CHAPTER 4

Getting Started:
The Project Members
Become a Team

RUP describes phases of a project, starting with Inception. Perhaps because we were starting a brand new project, we found that we needed some time to come together as a team before we started the Inception phase.

This chapter presents those things we did to address the teamwork aspects of the project. It would be nice if teams were given time to learn how to work well together, and then continue as a team from project to project. Unfortunately, this rarely happens. Our professions experience high turnover in personnel. Organizations often compensate for the possible loss of people by using a matrix-style organization where people are farmed out to projects as needed. As a result, the people who work on projects never seem to be able to put down roots and get to know their teammates.

Coalescing into a Team

Gary often repeats a saying commonly used when talking about organizations—that teams “form, storm, and perform.” We did assemble the team fairly quickly. But forming the team took a while.

Forming the Team

While working at Rational, we’d all become acquainted with each other. Some of us had worked on small, informal projects together or talked to each other in the course of our work. Some of us worked together on major

1 Chris and Jas, for example, have talked on the phone, but have yet to meet face-to-face.
projects. But we had never before all worked together as part of a team; it turned out that getting to know each other—our work styles, strengths, and weaknesses—took some time, more than we had anticipated.

We originally thought we could settle into working as a team fairly quickly; we just needed to decide what to build, assign tasks to team members, and wait for the software to be written. This plan seemed straightforward enough—what barriers could we possibly run into?

When we tried to work together, we found that we really didn’t know each other very well at all. We thought we understood each other’s abilities and strengths. We assumed that we were each making the same commitment to the project and that we would each be able to act on our commitment. It was not long before reality hit.

We all had good intentions. But we also each had a full-time job and other responsibilities in our lives. In this respect, the PSP Tools project was similar to an open-source project: We were a team of volunteers and we had to respect each other’s ability to offer different amounts of time to the project.

When we describe our project, people say that it sounds like an open-source project. In some ways it is, but we think there are many active projects in large organizations that are similar to ours. In fact, there are probably more than you might realize. We see project teams forming and re-forming all the time. The practice of “borrowing” someone from another project for a short time is not uncommon. Managers frequently ask their developers to fit in some small project along with larger projects. All too often, this is done with the suggestion that the smaller project “should not take away from the rest of the work the employees are responsible for.”

We also had to learn to say no in response to requests. There were times when it was important to decline a request because we lacked expertise or time. In the beginning of the project, though, we each resisted saying no because we felt that we’d be letting the team down.

Of course, by accepting an assignment that we were unable to complete, we still let the team down. The team members were not disciplined about demanding results by the promised dates. As a result, we had to become more effective managers of our individual schedules and workload.

As an experiment, we tried working with well-formed requests described in Chapter 3, but soon dropped the practice. Looking back, we were all just as eager to accept well-formed requests as we were to accept-

---

2 Sometimes a team is formed for a short project that is critical to the organization. These teams are called by different names, such as SWAT teams or Tiger teams, to indicate their urgency.
casual requests, with the same frustrating results. Using well-formed requests did not make it any easier to decline tasks.\textsuperscript{4}

We went into the project believing that we could all get started working right away. In retrospect, we realize that we should have held one or two daylong meetings to initiate the project. We were able to justify omitting the meetings because we were a small team. But by trying to save time in the beginning, we ultimately wasted a lot of time in the long run.

On our next project, we’d organize meetings to discuss the nature of the work and the organization of the team, including the following topics:

- **Introductions**: Who are you? What is your background? What are your interests at work? What are you good at doing? What new area do you want experience in?
- **Moving toward a vision**: What are the goals of the project? Can the team produce at least a draft of the team’s goals? Who volunteers to polish the draft and send it out for review?
- **Matching the team to the project**: What roles are needed on the project? By examining experience and interest, who is best suited for each role? What are the first deliverables for each role, and when are they due?
- **Establishing ownership of work**: How will we assign work? How does a team member either accept the work or decline it? How do team members support each other?
- **Next steps**: Review deliverables; decide how the team will communicate; schedule next meeting.

### Dealing with Employer and Geographical Distribution

On a typical small team, the team members work for the same company in the same location. This was not the case for us. Most of us work in the same town, but one team member lives 3,000 miles away. Another team member travels frequently, and the three of us who do work in the same town work in different buildings that are walking distance apart.

