Part IV: Implementing Requirements Engineering

Chapter List

Chapter 22: Improving Your Requirements Processes

Overview

Previous chapters have described several dozen requirements engineering "good practices" for you to consider applying in your software organization. Putting better practices into action is the essence of software process improvement. In a nutshell, process improvement consists of using more of the approaches that work well for us and avoiding those that have given us headaches in the past. However, the path to improved performance is paved with false starts, resistance from those who are affected, and the frustration of having too little time to handle current tasks, let alone improvement programs.

The ultimate objective of software process improvement is to reduce the cost of creating and maintaining software. There are several ways to accomplish this:

- Correcting problems encountered on previous or current projects that arose from process shortcomings
- Anticipating and preventing problems that you might encounter on future projects
- Adopting practices that are more efficient than the practices currently being used

If your team's current methods seem to work well—or if people insist that they do despite evidence to the contrary—people might not see the need to change their approach. However, even successful software organizations can struggle when confronted with larger projects, different customers, long-distance collaborations, tighter schedules, or new application domains. Approaches that worked for a team of five people with a single customer don't scale up to 125 people located in two different time zones who are serving 40 corporate customers. At the least, you should be aware of other approaches to requirements engineering that could be valuable additions to your software engineering tool kit.

This chapter describes how requirements relate to various key project processes and stakeholders. I'll present some basic concepts about software process improvement and a suggested process improvement

cycle. I'll also list several useful requirements "process assets" that your organization should have available. The chapter concludes by describing a process improvement road map for implementing improved requirements engineering practices.

How Requirements Relate to Other Project Processes

Requirements lie at the heart of every well-run software project, supporting the other technical and management activities. Changes that you make in your requirements development and management approaches will affect these other processes, and vice versa. Figure 22-1 illustrates some connections between requirements and other processes; the sections that follow briefly describe these process interfaces.

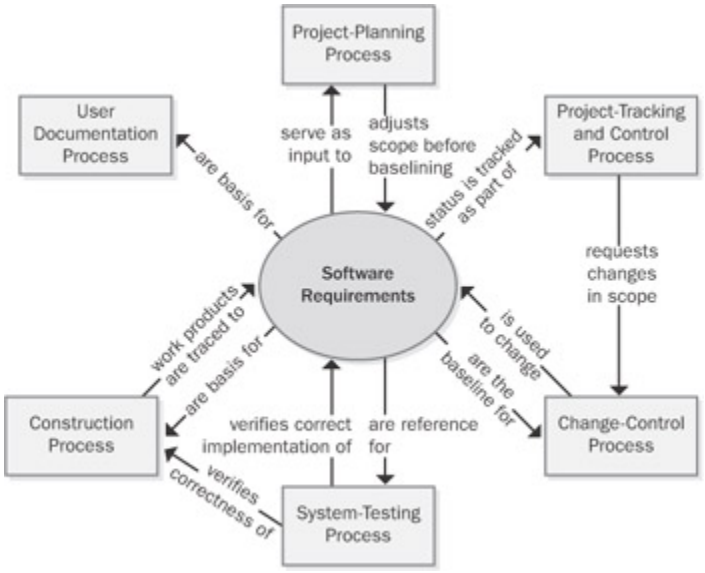


Figure 22-1: Relationship of requirements to other project processes.

Project planning Requirements are the foundation of the project-planning process. The planners select an appropriate software development life cycle and develop resource and schedule estimates based on the requirements. Project planning might indicate that it's not possible to deliver the entire desired feature set within the available bounds of resources and time. The planning process can lead to reductions in the project scope or to the selection of an incremental or staged-release approach to deliver functionality in phases.

Project tracking and control Project tracking includes monitoring the status of each requirement so that the project manager can see whether construction and verification are proceeding as intended. If not, management might need to request a scope reduction through the change-control process.

Change control After a set of requirements has been baselined, all subsequent changes should be made through a defined change-control process. This process helps ensure that

- The impact of a proposed change is understood.

- The appropriate people make informed decisions to accept changes.
- All people who are affected by a change are made aware of it.
- Resources and commitments are adjusted as needed.
- The requirements documentation is kept current and accurate.

System testing User requirements and functional requirements are essential inputs to system testing. If the expected behavior of the software under various conditions isn't clearly specified, the testers will be hard-pressed to identify defects and to verify that all planned functionality has been implemented as intended.

Construction Although executable software is the ultimate deliverable of a software project, requirements form the foundation for the design and implementation work and they tie together the various construction work products. Use design reviews to ensure that the designs correctly address all of the requirements. Unit testing can determine whether the code satisfies the design specifications and the pertinent requirements. Requirements tracing lets you document the specific software design and code elements that were derived from each requirement.

True Stories **User documentation** I once worked in an office area that also housed the technical writers who prepared user documentation for complex software products. I asked one of the writers why they worked such long hours. "We're at the end of the food chain," she replied. "We have to respond to the final changes in user interface displays and the features that got dropped or added at the last minute." The product's requirements provide input to the user documentation process, so poorly written or late requirements will lead to documentation problems. It's not surprising that the long-suffering people at the end of the requirements chain, such as technical writers and testers, are often enthusiastic supporters of improved requirements engineering practices.

PREVIOUS NEXT
PREVIOUS NEXT

Requirements and Various Stakeholder Groups

Figure 22-2 shows some of the project stakeholders who can interface with a software development group and some of the contributions they make to a project's requirements engineering activities. Explain to your contact people in each functional area the information and contributions you need from them if the product development effort is to succeed. Agree on the form and content of key communication interfaces between development and other functional areas, such as a system requirements specification or a market requirements document. Too often, important project documents are written from the author's point of view without full consideration of the information that the readers of those documents need.

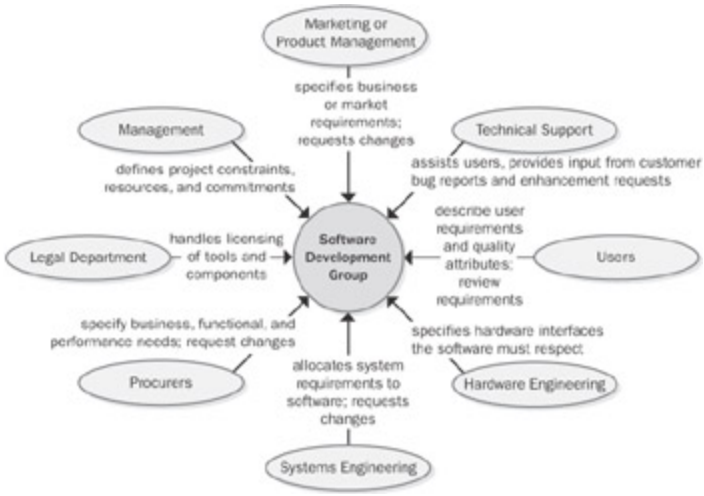


Figure 22-2: Requirements-related interfaces between software development and other stakeholders.

