An Introduction to the Rational Unified Process

The purpose of this article is to introduce the Rational Unified Process (RUP). We start with a discussion of the software best practices that form the foundation of RUP. We then discuss the key concepts and overall organization of RUP (the RUP "architecture"), and then close with a discussion of the use of RUP as a process framework for developing a customized process.

Note: This article is an excerpt from the book, *Building J2EE Applications with the Rational Unified Process* (Addison Wesley, 2002).

Best Practices–The Foundation of RUP

The Rational Unified Process is a software development process framework that provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget ("better software faster").

Best practices are a set of commercially proven approaches to software development. When used in combination, best practices ensure the success of a software development project by striking at the root causes of typical software development problems. RUP was explicitly designed to support the implementation of six best practices[1]:

- **Develop iteratively**. Deliver the functionality of the system in a successive series of releases of increasing completeness, where each release is an iteration. The selection of which requirements are developed within each iteration is driven by the mitigation of project risks, with the most critical risks being addressed first.
- **Manage requirements**. Use a systematic approach to elicit and document the system requirements, and then manage changes to those requirements, including assessing the impact of those changes on the rest of the system. Effective requirements management involves maintaining a clear statement of the requirements, as well as maintaining traceability from these requirements to the other project artifacts.
- **Use component architectures**. Structure the software architecture using components[2]. A component-based development approach to architecture tends to reduce the complexity of the solution, and results in an architecture that is more robust and resilient, and which enables more effective reuse.

[1] There are additional best practices in RUP. These are just the ones we chose to highlight.

[2] *Components* are cohesive parts of a system with well-defined interfaces that provide strong encapsulation of their contents. Because their contents are "hidden," components may be replaced by other components that offer compatible interfaces.

- **Model visually**. Produce a set of visual models of the system, each of which emphasizes specific details, and "ignores" (abstracts, filters away) others. These models promote a better understanding of the system being developed and provide a mechanism for unambiguous communication among team members ("a picture is worth a thousand words").
- **Continuously verify quality**. Continuously assess the quality of the system with respect to its functional and nonfunctional requirements. Perform testing as part of every iteration. It is a lot less expensive to correct defects found early in the software development lifecycle than it is to fix defects found later.
- **Manage change**. Establish a disciplined and controlled approach for managing change (changing requirements, technology, resources, products, platforms, and so on). Control how changes are introduced into the project artifacts, who introduces the changes, and when those changes are introduced. Provide a means to efficiently synchronize those changes across the different development teams, releases, products, platforms, and so forth.

These best practices are the result of Rational's experiences in developing its software products together with the experiences of Rational's many customers. Implementing these best practices puts a software development organization in a much better position to deliver high-quality software in a repeatable and predictable fashion.

RUP Key Concepts

RUP can be described in terms of two dimensions: *time* and *content*. Figure 1 provides a graphical representation of these dimensions. The horizontal axis represents *time* and shows the life cycle aspects of the process. This dimension is described in terms of phases and iterations. The vertical axis represents *content* and shows the disciplines, which logically group the process content.