Originally, we all worked for the same company. Part way through the project, Jas left the company but wanted to continue working on this project. We had anticipated several risks but did not anticipate this one.

\textsuperscript{3} Well-formed requests are also called “precision requests.” They are a way for a team to communicate precisely what needs to be done, who should do it, and by when. When they are used effectively, they can help a team work in a highly productive manner.

\textsuperscript{4} See *The Phoenix Agenda* by John Whiteside.
One effect of this change was that we lost our comfortable way of communicating. We no longer all shared the same intranet, so we could no longer share calendars, use a common network location for files, or even have access to the same set of tools. We had to find other ways of staying in touch. And we needed to figure out how to continue to work together effectively.

Throughout the rest of the book we return to the topic of our geographical distribution and the problems it caused. This practice is becoming increasingly common, and with more companies encouraging telecommuting and distributed teams, it is sometimes even normal. The main difference between a distributed project in one company and our project is our lack of a common intranet for sharing our tools and data. Solutions exist today that address the intranet problems, and we mention some of them in the final chapter. Although the common intranet problem can be addressed by technology, most other problems are not solved by technology. They are solved by team members who pay attention to people issues.

Most team members enjoy the informality of being able to communicate by talking over the cubicle wall. We had to teach ourselves to pick up the phone and call each other, to have regular meetings, and to use tools to facilitate our communication.

**Losing a Member**

Throughout most of the book, we talk about our four-member team. For a while, though, we had a fifth member, whom we will call “John.” John left the team partway through the project, shortly after we began our Inception phase. What happened? We discuss it here, rather than in the Inception chapter, because it is a team issue, not an Inception topic.

Gary asked John to be the architect. Although John had limited experience in that area, and felt that he wasn’t ready for the role, he reluctantly agreed to give it a try. Gary attempted to coach John by urging him to design the simplest thing that could work while keeping in mind future changes, but John seemed to be bogged down by “analysis paralysis.” It appeared to the rest of us that John was afraid to do something wrong, so it was very hard for him to get started.

Because none of us were database experts, we identified the design of the database as one of our most important technology risks. We were all concerned about our ability to create a working database application. John had the most experience with databases and he wanted to tackle our database design. This was a good fit for the architect.
During this period we entered into a long cycle. John had difficulty getting started with the database design. He felt that he would fail us if he could not deliver. Most likely, his fears caused him to work slowly and carefully. The team wanted to support John and not pressure him. Our lack of feedback caused John to spend even more time perfecting the design. In retrospect, Gary says that he should have taken more of a leadership role to support John, and also to establish stronger expectations.

And then, John’s commitments in his work and his personal life interfered with his ability to deliver work to this project. At the same time, it was hard for John to leave this project because he felt that we were depending on him and he would be letting us down.

We learned and re-learned many lessons on this most human part of the project.

- **Listen to the team.** When a team member says they’re not ready to step into a particular role, the manager needs to make a delicate decision—does the team member need to be encouraged to grow into the role or should the manager listen to the team member’s response? For a long time, we erred by wishing John into this role, and perhaps by not coaching him enough. In the future, we’ll know to be better listeners in this area, and perhaps to be better coaches, but we don’t know if we’ll have a happier outcome. This is one of the hardest areas of management.

- **Become an effective coach.** During our regular meetings, we’d ask John to produce a piece of work, and he’d say that he’d try. Because we were asking him to work in unfamiliar territory, we could have helped him succeed by having another team member work with him outside of the team meetings. In retrospect, we wish we had tried to use a pairing technique. John worked and lived near other team members, so it would have been relatively easy to arrange times to work with him. In a similar technique, a coach works with a team member to produce a piece of the work together. Together, they create and review the work and produce another piece. By breaking the work down into smaller pieces, the coach can help the learner understand what it means to do this new work—in this case, to create an

---

5 In a pairing technique, two teammates work together on a specific activity. The most popular pairing method today is *pair programming*, described in Appendix C.
architecture. For more about this, see the sidebar discussion of Couch, Mentor, Guru, Companion.

- **Allow a team member to leave gracefully.** John felt torn between commitments he had made to the team and his personal commitments. We believed John when he said he would perform project tasks, and we held up the project waiting for him to deliver the work. It was hard for him to say no, and it was hard for him to come to the conclusion that he needed to step away from the project in order for us to make progress. We could have helped more by intervening earlier, instead of relying on him to resign; we should have made this a team decision rather than placing the onus solely on John.