On the flip side, ask the other organizations what they need from the development group to make their jobs easier. What input about technical feasibility will help marketing plan their product concepts better? What requirements status reports will give management adequate visibility into project progress? What collaboration with system engineering will ensure that system requirements are properly partitioned among software and hardware subsystems? Strive to build collaborative relationships between development and the other stakeholders of the requirements process.

When the software development group changes its requirements processes, the interfaces it presents to other project stakeholder communities also change. People don't like to be forced out of their comfort zone, so expect some resistance to your proposed requirements process changes. Understand the origin of the resistance so that you can both respect it and defuse it. Much resistance comes from fear of the unknown. To reduce the fear, communicate your process improvement rationale and intentions to your counterparts in other areas. Explain the benefits that these other groups will receive from the new process. When seeking collaboration on process improvement, begin from this viewpoint: "Here are the problems we've all experienced. We think that these process changes will help solve those problems. Here's what we plan to do, this is the help we'll need from you, and this is how our work will help us both." Following are some forms of resistance that you might encounter:

- A change-control process might be viewed as a barrier thrown up by development to make it harder to get changes made. In reality, a change-control process is a structure, not a barrier. It permits well-informed people to make good business decisions. The software team is responsible for ensuring that the change process really does work. If new processes don't yield better results, people will find ways to work around them—and they probably should.

- Some developers view writing and reviewing requirements documents as bureaucratic time-wasters that prevent them from doing their "real" work of writing code. If you can explain the high cost of continually rewriting the code while the team tries to figure out what the system should do, developers and managers will better appreciate the need for good requirements.

- If customer-support costs aren't linked to the development process, the development team might not be motivated to change how they work because they don't suffer the consequences of poor product quality.

- If one objective of improved requirements processes is to reduce support costs by creating higher-quality products, the support manager might feel threatened. Who wants to see his empire shrink?

- Busy customers sometimes claim that they don't have time to spend working on the requirements. Remind them of earlier projects that delivered unsatisfactory systems and the high cost of responding to customer input after delivery.

Anytime people are asked to change the way they work, the natural reaction is to ask, "What's in it for me?" However, process changes don't always result in fabulous, immediate benefits for every individual involved. A better question—and one that any process improvement leader must be able to answer convincingly—is, "What's in it for *us*?" Every process change should offer the prospect of clear benefits to the project team, the development organization, the company, the customer, or the universe. You can often sell these benefits in terms of correcting the known shortcomings of the current ways of working that lead to less than desirable business outcomes.

PREVIOUS NEXT
PREVIOUS NEXT

Fundamentals of Software Process Improvement

You're reading this book presumably because you intend to change some of the current approaches your organization uses for requirements engineering. As you begin your quest for excellent requirements, keep the following four principles of software process improvement in mind (Wiegers 1996a):

1. **Process improvement should be evolutionary, continuous, and cyclical.** Don't expect to improve all your processes at once, and accept that you won't get everything right the first time you try to make changes. Instead of aiming for perfection, develop a few improved templates and procedures and get started with implementation. Adjust your approaches as your team gains experience with the new techniques. Sometimes simple and easy changes can lead to substantial gains, so look for the low-hanging fruit.

2. **People and organizations change only when they have an incentive to do so.** The strongest incentive for change is pain. I don't mean artificially induced pain, such as management-imposed schedule pressure intended to make developers work harder, but rather the very real pain you've experienced on previous projects. Following are some examples of problems that can provide compelling drivers for changing your requirements processes:

    - The project missed deadlines because the requirements were more extensive and complicated than expected.

    - Developers worked a lot of overtime because misunderstood or ambiguous requirements were addressed late in development.

    - System test effort was wasted because the testers didn't understand what the product was supposed to do.

    - The right functionality was present, but users were dissatisfied because of sluggish performance, poor usability, or other quality shortcomings.

    - The organization experienced high maintenance costs because customers requested many enhancements that should have been identified during requirements elicitation.

- The development organization acquired a reputation for delivering software that customers don't want.

3. **Process changes should be goal-oriented.** Before you begin the journey to superior processes, make sure that you know where you're headed (Potter and Sakry 2002). Do you want to reduce the amount of work that is redone because of requirements problems? Do you want better schedule predictability? Do you want to stop overlooking requirements during implementation? A road map that defines pathways to your business objectives greatly improves your chances of successful process improvement.

4. **Treat your improvement activities as miniprojects.** Many improvement initiatives founder because they're poorly planned or because resources never materialize. Include process improvement resources and tasks in your project's overall plans. Perform the planning, tracking, measurement, and reporting that you'd do for any project, scaled down for the size of the improvement project. Write an action plan for each process improvement area you tackle. Track the time the participants spend executing the action plan to check whether you're getting the level of effort you expected and to know how much the improvement work is costing.

   Trap The single biggest threat to a software process improvement program is lack of management commitment, followed closely by reorganizations that shuffle the program's participants and priorities.

All team members have the opportunity—and the responsibility—to actively improve how they do their work. Professional software practitioners don't need permission from their managers to better themselves and their teams. Grass-roots improvement programs that grow out of frustration with the status quo or are galvanized by a charismatic leader can be quite successful. However, a broad process improvement effort can succeed only if management is motivated to commit resources, set expectations, and hold team members accountable for their contributions to the change initiative.

Process Improvement One-Liners

The experienced software process improvement leader accumulates a list of short, pithy observations about this difficult domain. Here are some that I've picked up over the years:

- Take chewable bites. (If you bite into too large a process change, the team will likely choke on it.)
- Take a lot of satisfaction from small victories. (You won't have many big victories.)
- Use gentle pressure, relentlessly applied. (The process improvement leaders and committed managers steer the team toward a better future by keeping the change initiative visible and continually chipping away at it.)
- Focus, focus, focus. (A busy software team can work on only three, or two, or perhaps just one improvement initiative at a time. But never work on fewer than one.)
- Look for allies. (Every team has its early adopters who will try out new templates and procedures and give the improvement leaders feedback. Cultivate them. Thank them. Reward them.)
- Action plans that don't turn into actions are not useful. (It's easy to do a process assessment and to write an action plan. It's hard to get people to work in new ways that hold the promise of better results, yet that's the only useful outcome of process improvement.)

The Process Improvement Cycle

Figure 22-3 illustrates a process improvement cycle I've found to be effective. This cycle reflects the importance of knowing where you are before you take off for someplace else, the need to chart your course, and the value of learning from your experiences as part of continuous process improvement.