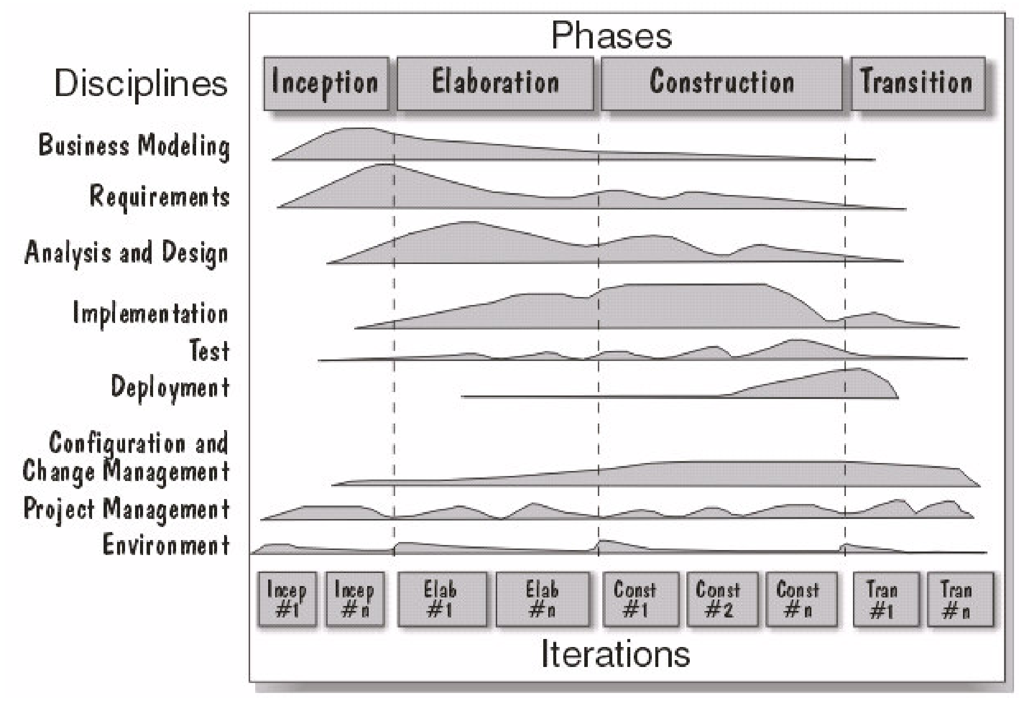

Figure 1
Time and Content Dimensions of RUP

As the "humps" in Figure 1 illustrate, the relative emphasis of the disciplines changes over the life of the project. For example, in early iterations more time is spent on Requirements, and in later iterations more time is spent on Implementation. Configuration and Change Management, Environment, and Project Management activities are performed throughout the project. Keep in mind, however, that all disciplines are considered within every iteration.

An effective software development process should describe *who* does *what*, *how*, and *when*. RUP does exactly that in terms of the following key concepts:

- Roles: The *who*
- Artifacts: The *what*
- Activities: The *how*
- Phases, iterations, disciplines and workflow details: The *when*

The RUP key concepts and their relationships are shown in Figure 2.

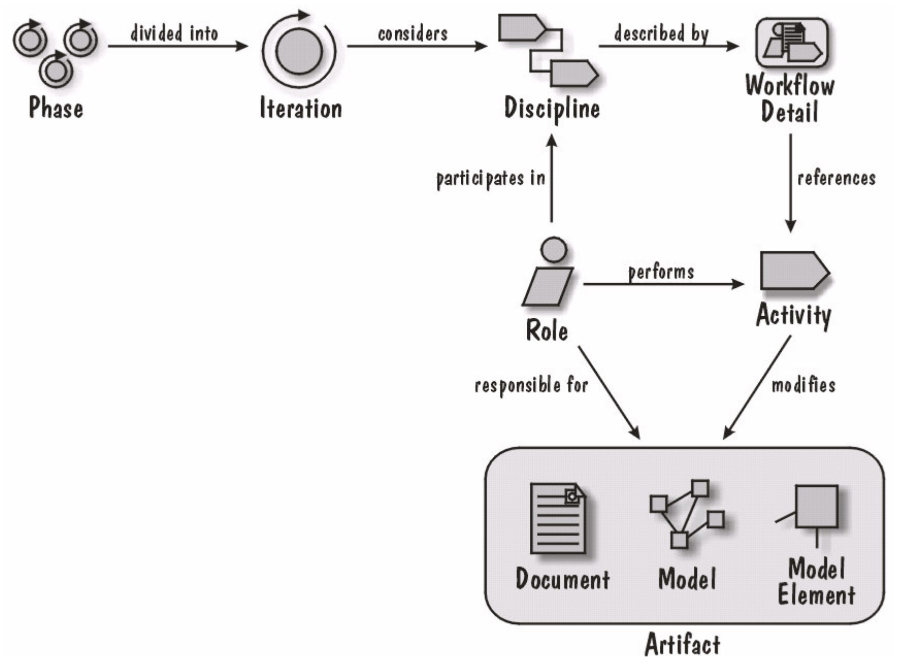

Figure 2
RUP Key Concepts

A software development project moves through a number of *phases* each of which is divided into a number of *iterations*. Within each iteration, we consider the various *disciplines*. The activities within a discipline are described in terms of *workflow details*. Workflow details describe *activities* that are usually performed together, the *roles* that perform those activities, and the resulting artifacts.

