CHAPTER 27

Requirements management practices

"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 in disgust. "These documents should be identical, but obviously they're not. So, is this feature still needed, or did I just waste 30 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 specifications. Having great requirements gets you only partway to a solution; they also have to be well managed and effectively communicated among the project participants. Version control of individual requirements and sets of requirements is one of the core activities of requirements management.

Chapter 1, "The essential software requirement," divided the domain of software requirements engineering into requirements development and requirements management. (Some people refer to the entire domain as "requirements management," but we favor a narrower definition of that term.) This chapter addresses some principles and practices of requirements management. The other chapters in Part IV describe certain requirements management practices in more detail, including change control (Chapter 28, "Change happens"), change impact analysis (also Chapter 28), and requirements tracing (Chapter 29, "Links in the requirements chain"). Part IV concludes with a discussion of commercial tools that can help a project team develop and manage its requirements (Chapter 30, "Tools for requirements engineering"). Note that a project might be managing certain sets of agreed-upon requirements while concurrently performing requirements development activities on other portions of the product's requirements.

Requirements management process

Requirements management includes all activities that maintain the integrity, accuracy, and currency of requirements agreements throughout the project. Figure 27-1 shows the core activities of requirements management in four major categories: version control, change control, requirements status tracking, and requirements tracing.

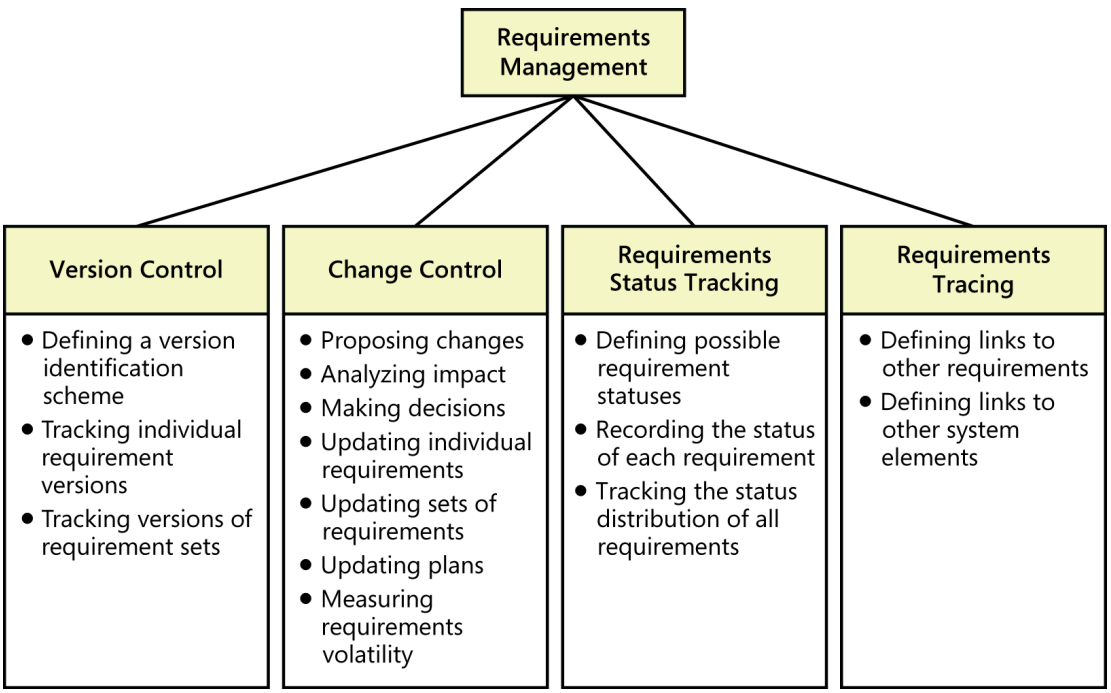

FIGURE 27-1 Major requirements management activities.