---

**Coach, Mentor, Guru, Companion**

We feel it is important to have one or more people who act in an advisory role to the team. These people are known by different names, such as coach or mentor, but they all serve the same purpose—to help the team function better as a team. Regardless of what you call these people, we recommend that you identify them and enlist them to help your team as early as possible.

We prefer the term *companion* to describe these people. Many spiritual traditions have the notion of a companion. This is a person who walks with you on part of your journey. The companion has been on this path and knows some of the pitfalls and stopping points along the way. Every path is different, but many are similar. The companion walks with you as long as your paths converge. At some point, your paths diverge and you bid farewell to each other.

The companion on a software development project serves a similar purpose to a spiritual companion. The companion has participated on projects in the same domain, technology sphere, or other area that is common to yours. The companion has learned some of the pitfalls and obstacles you will most likely encounter. The companion can talk about the experiences and relate them to what your team will experience.

The purpose of the companion is not to tell you how to do your job. It is likely that when you expect the companion to give you an answer, no answer is forthcoming. The purpose of the companion is to help you find your own answers and take charge of your own destiny. This is the only way your team can grow and develop its own unique culture.

**Who Are Companions?**

Companions can come from anywhere. You might discover an excellent companion who is already a member of your team. In organizations where work occurs...
In the Rational Unified Process, a Development Case is a description of how you will customize the process for your project. Even on this small, fairly well-defined project, we immediately decided that we wanted to use RUP to help guide us and keep us on track. But we were not interested in process for its own sake.

We started from the premise of Gary’s Prime Directive: “Only do those activities and produce those artifacts that directly lead to delivering value to your customers and stakeholders.” Another way to think about this directive is to ask “If I don’t perform a particular process step, will anything bad happen?” (Read the “Process” section in Chapter 3.) It turned out that we didn’t eliminate many RUP steps, but we took a much more informal approach than a larger project might have. For example, instead of creating formal artifacts, we had discussions or wrote informal documents.

As a small team, we were also interested in techniques from the agile community. So Gary and Chris did some pair programming when it was appropriate, and some test-first development (see Appendix C and the sidebar on Agility in Chapter 1), and were pleased with the results. We’ll talk more about these points in a later chapter.
We wrote a brief development case using the RUP template. Our development case is available on the project Web site. It is longer than we would like. Most of the length is due to the RUP template. The first five-and-a-half pages contain introductory material and descriptions of how to read the rest of the document. Our team did not need this information, but we recognize that this part of the document can help someone new to the project, or any interested person, understand the purpose and format of the document.

We want to be clear that this is the way we decided to document our development case. You can choose to document, or not document, the parts of your development artifacts in whatever way is useful to your team. RUP provides templates that serve as examples and as possible starting points. You must decide what is right for your team.

The next sections describe the key features of our development case.

**Conventions Used in the Development Case**

Each project team tailors a development case for their project, even if there is a common development case for the larger organization. The Development Case template contains a section that lets you describe how you tailored the Development Case and identify the conventions you used to represent your process.

We focused on artifacts rather than activities; therefore, the main descriptive vehicle in our development case is a table that describes each artifact. Figure 4.1 shows its format.

We created one artifact table for each RUP discipline we planned on using. RUP provides guidance for what to put into the table, but we chose our own style. So, in our Development Case, we described our conventions. These are shown in Table 4.1.

For each discipline, we included placeholders for notes and other issues in the artifact table. The real content that describes our planned process takes up about three pages in the Development Case document. Most of

<table>
<thead>
<tr>
<th>Artifacts</th>
<th>How to use</th>
<th>Review details</th>
<th>Tools used</th>
<th>Responsible</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inception</td>
<td>Elaboration</td>
<td>Construction</td>
<td>Transition</td>
</tr>
</tbody>
</table>