Artifacts

An *artifact*[3] is a piece of information that is produced and/or used during the execution of the process. Artifacts are the tangible by-products of the process. The deliverables that end up in the hands of the customers and end users are only a subset of the artifacts that are produced on a project.

Artifacts may take various shapes or forms:

- A model, such as a **Use-Case Model** or **Design Model**[4], which contains model elements
- A model element, such as a **Use Case, Design Class**, or **Design Subsystem,** which is part of a model
- A document, such as a **Software Architecture Document**
- Source code
- An executable
- Project plan

[3] Other common terms that have the same meaning as "artifact" include work product, work unit, deliverable, and so on.
[4] Throughout this article, we represent artifacts in the text using bold-faced text.

Artifacts are the responsibility of a single role. Roles use artifacts as input to activities, and roles produce or modify artifacts in the course of performing activities.

Artifacts are represented in RUP using graphical symbols. Some examples of RUP artifacts are shown in Figure 3.

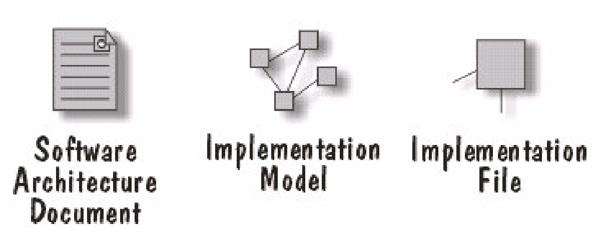

Figure 3
Examples of RUP Artifacts

Roles

A *role* defines the behavior and responsibilities of an individual, or a set of individuals working together as a team, within the context of a software development organization. A role is responsible for one or more artifacts and performs a set of activities. For example, the **Designer** role, as described in the J2EE Developer Roadmap[5], defines the responsibilities, operations, attributes, and relationships of one or several **Design Classes**, and determines how they will be implemented.

It is important to emphasize that roles are not individuals. Individuals may play multiple roles ("wear multiple hats") and multiple individuals may play a single role. The **Project Manager** performs the mapping from individuals to roles when planning and staffing the project.

Roles are also represented in RUP graphically as shown in Figure 4.

Software Architect
Designer
Implementer

Figure 4
Examples of RUP Roles

Activities

An *activity* is a unit of work that provides a meaningful result in the context of the project. It has a clear purpose, which usually involves creating or updating artifacts. Every activity is assigned to a specific role. Activities may be repeated several times, especially when executed in different iterations.

[5] The J2EE Developer Roadmap is a customized version of the RUP described in the book, *Building J2EE Applications with the Rational Unified Process* (Addison Wesley, 2002).

Activities are composed of one or more steps. For example, the activity **Implement Design Elements** of the Implementation discipline in the J2EE Developer Roadmap has the following steps:

- Implement Design Subsystems
- Implement Framework Components
- Implement Design Classes and Interfaces
- Implement Deployment Elements

An activity is displayed in RUP as shown in Figure 5.

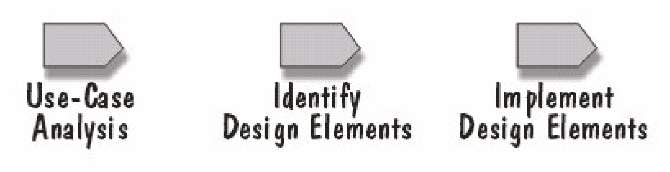

Figure 5
Examples of RUP Activities

Disciplines

A *discipline* is a collection of activities that are related to a major "area of concern" within the overall project. Disciplines group activities logically. As we show in Figure 1, RUP is organized around nine disciplines. Table 1 provides a brief description of each of these disciplines.