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 performance enables the members of the organization to conduct them consistently and effectively. Consider addressing the following topics:

- Tools, techniques, and conventions for distinguishing versions of individual requirements and of requirements sets
- The way that sets of requirements are approved and baselined (see Chapter 2, "Requirements from the customer's perspective")
- The ways that new requirements and changes to existing ones are proposed, evaluated, negotiated, and communicated
- How to assess the impact of a proposed change
- Requirement attributes and requirements status-tracking procedures, including the requirement statuses that you will use and who can change them
- Who is responsible for updating requirements trace information and when

- How to track and resolve requirements issues
- How the project's plans and commitments will reflect requirements changes
- How to use the requirements management (RM) tool effectively

You can include all this information in a single requirements management process description. Alternatively, you might prefer to write separate version control, change control, impact analysis, and status tracking procedures. These procedures should apply across your organization because they represent common functions that every project team ought to perform. Chapter 31, "Improving your requirements processes," describes several useful process assets for requirements management.

Your process descriptions should identify the team role that owns each of the requirements management activities. The project's business analyst typically has the lead responsibility for requirements management. The BA will set up the requirements storage mechanisms, define requirement attributes, coordinate requirement status and trace data updates, and monitor change activity as needed. The process description should also indicate who has authority to modify the requirements management process, how exceptions should be handled, and the escalation path for impediments encountered.

> **Trap** If no one on the project has responsibility for performing requirements management activities, don't expect them to get done. Similarly, if "everyone" has the responsibility, each person might expect that someone else is covering the necessary activities, so they can easily be overlooked.

The requirements baseline

Requirements development involves activities to elicit, analyze, specify, and validate a software project's requirements. Requirements development deliverables include business requirements, user requirements, functional and nonfunctional requirements, a data dictionary, and various analysis models. After they are reviewed and approved, any defined subset of these items constitutes a requirements baseline. As was described in Chapter 2, a requirements *baseline* is a set of requirements that stakeholders have agreed to, often defining the contents of a specific planned release or development iteration. The project might have additional agreements regarding deliverables, constraints, schedules, budgets, transition requirements, and contracts; those lie beyond the scope of this book.

At the time a set of requirements is baselined—typically following review and approval—the requirements are placed under configuration (or change) management. Subsequent changes can be made only through the project's defined change control procedure. Prior to baselining, the requirements are still evolving, so there's no point in imposing unnecessary process overhead on those modifications. A baseline could consist of some or all the requirements in a particular SRS (whether for an entire product or a single release), or a designated set of requirements stored in an RM tool, or an agreed-on set of user stories for a single iteration on an agile project.

If the scope of a release changes, update the requirements baseline accordingly. Distinguish the requirements in a particular baseline from others that were proposed but not accepted, are allocated to a different baseline, or remain unallocated in the product backlog. If the requirements are specified in the form of a document such as an SRS, clearly identify it as a baseline version to distinguish it from prior drafts. Storing requirements in an RM tool facilitates the identification of those that belong to a specific baseline and the management of changes to that baseline.

A development team that accepts proposed requirement changes or additions 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 accommodate new or changed requirements in various ways:

- By deferring lower-priority requirements to later iterations or cutting them completely
- By obtaining additional staff or outsourcing some of the work
- By extending the delivery schedule or adding iterations to an agile project
- By sacrificing quality to ship by the original date

No single approach is universally correct, because projects differ in their flexibility of features, staff, budget, schedule, and quality (Wiegers 1996). The choice should be based on the project's business objectives and the priorities the key stakeholders established during project initiation. 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 budget overruns, team member burnout, or quality compromises.

Requirements version control

Version control—uniquely identifying different versions of an item—applies at the level of both individual requirements and requirements sets, most commonly represented in the form of documents. Begin version control as soon as you draft a requirement or a document so you can retain a history of changes made.

Every version of the requirements must be uniquely identified. Every team member must be able to access the current version of the requirements. Changes must be clearly documented and communicated to everyone affected. To minimize confusion and miscommunication, permit only designated individuals to update the requirements, and make sure that the version identifier changes whenever an update is made. Each circulated version of a requirements document or each requirement in a tool 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.

