CHAPTER 1

Leveraging Key Development Principles

Practices can be adopted individually.

This book provides a set of software engineering best practices that your project can start using right away to improve the way you develop software. The practices can be adopted individually, but they also support each other. This means that you can pick a set of practices to adopt and be able to make sense of them without having to adopt all of the practices. As you adopt more and more of the practices, you will start to notice the synergy among them. Each practice becomes a piece of a puzzle, and taken together, though not complete, they constitute the backbone of a process that is iterative, agile, and scales to your project needs. See the Preface for how to use this book.

Where Do the Practices Come From?

We have chosen the best practices in this book based on principles gleaned from a huge number of successful projects and distilled into a few simple guidelines that we call key development principles (see the section Key Development Principles later in this chapter). Grouping the practices under these key development principles allows you to identify proven principles that address the issues your project is

facing and review the practices supporting each principle to see how you can make progress toward adopting the principle. We will discuss the topic of adoption in more detail in Chapter 8, "Making Practical Use of the Best Practices."

These practices borrow from a large number of iterative and agile processes, all sharing a focus on iterative development; continuous integration; early and continuous testing; addressing customer needs; people and team collaboration; and minimizing process overhead.

OpenUp is an open-source version of the Unified Process.

EPF is an open source process framework.

The practices capture much of the thought that was formative in the creation of the **Open Unified Process (OpenUP)**, especially its most agile and lightweight form, OpenUP/Basic, targeting smaller and collocated teams interested in agile and iterative development. OpenUP is a part of the **Eclipse Process Framework**[1] **(EPF)**, an open source process framework developed within the Eclipse open source organization.[2] It provides best practices from a variety of software development thought leaders and the broader software development community that cover a diverse set of perspectives and development needs. Some of the practices in the EPF may differ from the ones you find in this book, because they were produced by people with a different view of how to develop software or targeted a different type of project. This is one of EPF's strengths: allowing diversity of ideas, while encouraging learning from each other, thereby driving unification and evolution of best practices. We believe that reading this book will help you gain insights into key aspects of OpenUP/Basic as well as EPF; see the section OpenUP/Basic later in this chapter and Appendix A for more detail.

RUP is a continually evolving process framework that extends EPF.

The key development practices also have a strong connection to RUP. RUP is a continually evolving process framework. It started in 1987 as the Objectory Process[3], a use-case-driven and object-oriented process, and in 1996 was integrated with the Rational Process[4], an iterative and architecture-centric process. Since 1996, it has integrated best practices

1. www.eclipse.org/epf/.
2. www.eclipse.org.
3. See Jacobson 1992.
4. See Devlin 1995.

from a broad set of sources, including processes for testing, configuration management, agile development, service-oriented architecture, and business engineering[5]. RUP extends EPF with in-depth process content for specific technologies and tools, such as J2EE and IBM tools; for specific domains, such as packaged application development, systems engineering, and program management; and for process maturity standards and frameworks, such as SEI CMMI.

Some of the practices in this book are more appropriate for projects that need to follow a higher-ceremony process, making them better reflect RUP than the lighter-weight guidance in EPF. We believe that RUP users will find the practices in this book valuable, because they articulate a clear set of principles for developing software. Unfortunately, too many RUP users do not adhere to these principles (see Key Development Principles later in this chapter) and so misinterpret the underlying spirit of RUP.

Other iterative and agile processes, including eXtreme Programming (XP),[6] Scrum,[7] Agile Modeling,[8] Crystal,[9] Agile Data Method,[10] and Dynamic Systems Development Method (DSDM)[11] have influenced the practices in this book, just as they have influenced RUP and OpenUP.

XP, Scrum, Agile Modeling, Crystal, Agile Data Method, and DSDM have influenced this book.

Using Practice Descriptions

Each key development principle is discussed in its own chapter and contains the practices that support that principle. For example, the principle Demonstrate Value Iteratively is supported by the practice Manage Risk, which gives concrete guidance on one of several practices

5. See Appendix B and Kroll 2001 for the history of RUP.
6. See Beck 2004.
7. See Schwaber 2002.
8. See Ambler 2002.
9. See Cockburn 2002
10. See Ambler 2003.
11. See Stapleton 2003.

that will help you to adhere to the principle effectively. Each practice description discusses the following:

- **Problem:** the problem that the practice addresses
- **Background:** background information
- **Application:** how to apply the practice
- **Comparison with other practices:** how the practice compares with practices found in major iterative and agile development processes, including XP and Scrum
- **Adoption:** how to implement the practice at different levels of adoption
- **Related best practices:** additional supporting practices
- **Additional information:** additional reference information available in OpenUP and RUP, and in books relevant to the practice

There may be practices that do not fit your project or organization at this time for technical, cultural, business, or other reasons. You may find that you are already following other practices. Most practices can be adopted at different levels, allowing you to adopt more practices over time as well as implement them at a higher level. We believe that gradually adopting a distinct set of practices at a level appropriate for your organization will enable you to start improving today, and to continue improving over subsequent projects, without requiring that you change everything at once. See the following section for more information on the different levels of adopting a practice.

