Requirement functionality

Is every requirement correct?

Are all inputs and outputs clearly specified?

Requirement performance

Are all nonfunctional requirements that constrain performance (speed, resource utilization, etc.) clearly and quantitatively defined?

Requirement testability

Is each requirement testable?

> **NOTE**
>
> **More information on SRS development can be found in *Managing Software Requirements: A Use Case Approach* by Dean Leffingwell and Don Widrig (Addison Wesley, 2003) and *Software Requirements* by Karl Wiegers (Microsoft Press, 2003).**

Change Control

Throughout the course of most projects, many of the people involved come up with changes to the planned software that could be implemented. Many poorly managed software projects have been driven to failure because the designers, developers, and testers had to repeatedly switch directions because of uncontrolled changes. Changes originate from all over the project: a stakeholder may discover a new need that should be addressed, a senior manager could change his mind about a feature, a programmer could figure out a way to combine behaviors to make the software more efficient, a tester could discover conflicting requirements, etc. Some of these changes will be worth doing, while others should probably be scrapped. But every change will come with a sense of urgency, and the project manager needs a way to sort through them to make sure that only the right changes are made.

Change control is a method for implementing only those changes that are worth pursuing, and for preventing unnecessary or overly costly changes from derailing the project. Change control is essentially an agreement between the project team and the managers that are responsible for decision-making on the project to evaluate the impact of a change before implementing it. Many changes that initially sound like good ideas will get thrown out once the true cost of the change is known. The potential benefit of the change is written down, and the project manager works with the team to estimate the potential impact that the change will have on the project. This gives the organization all of the information necessary to do a real cost-benefit analysis. If the benefit of the change is worth the cost, the project manager updates the plan to reflect the new estimates. Otherwise, the change is thrown out and the team continues with the original plan.

Changes that seem "small" because they are easy to describe in words can have an unexpectedly large impact on the project. Even the most carefully planned and tracked project can be thrown off course by unexpected changes. A project manager can use change control to keep this from happening.

Establish a Change Control Board

The most important part of change control is a *change control board* (CCB). There are certain people in the organization who have the power to change the scope of the project. Usually there is a senior manager or decision-maker who has the authority to make sweeping changes at will; sometimes there are several people in this position. For change control to be effective, these people must be part of the CCB.

In addition, the CCB should contain people from the project team:

- The project manager
- An important stakeholder or user (or someone who understands and advocates the team's perspective)
- Someone who understands the effort involved in making the change (usually, this is a representative from the programming team)
- Someone who understands the engineering decisions that the team makes over the course of the project (a design team member, requirements analyst, or, if neither is available, a programmer who participated in the design of the software)
- Someone who is familiar with the expected functionality of the software and with the behavior being discussed for each individual change (typically a tester)

This last person fulfills a very important role in the change control process. Typically, she is involved in the tracking of changes and defects in the product. When a bug is reported, part of her job is to figure out whether it is a defect (meaning that the software does not behave the way its specification requires it to behave) or a change (meaning that the software behaves as designed, but that this behavior is not what the users or stakeholders need).

Changes will generally be reported through the defect tracking system. (If there is currently no defect tracking system in place, establishing one will have a far greater impact on controlling changes than a change control process will—see Chapter 8). The job of the tester on the CCB is to understand the change: the behavior specified in the requirements, why this behavior is incorrect, and how and why the software should be changed.

Before the project begins, the list of CCB members should be written down and agreed upon, and each CCB member should understand why the change control process is needed and what her role will be in it. The project manager must ensure that everyone buys into the idea of change control and agrees that it is their job to evaluate each change before the software project plan can be altered. The project manager must also reassure them that the programmers and other team members cannot deviate from the plan. This agreement between CCB members is the most important part of the change control process.

Change Control Process

Table 6-14 shows the script for a change control process.

TABLE 6-14. Change control process script

Name	Change control process script
Purpose	To control changes by evaluating their impact before agreeing to implement them
Summary	The change control process ensures that the impact of each change is evaluated before the decision is made to implement that change. A change is proposed by anyone evaluating the software. A *change control board* (CCB), made up of the decision-makers, project manager, stakeholder or user representatives, and selected team members, evaluates the change. The CCB either accepts or rejects the change.
Work products	*Input* Issue report in the defect tracking system that describes the change *Output* Modified issue report that reflects the impact of the change and the decision on whether or not to move forward with it
Entry criteria	A change has been discovered, and an issue report that describes it has been entered into the defect tracking system.
Basic course of events	1. A CCB member (typically a tester) who is familiar with the expected functionality of the software reads and understands the issue report, which describes the requested change. 2. The CCB member familiar with the change meets with the project manager to explain its scope and significance. Together, they identify all project team members who will be impacted by the change, and work with them to evaluate its impact. The project manager updates the issue report to reflect the result of that evaluation. 3. At the next CCB meeting, the project manager presents the scope and significance of the change, along with its expected impact. The CCB discusses the change, and performs a cost-benefit analysis to determine if its benefits are worth the cost. The CCB approves the change. 4. The project manager updates the issue report to indicate that the change has been approved, and then updates the project plan to reflect the change. The team members begin implementing the change.
Alternative paths	1. In Step 1, if the CCB member does not understand the change, it can be returned to the submitter for further explanation. The submitter may choose to either update the issue report to clarify the change (in which case, the script returns to step 1) or drop it entirely (in which case, the change control process ends). 2. In Step 3, if the CCB determines that the benefits of the change are not worth the cost, it can reject it. The change control process ends, and no changes are made to the project. The project manager updates the issue report to reflect the fact that it was rejected.
Exit criteria	The project plan has been updated to reflect the impact of the change, and work to implement the change has begun.