It's not a bug; it's a feature!

A contract development team received a flood of bug reports from the testers of the latest release they had just delivered to a customer. The contract team was perplexed—the system had passed all their own tests. After considerable investigation, it turned out that the customer was testing the new software against an obsolete version of the SRS. What the testers were reporting as bugs truly were features. Normally, this is just a little joke that software people like to make. The testers spent considerable time rewriting the tests against the correct version of the SRS and retesting the application, all because of a version control problem. Another colleague who once experienced the same kind of testing confusion because of an uncommunicated change said, "We probably wasted four to six hours of effort that our department had to absorb and couldn't spend on actual billable hours. I think software professionals would be shocked if they multiplied out these wasted hours times their bill rate to see what the loss in revenue is."

Similar confusion can arise when multiple BAs are working on a project. One BA begins to edit version 1.2 of the requirements specification. A few days later, another BA starts to work on some requirements and also labels his version 1.2, not knowing about the conflict. Pretty soon changes are lost, requirements are no longer up to date, work is overwritten, and confusion ensues.

The most robust approach to version control is to store the requirements in a requirements management tool, as described in Chapter 30. RM tools track the history of changes made to every requirement, which is valuable when you need to revert to an earlier version. 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.

If you're storing requirements in documents, 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 and underscores for additions. When you baseline a document, first archive a marked-up version, then accept all the revisions, and then store the now clean version as the new baseline, ready for the next round of changes. Store requirements documents in a version control tool, such as the one your organization uses for controlling source code through check-out and check-in procedures. This will let you revert to earlier versions if necessary and to know who changed each document, when, and why. (Incidentally, this describes exactly how we wrote this book. We wrote the chapters in Microsoft Word, using revision marks as we iterated on the chapters. We had to refer back to previous versions on several occasions.)

I know of one project that stored several hundred use case documents written in Microsoft Word in 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 BA and her backup person had read-write access to the documents stored in the tool; the other team members had read-only access. This approach worked well for this team.

The simplest version control mechanism is to manually label each revision of a document according to a standard convention. Schemes that try to differentiate document versions based on dates are prone to confusion. I use a convention that labels the first version of any new document with its title and "Version 1.0 draft 1." The next draft keeps the same title but is identified as " 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 version identifier is changed to "Version 1.0 approved," again keeping the same document title. The next version is 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 and depend on the context.) This scheme clearly distinguishes between draft and baselined document versions, but it does require manual discipline on the part of those who modify the documents.

Requirement attributes

Think of each requirement as an object with properties that distinguish it from other requirements. In addition to its textual description, each requirement should have supporting pieces of information or *attributes* associated with it. These attributes establish a context and background for each requirement. You can store attribute values in a document, a spreadsheet, a database, or—most effectively—a requirements management tool. It's cumbersome to use more than a couple of requirements attributes with documents.

RM tools typically provide several system-generated attributes in addition to letting you define others, some of which can be automatically populated. The tools let you 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 have a status of Approved. Following is a list of potential requirement attributes to consider:

- Date the requirement was created
- Current version number of the requirement
- Author who wrote the requirement
- Priority
- Status
- Origin or source of the requirement
- Rationale behind the requirement
- Release number or iteration to which the requirement is allocated
- Stakeholder to contact with questions or to make decisions about proposed changes
- Validation method to be used or acceptance criteria

Wherefore this requirement?

The product manager at a company that makes electronic measurement devices wanted to track which requirements the team included simply because a competitor's product had the same capability. A good way to note such features is with a Rationale attribute, which indicates why a specific requirement is included in the product. Suppose you included some requirement because it meets the need of a particular user group. Later on, your marketing department decides they don't care about that user group any more. Having the justification present as a requirement attribute would help people decide whether that requirement could be omitted.

