Chapter 4

Project Managers

In this chapter:

Project managers can go by many names: product manager, program manager, business manager, stakeholder, analyst, advocate, project lead, team lead, or the lone consultant. Whatever title is used, the responsibility is the same—to see the project through to its successful completion. In other words, the project manager needs to ensure that the project goes live on *time*, on *budget*, *bug free*, and meeting all requirements. It is with the project manager role that we begin the Team System story.

In Team System, the Project Manager role doesn't contribute designs, code, or tests. Project managers remain strictly in the management position and perform such duties as creating and configuring a project, managing work items, managing the content on the Project Portal, and reviewing any generated reports.

Organizing the Team

Before we launch Team System, let's set up the team. I'm assuming that you have a team, you know their names, and, most important, you know what roles each member will play on the team *and* inside Team System. Remember that a team member can play more than one role. For example, Sharon could be both an architect and a developer. As project manager, you need to know all these details before starting a new project—primarily to configure security.

Note From a software standpoint, Team System comes in editions specific to the various roles. If you have a team member who plays two roles, such as architect and developer, you'll need to install licensed copies of both Microsoft® Visual Studio® 2005 Team Edition for Software Architects and Visual Studio 2005 Team Edition for Software Developers. Installing a licensed copy of the Team System Suite is another option.

Even team members who don't correlate directly to an edition of Team System will still be able to interact by one of the other client interfaces, such as Microsoft Office Excel®, Microsoft Office Project, or the browser. Anyone in the company who has the correct permission levels can add or edit documents to the project portal.

Note Team System doesn't provide any Web-based work item support. Only collateral documents uploaded to the portal can be viewed and edited. This might change in future versions. Until then, it certainly is a great opportunity for integration. See Chapter 9 for more information on customizing and extending Team System.

To prepare for the new project, you should know the following information about each Team System member:

- Name
- Windows domain and user ID
- Role that the person will play in Team System (project manager, network infrastructure architect, solutions architect, developer, tester, or observer)
- Which edition of Team System the person has installed (because this will affect what he or she can and cannot physically accomplish)

Tip I know of several one-man-shop consultants who plan to use Team System. They will most likely decide to install and use Team System Suite. Installing Team Foundation Server might be a way to gain access to the Team Foundation Version Control and Team Build utilities. However, they will still be underutilizing Team System because they won't really be using the collaboration features.

Starting a New Project

You can create a new project from Team Explorer. It's a simple process—if you have all the answers to the questions. Besides the name and description of the project, you'll be prompted to configure the following project areas:

- **Methodology** Choose between MSF for Agile Development, MSF for CMMI Process Improvement, or any other methodologies that have been loaded into Team Foundation Server
- **Project portal title and description** The friendly name and description of the Microsoft Windows® SharePoint® Services portal your team will use
- **Source version control folder and branching information** New folder, branched folder, or no source control

Selecting a Methodology

As I mentioned in Chapter 1, Team System will ship with two methodologies: MSF for Agile Development (MSF Agile) and MSF for CMMI Process Improvement. Depending on the type of project and the characteristics of your team, either methodology might make sense. Each has its own strengths and weaknesses. When creating a new team project, you'll be

able to select one of these two methodologies. (See Figure 4-1.) If you have any other custom methodologies installed, they will also be available in the drop-down list here.

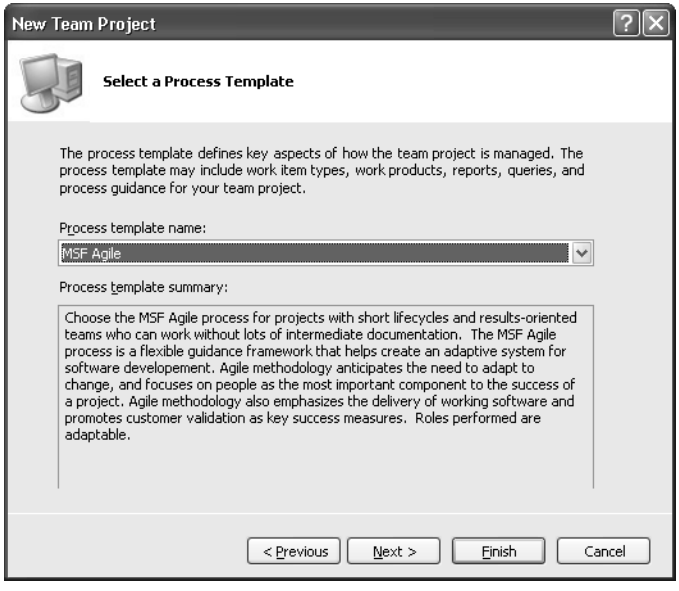

Figure 4-1 Selecting a new project's methodology

Team System's extensibility allows it to be modified and extended as your software development process evolves and improves over time. As your team evolves and matures, its methodology needs might also evolve and mature as well. Team System is designed to support that. With some free time and an understanding of how Team System integrates with a methodology, you can export an existing methodology—MSF for Agile Development, for example. These exported files can be edited to suit your development team's needs. The new files can be imported back into Team System and named something more topical, such as Adventure Works Agile Development.