**Figure 4.1** Format for the artifact table in the Development Case
Writing the Development Case

Table 4.1  Artifact table explanations

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Purpose</th>
<th>Contents/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artifact</td>
<td>The name of the artifact.</td>
<td>A reference to the artifact in RUP, or to a local artifact defined as part of the development case.</td>
</tr>
<tr>
<td>How to use</td>
<td>Describe how the artifact is used across the lifecycle.</td>
<td>Decide for each phase whether the artifact is produced or modified significantly. The possible values of this field are: C—create in the phase. M—modify in the phase. Blank—not used or not changed in this phase.</td>
</tr>
<tr>
<td>Review details</td>
<td>Define the review level, and review procedures to be applied to the artifact.</td>
<td>Formal—reviewed and signed off by the customer or relevant stakeholders. Informal—reviewed by one or more team members. No sign off required. Blank—no review required.</td>
</tr>
<tr>
<td>Tools used</td>
<td>Definition of the tool (or tools), used to produce the artifact.</td>
<td>References to the details of the tools used to develop and maintain the artifact.</td>
</tr>
<tr>
<td>Responsible</td>
<td>The role responsible for the artifact.</td>
<td>Describe which role, for example, Project Manager or Developer, is responsible for ensuring that the artifact is completed.</td>
</tr>
</tbody>
</table>

this space is occupied by tables like Figure 4.2, which shows our planned requirements artifacts.

Role Map

Every project distributes responsibilities to team members differently. Few projects have a direct one-to-one mapping between roles described in RUP and actual people and jobs. Therefore, it is important to be clear about what each person is responsible for. On small projects, this is critical because there is a lower threshold to tolerate duplication of effort.
Table 4.2 shows our initial role map. We simply took the different “Responsible” roles from the discipline artifact tables in the Development Case and created a row in the table for each one. Then we ensured that at least one person on the team was assigned to that set of responsibilities.

Table 4.2 Role map from the Development Case

<table>
<thead>
<tr>
<th>Role</th>
<th>Liz</th>
<th>Chris</th>
<th>Jas</th>
<th>Gary</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Analyst</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>User-Interface Designer</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Designer</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Software Architect</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Integrator</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Implementer</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
This role map provided an initial guess about how the project would proceed. It helped us ensure that every responsibility was covered and it helped us decide whether anyone was overcommitted.

**Artifacts in Our Development Case**

You may wonder which artifacts we originally thought we would need to produce for our project. We identified over twenty that we thought would be helpful. We did not create all of them. As we progressed, we found that some of our early assumptions were wrong. Rather than create something just because we planned to do it, we chose to do the right thing and create only those artifacts that were truly helpful.

If you download the material from the book’s Web site, you can compare our original Development Case to the following list of those artifacts that we actually produced.

- Vision
- Risk list
- Use-case model
- Design model
- Build
- Component
You would probably produce all these artifacts for any project you undertake. You might not produce volumes of paper, diagrams, or data, but you would produce the artifacts in some form. In some cases, there may be existing artifacts that you can just reuse, for example, Tools or Programming guidelines.

**Importance of the Development Case**

Some people think the Development Case is an important artifact in RUP and others think it should just be part of the project plan. Our thoughts on the Development Case are somewhere in between.

We believe that the Development Case is necessary. Team members need to understand their responsibilities. They need to know what they are expected to produce, what artifacts will be available for their use, and the form of the artifacts. The real question is how much time and effort you should spend creating and maintaining the Development Case.

Our advice is simple. If you have a small team, especially one that has worked together before and has a shared understanding of the process, then the Development Case can be an oral artifact. That is, it can be communicated to the team via discussions. The further you stray from a small, familiar team—whether in terms of size of the team or team members’ lack of familiarity with each other or the common process—the more you need to write your Development Case and keep it up to date.

We probably did not have to write down our Development Case. The hour or so that it took was not totally lost, though. It gave our team a good topic of conversation for early meetings.
To get a flavor for the PSP, Gary asked each of us to track our time spent on the project in 15-minute increments. We each used Microsoft Word or Excel to keep a diary of our activities. We dutifully recorded meeting time, design time, development time, and so on. This was tedious, and it didn’t last long because we found the effort of recording how we were spending our time to be more than we were willing to do. But it did help us understand why it was important to build the tools to assist users with capturing the relevant information.

Gary had been using PSP for a while. Even though he didn’t record overhead time, such as time spent in meetings, he did keep track of his development effort. At first, he recorded his time and defects in a spreadsheet. By our second Construction iteration, the tool was stable enough that Gary could use the tool we were building, so he became our first Beta tester.