Before we dive into the practices, let's discuss the process of adopting the practices: iterative development, levels of ceremony and agility, followed by the key development principles. We will then provide an overview of the Unified Process lifecycle, followed by an overview of OpenUP/Basic, RUP, XP, and Scrum in turn.

Adopting the Practices: Iterative Development, Levels of Ceremony, and Agility

A key aspect of this book is to allow you to move incrementally from where you are today to where you would like to be a couple of years from now, adopting the practices at your own pace.

Levels of Adopting the Practices

Each practice can be adopted at three levels: basic, intermediate, and advanced.

Each practice can be adopted at three different levels: basic, intermediate, and advanced. The basic level represents a level of adoption we think most projects can adhere to with limited investments, that is, with a limited skill set and with no or small amounts of tool investments. This level thus represents a reasonable starting point on your journey toward improved ability to develop software effectively, but it is not necessarily an acceptable end goal. As you move to the intermediate and advanced levels of adoption, you will have to make additional investments in building skills and/or tools. For some teams and practices, the basic or intermediate adoption level is preferable. Other teams should aim at adopting advanced-level practices for maximum productivity.

Process Map

To understand the level that is appropriate for your team, we will use a process map (see Figure 1.1) characterized by two dimensions discussed below:[12]

- **Low ceremony/High ceremony**[13] dimension on the horizontal axis. **Low ceremony** produces minimum supporting documentation and has little formality in the working procedure; **High ceremony** has comprehensive supporting documentation, traceability maintained between artifacts, Change Control Boards, and so on.
- **Waterfall/Iterative** dimension on the vertical axis. **Waterfall** is a linear approach with late integration and testing; **Iterative** is a risk-driven development approach with early and continuous implementation, integration, testing, and validation of software that delivers concrete value to the stakeholders.

12. See Chapter 3 in Kroll 2003 for an in-depth discussion on the process map.

13. Cockburn refers to "Ceremony" as "Methodology Weight." See page 123 in Cockburn 2002 for an interesting discussion on this topic.

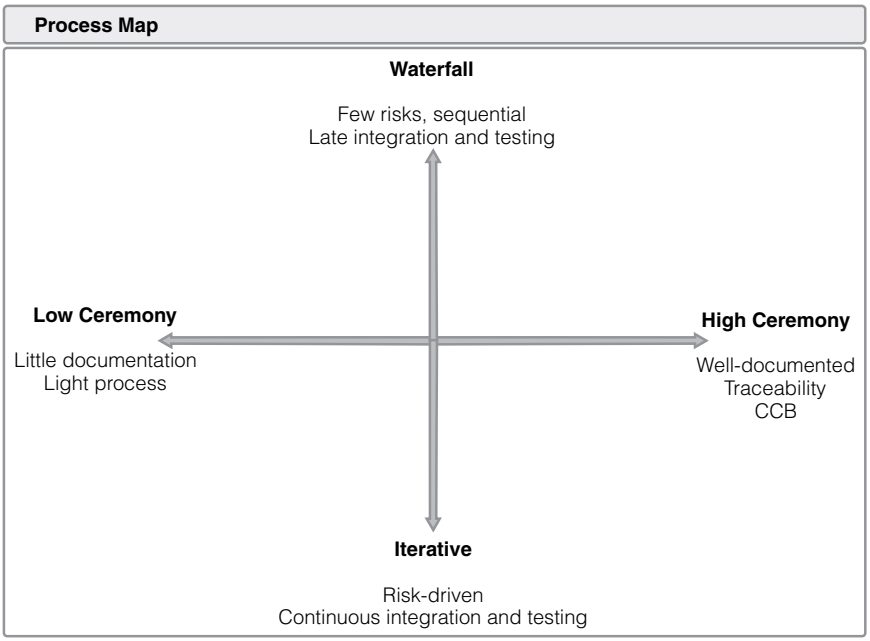

FIGURE 1.1 Process Map for Process and Practice Comparison. *By organizing processes and practices along two dimensions—Low ceremony/High ceremony and Waterfall/Iterative—you can compare them and analyze which are more suitable for your project or organization. Agility translates to being in the lower-left quadrant on the map. (Adapted from Kroll 2003.)*

Agility and Ceremony

Agility is the ability to respond rapidly to risks and change.

We define agility as *the ability to respond to risks rapidly; changing requirements or stakeholders needs, or other changes impacting the application we are building*.[14] Agility translates to being in the lower-left quadrant in our process map. Iterative development provides us with the rapid and timely feedback we need to understand when and what to

14. Compare Larman 2004, page 25, "agility—rapid and flexible response to change."

change, and the low ceremony provides us with the ability to execute changes rapidly.