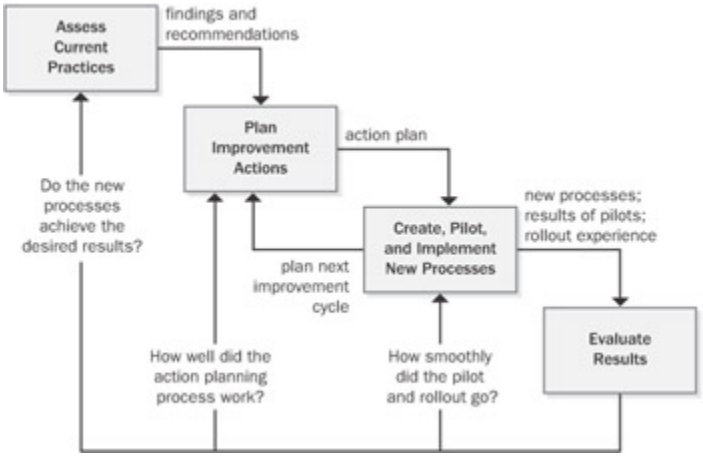


Figure 22-3: The software process improvement cycle.

Assess Current Practices

Step 1 of any improvement activity is to assess the practices currently being used in an organization and to identify their strengths and shortcomings. An assessment does not in itself provide any improvement—it provides information. The assessment lays the foundation for making the right choices about the changes you should make. It also brings visibility to the processes actually being used in the organization, which are frequently different from the stated or documented processes. And you'll find that different team members typically have very different perspectives as to what processes the team is using.

You can evaluate your current processes in several ways. If you tried any of the Next Steps at the end of previous chapters, you've already begun an informal evaluation of your requirements practices and their results. Structured questionnaires provide a more systematic approach, which can reveal insights about your current processes at low cost. Interviews and discussions with team members provide a more accurate and comprehensive understanding.

You can use the questionnaire in Appendix A to calibrate your organization's current requirements engineering practices.[1] This self-assessment helps you decide which of your requirements processes are most in need of improvement. Just because you give yourself a low rating on a particular question isn't reason enough to address it immediately or perhaps at all. Focus your energy on improving those practice areas that are causing your projects the most difficulties and those that pose risks to the success of your future projects.

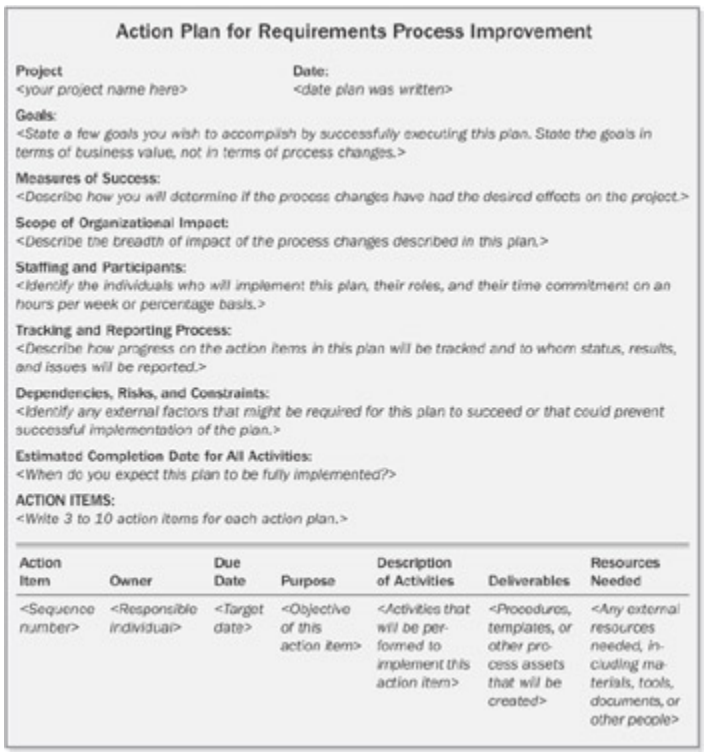
A more thorough approach is to have an outside consultant evaluate your current software processes. The most comprehensive process assessments are based on an established process improvement framework,

such as the Capability Maturity Model for Software (SW-CMM) developed by the Software Engineering Institute (Paulk et al. 1995). Outside assessors typically examine many of your software development and management processes, not just the requirements activities. The deliverables from a formal assessment include a list of findings—statements of both strengths and weaknesses in the current processes—and recommendations for addressing the improvement opportunities. Select an assessment approach that aligns with the business goals you want to achieve through your process improvement activities, and don't worry too much about satisfying the requirements of the SW-CMM or any other specific model. Appendix B, "Requirements and Process Improvement Models," describes how requirements fit into the SW-CMM and the newer CMM Integration model, the CMMI-SE/SW.

Plan Improvement Actions

In keeping with the philosophy of treating process improvement activities as projects, write an action plan following your assessment (Potter and Sakry 2002). A strategic plan describes your organization's overall software process improvement initiative. Tactical action plans target specific improvement areas, such as the way you gather requirements or your prioritization procedure. Each action plan should state the goals of the improvement activities, the participants, and the individual action items that must be completed to implement the plan. Without a plan, it's easy to overlook important tasks. The plan also provides a way to monitor progress as you track the completion of individual action items.

Figure 22-4 illustrates a process improvement action plan template I've used many times. Include no more than 10 items in each action plan, scoped such that the plan can be completed in two to three months. As an example, I saw a plan for requirements management improvements that included these action items:



Action Plan for Requirements Process Improvement

Project
<your project name here>

Date:
<date plan was written>

Goals:
<State a few goals you wish to accomplish by successfully executing this plan. State the goals in terms of business value, not in terms of process changes.>

Measures of Success:
<Describe how you will determine if the process changes have had the desired effects on the project.>

Scope of Organizational Impact:
<Describe the breadth of impact of the process changes described in this plan.>

Staffing and Participants:
<Identify the individuals who will implement this plan, their roles, and their time commitment on an hours per week or percentage basis.>

Tracking and Reporting Process:
<Describe how progress on the action items in this plan will be tracked and to whom status, results, and issues will be reported.>

Dependencies, Risks, and Constraints:
<Identify any external factors that might be required for this plan to succeed or that could prevent successful implementation of the plan.>

Estimated Completion Date for All Activities:
<When do you expect this plan to be fully implemented?>

ACTION ITEMS:
<Write 3 to 10 action items for each action plan.>

Action Item	Owner	Due Date	Purpose	Description of Activities	Deliverables	Resources Needed
<Sequence number>	*<Responsible individual>*	*<Target date>*	*<Objective of this action item>*	*<Activities that will be performed to implement this action item>*	*<Procedures, templates, or other process assets that will be created>*	*<Any external resources needed, including materials, tools, documents, or other people>*

Figure 22-4: Action plan template for software process improvement.

1. Draft a requirements change-control procedure.

2. Review and revise the change-control procedure.

3. Pilot the change-control procedure with Project A.
4. Revise the change-control procedure based on feedback from the pilot.
5. Evaluate problem-tracking tools, and select one to support the change-control procedure.
6. Procure the problem-tracking tool, and customize it to support the change-control procedure.
7. Roll out the new change-control procedure and tool to the organization.

Assign each action item to a specific individual owner who is responsible for seeing that the item is completed. Don't assign "the team" as an action item owner. Teams don't do work; individuals do.