Table 1
RUP Disciplines

RUP Discipline	**Brief Description**
Business Modeling	The purpose of the Business Modeling discipline is to: - Understand the structure and the dynamics of the organization in which a system is to be deployed (the target organization) - Understand current problems in the target organization and identify improvement potential - Ensure that customers, end users, and developers have a common understanding of the target organization - Derive the system requirements needed to support the target organization
Requirements	The purpose of the Requirements discipline is to: - Establish and maintain agreement with the customers and other stakeholders on what the system should do - Provide system developers with a better understanding of the system requirements - Define the boundaries of (delimit) the system - Provide a basis for planning the technical contents of iterations - Provide a basis for estimating the cost and time to develop the system.
Analysis and Design	The purpose of the Analysis and Design discipline is to: - Transform the requirements into a design of the system-to-be

	- Evolve a robust architecture for the system - Adapt the design to match the implementation environment
Implementation	The purpose of the Implementation discipline is to: - Define the organization of the implementation - Implement the design elements - Unit test the implementation - Integrate the results produced by individual implementers (or teams), resulting in an executable system
Test	The purpose of the Test discipline is to - Find and document defects in software quality - Provide general advice about perceived software quality - Prove the validity of the assumptions made in design and requirement specifications through concrete demonstration - Validate that the software product functions as designed - Validate that the software product functions as required (that is, the requirements have been implemented appropriately)
Deployment	The purpose of the Deployment discipline is to: - Ensure that the software product is available for its end users
Configuration and Change Management	The purpose of the Configuration and Change Management discipline is to: - Identify configuration items[6] - Restrict changes to those items. - Audit changes made to those items. - Define and manage configurations[7] of those items
Project Management	The purpose of the Project Management discipline is to: - Manage a software-intensive project - Plan, staff, execute, and monitor a project - Manage risk
Environment	The purpose of the Environment discipline is to: - Provide the software development organization with the software development environment—both processes and tools—that will support the development team. This includes configuring the process for a particular project, as well as developing guidelines in support of the project.

Workflow Details

It is not enough to describe the artifacts produced by a process and the roles that perform the activities that produce these artifacts. A process also needs to provide guidance on what activities are performed together and the order in which the activities are performed. Such guidance can be expressed in a set of workflows. A *workflow* is a sequence of activities that produces a result of observable value.

[6] *Configuration items* are elements that should be placed under configuration management control. Configuration items are individually versioned and thus can be uniquely identified. Configuration items are part of a configuration.

[7] A *configuration* is the set of configuration items that define a particular version of a system or part of a system.

In RUP, the process is described at two levels: the discipline level and the workflow detail level. A *workflow detail* is a grouping of activities that are often performed together to produce a specific result. In particular, workflow details describe groups of activities performed together in a discipline.

The workflows for the RUP disciplines and workflow details are described using Unified Modeling Language (UML) activity diagrams. Discipline diagrams contain the workflow details of the discipline. The diagram for the Implementation discipline, as configured in the J2EE Developer Roadmap, is shown in Figure 6. This discipline consists of two workflow details: **Structure the Implementation Model** and **Implement Design Elements**.

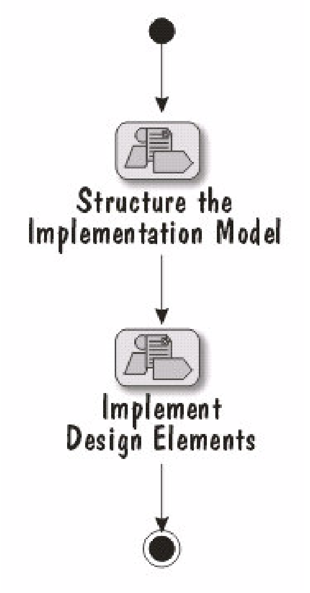

Figure 6
Implementation Discipline Workflow

Within a workflow detail, activities may be performed in parallel, and each activity may affect more than one artifact. Workflow detail diagrams show the key artifacts, activities, and roles involved in this workflow detail. The content of the **Implement Design Elements** workflow detail, as tailored in the J2EE Developer Roadmap, is shown in Figure 7. This workflow detail consists of three activities: **Implement Design Elements, Perform Unit Tests**, and **Review the Implementation**.