Another BA described his quandary with requirements that had no obvious justification. He said, "In my experience, many requirements exist without a real need behind them. They are introduced because the customer lacks an understanding of the technology, or because some key stakeholders get excited about the technology and want to show off, or because our sales team intentionally or unintentionally has misled the customer." If you can't provide a convincing rationale for a requirement and trace it back to a business need, the BA should question whether there's a real reason to devote effort to it.

Trap Selecting too many requirements attributes can overwhelm a team. They won't supply all attribute values for all requirements and won't use the attribute information effectively. Start with perhaps three or four key attributes. Add others only when you know how they will add value.

The requirements planned for a release will change as new requirements are added and existing ones are deleted or deferred. The team might be juggling separate requirements documents for multiple releases or iterations. Leaving obsolete requirements in the SRS can confuse readers as to whether those requirements are included in that baseline. A solution is to store the requirements in an RM 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. Handle deleted and rejected requirements by using a status attribute, as described in the next section.

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 3 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. The larger the project, the easier it is to experience time-wasting miscommunications.

Tracking requirements status

"How are you coming on implementing that subsystem, Yvette?" asked the project manager, Dave.

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

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

Yvette replied, "Yes, I thought I was, but now I'm really *90 percent done."*

Like nearly everyone, software developers are sometimes overly optimistic when they report how much of a task is complete. The common "90 percent done" syndrome doesn't tell Dave much about how close Yvette really is to finishing the subsystem. But suppose Yvette had replied, "Pretty good, Dave. Of the 84 requirements for the subsystem, 61 are implemented and verified, 14 are implemented but not yet verified, and I haven't implemented the other 9 yet." Tracking the status of each functional requirement throughout development provides a more precise gauge of project progress.

Status was one of the requirement attributes proposed in the previous section. Tracking status means comparing where you really are at a particular time against the expectation of what "complete" means for this development cycle. You might have planned to implement only certain flows of a use case in the current release, leaving full implementation for a future release. Monitor the status of just those functional requirements that were committed for the current release, because that's the set that's supposed to be 100 percent done before you declare success and ship 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 27-1 lists several possible requirement statuses. Some practitioners add others, such as Designed (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 acceptance or beta testing). It's valuable to keep a record of rejected requirements and the reasons they were rejected. Rejected requirements have a way of resurfacing later during development or on a future project. 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. You don't need to monitor all of the possible statuses in Table 27-1; choose the ones that add value to your requirements activities.

TABLE 27-1 Suggested requirement statuses

Status	Definition
Proposed	The requirement has been requested by an authorized source.
In Progress	A business analyst is actively working on crafting the requirement.
Drafted	The initial version of the requirement has been written.
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. The software that implemented the requirement is now ready for testing, review, or other verification.
Verified	The requirement has satisfied its acceptance criteria, meaning that the correct functioning of the implemented requirement has been confirmed. The requirement has been traced to pertinent tests. It is now considered complete.
Deferred	An approved requirement is now planned for implementation in a later release.
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 was never approved and is not planned for implementation in any upcoming release. Include an explanation of why and by whom the decision was made to reject it.

Classifying requirements into several status categories is more meaningful than trying to monitor the percent completion of each requirement or of the complete release baseline. Update a requirement's status only when specified transition conditions are satisfied. Certain status changes also require updates to the requirements trace data to indicate which design, code, and test elements addressed the requirement, as illustrated in Table 29-1 in Chapter 29.

Figure 27-2 illustrates how you can visually monitor the status of a set of requirements 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. Tracking the distribution by percentages doesn't show whether the number of requirements in the baseline is changing over time. The number of requirements increases as scope is added and decreases when functionality is removed from the baseline. 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 requirements allocated to it have a status of Verified, Deleted, or Deferred.