So, do we always want to be in the lower-left "agility" corner? If you are two developers building an application with a short life span, you may choose to save time by not documenting many of the requirements or the design. This may allow you to be more productive and to make rapid changes to the application. However, if you are doing distributed development with twenty people in different time zones, you will probably be more productive if you document the requirements and key design decisions. But the added ceremony will add time and cost when you have to implement a change. You therefore choose to move to the right on the process map to gain productivity, while losing agility. By deploying the right tools, however, you can counter this loss and reduce the cost of change, allowing you also to move down on the scale.

Most projects[15] benefit from being as low as possible on the waterfall/iterative axis to ensure rapid feedback, but not so low that the cost associated with the overhead of each iteration becomes prohibitive. One of the aims of iterative development is to reduce the overhead cost of each iteration so that you can do more iterations, and many of the practices in this book help you achieve that goal. As Professor Barry Boehm points out in *Get Ready for Agile Methods, with Care*,[16] each project should choose the appropriate level of ceremony to fit its specific needs (Boehm used the term "agile" to mean roughly the same as what we refer to as "ceremony.") This means that each project should be as agile as possible based on its specific characteristics, but not more agile.

Most, but not all, advanced practices lead to higher levels of ceremony. *If your project is better off following a low-ceremony process, you should probably not adopt an advanced practice if it leads to higher level of*

15. A few projects with very well understood requirements and architecture, facing very limited risk, are exceptions to this rule and may benefit from waterfall development.

16. See Boehm 2002.

ceremony. On the other hand, there are many reasons why projects may benefit from more ceremony, including large project size, regulatory requirements, distributed development, complex projects, long application life span, or complex stakeholder relations. In some cases an advanced adoption level leads to less ceremony and shorter and more frequent iterations, which is probably beneficial for most projects that are willing to take on the investment.

Where Does a Level of Adoption Take You on the Process Map?

We describe the levels of adoption for each practice, as well as giving an indication of the direction in which the adoption level will take you on the process map. Let's look at examples of some of the symbols we use:

If the arrow goes down and to the left, it means that adopting the practice at this level will lead to, or enable you to have, shorter iterations with a decreased level of ceremony. This is typical for many of the practices at the basic level but could also be true for the intermediate and advanced levels.

If the arrow goes down and to the right, it means that adopting the practice at this level will lead to, or enable you to have, shorter iterations, while increasing the level of ceremony or learning curve for new team members. This outcome may be desirable if your project needs more ceremony, or if team members need to build more skills, but if that is not the case, your project should consider not adopting the practice at this level.

If the arrow goes up and to the right, it means that adopting the practice at this level will increase the level of ceremony you are working with, as well as the overhead associated with doing shorter iterations, potentially forcing you to increase the iteration length. These are typically not good changes for a project, but the benefits of implementing the practice may be valuable enough to counteract the drawbacks.

Key Development Principles

Six fundamental principles characterize the most successful projects in the software industry.

The practices in this book are organized according to six fundamental principles observed by IBM to characterize the most successful projects in the software development industry.[17] Many of the following principles (see Figure 1.2) are found to some degree in most iterative and agile processes:

1. **A**dapt the process.
2. **B**alance stakeholder priorities.
3. **C**ollaborate across teams.
4. **D**emonstrate value iteratively.
5. **E**levate the level of abstraction.
6. **F**ocus continuously on quality.

FIGURE 1.2 Principles for Business-Driven Development. *These six principles characterize the software development industry's best practices in the creation, deployment, and evolution of software-intensive systems. They have been derived by IBM through workshops with thousands of development managers and executives and by synthesizing input from a large number of technical leaders within IBM who are working with software development organizations across all industries.*

17. These six principles are an evolution of what IBM Rational previously called the 6 Best Practices.

Each principle and supporting practices is described in a separate chapter.

Each of these principles is described in a separate chapter giving an overview of the principle, outlining the patterns of behavior that best embody each principle, and listing the most recognizable anti-patterns that can harm software development projects. The primary focus of each chapter is a set of supporting practices that will help your team to adhere to the principle. To provide a more logical sequence for the book, we have chosen to present the chapters in a different order from the alphabetical order in which the principles are listed above. There is no particular significance to the ordering of practices within a chapter. The principles and their supporting best practices are listed below in the order in which they appear in the book.

- **Demonstrate value iteratively.**
  Each iteration should deliver incremental capabilities that are assessed by stakeholders. This enables you to get feedback from stakeholders as well as on progress made so that you can adapt your plans as required. Iterative development also allows you to focus each iteration on addressing the key risks that the project is currently facing, allowing you to increase predictability.

  Practice 1 Manage risk.

  Practice 2 Execute your project in iterations.

  Practice 3 Embrace and manage change.

  Practice 4 Measure progress objectively.
- **Focus continuously on quality.**
  Continuously improving quality requires more than just testing to validate fitness for use. Rather, it involves all team members throughout the lifecycle—having them build quality into the process and the product. An iterative approach focuses on early testing and test-and-build automation throughout the lifecycle as a means to reduce the number of defects, provide fact-based quality metrics early on, and allow you to plan and adapt your product more effectively based on reality.