More info For more information on MSF for Agile Development, be sure to read Chapter 8. For more information on extending Team System, especially with regard to methodologies, be sure to read Chapter 9.

Configuring the Project Portal

Team System relies on Windows SharePoint Services (WSS) for disseminating information to the extended team. I use the word "extended" here to mean two things: team members who are geographically dispersed and team members who might not play a direct role in Team System. With WSS, integrating members of an extended team into Team System becomes a no-brainer. A team member with permissions can upload, edit, or simply view any type of document hosted in WSS.

Note Team members must authenticate to Team Foundation Server by way of Windows authentication. This might limit who can work remotely.

Anyone who has used WSS knows it has many collaborative advantages, including the following ones:

- Tight integration with Office 2003
- Document check-in, check-out, and version control
- Configurable security settings
- Customizable Web-part technology

Team System will set up the Project Portal site for you. When you reach that step in the wizard, you'll be prompted for the project portal name and description. Because it's assumed that Windows SharePoint Services is installed on the application tier of Team System, the project creation step will happen automatically. In other words, the Web service being called by Team Explorer will create the WSS site and configure it for you. You don't have to know anything about WSS to make this happen. The URL is provided at the bottom of the dialog box. Just copy it and e-mail it to your team members, letting them know that the portal has been created.

After the wizard has run and the site is created, you may want to customize it for your team. As with any WSS site, you can customize it in many ways. New document libraries can be created from Team Explorer, and documents can be uploaded and managed. Beyond that, you'll need to tweak and customize the portal by using the WSS administrative screens.

Here are some customizations that you might want to perform on the portal:

- Grant any additional, non–Team System users permission to access the portal.
- Add Web parts, such as any third-party reports.
- Change site title and description.
- Redesign the page layout.
- Change the theme.

Configuring Version Control Settings

It's likely that all Team System projects will want to take advantage of the Team Foundation Server version control capability. This capability is enabled by default, but you can choose any of the following options (as shown in Figure 4-2):

- Create an empty version control folder.

- Create a new version control branch.
- Do not create a version control folder at this time.

The default action is to create an empty version control folder, which is nice and safe, but doing this doesn't help start your project in any way. What I mean is that if you can *branch* from an existing project, you'll already have some content from which to work. As I mentioned in Chapter 2, branching is a powerful feature of Team Foundation Version Control. This flexibility is evident when you're creating a team project that is a new version of an existing project, has similar or shared functionality, or must diverge in functionality or focus from its core.

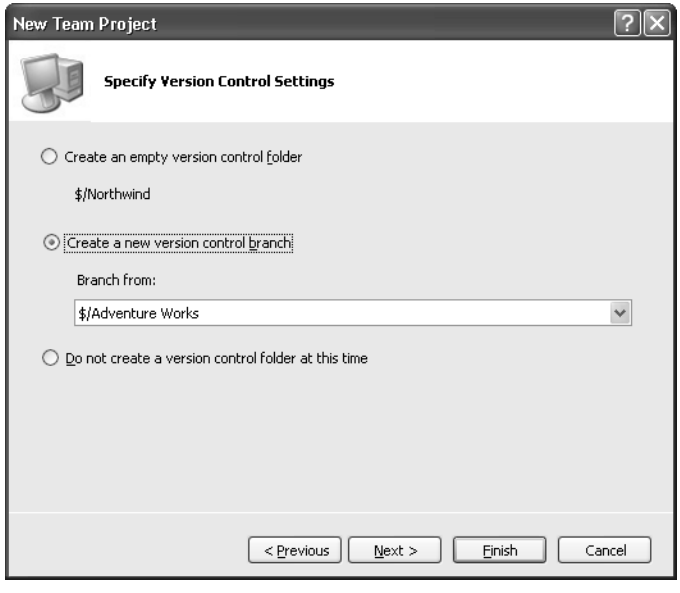

Figure 4-2 Branching an existing project

Selecting the source control option is the last step in creating a new project. The wizard will then begin creating the new project, which includes updating many database structures and copying many templates into the portal. This process can take several moments to complete.

Managing the Ongoing Project

Once Team System has created the project, your architects, developers, and testers can begin contributing diagrams, code, and tests to the project. No additional settings are required. In other words, work can commence immediately. I would, however, recommend configuring security, classifications, and check-in policies before you embark on any software development effort. To further aid your team, you might want to upload any process guidance or other related documents to the portal. As the project goes forward, you might need to tweak some of these settings, especially if new team members arrive or existing team members leave.

Configuring Security

Team System is very secure, which is good, because tight security, even when dealing with trusted developers, is a requirement at many companies nowadays. Financial institutions, government, and any Sarbanes-Oxley-compliant companies will judge a finished product by its security support and the level of security that surrounded its construction. Security is also a good way to protect the team against itself. If you lock down privileged operations to only a select few members, any inexperienced team members won't be able to accidentally change critical information.

Team System can integrate with Active Directory®, so it is able to map Microsoft Windows users and groups directly to its own predefined roles and permissions. Enabling this role-based security is done in two steps. First, you map in the Active Directory users and groups. Then you assign the relevant permissions.