If you need more than about 10 action items, focus the initial activity cycle on the most important issues and address the rest later in a separate action plan; remember, change is cyclical. The process improvement road map described later in this chapter illustrates how you can group multiple improvement actions into an overall software process improvement plan.

Create, Pilot, and Implement New Processes

So far, you've evaluated your current requirements practices and crafted a plan for addressing the process areas that you think are most likely to yield benefits. Now comes the hard part: implementing the plan. Many process improvement initiatives stumble when they try to turn action plans into actions.

Implementing an action plan means developing processes that you believe will yield better results than your current ways of working do. Don't expect to get the new processes perfect on the first try. Many approaches that seem like a good idea in the abstract turn out to be less pragmatic or less effective than anticipated. Therefore, plan a process *pilot* for most of the new procedures or document templates you create. Use the knowledge gained from the pilot to adjust the new process. This improves the chance that it will be effective and well received when you roll it out to the affected community. Keep the following suggestions in mind for the process pilots you conduct:

- Select pilot participants who will give the new approaches a fair try and provide helpful feedback. These participants could be either allies or skeptics, but they shouldn't strongly oppose the improvement effort.
- To make the outcome easy to interpret, quantify the criteria the team will use to evaluate the pilot.
- Identify the stakeholders who need to be kept informed of what the pilot is about and why it is being performed.
- Consider piloting portions of the new processes on different projects. This engages more people in trying new approaches, which increases awareness, feedback, and buy-in.
- As part of the evaluation, ask pilot participants how they would feel if they had to go back to their former ways of working.

Even motivated and receptive teams have a limited capacity to absorb change, so don't place too many new expectations on a project team at once. Write a roll-out plan that defines how you'll distribute the new methods and materials to the project team, and provide sufficient training and assistance. Also consider how management will communicate their expectations about the new processes, perhaps through

a formal requirements engineering policy.

Evaluate Results

The final step of a process improvement cycle is to evaluate the activities performed and the results achieved. This evaluation will help the team do an even better job on future improvement activities. Assess how smoothly the pilots ran and how effective they were in resolving the uncertainties about the new processes. Would you change anything the next time you conduct a process pilot?

Consider how well the general rollout of the new processes went. Was the availability of the new processes or templates communicated effectively to everyone? Did participants understand and successfully apply the new processes? Would you change anything about how you handle the next rollout?

A critical step is to evaluate whether the newly implemented processes are yielding the desired results. Some new technical and management practices deliver visible improvements quickly, but others take time to demonstrate their full value. For example, you should be able to tell quickly whether a new process for dealing with requirement changes facilitates incorporating changes into the project in a less chaotic way. However, a new software requirements specification template can take some time to prove its worth as analysts and customers get used to the discipline of documenting requirements in a particular format. Give new approaches adequate time to work, and select measures that will demonstrate the success of each process change.

Accept the reality of the learning curve—that is, the productivity drop that takes place as practitioners take time to assimilate new ways of working, as illustrated in Figure 22-5. This short-term productivity drop—sometimes called the "valley of despair"—is part of the investment your organization is making in process improvement. People who don't understand this might be tempted to abandon the process improvement effort before it begins to pay off, thereby achieving a zero return on their investment. Educate your managers and peers about the learning curve, and commit to seeing the change initiative through. The team *will* achieve superior project and business results with the help of superior requirements processes.

Figure 22-5: The learning curve is an unavoidable part of process improvement.

[1]Motorola developed a similar “Software Requirements Quality Model” for requirements process assessment (Smith 1998).

Requirements Engineering Process Assets

High-performance projects have effective processes for all of the requirements engineering components: elicitation, analysis, specification, validation, and management. To facilitate the performance of these processes, every organization needs a collection of *process assets* (Wiegers 1998c). A process encompasses the actions you take and the deliverables you produce; process assets help the team members perform processes consistently and effectively. These process assets will help those involved in the project understand the steps they should follow and the work products they're expected to create. Process assets include the types of documents described in Table 22-1.

Table 22-1: Process Asset Documents

Type	Description
checklist	A list that enumerates activities, deliverables, or other items to be noted or verified. Checklists are memory joggers. They help ensure that busy people don't overlook important details.
example	A representative of a specific type of work product. Accumulate good examples as your project teams create them.
plan	An outline of how an objective will be accomplished and what is needed to accomplish it.
policy	A guiding principle that sets a management expectation of behaviors, actions, and deliverables. Processes should enable satisfaction of the policies.
procedure	A step-by-step description of the sequence of tasks that accomplishes an activity. Describe the tasks to be performed and identify the project roles that perform them. Don't include tutorial information in a procedure. Guidance documents can support a process or procedure with tutorial information and helpful tips.
process description	A documented definition of a set of activities performed for some purpose. A process description might include the process objective, key milestones, participants, communication steps, input and output data, artifacts associated with the process, and ways to tailor the process to different project situations (Caputo 1998).
template	A pattern to be used as a guide for producing a complete work product. Templates for key project documents remind you to ask questions that you might otherwise overlook. A well-structured template provides many "slots" for capturing and organizing information. Guidance text embedded in the template will help the document author use it effectively.

Figure 22-6 identifies some valuable process assets for requirements engineering. No software process rule book says that you need all of these items, but they will all assist your requirements-related activities. The procedures listed in Figure 22-6 should be no longer than they need to be to let team members consistently perform the procedures effectively. They need not be separate documents; an overall requirements management process could include the change-control procedure, status-tracking procedure, and impact analysis checklist. Many of the process assets in Figure 22-6 are available at *http://www.processimpact.com/goodies.shtml*.

Requirements Development Process Assets	**Requirements Management Process Assets**
• Requirements Development Process	• Requirements Management Process

• Requirements Allocation Procedure • Requirements Prioritization Procedure • Vision and Scope Template • Use-Case Template • Software Requirements Specification Template • SRS and Use-Case Defect Checklists	• Change-Control Process • Requirements Status-Tracking Procedure • Requirements Traceability Procedure • Change Control Board Charter • Requirements Change Impact Analysis Checklist and Template

Figure 22-6: Key process assets for requirements development and requirements management.

Following are brief descriptions of each of the process assets listed in Figure 22-6, along with references to the chapters where they are discussed in detail. Keep in mind that each project should tailor the organization's assets to best suit its needs.

Requirements Development Process Assets

Requirements development process This process describes how to identify stakeholders, user classes, and product champions in your domain. It should address how to plan the elicitation activities, including selecting appropriate elicitation techniques, identifying participants, and estimating the effort and calendar time required for elicitation. The process also describes the various requirements documents and models your project is expected to create and points the reader toward appropriate templates. The requirements development process also should identify the steps the project should perform for requirements analysis and validation.

Requirements allocation procedure Allocating high-level product requirements to specific subsystems, including people, is necessary when developing systems that include both hardware and software components or complex products that contain multiple software subsystems (Nelsen 1990). Allocation takes place after the system-level requirements are specified and the system architecture has been defined. This procedure describes how to perform these allocations to ensure that functionality is assigned to the appropriate components. It also describes how allocated requirements will be traced back to their parent system requirements and to related requirements in other subsystems.