Evaluate Changes in the CCB Meeting

There are points in the course of the project when the CCB may need to meet regularly:

- During the requirements phase, the CCB will need to discuss the scope of the project if it turns out that there are major areas that the vision and scope failed to cover.
- During the design and programming of the software, the team may discover that the requirements need to be changed. For example, programmers may discover requirements in the SRS that seemed reasonable at the time but that turn out to be contradictory or convoluted, which could be changed to simplify the implementation.

- During the testing phase, the testers may discover omissions in the SRS or design that cause defects. For example, an enhancement to a software project may require that a new record type is to be added to one feature, but the requirements and design fail to specify how that record type is handled in another related feature. The programmers never implemented any handling for this new record type in the second feature because the design failed to indicate how that was to be handled.
- The users or stakeholders may discover during a design walkthrough, demo, user acceptance testing, or beta testing that the software does not fulfill their needs.

During the CCB meeting, the project manager explains the change to the rest of the CCB. Once the CCB is brought up to speed, it must determine what project work products will have to be changed. Each change will affect at least one work product that has already been approved—if this were not the case, the CCB would not have to meet because the change could be handled as part of the regular review process.

The CCB must evaluate every change that is requested. This ensures that nothing slips through the cracks, and that a real decision is made for each request. However, this could mean that at some very busy points in the project, the CCB must meet periodically—sometimes weekly or even daily. At each meeting, it may discuss many changes that have been submitted since the previous meeting.

It is important that when the CCB is being organized, the project manager makes sure that each member understands that this sort of time commitment may be necessary. These meetings are an important way for all of the CCB members to stay on top of the issues that come up during development. This knowledge is very valuable later on in the project when it is time to determine whether the software is ready for release (see Chapter 8).

Analyze the Impact of Each Change

It is vital that the project manager understand and manage all of the information produced in the change control process. While all of the project team members' opinions are necessary in evaluating the change, it is the project manager who owns the change, and who makes sure that it is properly understood and evaluated. This is generally a lot of work: the project manager needs to take the time to understand why each change is needed, what needs to be changed, and how much work it will be to make the change. His understanding must be complete enough that he can present this information at the CCB meeting.

The project manager who is responsible for guiding each change through the change control process. This is why it is important that the project manager be the one who updates the issue report to reflect both the effort that was estimated in the evaluation and the final decision of the CCB. And it is the project manager who is responsible for making sure that each CCB member is given enough information to understand the change.

During the change control process, the project manager works with the team to evaluate the impact. One effective way to do this is to use the Wideband Delphi estimation process (see Chapter 3). If this is done, the project manager can append the results of the Delphi meeting to the issue report and present them to the CCB for cost-benefit analysis. But

while a full estimation may be necessary for large changes, there are many small changes that do not require the project manager to gather an estimation team and hold a meeting. For example, a change that amounts to a small bug fix may simply require that the project manager talk to the programmer responsible for that code and ask how long it will take to fix it.

There is some overlap here between the responsibility of the project manager and that of the QA team. Some QA engineers may question the need for the project manager to be so involved in updating specific issue reports in the defect tracking system—this is generally a task that is exclusively controlled by QA. This is true both when the issue is a defect and when it is a change; in both cases, the impact must be evaluated, to determine whether the change should be implemented or whether the defect should be fixed.

The reason the project manager is responsible for this is because he serves as a conduit for all of the information relevant to the change. By updating the issue report, he ensures that he has gathered all the information relevant to the change, and that all of the information is in one place. The issue report is still owned by QA, but it makes sense to have the project manager directly update it rather than having to sit down with the QA team each time a change is evaluated. As long as a member of the QA team on the CCB initiates each change, they will still be kept in the loop.

Introduce Software Requirements Carefully

There are plenty of books that tell us that poor requirements are the most common cause of software quality problems. Yet many project managers have had difficulty bootstrapping efforts within their own teams to implement better software requirements practices. Software requirements should make intuitive sense: before a team can build software, they need to know what to build. In practice, however, many project managers have found that it is difficult to convince their teams to adopt good software requirements engineering practices. This is especially true for certain kinds of projects:

- Small projects in which the programmer is confident that he understands all of the requirements already
- Projects in which "everybody" knows what the software is supposed to do
- Projects without a user interface (like batch processes or back-end calculation software)
- Any project considered "technical"— one in which one or more of the stakeholders is a programmer

In all these projects, there is an expectation that a programmer can make all of the decisions about the behavior of the software. It seems intuitive (but incorrect) that since the programmer can already decide on the details of the implementation, he should also have a grasp on what is being implemented. The programmer is often the most confident person in the organization about his own ability to do this, which the rest of the team and the stakeholders always find reassuring.