Mapping users and groups to Team System is done by right-clicking the team project inside Team Explorer, selecting Team Project Settings, and clicking Groups. Team System defines both *global* groups as well *project* groups. (See Figure 4-3.)

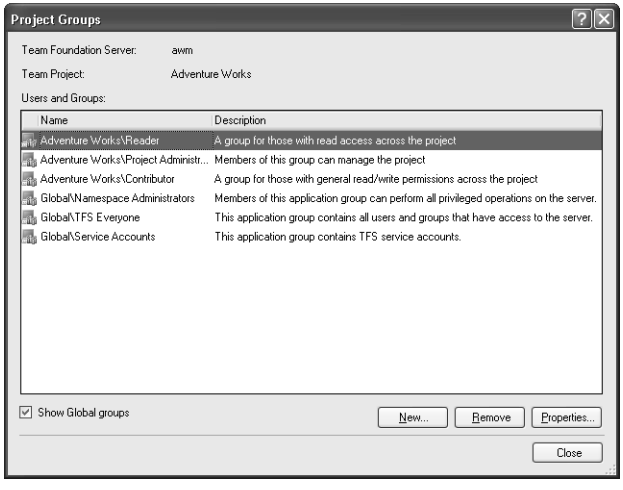

Figure 4-3 Built-in Team System project and global groups

Here are the built-in *global* groups:

- **Namespace Administrators** Includes members who can perform all privileged operations
- **TFS Everyone** Includes all users and groups that have access to the server
- **Service Accounts** Includes Team Foundation Server service accounts

Here are the built-in *project* groups:

- **Project Administrators** Includes members who can manage the project

- **Contributor** Includes members who have general read/write permissions across the project
- **Reader** Includes members who have read access across a project

Tip Team System allows you to create new project groups so that you can classify your team members any way that you want. These groups can then be assigned whatever permissions you'd like, thus enabling you to create your own customized Team System roles.

The next step is to assign the appropriate Team System permissions to these users and groups. Figure 4-4 shows the level of permissions allowed. Team System's level of security even allows you to secure the state transitions of work item types. In other words, you can specify which team members can log bugs in and, more important, which team members can transition them to fixed or closed.

Note Team Foundation Server supports Windows 2000 or 2003 native active directory. This environment is required for a dual-server installation. If you want to install Team Foundation Server onto a single-server, an Active Directory environment is not required, and you can simply install onto a workgroup server.

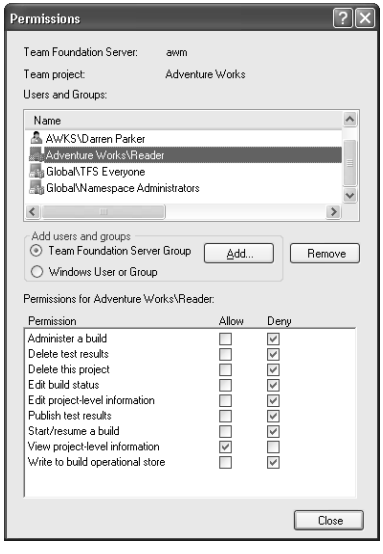

Figure 4-4 Assigning permissions to users and groups

Best Practice A best practice is to assign permissions to groups of users, and not directly to the individual users. This has long been a best practice in Windows, and it also applies to Team System.

Creating Classifications

Projects can have a hierarchy of *classifications*. These represent generic organization units by which you can classify your work items. You're free to add any classifications or child classifications to build your container hierarchy in whatever way you want it. I recommend good names—names that your team members immediately recognize as applying to them. Classifications are optional, however.

Here are some examples of classifications:

- Design, Code, Bugs
- North America, Europe, Asia
- Top Secret, Secret, Classified, For Official Use Only, Unclassified
- Red, Green, Blue

For example, you could have three geographically separated development teams, all working on the same project. To keep the work items, such as tasks and bugs, partitioned for each team, you could create North America, Europe, and Asia classifications, and then select the respective classification when creating work items.

Creating Iterations

Projects can also have *iterations*, which represent milestones or specific epochs in the project's software development life cycle, as seen in Figure 4-5. Depending on the methodology, Team System creates some default iterations, such as *Iteration 0*, *Iteration 1*, and *Iteration 2*. You should rename these and strive to have very self-evident iteration names, as you cannot associate any additional metadata with them.

Here are some examples of iterations:

- Alpha, Beta, Release Candidate, RC1, RC2, Gold, RTM
- Phase 1, Phase 2, Phase 3
- Proof of Concept, Executive Buyoff, Available to Public, Certified

For example, your project might go through several phases, such as Alpha, Beta, and then release. To keep the work items, such as tasks and bugs, partitioned for each phase, you could select the respective iteration when creating work items.

As you can see, the iterations are very flexible. They are just names. To fit your organization and methodology, you can have as many as you want and name them whatever you want. As you move through Team System, any work item you create can be associated with a classification and an iteration.