Requirements prioritization procedure To sensibly reduce scope or to accommodate added requirements within a fixed schedule, we need to know which planned system capabilities have the lowest priority. Chapter 14, "Setting Requirement Priorities," describes a spreadsheet tool for prioritization that incorporates the value provided to the customer, the relative technical risk, and the relative cost of implementation for each feature, use case, or functional requirement.

Vision and scope template The vision and scope document is a concise, high-level description of the new product's business requirements. It provides a reference for making decisions about requirements priorities and changes. Chapter 5, "Establishing the Product Vision and Project Scope," recommends a template for this document.

Use-case template The use-case template provides a standard format for describing tasks that users need

to perform with a software system. A use-case definition includes a brief description of the task, descriptions of alternative behaviors and known exceptions that must be handled, and additional information about the task. Chapter 8, "Understanding User Requirements," proposes a use-case template.

Software requirements specification template The SRS template provides a structured, consistent way to organize the product's functional and nonfunctional requirements. Consider adopting more than one template to accommodate the different types or sizes of projects your organization undertakes. This can reduce the frustration that arises when a "one size fits all" template or procedure isn't suitable for your project. Chapter 10, "Documenting the Requirements," describes a sample SRS template.

SRS and use-case defect checklists Formal inspection of requirements documents is a powerful software quality technique. An inspection defect checklist identifies many of the errors commonly found in requirements documents. Use the checklist during the inspection's preparation stage to focus your attention on common problem areas. Chapter 15, "Validating the Requirements," contains sample SRS and use-case defect checklists.

Requirements Management Process Assets

Requirements management process This process describes the actions a project team takes to deal with changes, distinguish versions of the requirements documentation, track and report on requirements status, and accumulate traceability information. The process should list the attributes to include for each requirement, such as priority, predicted stability, and planned release number. It should also describe the steps required to approve the SRS and establish the requirements baseline. For an example of a requirements management process description, see Appendix J of *CMM Implementation Guide* (Caputo 1998).

Change-control process A practical change-control process can reduce the chaos inflicted by endless, uncontrolled requirements changes. The change-control process defines the way that a new requirement or a modification to an existing requirement is proposed, communicated, evaluated, and resolved. A problem-tracking tool facilitates change control, but remember that a tool is not a substitute for a process. Chapter 19, "Change Happens," describes a change-control process in detail.

Requirements status tracking procedure Requirements management includes monitoring and reporting the status of each functional requirement. You'll need to use a database or a commercial requirements management tool to track the status of the many requirements in a large system. This procedure also describes the reports you can generate to view the status of the collected requirements at any time. See Chapter 18, "Requirements Management Principles and Practices," for more about requirements status tracking.

Change control board charter The change control board (CCB) is the body of stakeholders that decides which proposed requirements changes to approve, which to reject, and in which product release each approved change will be incorporated. As described in Chapter 19, the CCB charter describes the composition, function, and operating procedures of the CCB.

Requirements change impact analysis checklist and template Estimating the cost and other impacts of a proposed requirement change is a key step in determining whether to approve the change. Impact analysis helps the CCB make smart decisions. As illustrated in Chapter 19, an impact analysis checklist helps you contemplate the possible tasks, side effects, and risks associated with implementing a specific requirement change. An accompanying worksheet provides a simple way to estimate the labor for the tasks. A sample template for presenting the results of an impact analysis also appears in Chapter 19.

Requirements traceability procedure The requirements traceability matrix lists all the functional requirements, the design components and code modules that address each requirement, and the test cases that verify its correct implementation. The traceability matrix should also identify the parent system requirement, use case, business rule, or other source from which each functional requirement was derived. This procedure describes who provides the traceability data, who collects and manages it, and where it is stored. Chapter 20, "Links in the Requirements Chain," addresses requirements traceability.

Requirements Process Improvement Road Map

Haphazard approaches to process improvement rarely lead to sustainable success. Rather than just diving in, develop a road map for implementing improved requirements practices in your organization. This road map is part of your strategic process improvement plan. If you tried one of the requirements process assessment approaches described in this chapter, you have some ideas about the practices or process assets that would be most helpful to your team. The process improvement road map sequences improvement actions in a way that will yield the greatest benefits with the smallest investment.

Because every situation is different, I can't give you a one-size-fits-all road map. Formulaic approaches to process improvement don't replace careful thinking and common sense. Figure 22-7 illustrates one organization's road map for improving its requirements processes. The desired business goals are shown in the boxes on the right side of the figure, and the major improvement activities are shown in the other boxes. The circles indicate intermediate milestones along the paths to achieving the business goals. (*M1* means *milestone 1*.) Implement each set of improvement activities from left to right. Once you've created a road map, give ownership of each milestone to an individual, who can then write an action plan for achieving that milestone. Then turn those action plans into actions!

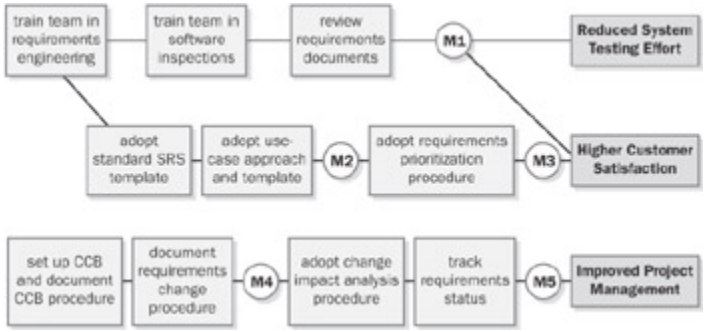


Figure 22-7: Sample requirements process improvement road map.

Next Steps

- Complete the Current Requirements Practice Self-Assessment in Appendix A. Identify your top three improvement opportunities for requirements practices, based on the consequences of shortcomings in your current practices.
- Determine which of the requirements engineering process assets listed in Figure 22-6 are not presently available in your organization but would be useful to have.
- Based on the two preceding steps, develop a requirements process improvement road map patterned after that shown in Figure 22-7. Persuade someone in your organization to take

responsibility for each milestone. Have each milestone owner use the template in Figure 22-4 to write an action plan for implementing the recommendations leading up to his or her milestone. Track the progress of the action items in the plan as they are implemented.

Chapter 23: Software Requirements and Risk Management

Overview

Dave, the project manager for the Chemical Tracking System at Contoso Pharmaceuticals, is meeting with his lead programmer, Helen, and the lead tester, Ramesh. All are excited about the new project, but they remember the problems they ran into on an earlier project called the Pharm-Simulator.

"Remember how we didn't find out that the users hated the Simulator's user interface until beta testing?" Helen asked. "It took us four weeks to rebuild it and retest it. I sure don't want to go through that death march again."