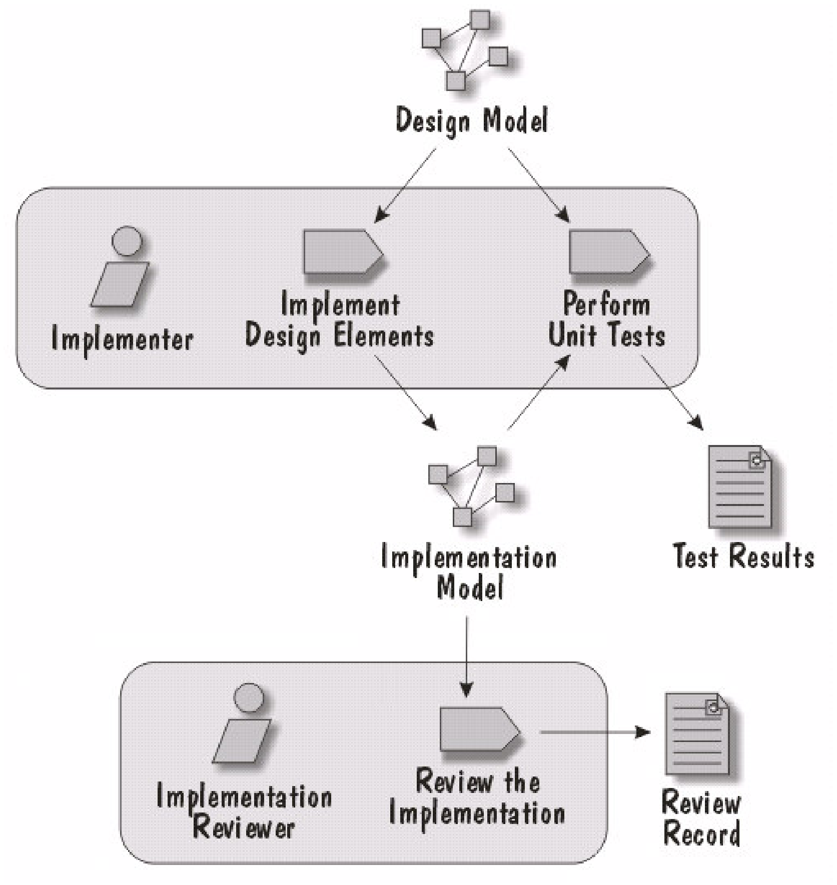

Figure 7
Implement Design Elements Workflow Detail

Iterations

Iterative development is a key characteristic of successful software development projects. Within an iterative software development lifecycle, several passes are made through each of the disciplines. Each pass is called an iteration. An *iteration* is a distinct, time-boxed sequence of activities that results in a release (internal or external) of an executable product. As the project progresses, releases evolve from a subset of the final product to the final system. An iterative development process is similar to "growing" software, where the end product matures over time. Each iteration results in a better understanding of the requirements, a more robust architecture, a more experienced development organization, and a more complete implementation.

Figure 8 illustrates how the focus of a project shifts across successive iterations. The size of the boxes within each of the disciplines illustrates the relative time spent performing the activities within that discipline. Each discipline is addressed during every iteration, but the relative emphasis shifts as the project progresses from Requirements to Analysis and Design to Implementation to Test, and finally to Deployment.