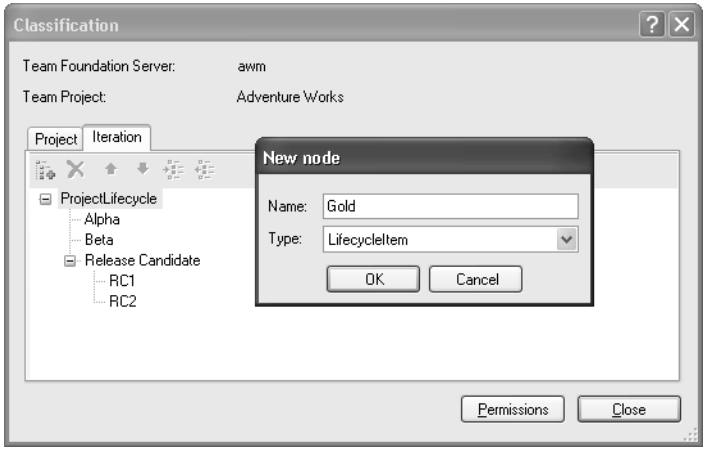

Figure 4-5 Creating iterations for the various phases of your project

Security can be applied to either classifications or iterations. In other words, you're able to lock down by user or Team System role who has permission to do what within these classifications and iterations. Figure 4-6 shows the level of permissions you can assign. Refer to the online documentation for the specifics of each permission.

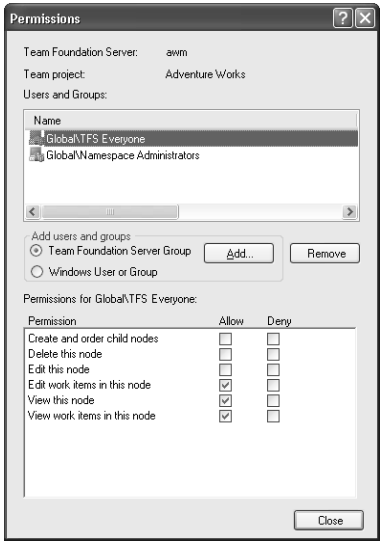

Figure 4-6 Assigning permissions to a classification or iteration

Setting Check-In Policies

One of the most powerful features of Team System is its support of *check-in policies*, which give the project manager, architect, or lead developer a great deal of control over the project and its team. Keep in mind that when used properly, these policies can result in better quality code being checked in; however, developers can get lazy and bypass the policies

or revolt altogether if placed in an environment with too many policies. Diplomacy and psychology should be used when introducing new policies to your team. These discussions, however, are beyond the scope of this book. Figure 4-7 shows a project with the Clean Build and Work Items policies enabled.

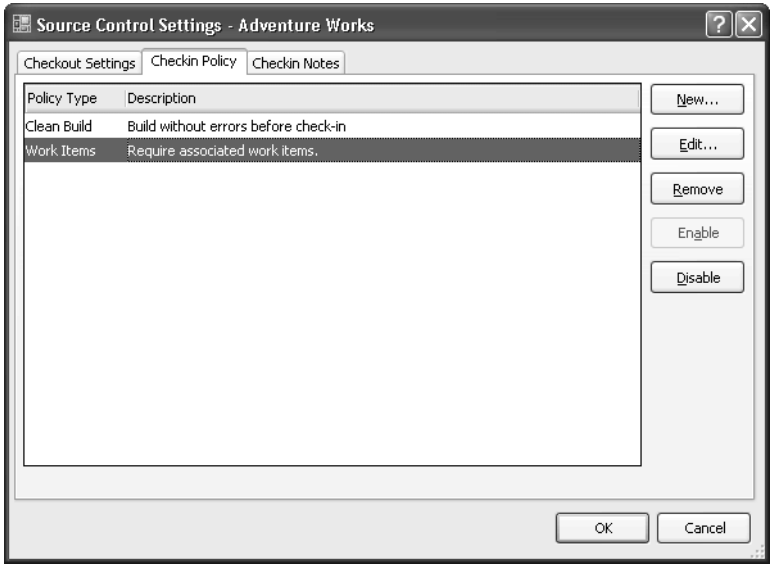

Figure 4-7 Enabling a project's check-in policies

Here are some check-in policies that Team System supports:

- **Clean Build** The code must be error free before check-in is allowed.
- **Static Analysis** The code must pass a static analysis test before check-in is allowed.
- **Testing Policy** The code must pass one or more tests before check-in is allowed.
- **Work Items** The code must be associated with one or more work items before check-in is allowed.
- **Notes** One or more notes (such as reviewer's name, issues, status, hours, and so on) must be associated with the code before check-in is allowed.
- **Custom** Custom check-in policies can be created as well. Be sure to read Chapter 9 for more information.

Note Project managers can also decide whether a project needs the ability to have multiple users checking out the same files at the same time. This is an advanced feature, but one that Team System supports very well with its merging technology. The ability for multiple users to check out and work on the same file is not appropriate for every team. In my opinion, training on the types of conflicts that can occur and hands-on experience resolving conflicts should be required before a team tries to use this feature.

Uploading Documents and Other Assets

Team System relies heavily on WSS for disseminating information to the distributed team. As I've already mentioned, a number of application clients interact with Team System, but none is more ubiquitous than the browser. The developers whom I've met are constantly online, searching for sample code or downloading guidance. Microsoft knows this and, as an acknowledgment, has even integrated the browser into the start page of Visual Studio. The project portal, however, is more than just a nice-looking site. It's a fully managed WSS portal, written in ASP.NET and fully configurable and customizable. Anyone who has used WSS knows it has many collaborative advantages over other, non-managed sites.