"That wasn't fun," Dave agreed. "It was also annoying that the users we talked to swore they needed a lot of features that no one has used so far. That drug interaction modeling feature took three times longer to code than we expected, and we wound up throwing it out anyway. What a waste!"

"We really had to rush on the Simulator and didn't have time to write detailed requirements," Ramesh remembered. "Half the time the testers had to ask a programmer how some feature was supposed to work so they could test it. Then it turned out that some of the functions the programmers implemented didn't do what the users needed anyway."

"I was really annoyed that the manager who requested the Pharm-Simulator signed off on the requirements without even looking at them," Dave added. "Then remember the constant stream of change requests from people in her department? It's no surprise the project came in five months late and cost almost twice what they budgeted. If that happens again, I'll probably get fired."

Ramesh had a suggestion. "Maybe we should make a list of these problems from the Simulator so we can try to avoid them on the Chemical Tracking System. I read an article on software risk management that said we should identify risks up front and figure out how to prevent them from hurting the project."

"I don't know about that," Dave protested. "We learned a lot from the Simulator, so we probably won't have those problems again. This project isn't big enough to need risk management. If we write down things that could go wrong on the Chemical Tracking System, it'll look like I don't know how to run a software project. I don't want any negative thinkers on this project. We have to plan for success!"

As Dave's final comments suggest, software engineers are eternal optimists. We often expect our next project to run smoothly, despite the history of problems on earlier projects. The reality is that dozens of potential pitfalls can delay or derail a software project. Contrary to Dave's beliefs, software project

managers *must* identify and control their project risks, beginning with requirements-related risks.

A *risk* is a condition that could cause some loss or otherwise threaten the success of a project. This condition hasn't actually caused a problem yet, and you'd like to keep it that way. These potential problems might have an adverse impact on the project's cost, schedule, or technical success, the product's quality, or team effectiveness. Risk management—a software industry best practice (Brown 1996)—is the process of identifying, evaluating, and controlling risks before they damage your project. If something untoward has already happened on your project, it's an issue, not a risk. Deal with current problems and issues through your project's ongoing status tracking and corrective action processes.

Because no one can predict the future with certainty, risk management is a way to minimize the likelihood or impact of potential problems. Risk management means dealing with a concern before it becomes a crisis. This improves the chance of project success and reduces the financial or other consequences of those risks that you can't avoid. Risks that lie outside the team's sphere of control should be directed to the appropriate level of management.

Because requirements play such a central role in software projects, the prudent project manager will identify requirements-related risks early and control them aggressively. Typical requirements risks include misunderstanding the requirements, inadequate user involvement, uncertain or changing project scope and objectives, and continually changing requirements. Project managers can control requirements risks only through collaboration with customers or their representatives. Jointly documenting requirements risks and planning mitigation actions reinforces the customer-development partnership that was discussed in Chapter 2, "Requirements from the Customer's Perspective."

Simply knowing about the risks doesn't make them go away, so this chapter presents a brief tutorial on software risk management (Wiegers 1998b). Later in the chapter, I'll also describe a number of risk factors that can raise their ugly heads during requirements engineering activities. Use this information to launch an attack on your requirements risks before they attack your project.

Fundamentals of Software Risk Management

Projects face many kinds of risks besides those related to project scope and requirements. Dependence on an external entity, such as a subcontractor or another project providing components to be reused, is a common source of risk. Project management is fraught with risks caused by poor estimation, rejection of accurate estimates by managers, insufficient visibility into project status, and staff turnover. Technology risks threaten highly complex and leading-edge development projects. Lack of knowledge is another source of risk, as with practitioners who have insufficient experience with the technologies being used or with the application domain. And ever-changing, imposed government regulations can disrupt the best-laid project plans.

Scary! This is why all projects need to take risk management seriously. Risk management involves scanning the horizon for icebergs, rather than steaming full speed ahead with great confidence that your ship is unsinkable. As with other processes, scale your risk management activities to your project's size. Small projects can get by with a simple risk list, but formal risk management planning is a key element of a successful large-scale project.

Elements of Risk Management

Risk management is the application of tools and procedures to contain project risk within acceptable limits. Risk management provides a standard approach to identify and document risk factors, evaluate their potential severity, and propose strategies for mitigating them (Williams, Walker, and Dorofee 1997). Risk management includes the activities shown in Figure 23-1.[1]

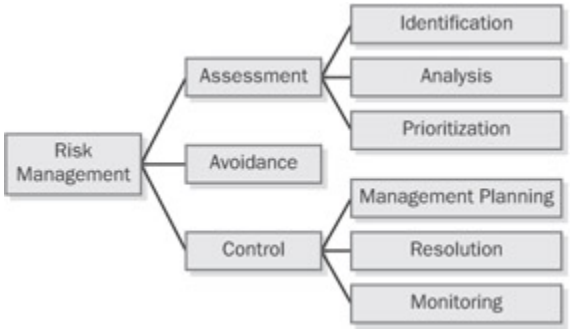


Figure 23-1: Elements of risk management.

Risk assessment is the process of examining a project to identify areas of potential risk. Facilitate *risk identification* with lists of common risk factors for software projects, including the requirements risk factors described in the "Requirements-Related Risks" section later in this chapter (Carr et al. 1993; McConnell 1996). During *risk analysis*, you'll examine the potential consequences of specific risks to your project. *Risk prioritization* helps you focus on the most severe risks by assessing the potential *risk exposure* from each. Risk exposure is a function of both the probability of incurring a loss due to the risk and the potential magnitude of that loss.

Risk avoidance is one way to deal with a risk: don't do the risky thing. You can avoid risks by not undertaking certain projects, by relying on proven rather than cutting-edge technologies, or by excluding features that will be especially difficult to implement correctly. Software development is intrinsically risky, though, so avoiding risk also means losing an opportunity.

Most of the time, you'll have to perform *risk control* activities to manage the top-priority risks you identified. *Risk management planning* produces a plan for dealing with each significant risk, including mitigation approaches, contingency plans, owners, and timelines. Mitigation actions try either to prevent the risk from becoming a problem at all or to reduce the adverse impact if it does. The risks won't control themselves, so *risk resolution* involves executing the plans for mitigating each risk. Finally, track your progress toward resolving each risk item through *risk monitoring*, which should become part of your routine project status tracking. Monitor how well your risk mitigation actions are working, look for new risks that have popped up, retire risks whose threat has passed, and update the priorities of your risk list periodically.

Documenting Project Risks

It's not enough to simply recognize the risks that face your project. You need to manage them in a way that lets you communicate risk issues and status to stakeholders throughout the project's duration. Figure 23-2 shows a template for documenting an individual risk statement. You might find it more convenient to store this information in tabular form in a spreadsheet, which makes it easy to sort the list of risks. Rather than embedding it in your project management plan or software requirements specification, keep the risk list as a stand-alone document so that it's easy to update throughout the project's duration.