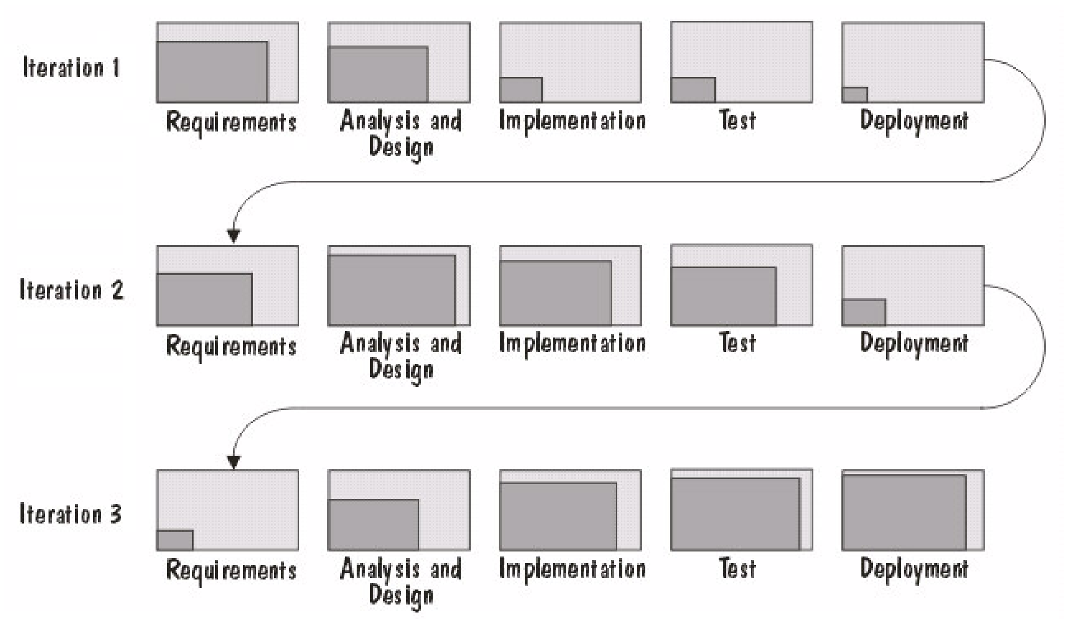

Figure 8
An Iterative Life Cycle

The following list provides some important characteristics of a successful iteration.

- The iteration has clear evaluation criteria.
- The iteration has a planned capability that is demonstrable.
- The iteration is concluded by a minor milestone, where the result of the iteration is assessed relative to the objective success criteria of that particular iteration.
- During the iteration, artifacts are updated (artifacts evolve with the system).
- During the iteration, the system is integrated and tested.

Phases

There is more to an iterative development process than a stream of iterations. There must be an overall framework in which the iterations are performed that represents the strategic plan for the project and drives the goals and objectives of each of the iterations. Such a framework is provided by phases. *Phases* provide well-defined business milestones that ensure that the iterations make progress and converge on a solution, rather than just iterating indefinitely.

Phases and iterations together provide the foundation for iterative development. The objectives of each phase are achieved by executing one or more iterations within the phase. Each phase concludes with a major milestone and an assessment to determine whether the objectives of the phase have been met. A satisfactory assessment allows the project to move to the next phase. Iterations are time-based (they are of a fixed duration), whereas phases are goal-based. A phase cannot be time-boxed since the completion of a phase is assessed based on the state of the project.

The software life cycle of the RUP is decomposed into four sequential phases: *Inception*, *Elaboration*, *Construction*, and *Transition*. These phases and their milestones are shown in Figure 9.

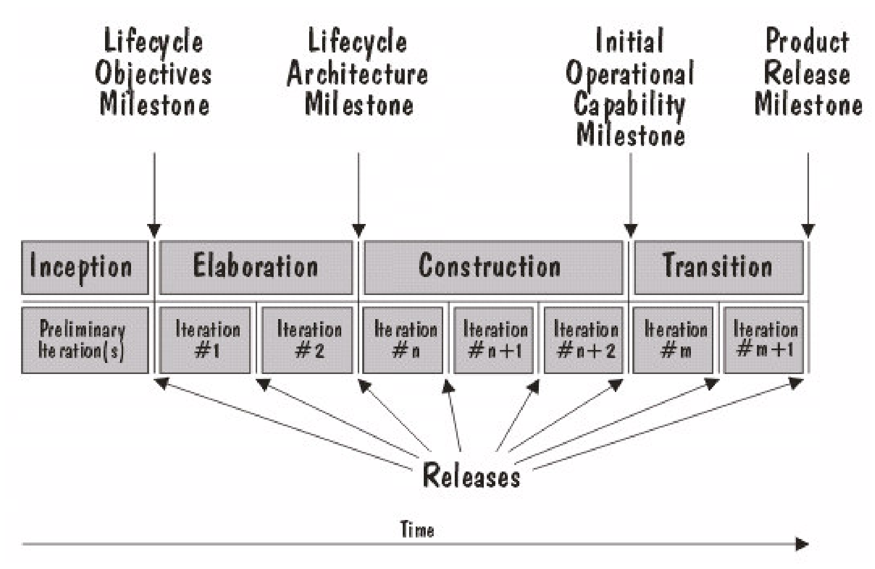

Figure 9
RUP Phases and Milestones

Inception

The Inception phase is where the "business case" for the project is established, and where concurrence among all stakeholders on the objectives for the project is also established. Inception is where we focus is on ensuring that the project is both valuable and feasible.