By default, Team System uploads process guidance documents based on the methodology selected. (See Figure 4-8.) This is just one way that the portal can empower the team—by providing how-to and best-practice documentation for the actual construction of the software.

Note As you might expect, this documentation can be customized, improved, and associated with the methodology template so that future Team System projects will contain newer, more focused documentation. Chapter 9 discusses how to do this.

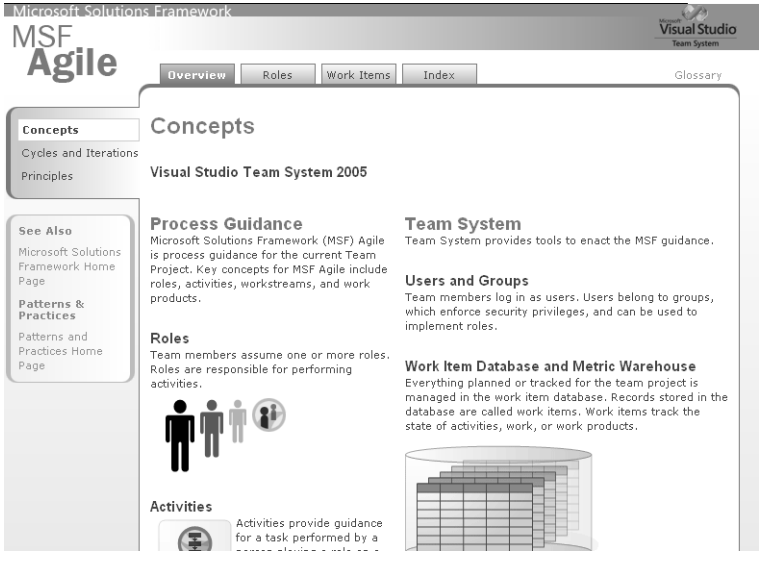

Figure 4-8 Default process guidance documents available through the project portal

Here are just a few examples of other documents you can upload to the project portal:

- **Business documents** Bids, estimates, accounting spreadsheets, functional specifications, team contact information

- **Technical documents** Technical specifications, UML diagrams, database designs, graphical representations of current systems, other Microsoft Visio® diagrams
- **Guidance** Best-practice examples, whitepapers, how-to guides

Adding and Managing Work Items

I think of work items as being the *currency* of Team System—that is, work items enable the collaboration and the "development commerce," if you will. In other words, these documents represent the communication packets traveling from teammate to teammate. Work items can be tasks, bugs, risks, scenarios, Quality of Service (QoS) requirements, or any other type defined by the methodology you're using. Because the team will be looking to the project manager to kick things off, he or she must be an expert in creating and assigning work items to initiate and maintain the workflow.

A work item is just a database record that Visual Studio 2005 Team Foundation Server uses to track the assignment and state of work. The MSF for Agile Software Development process defines five work items to assign and track work:

- **Scenario** Defines a single path of user interaction within the system
- **Quality of Service (QoS) requirement** Documents characteristics of the system, such as performance, load, availability, stress, accessibility, serviceability, and maintainability
- **Task** Work that needs to be done
- **Bug** A potential problem that exists or has existed in the system
- **Risk** Any probable event or condition that can have a potentially negative outcome on the project

More info Please see Chapter 8 for more information on MSF for Agile Software Development and these work item types.

The MSF for CMMI Process Improvement process defines seven work items to assign and track work:

- **Bug** A potential problem that exists or has existed in the system
- **Change Request** A proposed change to some part of the product or baseline
- **Issue** An event or situation that may block work or is currently blocking work on the product
- **Requirement** What the product needs to do to solve the customer problem
- **Review** Results of a design or code review

- **Risk** Any probable event or condition that can have a potentially negative outcome on the project
- **Task** Work that needs to be done

More info Please see Chapter 8 for more information on MSF for CMMI Process Improvement and these work item types.

Choosing the Best Work-Item Client Application

To add and manage work items, the project manager—or any team member for that matter—will need to use one of the three client applications that I mentioned in the previous chapter: Visual Studio 2005, Microsoft Excel, or Microsoft Project. Most developers feel at home inside Visual Studio 2005. They enjoy its do-anything attitude, meaning that they get 100 percent of the Team System features and capabilities. Visual Studio 2005, however, makes a terrible spreadsheet, and its Gnatt chart capabilities are also lacking. I think you see my point—each application has its own strengths and weaknesses. For more information on how to decide which client application is best for you, please refer to the previous chapter.

Adding Scenarios

In MSF for Agile Software Development, a *scenario* records a single path of user interaction through the system. As that user attempts to reach a goal, the scenario records the specific steps that he or she will take in attempting to reach it. Some scenarios will record a successful path; others will record an unsuccessful one. When writing scenarios, you should be as specific as possible. There are an infinite number of possible scenarios for all but the most trivial of systems, so it is important to be discerning in deciding which scenarios to write.