On all but the smallest of projects, we recommend using specialized tools like a project planner (for example, Microsoft Project), and a change request management tool (for example, Rational ClearQuest). We have used these tools successfully on larger projects. But on this project, we used Groove to keep track of our engineering backlog and defects; for these two items, we created a new Groove discussion tab. This was sufficient, but even on our small project, it became difficult to find information.

Groove

Groove is a collaborative workspace product that helps geographically distributed teams collaborate effectively. When we began the project for this book, Groove was a new product. It consisted of a workspace where we could store files, a calendar, and tools such as an outliner tool and a discussion tool where we could create discussion items and reply to them. Today, Groove includes tools to help with project management and meeting management. Other tools are available as plug-ins from Groove partners.

Groove ships in several editions. The Preview edition is free for personal use and free for a 90-day evaluation for businesses. The Standard edition and the Professional edition are available for purchase. The Preview edition limits the number of workspaces and meetings you can work with. The Standard edition gives you access to all Groove features.

continues
When we talk about a collaborative workspace, we mean a repository that is available to the whole team, regardless of where the team members are located. Additionally, the workspace provides tools that let you perform certain tasks effectively. Groove works as follows:

- Each team member downloads the Groove software and creates an account.
- One team member creates a project. A project is a workspace that can have any number of members. The creator or another workspace member with privileges can invite other members to join.
- When you accept the invitation, the workspace is copied from the computer of the person who invited you to your computer, through the Groove server.
- When you create a new file in your workspace, usually by dragging a file into the workspace, or by adding an item to a tool such as a discussion tool, the information is transmitted to the Groove server. If you are not online at the time, it will be transmitted the next time you start the Groove product when you are online.
- The Groove server transmits the new information to everyone on the project. When everyone has the file, it is removed from the Groove server.
- When changes are made to files or any type of item, a similar process occurs. If there are conflicts (for example, two people change the same file), Groove notifies you and copies both files to your workspace.

Groove became our most-used tool for this project. Especially in the beginning, when it was newly released, it didn’t always work flawlessly. As Groove matured, it became more robust and indispensable to the team.

Some of our team members have used only the Preview version while others have purchased licenses for the Standard version. The cost is, in our opinion, quite reasonable for what you get. To learn more, visit the Groove Web site: www.groove.net.

We used a stylized way of entering the information. Each headline for a change request had the following format:

[status] (priority for defects) Description (assigned) [resolution]

The “status” was simply *** (three asterisks) for open defects or engineering tasks; we removed the asterisks when the task was complete. We had a simple Priority scheme for defects:
P1—Showstopper, can’t continue testing.

P2—Fix as soon as possible, but testing can continue or the customer can still use the product.

P3—Fix as time permits.

When team members started working on an item, they put their initials after the description, and when the item was resolved, they inserted the build or resolution status. This allowed us to sort the items based on open/close status, and print out reports as needed. Simple, but effective. Figure 4.3 shows a portion of the Defects tab in the Groove workspace.

![Defect tracking with Groove](image_url)

Figure 4.3 Defect tracking with Groove
Creating an Iteration Plan for Inception

RUP is a process that is based on iterative development. If you are not working iteratively, you are not really using RUP. Before each iteration you create an Iteration Plan. The plan does not have to be very detailed, nor does it even need to be a written plan, but it must be one that can be communicated to all members of the team and to all stakeholders.

As our project progressed and we got into the rhythm of regular iterations, we became less worried about creating written plans. In the beginning, though, for the Inception phase, we felt the need to write out the Iteration Plan.

We began with the RUP Iteration Plan template and tailored it to our needs. The plan has these sections:

- **Scope and Objectives.** An overview paragraph for the plan.
- **Plan.** The list of items we expect to complete during the iteration.
- **Resources.** The resources—people, tools, and any other type of resource—we think we’ll need in order to complete the planned items.
- **Use Cases.** The use cases or scenarios that we expect to complete for the iteration.
- **Evaluation Criteria.** A list of items we will use to assess the success of the iteration and to determine whether we’re ready to start the next iteration.

Figure 4.4 illustrates the Plan section of our Inception Iteration Plan. We used a simple tabular list for our plan.

On a more complex project, we would use a project-planning tool such as Microsoft Project.