Table 2
Inception Phase Objectives and Evaluation Criteria

Primary Objectives	**Milestone Evaluation Criteria**
Establish the project's scope	Stakeholders concur with the scope definition.
Establish the project's acceptance criteria	Stakeholders concur with the project acceptance criteria.
Identify the features of the system, and select those that are critical	Stakeholders concur that the right set of requirements has been captured and there is a shared understanding of the requirements. All requirements are prioritized.
Estimate the overall cost and schedule for the entire project (and more detailed estimates for the Elaboration phase that will immediately follow)	Stakeholders concur that the cost/schedule estimates, priorities, risks, and development process are appropriate.
Estimate potential risks	All known risks have been recorded and assessed, and a risk mitigation strategy has been defined.
Set up the supporting environment for the project (for example, hardware, software, process, resources)	The supporting environment is in place.

Elaboration

The Elaboration phase is where the software architecture is established that provides a stable foundation for the design and implementation that is performed during the Construction phase.

Table 3
Elaboration Phase Objectives and Evaluation Criteria

Primary Objectives	Milestone Evaluation Criteria
Ensure that the architecture, requirements and plans are stable. Establish a baselined architecture.	The product vision, requirements and baselined architecture are stable.
Ensure that the risks are sufficiently mitigated to be able to predictably determine the cost and schedule for the completion of the development.	The major risk elements have been addressed and have been credibly resolved.
Demonstrate that the baselined architecture will support the requirements of the system at an acceptable cost and within an acceptable timeframe.	All architecturally significant aspects of the system, and selected areas of functionality, have been evaluated in an evolutionary prototype[8].

Construction

The Construction phase is where the remaining requirements are clarified and where development of the system is completed based on the baselined architecture established during the Elaboration phase. Between the Elaboration and Construction phases, the focus shifts from understanding the problem and identifying key elements of the solution, to the development of a deployable product.

Table 4
Construction Phase Objectives and Evaluation Criteria

Primary Objectives	Milestone Evaluation Criteria
Achieve useful versions (alpha, beta, and other test releases) in a timely fashion.	The system has been developed in line with the expectations specified in the phase plan and iteration plans.
Complete the analysis, design, implementation and testing of all required functionality.	All required functionality has been incorporated into the system.
Make sure the system is ready to be deployed into the end user's environment.	The system has met all acceptance criteria when tested in the development environment.

Transition

The Transition phase is where we ensure that the software is available to, and accepted by, its end users. This is where the system is deployed into the user's environment for evaluation and testing. The focus is on fine-tuning the product, and on addressing configuration, installation, and usability issues. All the major structural issues should have been worked out much earlier in the project's life cycle. By the end of the Transition phase, the project should be in a position to be closed out.

Table 5
Transition Phase Objectives and Evaluation Criteria

Primary Objectives	Milestone Evaluation Criteria
Successfully roll out the system to the delivery channels	The system has passed the formal acceptance criteria in the end user's environment.

[8] An *evolutionary prototype* gradually evolves to become the real system, as opposed to an *exploratory prototype*, which is thrown away when it has served its purpose.

RUP as a Process Framework

RUP is comprehensive and complete. It provides detailed activity steps, artifact templates, guidelines, checkpoints, and examples. However, a "one size fits all" perspective does not apply to a software development process. Thus, RUP was designed to be a process framework from which customized processes could be derived. In fact, in addition to *software development* guidance, RUP contains *process customization* guidance. In other words, RUP contains detailed information on how to tailor RUP for a specific project, type of solution, or organization.

Summary

In this article, we provided a brief introduction to the Rational Unified Process (RUP), including the software engineering best practices from which it originates, the key concepts used to describe it, and the support provided for tailoring it. In summary, RUP captures software development best practices in a form that can be adapted for a wide range of purposes.

For an example of how to develop a J2EE application using a customized version of the RUP explicitly tailored for the J2EE developer, we encourage you to take a look at our book, *Building J2EE Applications with the Rational Unified Process*, by Peter Eeles, Kelli Houston and Wojtek Kozaczynski (ISBN 0-201-79166-8, published by Addison-Wesley), from which this article is an excerpt.