

3. Distribute 100 points among the selection factors that you listed in step 2, giving more points to the more important factors.

4. Obtain current information about the available requirements management tools, and rate the candidates against each of your selection factors. Scores for the subjective factors will have to wait until you can actually work with each tool. A vendor demonstration can fill in some of the blanks, but the demo will likely be biased toward the tool's strengths. A demo isn't a substitute for using the product yourself for several hours.

5. Calculate the score for each candidate based on the weight you gave each factor to see which products appear to best fit your needs.

6. Solicit experience reports from other users of each candidate product, perhaps by posting queries in online discussion forums, to supplement your own evaluation and the vendor's literature, demo, and sales pitch.

7. Obtain evaluation copies from the vendors of your top-rated tools. Define an evaluation process before you install the candidates to make sure you get the information you need to make a good decision.

8. Evaluate the tools by using a real project, not just the tutorial project that comes with the product. After you complete your evaluations, adjust your rating scores if necessary and see which tool now ranks highest.

9. To make a decision, combine the ratings, licensing costs, and ongoing costs with information on vendor support, input from current users, and your team's subjective impressions of the products.

Changing the Culture

Buying a tool is easy; changing your culture and processes to accept the tool and take best advantage of it is much harder. Most organizations already have a comfort level with storing their requirements in word-processing documents. Changing to an online approach requires a different way of thinking. A tool makes the requirements visible to any stakeholder who has access to the database. Some stakeholders interpret this visibility as reducing the control they have over the requirements, the requirements-engineering process, or both. Some people prefer not to share an incomplete or imperfect SRS with the world, yet the database contents are there for all to see. Keeping the requirements private until they're "done" means you miss an opportunity to have many pairs of eyes scan the requirements frequently for possible problems.

True Stories There's little point in using a requirements management tool if you don't take advantage of its capabilities. I encountered one project team that had diligently stored all its requirements in a commercial tool but hadn't defined any requirements attributes or traceability links. Nor did they provide online access for all the stakeholders. The fact that the requirements were stored in a different form didn't really provide significant benefits, although it consumed the effort needed to get the requirements into the tool. Another team stored hundreds of requirements in a tool and defined many traceability links. Their only use of the information was to generate massive printed traceability reports that were supposed to be reviewed manually for problems. No one really studied the reports, and no one regarded the database as the authoritative repository of the project's requirements. Neither of these organizations reaped the full benefits of their considerable investments of time and money in the requirements management tools.

Consider the following cultural and process issues as you strive to maximize your return on investment from a commercial requirements management tool:

- Don't even pilot the use of a tool until your organization can create a reasonable software requirements specification on paper. If your biggest problems are with gathering and writing clear, high-quality requirements, these tools won't help you.
- Don't try to capture requirements directly in the tool during the early elicitation workshops. As the requirements begin to stabilize, though, storing them in the tool makes them visible to the workshop participants for refinement.
- Use the tool as a groupware support aid to facilitate communication with project stakeholders in various locations. Set the access and change privileges to permit sufficient input to the requirements by various people without giving everyone complete freedom to change everything in the database.
- Think carefully about the various requirement types that you define. Don't treat every section of your current SRS template as a separate requirement type, but don't simply stuff all the SRS contents into a single requirement type either. The tools let you create different attributes for each requirement type you define, so selecting appropriate attributes will help you determine how many different types of requirement to define.
- Define an owner for each requirement type, who will have the primary responsibility for managing the database contents of that type.
- Use business, not IT, terminology when defining new data fields or requirements attributes.
- Don't define traceability links until the requirements stabilize. Otherwise, you can count on doing a lot of work to revise the links as requirements continue to change.
- To accelerate the movement from a document-based paradigm to the use of the tool, set a date after which the tool's database will be regarded as the definitive repository of the project's requirements.

After that date, requirements residing only in word-processing documents won't be recognized as valid requirements.

- Instead of expecting to freeze the requirements early in the project, get in the habit of baselining a set of requirements for a particular release. Dynamically shift requirements from the baseline for one release to another as necessary.

  More Info Chapter 18 discussed using requirement attributes to manage requirements destined for different releases.

As the use of a requirements management tool becomes ingrained in your culture, project stakeholders will begin to regard requirements as life cycle assets, just as they do code. The team will discover ways to use the tool to speed up the process of documenting requirements, communicating them, and managing changes to them. Remember, though: even the best tool can't compensate for an ineffective requirements development process. The tool won't help you scope your project, identify users, talk to the right users, or collect the right requirements. And it doesn't matter how well you manage poor requirements.

Making Requirements Management Tools Work for You

For the smoothest transition, assign 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. 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. The initial tool advocate will manage its use in the pilot, and then he'll train and mentor others to support the tool as other projects adopt it. 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.

Recognize that it will take effort to load a project's requirements into the database, define attributes and traceability links, keep the database's contents current, define access groups and their privileges, and train users. Management must allocate the resources needed for these operations. Make an organization-wide commitment to actually use the product you select, instead of letting it become expensive shelfware.

Provided you remember that a tool can't overcome process deficiencies, you're likely to find that commercial requirements management tools enhance the control you have over your software requirements. Once you've made a requirements database work for you, you'll never go back to plain paper.

Next Steps

- Analyze shortcomings in your current requirements management process to see whether a 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 correct them.

- Before launching a comparative evaluation, assess your team'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.

PREVIOUS NEXT