Scenarios begin in the Active state. The business analyst or project manager creates the scenario, provides a descriptive title, and fills in the description field with as much detail as possible about the scenario. (See Figure 4-9.) When the scenario is fully written, the business analyst assigns it to a lead developer. The scenario remains in the Active state while it is being implemented. The lead developer coordinates efforts with other developers to implement the scenario.

During a project's life cycle, a scenario can be any of the following states:

- **New** A scenario is activated as a new scenario when it is first created.
- **Active** Scenarios begin in the Active state. The business analyst creates the scenario, provides a descriptive title, and fills in the Description field with as much detail as possible about the scenario. When the scenario is fully written, the business analyst assigns it to a lead developer. The Specified field is set to Yes, and the scenario remains in the Active state while it is being implemented. The lead developer coordinates efforts with other developers to implement the scenario.

- **Resolved** When the scenario is implemented in code, the lead developer sets the state to Resolved. The lead developer also assigns the scenario to a tester so that testing can begin.
- **Closed** The tester closes a scenario if it passes its tests. A scenario is also closed if it is Deferred, Removed, or Split into more scenarios.

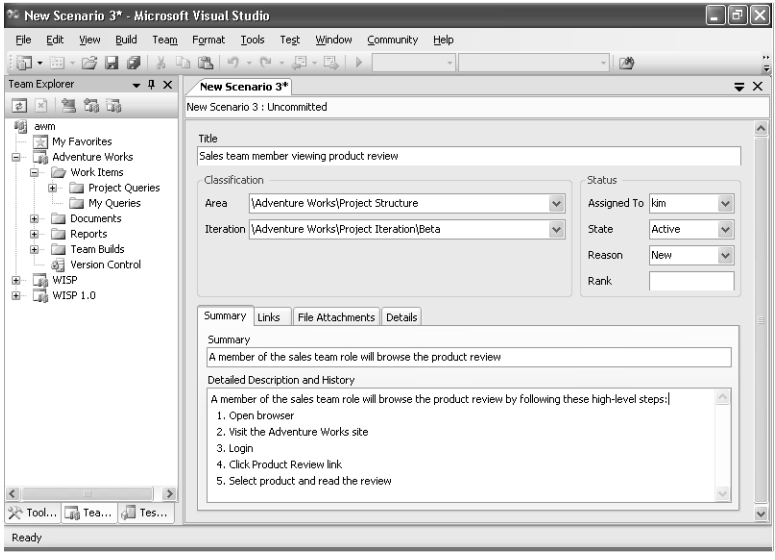

Figure 4-9 Adding a new scenario to Team System

During a scenario's life cycle, a scenario may make any of the following state transitions:

- New to Active—Occurs when a new scenario is first created.
- Active to Resolved (Completed)—Occurs when the development team completes writing code for the scenario and the lead developer assigns the scenario to a tester.
- Active to Resolved (Split)—Occurs when further review indicates that the scenario is too large or that it needs more granular definition.
- Active to Resolved (Deferred)—Occurs if it cannot be implemented in the current iteration. A scenario can be deferred because the team does not have enough time or because blocking issues were discovered.
- Active to Resolved (Removed)—Occurs if it is no longer deemed necessary to implement.
- Resolved to Closed (Completed)—Occurs when the tester indicates that it has passed its tests.
- Resolved to Closed (Split)—Occurs because further review indicated that the scenario was too large or that it needed more granular definition.
- Resolved to Closed (Deferred)—Occurs because it could not be implemented in the current iteration.

- Resolved to Closed (Removed)—Occurs because it is no longer deemed necessary to implement.
- Resolved to Active (Test Failed)—Occurs if the scenario fails to pass any tests and the tester must return the scenario to an active state and reassign it to the original lead developer.
- Closed to Active (Reactivated)—Occurs because of a change in functionality.

Adding Quality of Service Requirements

In MSF for Agile Software Development, a Quality of Service(QoS) requirement documents characteristics of the system, such as performance, load, availability, stress, accessibility, serviceability, and maintainability. These requirements usually take the form of constraints on how the system should operate.

Note QoS requirements are not the same thing as *requirements*. MSF for Agile Software Development doesn't directly help with the gathering, synthesizing, or storing of software requirements. A project manager could work around this by using another methodology or Excel to maintain a list of requirements and map them to other Team System work items. Another option would be to customize Team System and add a requirement work item type.

The QoS requirements can be created in the QoS requirements list found in the requirements folder in the document library or by using the new work item menu in Team Explorer. The business analyst or project manager creates the requirement, provides a descriptive title, and fills in the description field with as much detail as possible about the requirement. When the requirement is fully written, the business analyst or project manager assigns it to a lead developer. The requirement remains in the Active state while it is being implemented. The lead developer coordinates efforts with other developers to implement the requirement.

During the project's life cycle, a QoS requirement can change to any of the following states:

- **New** A QoS requirement is activated as a new requirement when it is first created.
- **Active** QoS requirements begin in the Active state. The business analyst creates the requirement, provides a descriptive title, and fills in the Description field with as much detail as possible about the requirement. When the requirement is fully written, the business analyst assigns it to a lead developer.
- **Resolved** When the QoS requirement is implemented in code, the lead developer sets the state to Resolved. The lead developer also assigns the requirement to a tester so that testing can begin.
- **Closed** The tester closes a QoS requirement if it passes its tests. A requirement is also closed if it is Deferred, Removed, or Split into more requirements.