  Practice 5 Test your own code.

  Practice 6 Leverage test automation appropriately.

  Practice 7 Everyone owns the product.
- **Balance stakeholder priorities.**
  There will always be many competing stakeholder priorities, such as producing a solution rapidly and inexpensively versus address-

ing all the business requirements. We need to work closely with the stakeholders to make sure that we understand their priorities and to prioritize the right projects and the project requirements. We also need to strike the right balance between leveraging existing assets and building custom software, acknowledging that in some cases the former may require compromising on what requirements to address.

Practice 8 Understand the domain.

Practice 9 Describe requirements from the user perspective.

Practice 10 Prioritize requirements for implementation.

Practice 11 Leverage legacy systems.

- **Collaborate across teams.**
  We need to enable people to work at their best. This means that we need to equip talented people with the right skills, break down walls that prevent a project team from collaborating effectively, and put in place the right environments to facilitate meaningful collaboration. As software becomes increasingly critical to how we run our business, we also need to make sure that we work well together across business, software, and operational teams.

  Practice 12 Build high-performance teams.

  Practice 13 Organize around the architecture.

  Practice 14 Manage versions.

- **Elevate the level of abstraction.**
  Complexity is a major enemy to project success, and minimizing the amount of code, data structures, components, model elements, or other constructs humans produce during a project is crucial to reducing complexity. You can achieve this goal by reusing existing assets—such as business models, components, patterns, and services—instead of custom-building new ones. You can also leverage higher-level languages, frameworks, and tools that can generate code from higher-level models; automate unit testing; and manage the complexity of configuration management. Another approach to reducing complexity is to promote implicity. You can do this by refactoring, keeping code and models clean, and implementing key aspects of the architecture first in what we call architecture-driven development.

Practice 15 Leverage patterns.

Practice 16 Architect with components and services.

Practice 17 Actively promote reuse.

Practice 18 Model key perspectives.

- **Adapt the process.**
  More process is not necessarily better. Rather, you need to adapt the process to the specific needs of your project, based on size, complexity, needs for compliance, and so on. In addition, you need to adapt the process to different lifecycle phases, so you may, for example, use less ceremony at the start of a project and more ceremony toward the end. You must also continuously improve the process, for example by assessing how well it works at the end of each iteration.

  Practice 19 Rightsize your process.

  Practice 20 Continuously reevaluate what you do.

Unified Process Lifecycle

*The Unified Process lifecycle divides a project into four **phases:** Inception, Elaboration, Construction, and Transition.*

Throughout this book you will see references to the **Unified Process lifecycle.** This is the lifecycle used in RUP and OpenUP, and all other processes part of the **Unified Process** family. It is one of several lifecycles supported in the EPF. Even though the practices in this book typically apply to any iterative lifecycle, they work particularly well with the Unified Process lifecycle.

The lifecycle describes the time dimension of project, that is, how a project is divided into phases and iterations. It divides a project into four **phases: Inception, Elaboration, Construction,** and **Transition,** each ending with a well-defined milestone.[18] Each phase has specific objectives:

1. **Inception.** Establish the scope of the system, including a good understanding of what system to build, by reaching a high-level understanding of the requirements. Mitigate many of the business

18. See Kroll 2003 for an in-depth overview of what is done in each phase.

risks and produce the business case for building the system and a vision document to get buy-in from all stakeholders on whether or not to proceed with the project. This is similar to what many agile processes refer to as *Iteration 0*.

2. **Elaboration.** Reduce major risks to enable cost and schedule estimates to be updated and to get buy-in from key stakeholders. Mitigate major technical risks by taking care of many of the most technically difficult tasks. Design, implement, test, and baseline an executable **architecture,** including subsystems, their interfaces, key components, and architectural mechanisms such as how to deal with interprocess communication or persistency. Address major business risks by defining, designing, implementing, and testing key capabilities, which are validated with the customer. Do not define and analyze all requirements at this time, as doing so would lead to waterfall development. Detail and analyze only the requirements required to address the above risks.
3. **Construction.** Undertake a majority of the implementation as you move from an executable architecture to the first operational version of your system. Deploy several internal and alpha releases to ensure that the system is usable and addresses user needs. End the phase by deploying a fully functional beta version of the system, including installation and supporting documentation and training material (although the system will likely still require fine-tuning of functionality, performance, and overall quality).
4. **Transition.** Ensure that software addresses the needs of its users by testing the product in preparation for release and making minor adjustments based on user feedback. At this point in the lifecycle, user feedback focuses mainly on fine-tuning, configuration, installation, and usability issues; all the major structural issues should have been worked out much earlier in the project lifecycle.

Each phase contains one or more iterations, each producing a product increment.