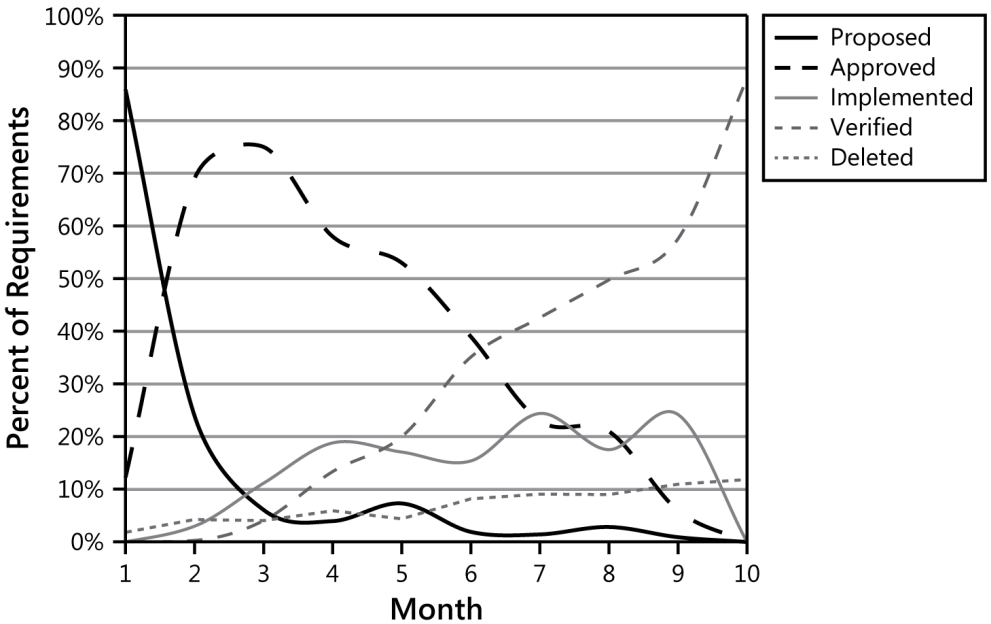

FIGURE 27-2 Tracking the distribution of requirements status throughout a project's development cycle.

Resolving requirements issues

Numerous questions, decisions, and issues related to requirements will arise during the course of a project. Potential issues include items flagged as TBD, pending decisions, information that is needed, and conflicts awaiting resolution. It's easy to lose sight of these open issues. Record issues in an issue-tracking tool so all affected stakeholders have access to them. Keep the issue-tracking and resolution process simple to ensure that nothing slips through the cracks. Some of the benefits of using an issue-tracking tool are:

- Issues from multiple requirements reviews are collected so that no issue ever gets lost.
- The project manager can easily see the current status of all issues.
- A single owner can be assigned to each issue.
- The history of discussion around an issue can be retained.
- The team can begin development earlier with a known set of open issues rather than having to wait until the SRS is complete.

Resolve requirements issues so they don't impede the timely baselining of a high-quality requirements set for your next release or iteration. A burndown chart that shows remaining issues and the rate at which they are being closed can help predict when all of the issues will be closed so you can accelerate issue resolution if necessary. (See "Managing requirements on agile projects" later in this chapter for a sample burndown chart.) Categorizing issues will help you determine which sections of requirements still need work. Few open issues on a section could mean either that the requirements haven't been reviewed yet or that the open issues are mostly resolved.

Nearly all of the defects logged early in a project are related to issues in the requirements, such as asking for clarification on a requirement, scope decisions, questions about development feasibility, and to-do items on the requirements themselves. All stakeholders can log questions as they review the requirements. Table 27-2 lists several common types of requirements issues that can arise.

TABLE 27-2 Common types of requirements issues

Issue type	Description
Requirement question	Something isn't understood or decided about a requirement.
Missing requirement	Developers uncovered a missed requirement during design or implementation.
Incorrect requirement	A requirement was wrong. It should be corrected or removed.
Implementation question	As developers implement requirements, they have questions about how something should work or about design alternatives.
Duplicate requirement	Two or more equivalent requirements are discovered. Delete all but one of them.
Unneeded requirement	A requirement simply isn't needed anymore.