RUP 2003 provides help for small, informal projects by including a separate set of informal templates. The informal Iteration Plan template is even simpler than the one we used (see Figure 4.5). You can see how easy it is to plan an iteration. If you communicate regularly with your team, you may even decide not to produce a written plan. If you do create a written plan, using the informal template does not impose any significant overhead.

---

6 Some people argue that you can consider a project using a waterfall process as a project with a single iteration. We don’t think this is worth arguing. You can choose to accept this view, but we think that multiple iterations deliver the best results.

7 If you use Microsoft Project 2002, RUP provides a compatible project-planning guide that guides you through building a project plan based on RUP activities and artifacts (available on the Rational Developer Network).
Creating an Iteration Plan for Inception

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Due</th>
<th>Responsible</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial project plan</td>
<td>10/11/2002</td>
<td>Gary</td>
<td>Through Elaboration, reviewed and agreed upon</td>
</tr>
<tr>
<td>Vision and product feature</td>
<td>10/13/2002</td>
<td>Jas, Gary</td>
<td>Reviewed</td>
</tr>
<tr>
<td>requirements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supplementary requirements</td>
<td>10/13/2002</td>
<td>Jas</td>
<td>Deferred until needed</td>
</tr>
<tr>
<td>Tools environment</td>
<td>10/15/2002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clear Case</td>
<td>10/12/2002</td>
<td>Gary, Liz</td>
<td>VOBs</td>
</tr>
<tr>
<td>Test environment</td>
<td></td>
<td>Chris</td>
<td>Not completed</td>
</tr>
<tr>
<td>Requirements</td>
<td></td>
<td>Jas</td>
<td>Deferred</td>
</tr>
<tr>
<td>Project Web site</td>
<td></td>
<td>Gary</td>
<td>Deferred</td>
</tr>
<tr>
<td>Initial use-case model</td>
<td>10/15/2002</td>
<td>Jas</td>
<td>Actors and use cases with brief descriptions</td>
</tr>
<tr>
<td>Initial risk list</td>
<td>10/15/2002</td>
<td>Gary</td>
<td>Reviewed and understood</td>
</tr>
<tr>
<td>Test plan</td>
<td>10/15/2002</td>
<td>Chris</td>
<td>Draft reviewed and agreed upon</td>
</tr>
<tr>
<td>Elaboration Iteration Plan</td>
<td>10/15/2002</td>
<td>Gary</td>
<td>Reviewed and agreed upon</td>
</tr>
</tbody>
</table>

Figure 4.4 Planned items for the Inception phase

<Project Name>
Iteration Plan

1. Key Milestones

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration Start</td>
<td></td>
</tr>
<tr>
<td>Iteration Stop</td>
<td></td>
</tr>
</tbody>
</table>

2. Iteration Objectives

[Objectives may include creating or refining specific artifacts, addressing risks, or implementing specific requirements, or performing supporting tasks. Some example objectives are listed below.]

<table>
<thead>
<tr>
<th>Objective/Task</th>
<th>Assigned to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement Use Case: Register for Course, Basic flow, Alternative 1, Alternate 2</td>
<td>Fred</td>
</tr>
<tr>
<td>Complete Vision</td>
<td>Jill</td>
</tr>
<tr>
<td>Detail UC3: Publish Calendar</td>
<td>John</td>
</tr>
<tr>
<td>Test all developed requirements</td>
<td>Lance</td>
</tr>
<tr>
<td>Create plan for next iteration</td>
<td>Jill</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 4.5 RUP 2003 Iteration Plan template
Summary

On this project, our team understood what we needed to build and we were eager to get started. But it took much longer than we’d predicted before we were a productive team. If we had the project to do over again, we’d spend more time in the beginning focused on the people issues. We’d make it easier for the team to coalesce by launching the project in a more formal way—by holding meetings—instead of expecting things to just “work out.” Our experience indicates that spending time in the beginning of the project to create the team will save you time later.

In parallel with attending to the team issues, you need to prepare for the work ahead. Create simple plans that communicate your intentions. Remember: At this stage of the project all plans you produce are best guesses rather than accurate statements. Don’t spend a lot of time on details; most details will probably change later.

Finally, select your initial process. As with your other plans, this is a best-guess process. It too will change. Most project teams put too much effort into their initial plans; remember that your initial attempt does not have to convey all the details. However, when you’ve assembled your plans, make sure you communicate them to the whole team and that the team members agree to work with the plans and process you devise.