Each phase contains one or more **iterations** (see Figure 1.3), which focus on producing a product increment, that is, the working code and other deliverables necessary to achieve the business objectives of that phase. There are as many iterations as it takes to adequately address the objectives of that phase, but *no more*. If objectives cannot

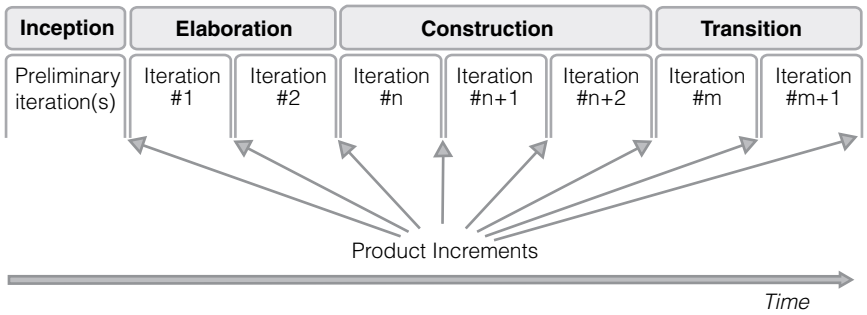

FIGURE 1.3 The Unified Process Lifecycle. *Each of the four phases in the Unified Process lifecycle consists of one or several iterations. Each iteration builds on the result of previous iterations, delivering a product increment one step closer to the final release. Product increments should include working software, with a possible exception being the product increments produced in the Inception phase for new applications.*

be adequately addressed within the planned phase, another iteration should be added to the phase, but this will delay the project. To avoid such a delay, make sure that each iteration is sharply focused on *just* what is needed to achieve the business objectives of that phase, but no less. For example, focusing too heavily on requirements in Inception is counterproductive; so is not involving stakeholders.

The Unified Process lifecycle provides a great deal of flexibility. Product increments may be internal only and may be demonstrated or deployed only to select project stakeholders. Other product increments may also be released for use by customers. For example, some projects benefit from the deployment of several product increments to the production environment, which allows end users to adopt the most critical capabilities more rapidly. You can do this by rapidly moving into the Transition phase and having several Transition iterations, each deploying a release into the production environment (see Figure 1.4).

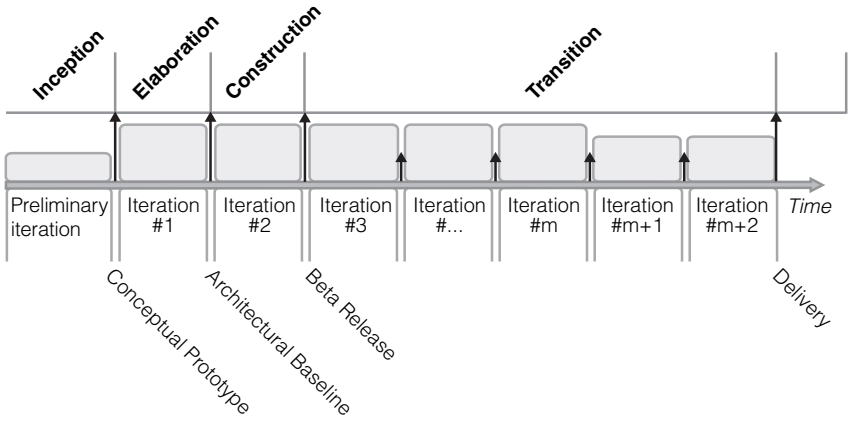

FIGURE 1.4 Incremental Delivery Using the Unified Process Lifecycle. *Projects that deliver product increments into the production environment often rapidly move into the Transition phase. Once in Transition, they deliver a new product increment to production at the end of each iteration. The milestone at the end of the Construction phase aims at ensuring that all pieces are together so that the system can be deployed. You should pass this management milestone before undertaking deployments into the production environment.*

OpenUP/Basic

OpenUP/Basic is an agile and iterative process focusing on the needs of small collocated teams.

OpenUP is an open-source process framework that over time is expected to cover a broad set of development needs.[19] OpenUP/Basic is a subset of OpenUP, and provides a simplified set of artifacts relative to RUP and a much smaller set of roles, tasks, and guidelines. Let's look briefly at the key artifacts, the essentials of the process, and how these relate to the practices in this book.

See Practice 19.

The project team maintains a **work item list**[20] of all work that needs to be done, including requirements and change requests to be addressed and random tasks, such as delivering training. At the beginning of

See Practices 1 and 10.

19. At the time of this book's printing, OpenUP is almost equivalent to OpenUP/Basic, and we have chosen to primarily refer to OpenUP/Basic for the remainder of this book. OpenUP is expected to grow over time, while OpenUP/Basic will continue to be a small process.