During a QoS requirement's life cycle, a requirement may make any of the following state transitions:

- New to Active—Occurs when a new QoS requirement is first created.
- Active to Resolved (Completed)—Occurs when the development team completes writing code for the requirement. The lead developer assigns the requirement to a tester.
- Active to Resolved (Split)—Occurs when further review indicates that the requirement is too large or that it needs a more granular definition.
- Active to Resolved (Deferred)—Occurs if the requirement cannot be implemented in the current iteration. A requirement could be deferred because the team does not have enough time or because blocking issues were discovered.
- Active to Resolved (Removed)—Occurs if it is no longer deemed necessary to implement.
- Resolved to Closed (Completed)—Occurs when the tester indicates that it has passed its tests.
- Resolved to Closed (Split)—Occurs because further review indicated that the requirement was too large or that it needed more granular definition.
- Resolved to Closed (Deferred)—Occurs because it could not be implemented in the current iteration.
- Resolved to Closed (Removed)—Occurs because it is no longer deemed necessary to implement.
- Resolved to Active (Test Failed)—Occurs if the QoS requirement fails to pass any tests and the tester must return the scenario to an active state and reassign it to the original lead developer.
- Closed to Active (Regression)—Occurs if a regression test indicates that the requirement is no longer passing.
- Closed to Active (Reactivated)—Occurs when a QoS requirement's assigned iteration begins.

Adding Tasks

Probably the most understandable and commonly used Team System work item is the *task*. A task work item simply communicates the need to do some work. Each role will get tasked to do different things. For example, a developer uses development tasks to assign work derived from scenarios or QoS requirements to component owners. The tester uses test tasks to assign the job of writing and running test cases. A task can also be used to signal regressions or to suggest that exploratory testing be performed. Finally, a task can be used generically to assign work within the project. On the work item form, certain fields are used only in cases when a task relates to a particular role.

A new task work item can be created whenever new work is identified that needs to be done. To create a new task work item, use the new work item menu in Team Explorer. Initially, the state, or status, of a task is automatically set to Active. An Active task indicates there is some element of work to be done. All tasks should be assigned to an owner and a discipline if they are development or test tasks.

During the project's life cycle, a task can change to any of the following states:

- **New** A new task work item can be created whenever new work is identified that needs to be done. There are two types of tasks. Development tasks identify development work to be done as part of implementing scenarios, QoS requirements, or architecture. Test tasks identify testing to be completed. Tasks can also be used to identify work outside of these two areas.
- **Active** When a new task is created using Team Explorer, the state, or status, is automatically set to Active. An active task indicates there is some element of work to be done.
- **Closed** A closed task means that no further work is to be done for the current product version. A development task is closed after the code changes have been integrated. A test task is closed when all the tests are complete for that area.

During the project's life cycle, a task may make any of the following state transitions:

- New to Active—Occurs when a new task is first created.
- Active to Closed (Completed)—Occurs when there is no further work to be done.
- Active to Closed (Deferred)—Occurs if the task cannot be implemented in the current iteration. A task can be deferred because the team does not have enough time or because blocking issues were discovered.
- Active to Closed (Obsolete)—Occurs if the work that the task represents is no longer applicable to completion of the product.
- Active to Closed (Cut)—Occurs if the functionality for that task is removed from the product.
- Closed to Active (Reactivated)—Occurs when a QoS requirement's assigned iteration begins.

Adding Bugs

I'm pretty sure you know what bugs are. Some teams refer to them as "defects." Whatever term you use for them, Team System bugs are work items that indicate a potential problem exists or has existed in the system. The goal of creating a bug work item is to accurately report bugs in a way that allows the reader to understand the full impact of the problem. The descriptions in the bug work item should make it easy to trace through the steps used when the bug was encountered, thus allowing it to be easily reproduced. The test results

should clearly show the problem. The clarity and understandability of this description often affects the probability that the bug will be fixed.

As bugs are detected in your software product, they must be documented as quickly as possible so that developers can resolve them. When you discover a new bug and enter it using Team Explorer, the bug work item is automatically set to an Active state. An Active bug indicates that a problem exists and must be addressed. Bugs can be discovered and documented as a result of a build failure or other causes, such as those discovered by developers, testers, or users.

Tip Before creating a bug work item, you should query existing bugs to be sure the bug you discovered hasn't already been reported. You can do this from Team Explorer by browsing the Active Bugs list of work items, or by creating a new query for bug work items by title, date, iteration, and so on.

During the project's life cycle, a bug can change to any of the following states:

- **New** As bugs are detected in the software product, they must be documented as quickly as possible so that developers can resolve them.
- **Active** When you discover a new bug and enter it using Team Explorer, the bug work item is automatically set to an Active state. An active bug indicates that a problem exists that must be addressed.
- **Resolved** A bug is in the Resolved state when it has been addressed by a developer or during triage. A bug is resolved as either Fixed or As Designed.
- **Closed** A closed bug means that no further work is to be done for the current product version. A bug is closed after the resolution has been verified.