Bad things can happen if you don't have an organized process for handling your requirements issues. On one project, a stakeholder mentioned very early on that we would handle something in "the portal." This was the first I had heard of a portal as part of the solution, so I asked about it. The stakeholder assured me that the COTS package being acquired included a portal component that simply had to be configured properly. We hadn't included any time for portal requirements in our plan, so I thought we might have a gap. I asked a teammate to record an issue about the portal so we wouldn't overlook that need. I left the project a few weeks later.

As it turned out, my teammate jotted the portal issue on a whiteboard that was later erased; she didn't record it in our issue-tracking tool. Six months into the project, our executive stakeholder came to me absolutely furious that no one had elicited requirements for the portal. I had to find out why we hadn't developed portal requirements: we simply forgot about it. Recording the issue in a tracking tool would have kept us from scrambling at the last minute and avoided upsetting the customer.

Measuring requirements 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 development and management activities, you can evaluate whether it was too little, about right, or too much, and adjust your future planning accordingly. Karl Wiegers (2006) discusses measuring various other aspects of the requirements work on a project.

Measuring effort requires a culture change and the individual discipline to record daily work activities (Wiegers 1996). Effort tracking isn't as time-consuming as people sometimes fear. Team members gain valuable insight from knowing how they *actually* spent their time, compared to how they *thought* they spent their time, compared to how they were *supposed* to spend their time. Effort tracking also indicates whether the team is performing the intended requirements-related activities.

Note that work effort is not the same as elapsed calendar time. Tasks can be interrupted; they might require interactions with other people that lead to delays. The total effort for a task, in units of labor hours, might not change because of such factors (although frequent interruptions do reduce an individual's productivity), but the calendar duration increases.

When tracking requirements development effort, you might find it valuable to separate the time spent by people in the BA role from time spent by other project participants. Tracking the BA's time will help you plan how much BA effort is needed on future projects (see Chapter 19, "Beyond requirements development," for more about estimating BA time). Measuring the total effort spent on requirements activities by all stakeholders gives you a sense of the total cost of requirements activities on a project. Record the number of hours spent on requirements development activities such as the following:

- Planning requirements-related activities for the project
- Holding workshops and interviews, analyzing documents, and performing other elicitation activities
- Writing requirements specifications, creating analysis models, and prioritizing requirements
- Creating and evaluating prototypes intended to assist with requirements development
- Reviewing requirements and performing other validation activities

Count the effort devoted to the following activities as requirements management effort:

- Configuring a requirements management tool for your project
- Submitting requirements changes and proposing new requirements
- Evaluating proposed changes, including performing impact analysis and making decisions
- Updating the requirements repository
- Communicating requirements changes to affected stakeholders
- Tracking and reporting requirements status
- Creating requirements trace information

Remember, the time you spend on these requirements-related activities is an investment in project success, not just a cost. To justify the activities, compare this time investment with the time the team spends dealing with issues that arose because these things were *not* done—the cost of poor quality.

Managing requirements on agile projects

Agile projects accommodate change by building the product through a series of development iterations and managing a dynamic product backlog of work remaining to be done. As described in Chapter 2, the stakeholders reach agreement on the stories to be implemented in each iteration. New stories that customers add while an iteration is under way are prioritized against the remaining

backlog contents and allocated to future iterations. New stories might displace lower-priority stories if the team wants to keep the original delivery schedule. The goal—as it should be for all projects—is to always be working on the highest-priority stories to deliver the maximum value to customers as quickly as possible. See Chapter 28 for more information about handling requirement changes on agile projects.