20. Work item list corresponds to product backlog in Scrum; see Schwaber 2002.

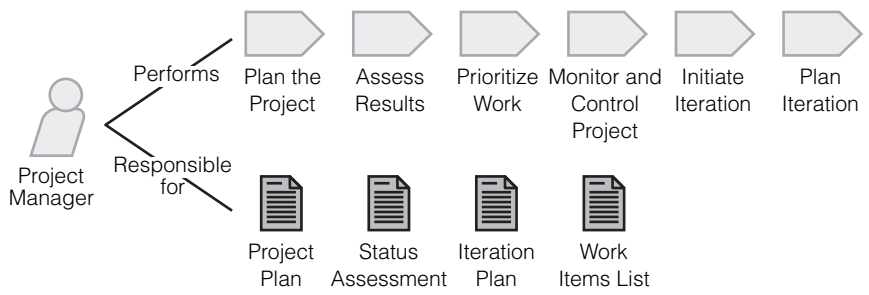

FIGURE 1.5 Managing an OpenUP/Basic Project. *The project manager is responsible for producing a high-level project plan, maintaining a work item list of all things to be done, prioritizing work items into an iteration plan, and assessing the result of the iterations. The entire team is typically involved in producing each of these artifacts.*

each iteration, the team prioritizes which work items should be addressed within that iteration, and that subset of the work item list is called the *Iteration Plan* (see Figure 1.5). The team prioritizes work items to drive down risks and to deliver high-priority end-user capabilities in each iteration.

See Practices 2, 3, 4, and 20.

An iteration is typically a few weeks long, and each iteration will deliver an executable that is one step closer to the final product, with the potential exception of the first iteration. During an iteration, the team will work on defining, designing, implementing, and testing the requirements listed in the iteration plan, as well as any other planned work items. At the end of each iteration, the team will assess what was accomplished and demonstrate the executable to stakeholders. The resulting feedback will enable the team to improve upon the solution and understand what are the most important work items for the next iteration. Lessons learned during the retrospective improve the process for the next iteration.

See Practices 5–6, 8–10, 14, and 18.

A *vision* outlines stakeholder needs and high-level features, establishing a common understanding of the domain. Functional requirements are documented as use cases and scenarios, and other requirements are documented as supplementary requirements. The requirements are incre-

mentally implemented by growing the design and implementation in stages, while continuously testing and integrating the code into builds. A strong emphasis is placed on test-and-build automation to enable frequent and high-quality builds.

OpenUP/Basic is steeped in the belief that the team needs to take responsibility for the end product, to which everybody chips in as needed. To improve productivity and quality, the team should reuse existing assets, such as patterns and common components, as appropriate. OpenUP/Basic also puts a strong emphasis on the architecture; a stable architecture is established early in the project and evolves continuously along with the application. Components and services are leveraged as appropriate, and the architecture also impacts how responsibilities are divided within the team.

See Practices 7, 11–13, and 15–17.

OpenUP/Basic is a subset of OpenUP, and is delivered through the EPF (see Appendix A). It is also the basis for RUP.

Rational Unified Process (RUP)

RUP is a widely adopted process framework used by tens of thousands of projects ranging from teams of two to teams with hundreds of members, in a broad variety of industries worldwide. It extends EPF with a large volume of additional process content, allowing development teams to scale their process to do the following:

RUP extends EPF with additional process content.

- Carry out distributed or large-scale development requiring more ceremony, such as requirements traceability, analysis models, model-driven architecture (MDA), or comprehensive testing of load and performance.
- Develop systems using IBM tools, providing specific guidance on relevant technologies such as J2EE and .NET, and using IBM and partner tools.
- Develop systems adhering to industry standards such as ISO 9001, SEI CMMI, or SOX.
- Scale from project-oriented processes to enterprise processes, such as program and portfolio management; systems engineering;

enterprise reuse; business modeling and simulation; or enterprise-scale SOA.

RUP provides a collection of processes that can either be used "out of the box" or further customized.

RUP follows the Unified Process lifecycle and adheres to the key principles described above. RUP emphasizes the importance of adapting the process to the needs of your project. Rather than providing one process, such as OpenUP/Basic, RUP provides a collection of processes that can either be used out-of-the-box or further customized. These processes are variations built on OpenUP/Basic and include processes for Large-Scale Custom Application Development, Commercial-Off-The-Shelf (COTS) Packaged Delivery (see Figure 1.6), Service-Oriented Architecture Development, Systems Engineering, Small Projects, Maintenance Projects, Portfolio Management, and so on. More processes are continually being added. RUP allows you to customize these processes or build your own process through the product IBM Rational Method Composer, which is the delivery vehicle for RUP. The product is described in more detail in Appendix B.

One of the key ideas behind RUP is that a process is much more valuable when automated by tools. Therefore, a fundamental aspect of RUP is the tight integration with developer tools through context-sensitive process guidance available within the tools and tool-specific guidance available in the process, which makes RUP an integral part of the development environment. RUP is also tightly integrated with tools that allow teams to instantiate their process, enabling adaptive planning of projects and collaborative software development (see Appendix B for more information). Even though RUP is integrated with IBM and other tools, it does not require the use of any one set of tools.