ID:

<sequence number>

Date Opened:

<date the risk was identified>

Date Closed:

<date the risk was closed out>

Description:

<description of the risk in the form "condition-consequence">

Probability:

<the likelihood of this risk becoming a problem>

Impact:

<the potential damage if the risk does become a problem>

Exposure:

<probability multiplied by impact>

Mitigation Plan:

<one or more approaches to contril, avoid, minimize, or otherwise mitigate the risk>

Owner:

<individual responsible for resolving the risk>

Date Due:

<date by which the mitigation actions are to be implemented>

Figure 23-2: Risk item tracking template.

Use a *condition-consequence* format when you document risk statements. That is, state the risk condition that you are concerned about, followed by the potential adverse outcome—the consequence—from that condition. Often, people who suggest risks state only the condition ("the customers don't agree on the product requirements") or the consequence ("we can satisfy only one of our major customers"). Pull these statements together into the condition-consequence structure: "The customers don't agree on the product requirements, so we will only be able to satisfy one of our major customers." One condition might lead to several consequences, and several conditions can result in the same consequence.

The template provides locations to record the probability of a risk materializing into a problem, the negative impact on the project as a result of that problem, and the overall risk exposure. I estimate the probability on a scale from 0.1 (highly unlikely) to 1.0 (certain to happen), and the impact on a relative scale of 1 (no sweat) to 10 (deep tapioca). Even better, try to rate the potential impact in units of lost time or money. Multiply the probability by the impact to estimate the exposure from each risk.

Don't try to quantify risks too precisely. Your goal is to differentiate the most threatening risks from those you don't need to tackle immediately. You might find it easier simply to estimate both probability and impact as high, medium, or low. Those items that have at least one *high* rating demand your early attention.

Use the Mitigation Plan field to identify the actions you intend to take to control the risk. Some mitigation strategies try to reduce the risk probability, and others reduce the impact. Consider the cost of mitigation when planning. It doesn't make sense to spend $20,000 to control a risk that could cost you only $10,000. You might also devise contingency plans for the most severe risks to anticipate what actions to take if, despite your efforts, the risk does affect your project. Assign every risk that you're going to control to an individual owner, and set a target date for completing the mitigation actions. Long-term or complex risks might require a multistep mitigation strategy with multiple milestones.

Figure 23-3 illustrates a risk that the Chemical Tracking System team leaders discussed at the beginning of this chapter. The team estimated the probability and impact on the basis of their previous experience. Until they evaluate other risk factors, they won't know how serious a risk exposure of 4.2 is. The first two mitigation approaches reduce the probability of this risk becoming a problem by increasing user involvement in the requirements process. Prototyping reduces the potential impact by seeking early feedback on the user interface.

ID:

1

Date Opened:

3/4/2003

Date Closed:

(open)

Description:

Insufficient user involvement in requirements elicitation could lead to extensive user interface rework after beta testing.

Probability:

0.6

Impact:

7

Exposure:

4.2

Mitigation Plan:

1. Gather usability requirements early in Phase 1.
2. Hold facilitated workshops with product champions to develop the requirements.
3. Develop a throwaway user interface prototype of core functionality with input from the product champions and a hum an factors consultant. Have other users evaluate the prototype.

Owner:

Helen

Date Due:

Hold workshop by 4/16/2003

Figure 23-3: Sample risk item from the Chemical Tracking System.

Planning for Risk Management

True Stories A risk list is not the same as a risk management plan. For a small project, you can include your plans for controlling risks in the software project management plan. A large project should write a separate risk management plan that spells out the approaches it intends to take to identify, evaluate, document, and track risks. This plan should include the roles and responsibilities for the risk management activities. A risk management plan template is available from *http://www.processimpact.com/goodies.shtml*. Many projects appoint a project risk manager to be responsible for staying on top of the things that could go wrong. One company dubbed their risk manager "Eeyore," after the gloomy Winnie-the-Pooh character who constantly bemoaned how bad things could become.

Trap Don't assume that risks are under control just because you identified them and selected mitigation actions. Follow through on the risk management actions. Include enough time for risk management in the project schedule so that you don't waste your investment in risk planning. Include risk mitigation activities, risk status reporting, and updating of the risk list in your project's task list.

Establish a rhythm of periodic risk monitoring. Keep the ten or so risks that have the highest risk exposure highly visible, and track the effectiveness of your mitigation approaches regularly. When a mitigation action is completed, reevaluate the probability and impact for that risk item and then update the risk list and any other pending mitigation plans accordingly. A risk is not necessarily under control simply because the mitigation actions have been completed. You need to judge whether your mitigation approaches have reduced the exposure to an acceptable level or whether the opportunity for a specific risk to become a problem has passed.

Out of Control

True Stories A project manager once asked me what to do if the same items remained on his top five risk list week after week. This suggests that the mitigation actions for those risks either aren't being implemented or aren't effective. If your mitigation actions are doing the trick, the exposure from risks that you are actively attempting to control will decrease. This lets other risks that were less threatening than the initial top five float up the risk list and engage your attention. Periodically reassess the probability of each risk materializing and the potential loss if it does to see whether your risk mitigation activities are getting the job done.

[1] Adapted from McConnell, Steve. 1996. *Rapid Development: Taming Wild Software Schedules*. Redmond, Wash.: Microsoft Press.

PREVIOUS NEXT
PREVIOUS NEXT

Requirements-Related Risks

The risk factors described on the following pages are organized by the requirements engineering subdisciplines of elicitation, analysis, specification, validation, and management. Techniques are suggested that can reduce the risk's probability or impact. This list is just a starting point; accumulate your own list of risk factors and mitigation strategies, based on the lessons you learn from each project. Leishman and Cook (2002) describe additional risks related to software requirements. Use the items here to prompt your thinking when identifying requirements risks. Be sure to write your risk statements in the condition-consequence format.

Requirements Elicitation

Product vision and project scope Scope creep is more likely if the stakeholders lack a clear, shared understanding of what the product is supposed to be (and not be) and do. Early in the project, write a vision and scope document that contains your business requirements, and use it to guide decisions about new or modified requirements.

Time spent on requirements development Tight project schedules often pressure managers and customers into glossing over the requirements because they believe that if the programmers don't start coding immediately, they won't finish on time. Projects vary widely depending on their size and application class (such as information systems, systems software, commercial, or military), but a rough guideline is to spend about 10 to 15 percent of your project effort on requirements development activities (Rubin 1999). Record how much effort you actually spend on requirements development for each project so that you can judge whether it was sufficient and improve your planning for future projects.

Completeness and correctness of requirements specifications To ensure that the requirements specify what the customer really needs, apply the use-case technique to elicit requirements by focusing on user tasks. Devise specific usage scenarios, write test cases from the requirements, and have customers develop their acceptance criteria. Create prototypes to make the requirements more meaningful for users and to elicit specific feedback from them. Enlist customer representatives to inspect the requirements specifications and analysis models.

Requirements for highly innovative products It's easy to misgauge market response to products that

are the first of their kind. Emphasize market research, build prototypes, and use customer focus groups to obtain early and frequent feedback about your innovative product visions.