Some agile teams, particularly large or distributed teams, use an agile project management tool to track the status of an iteration and the stories allocated to it. The stories and their associated acceptance criteria and acceptance tests might all be placed in a product backlog or user story–management tool. Story status can be monitored by using statuses analogous to those described earlier in Table 27-1 (Leffingwell 2011):

- In backlog (the story is not yet allocated to an iteration)
- Defined (details of the story were discussed and understood, and acceptance tests were written)
- In progress (the story is being implemented)
- Completed (the story is fully implemented)
- Accepted (acceptance tests were passed)
- Blocked (the developer is unable to proceed until something else is resolved)

Agile projects typically monitor their progress with an iteration burndown chart (Cohn 2004; Cohn 2005). The team estimates the total amount of work to do on the project, often sized in units of story points, which are derived from an understanding of the user stories in the product backlog (Cohn 2005; Leffingwell 2011). The story point total is thus proportional to the amount of effort the team must expend to implement the requirements. The team allocates certain user stories to each iteration based on their priority and their estimated size in story points. The team's past or average velocity dictates the number of story points the team plans to deliver in an iteration of a particular calendar duration.

The team charts the story points remaining in the product backlog at the end of each iteration. This total will change as work is completed, as current stories are better understood and re-estimated, as new stories are added, and as customers remove pending work from the backlog. That is, rather than monitoring the count and status of individual functional requirements or features (which can come in a variety of sizes), the burndown chart shows the total work remaining to be done at a specific time.

Figure 27-3 illustrates a burndown chart for a hypothetical project. Notice that the scope remaining, as measured in story points, actually increased in iterations 2, 3, and 5. This indicates that more new functionality was added to the backlog than was completed or removed during the course of the iteration. The burndown chart helps the team avoid the "90 percent done" syndrome by making visible the amount of work remaining, as opposed to the amount of work completed, which doesn't reflect the inevitable scope increases. The slope of the burndown chart also reveals the projected end date for the project, the point at which no work remains in the backlog.

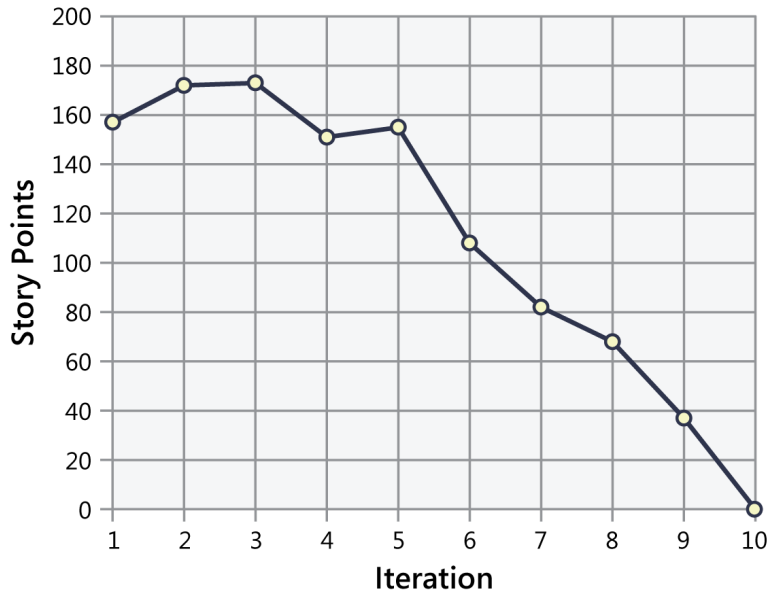

FIGURE 27-3 Sample iteration burndown chart for monitoring the product backlog on an agile project.

Why manage requirements?

Whether your project is following a sequential development life cycle, one of the various agile life cycles, or any other approach, managing the requirements is an essential activity. Requirements management helps to ensure that the effort you invest in requirements development isn't squandered. Effective requirements management reduces the expectation gap by keeping all project stakeholders informed about the current state of the requirements throughout the development process. It lets you know where you're headed, how the trip is going, and when you've arrived at your destination.

Next steps

- Document the processes your organization will follow to manage the requirements on each project. Engage several business 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 each affected project.
- If you're not using a requirements management tool, define a version labeling scheme to identify your requirements documents. Educate the BAs about this scheme.
- Select the statuses that you want to use to describe the life cycle of your functional requirements or user stories. Draw a state-transition diagram to show the conditions or events that trigger a change from one status to another.
- Define the current status for each requirement in your baseline. Keep the status current as development progresses.