RUP is described in a variety of white papers and books.[21] The most comprehensive information can be found in the Rational Method Composer (RMC) product, which contains detailed guidelines, examples, and templates covering the full project lifecycle. However, RUP underwent extensive modernization in 2005, and material or product versions predating 2006 may use different terminology.

21. See Kruchten 2003 and Kroll 2003.

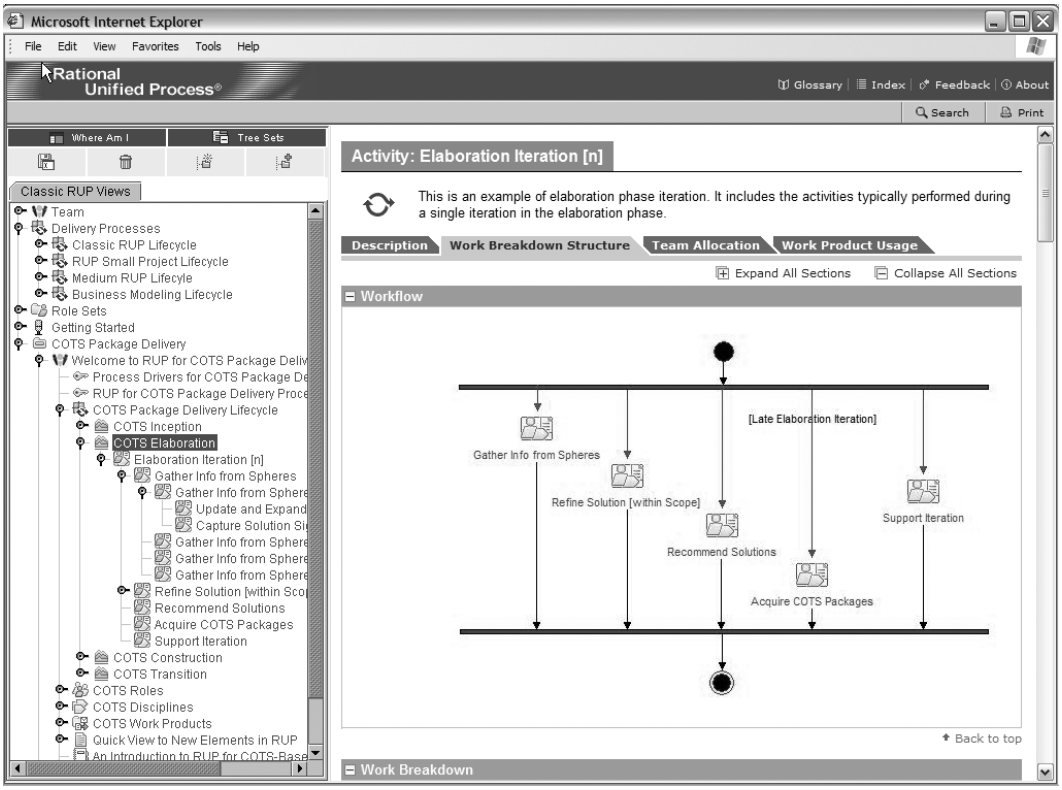

FIGURE 1.6 RUP for Commercial-Off-The-Shelf (COTS) Packaged Application Delivery. *RUP provides many out-of-the-box processes for different types of projects, including COTS packaged application delivery. For COTS development, RUP provides specific guidance on how to evaluate commercial components or packages, how to deal with a Request for Information (RFI) and a Request for Proposals (RFP), and how to make trade-offs to balance the many conflicts between stakeholder needs, architecture, program risk, and market concerns.*

eXtreme Programming (XP)

eXtreme Programming (XP) is one of the best-known agile processes. Created by Kent Beck,[22] it is considered by many to be "glorified hacking,"

22. Beck 2004.

but that is far from the case. XP is a disciplined approach, requiring skilled people who are committed to adhering closely to a core set of principles.

XP is a disciplined approach, requiring skilled people adhering to a core set of principles.

XP articulates five values to guide you in your project: *communication, simplicity, feedback, courage,* and *respect.* Further, it prescribes a set of practices to make these values concrete. Although it may be unclear whether somebody really adheres to a value, you can easily tell whether somebody adheres to the practice. XP practices are divided into *primary* and *secondary.* The primary practices are listed below.

- *Sit together* helps you to communicate more effectively by being physically collocated in the same room or office space.
- *Whole team* talks about the importance of building a cohesive team with a diverse set of skills required to complete the project.
- *Informative workspace* tells you that if an outsider spends 15 seconds in your workspace, he or she should be able to get a general idea of how the project is going. What are the issues you are facing and items you are working on?
- *Energized work* guides you in adjusting your work hours so that you function effectively when working and avoid burnout.
- *Pair programming* tells you to write all production code in pairs, with each person taking turns watching and assisting the other programmer write code.
- *Stories* allow you to specify, in one or two sentences, capabilities that typically take one or two days to implement. The customer prioritizes which stories to implement and in what order.
- *Weekly cycle* means that at the beginning of each week you plan what should be accomplished for that week by assessing status, prioritizing user stories, and dividing user stories into tasks that programmers sign up for.
- *Quarterly cycle* allows you to step back and determine how to improve process, remove bottlenecks, focus on the big picture of where to take the projects, and do coarse-grained planning for the next quarter.
- *Slack* is built in to the schedule as tasks that can be dropped or by assigning certain time slots as slack time.