Defining nonfunctional requirements Because of the natural emphasis on product functionality, it's easy to neglect nonfunctional requirements. Query customers about quality characteristics such as performance, usability, integrity, and reliability. Document these nonfunctional requirements and their acceptance criteria as precisely as you can in the SRS.

Customer agreement on product requirements If the diverse customers for your product don't agree on what you should build, someone will be unhappy with the result. Determine who the primary customers are, and use the product champion approach to get adequate customer representation and involvement. Make sure you're relying on the right people for decision-making authority on the requirements.

Unstated requirements Customers often hold implicit expectations that are not communicated or documented. Try to identify any assumptions the customers might be making. Use open-ended questions to encourage customers to share more of their thoughts, wishes, ideas, information, and concerns than you might otherwise hear.

Existing product used as the requirements baseline Requirements development might not be deemed important on next-generation or reengineering projects. Developers are sometimes told to use the existing product as their source for requirements, with a list of changes and additions. This forces the developer to glean the bulk of the requirements through reverse engineering of the current product. However, reverse engineering is an inefficient and incomplete way to discover requirements, and no one should be surprised if the new system has some of the same shortcomings as the legacy system. Document the requirements that you discover through reverse engineering, and have customers review those requirements to ensure that they are correct and still relevant.

Solutions presented as needs User-proposed solutions can mask the users' actual needs, lead to automating ineffective business processes, and pressure developers into making poor design decisions. The analyst must drill down to understand the intent behind a solution the customer has presented.

Requirements Analysis

Requirements prioritization Ensure that every functional requirement, feature, or use case is prioritized and allocated to a specific system release or iteration. Evaluate the priority of every new requirement against the body of work remaining to be done so that you can make smart trade-off decisions.

Technically difficult features Evaluate the feasibility of each requirement to identify those that might take longer than anticipated to implement. Success always seems just around the corner, so use your project status tracking to watch for requirements that are falling behind their implementation schedule. Take corrective action as early as possible.

Unfamiliar technologies, methods, languages, tools, or hardware Don't underestimate the learning curve of getting up to speed with new techniques that are needed to satisfy certain requirements. Identify those high-risk requirements early on, and allow sufficient time for false starts, learning, experimentation, and prototyping.

Requirements Specification

Requirements understanding Different interpretations of the requirements by developers and

customers lead to expectation gaps, in which the delivered product fails to satisfy customer needs. Formal inspections of requirements documents by teams that include developers, testers, and customers can mitigate this risk. Trained and experienced requirements analysts will ask the right questions of customers and write high-quality specifications. Models and prototypes that represent the requirements from multiple perspectives will also reveal fuzzy, ambiguous requirements.

Time pressure to proceed despite TBDs It is a good idea to mark areas of the SRS that need further work with TBD (to be determined), but it's risky to proceed with construction if these TBDs haven't been resolved. Record the name of the person responsible for closing each TBD and the target date for resolution.

Ambiguous terminology Create a glossary to define business and technical terms that might be interpreted differently by different readers. In particular, define any terms that have both common and technical or domain-specific meanings. Create a data dictionary that defines the data items and structures. SRS reviews can help participants reach a common understanding of key terms and concepts.

Design included in requirements Designs that are included in the SRS place unnecessary constraints on the options available to developers. Unnecessary constraints inhibit the creation of optimal designs. Review the requirements to make sure they emphasize what needs to be done to solve the business problem, rather than stating how it will be solved.

Requirements Validation

Unvalidated requirements The prospect of inspecting a lengthy SRS is daunting, as is the idea of writing test cases very early in the development process. However, if you confirm the correctness and quality of the requirements before construction begins, you can avoid considerable expensive rework later in the project. Include time and resources for these quality activities in the project plan. Gain commitment from your customer representatives to participate in requirements inspections, because only customers can judge whether the stated requirements will meet their needs. Also, perform incremental, informal reviews to find problems as early and cheaply as possible.

Inspection proficiency If inspectors do not know how to properly inspect requirements documents and how to contribute to effective inspections, they might miss serious defects. Train all team members who will participate in inspections of requirements documents. Invite an experienced inspector from your organization or an outside consultant to observe, and perhaps to moderate, your early inspections to coach the participants.

Requirements Management

Changing requirements You can reduce scope creep by using a vision and scope document as the benchmark for approving changes. A collaborative requirements elicitation process with extensive user involvement can cut requirements creep nearly in half (Jones 1996a). Quality-control practices that detect requirements errors early reduce the number of modifications requested later on. To reduce the impact of changing requirements, defer implementation of those requirements that are most likely to change until they're pinned down, and design the system for easy modifiability.

Requirements change process Risks related to the way changes to requirements are handled include not having a defined change process, using an ineffective change mechanism, and incorporating changes that bypass the process. It takes time to develop a culture and discipline of change management. A requirements change process that includes impact analysis of proposed changes, a change control board to make decisions, and a tool to support the defined procedure is an important starting point.

Unimplemented requirements The requirements traceability matrix helps to avoid overlooking any requirements during design, construction, or testing.

Expanding project scope If requirements are poorly defined initially, further definition can expand the scope of the project. Vaguely specified areas of the product will consume more effort than anticipated. The project resources that were allocated according to the initial incomplete requirements might be insufficient to implement the full scope of user needs. To mitigate this risk, plan on a phased or incremental delivery life cycle. Implement the core functionality in the initial release, and elaborate the system's capabilities in later iterations.

PREVIOUS NEXT
PREVIOUS NEXT

Risk Management Is Your Friend

True Stories A project manager can use risk management to raise the awareness of conditions that could cause the project to suffer. Consider the manager of a new project who's concerned about getting appropriate users involved in requirements elicitation. The astute manager will realize that this condition poses a risk and will document it in the risk list, estimating the probability and impact based on previous experience. If time passes and users still are not involved, the risk exposure for this item will increase, perhaps to the point where it compromises the project's success. I've been able to convince managers to postpone a project that could not engage sufficient user representatives by arguing that we shouldn't waste the company's money on a doomed project.

Periodic risk tracking keeps the project manager apprised of the threat from identified risks. Escalate risks that aren't adequately controlled to senior managers, who can either initiate corrective actions or make a conscious business decision to proceed despite the risks. Risk management helps you keep your eyes open and make informed decisions, even if you can't control every adversity your project might encounter.

Next Steps

- Identify several requirements-related risks facing your current project. Don't identify current problems as risks, only things that haven't happened yet. Document the risk factors by using the template in Figure 23-2. Suggest at least one possible mitigation approach for each risk.
- Hold a risk brainstorming session with key project stakeholders. Identify as many requirements-related risk factors as you can. Evaluate each factor for its probability of occurrence and relative impact, and multiply these together to calculate the risk exposure. Sort the risk list in descending order by risk exposure to identify your top five requirements-related risks. Assign each risk to an individual to implement mitigation actions.

PREVIOUS NEXT