During the project's life cycle, a bug may make any of the following state transitions:

- New to Active (New)—Occurs when a new bug is first created.
- New to Active (Build Failure)—Occurs as a direct result of a build failure.
- Active to Resolved (Fixed)—Occurs when the changed code is checked in.
- Active to Resolved (As Designed)—Occurs if the bug describes an expected condition or behavior of the system.
- Active to Resolved (Deferred)—Occurs if the bug will not be fixed in the current iteration. It will be postponed until it can be reevaluated in a future iteration or version of the product.
- Active to Resolved (Duplicate)—Occurs if the bug describes the same problem as another bug.

- Active to Resolved (Obsolete)—Occurs if the bug is no longer applicable to the product. For example, if the bug describes a problem in a feature area that no longer exists in the product, it is obsolete.
- Active to Resolved (Unable to Reproduce)—Occurs if the developer cannot reproduce the bug on his or her computer.
- Resolved to Closed (Fixed)—Occurs when the author of the bug has verified that the fix is in place in a build.
- Resolved to Closed (As Designed)—Occurs if the bug author agrees that the bug describes something that is intentional by design.
- Resolved to Closed (Deferred)—Occurs if the bug author agrees that the bug should be deferred.
- Resolved to Closed (Duplicate)—Occurs if the bug author confirms that the bug describes the same problem as another bug.
- Resolved to Closed (Obsolete)—Occurs if the bug author agrees that the problem described is no longer applicable to the product.
- Resolved to Closed (Unable to Reproduce)—Occurs if the bug author is unable to produce a working example of the bug or provide more specific instructions for reproducing the bug.
- Resolved to Active (Resolution Denied)—Occurs if the resolution is unacceptable.
- Resolved to Active (Wrong Fix)—Occurs if the fix was incorrect.
- Resolved to Active (Test Failed)—Occurs if a test demonstrates that the bug still exists.
- Closed to Active (Regression)—Occurs when a regression test indicates that the bug exists again.

Adding Risks

An essential aspect of project management is to identify and manage the inherent risks of a project. A *risk* is any probable event or condition that can have a potentially negative outcome on the project in the future. A risk work item documents and tracks the technical or organizational risks of a project. When concrete action is required, these risks can translate into tasks to be performed to mitigate the risk. For example, a technical risk can set off an architectural prototyping effort. In this case, the project manager or another team member would create new Task work items, linking them back to the original Risk task item or items. The linking of these work items is important to establish a proper work item lineage for management.

The team should always regard risk identification in a positive way to ensure team members contribute as much information as possible about the risks the team faces. The environment should allow individuals identifying risks to do so without fear of retribution for their honest expression of tentative or controversial views. Teams creating a positive risk

management environment will be more successful at identifying and addressing risks earlier than teams operating in a negative risk environment.

A new risk work item can be created whenever a new risk is identified. These new work items should contain a title and description that clearly states the potential adverse outcome. The potential impact should be selected. New risks can be created by any team member and should be assigned to the person who will track the risk until it is closed or reassigned. To create a new risk work item, use the New Work Item menu in Team Explorer. An Active risk indicates there is some potential unrealized event that could affect the project. All risks should be assigned to an owner.

During the project's life cycle, a risk can change to any of the following states:

- **New** A new risk work item can be created whenever a new risk is identified. These new work items should contain a title and description that clearly state the potential adverse outcome. The potential impact should be selected. New risks can be created by any team member and should be assigned to the person who will track the risk until it is closed or reassigned.
- **Active** When a new risk is created using Team Explorer, the state is automatically set to Active. An active risk indicates there is some potential unrealized event that could affect the project.
- **Closed** A risk that is closed is no longer a threat to the project. However, a risk that was closed during an iteration may be worth discussing in the iteration retrospective.

During the project's life cycle, a risk can make any of the following state transitions:

- New to Active—Occurs whenever a potential risk may affect the project.
- Active to Closed (Mitigated)—Occurs when some actions are performed ahead of time to prevent the risk from occurring altogether or to reduce the impact or consequences of its occurring to an acceptable level.
- Active to Closed (Inactive)—Occurs when the probability of the risk occurring drops low enough to be negligible.
- Active to Closed (Transferred)—Occurs when the risk can be moved outside the project. The risk has not been eliminated, but it is no longer in the scope of the current project. Example methods of transferred risk include moving the risk to the next release, using external consultants, or purchasing a component instead of building it.
- Active to Closed (Accepted)—Occurs when a risk is such that it is simply not feasible to intervene with effective preventative or corrective measures. The team elects simply to accept the risk to realize the opportunity.
- Active to Closed (Avoided)—Occurs because of a change to the project or the risk itself. The risk may have failed to materialize, so the project can stop tracking it.
- Closed to Active (Regression)—Occurs when a risk reappears.

Summary

In Team System, the project manager is both the technical and functional administrator of the project. He or she is responsible for the creation, configuration, and ongoing maintenance of the project. In other words, the project manager launches and manages the project by creating and assigning work items—such as scenarios, QoS requirements, and tasks—setting up policies and constraints, and maintaining the project's status.