- *Ten-minute build* forces you to trim your automated build and automated tests so that they take no more than 10 minutes.
- *Continuous integration* aims at reducing the overall cost of integration by forcing it to happen at least once every couple of hours.
- *Test-first programming* tells you to write automated tests before writing the code to be tested.
- *Incremental design* guides you in doing a little bit of design every day, but designing only for what you need today rather than for future possibilities.

XP also articulates a set of fourteen principles that function as the bridge between values and practices, guiding you in how to apply the practices effectively in order to adhere to the values. The principles are humanity, economics, mutual benefit, self-similarity, improvement, diversity, reflection, flow, opportunity, redundancy, failure, quality, baby steps, and accepted responsibility.

Scrum

Scrum focuses on the management of a project.

Scrum[23] was introduced in 1996 by Ken Schwaber and Jeff Sutherland.[24] The term *scrum* is derived from rugby, in which it refers to restarting play after the game is stuck in a so-called maul. Scrum focuses on the management of a project without describing the specifics of how to undertake design, implementation, or testing. Consequently, it can be combined with a number of different processes, such as XP or RUP.

Each iteration is 30 days long and is referred to as a sprint. The sprint starts with a half-day planning meeting to determine what should be done within the sprint. During the planning meeting, the Product Owner provides a prioritized **Product Backlog,** which is an evolving list containing all requirements, defects, tasks, and change requests. The development team will then determine how many of the high-priority items they can take on within that sprint, which constitutes

23. Schwaber 2002.
24. Scrum was formalized and presented at OOPSLA'96.

the **Sprint Backlog,** and a sprint goal is crafted. The sprint goal provides the team with a clear focus by establishing an objective that will be met by implementing the backlog.

After the planning meeting, the team spends 29 days developing an executable that delivers incremental functionality. The sprint ends with a half-day Sprint Review Meeting, at which the team and management inspect the product increment together and capture lessons learned. The team will then start the next sprint, delivering an executable every 30 days that is one step closer to the final product.

During a sprint the team will get together for a daily 15-minute meeting, also called scrum. During the scrum, each team member will briefly answer three (and *only* three) questions:

1. What have you done since last meeting?
2. What will you do between now and next meeting?
3. What obstacles stood in the way of doing work?

The "scrum master" makes sure that the meeting is kept on track, makes decisions as appropriate, and is also responsible for ensuring that identified obstacles are addressed as soon as possible. As issues arise that need further discussion, the scrum master sets up additional meetings with involved parties to continue the discussion, to ensure that the daily scrum is kept to 15 minutes. The scrum master also makes sure that no outside changes are imposed on the plan during a sprint, so that the team can focus on executing the plan without any distracting interruptions.

The team in scrum should have seven (give or take two) members. If your team has more, split the team into several teams and divide their work so that they can operate semi-independently. Scrum teams should be cross-functional so that each team can do the necessary analysis, design, implementation, testing, and documentation.

Scrum teams are self-organized, meaning that the team is left loose during the sprint to use whatever approach is considered appropriate to convert the Sprint Backlog to a product increment. Management should not intervene. At the end of the sprint, management can assess what was done and make changes as necessary.

Scrum focuses on collaboration. Open workspaces are preferred, and each team member should attend the daily scrum meeting in person. No titles or job descriptions are used, and people are asked to leave their egos at home.

Scrum focuses on collaboration. Each team member attends the daily scrum meeting.

Summary

This chapter introduces key principles and practices that support them. It also presents the framework for this book, as outlined below:

- **Comparison with other methods.** For each practice, we describe how the practice is similar to, or different from, other methods.
- **Levels of adoption.** Each practice can be adopted at three levels: basic, intermediate, and advanced. The advanced level typically requires a combination of more advanced skills, more advanced tooling, or a higher-ceremony process.
- **Process map.** For each level of adoption of a practice, we indicate whether the practice enables agility or discipline and whether it enables shorter iterations.
- **Information in the Unified Process.** For each practice, we describe how OpenUP/Basic and RUP support these practices.

To enable you to make better use of this framework, this chapter provides a summary of the Unified Process lifecycle, OpenUP/Basic, and RUP and explains how they fit together in the family of Unified Process methods built on EPF and RMC. This chapter also summarizes Scrum and XP, because we have chosen to focus on these methods as part of the "comparison with other methods" section in each practice.

With this background, we hope you can now pick and choose a practice chapter of interest and gain valuable insight into how to improve your process. We recommend that you start with the basic level of adoption and incrementally adopt some intermediate and advanced practices based on whether you need to be more iterative, agile, or disciplined.