Applying the Rational Unified Process:
A Web Service Sample

The Rational Unified Process (RUP) is a disciplined approach to engineering a piece of software. In this article, we'll walk through using the RUP and related products from Rational to develop a sample application. We'll describe the process used by the imaginary Catapulse Pacific Bank to migrate their currency exchange services onto the Web by providing a Web service — a component that runs on a server and can be invoked using an XML-based, operating system–independent protocol. Where relevant, we'll briefly describe the key technologies used to develop the project.

If you're new to the RUP, you'll find this sample useful as a starting point. Readers already familiar with the RUP (who may want to skip the next section) will be able to observe the RUP in action as applied to relatively new technologies.

About the RUP

The RUP is flexible and scalable enough to be applied to projects of varying sizes. Moreover, it's fairly independent of the type of software being developed. You can apply the RUP to standalone applications, client/server applications, and Web applications with equal facility.

In this article, we'll look at the key RUP **roles**, **activities**, and **artifacts** for our sample project. These are three of many terms defined in the RUP; most are self-explanatory, but a good source for additional information is the book *The Rational Unified Process: An Introduction*.

The RUP is an iterative rather than a sequential process. Nevertheless, it's broken up roughly into four phases — inception, elaboration, construction, and transition — each of which can be cycled through multiple times. We'll take our sample project through the first three phases and present artifacts to mark key milestones reached during these phases.

About Catapulse Pacific Bank

Catapulse Pacific Bank built its business on the basis of its currency exchange services in the 1980s and early part of the 1990s. Customers carried out currency exchange over the phone or on a walk-in basis at the bank. Soon, the popularity of the service prompted Catapulse Pacific to open counters at other banks, malls, and international tourist attractions.

When the Internet emerged as a powerful medium of conducting business, Catapulse Pacific seized the opportunity to put its currency exchange services online. The benefits would be enormous. By making its services globally accessible and round-the-clock, the bank could save its customers the inconvenience of finding an exchange location, making sure they arrived during business hours, and waiting in line. Catapulse Pacific would also save on the operational costs of running counters at multiple locations.

The Beginning: Inception

At the very beginning of a project — the **inception** phase — the business-process analyst plays a key role. (Like all roles, this one can be played by a team of people as well.) The business-process analyst's job is to capture the business requirements and the critical use cases for the project. Later we'll see how another analyst — the system analyst — contributes to some of the artifacts created during this phase.

The Vision Document

A key artifact to emerge from the business-process analyst's efforts is the **vision document**, which does the following:

- Describes the market and the reason or idea behind the product being developed.

- Defines the problem, explains the product concept, and positions the product vis-à-vis the company's priorities.

- Lists the different types of stakeholders and customers that are important to the product. (**Stakeholder** means someone with a vested interest in the end result of the system — for example, clients, business partners, and investors.) Thumbnail sketches of the different stakeholder and customer types describe the nature of their involvement in the system, why they're interested in the end result, and what deliverables they're expecting. This information allows subsequent roles involved in the project to identify the type of people they should be collecting input from and what needs to be delivered to them at various points in the project.

Since projects can produce copious output in the form of documents, Rational provides an application, called Rational RequisitePro, that not only manages requirements but also acts as a repository for all the documents generated during any phase of the system. (The documents are actually Microsoft Word documents, although you can access them only through RequisitePro; we'll learn more about RequisitePro in a moment.) Table 1 sums up the key points about the vision document; a similar table follows the description of each artifact introduced in this article.

Phase	Artifact(s)	Role(s)	Program used
Inception	Vision document	Business-process and system analysts	Rational RequisitePro

Table 1: Vision document

The Business Glossary

At the same time that the vision document is being written, another artifact is taking shape: the **business glossary**. This document keeps track of terms and acronyms specific to the problem domain, explaining terms that may be unfamiliar to the reader of the use case descriptions or other project documents. Functioning as a project dictionary of sorts, it's constantly updated throughout the life of the project. This brings us to a key point: artifacts generated in the RUP are not static but rather dynamic pieces of information; they can be modified during an iteration of the phase (or phases) they belong to.

Phase	Artifact(s)	Role(s)	Program used
All	Business glossary	Business-process and system analysts	Rational RequisitePro

Table 2: Business glossary

Using RequisitePro

Before you can create the first document in RequisitePro, you must identify the different types of documents that the project will produce. You do this by choosing Properties from the File menu and then clicking the Document Types tab; you can create new document types from there. RequisitePro provides templates for the entire set of documents that can be created in the RUP. When creating a new document, you can simply inherit from its template, and a sample document will be generated that contains the appropriate sections along with a brief explanation of what should be in each section. This provides an excellent jump-start; you don't need a thorough understanding of the RUP to get going.

It's not necessary to define all document types at the beginning of a project. You can come back and set up new types later. It's good practice to decide on the minimal set of documents your project requires and create document types for them up front, then create additional types as the need evolves.

The Business Use-Case Model

The business-process analyst also captures use cases during the inception phase, in the **business use-case model**. The use cases are focused around business entities; in other words, the analyst looks at the use cases the way a customer would. Use-case diagrams and other Unified Modeling Language (UML) diagrams are best captured in Rational Rose.

Phase	Artifact(s)	Role(s)	Program used
Inception	Business use-case model	Business-process analyst	Rational Rose

Table 3: Business use-case model

If you start Rose and load the sample use-case model referred to in this table, you can take a look at the use cases for Catapulse Pacific Bank. Rose takes you first to the main class diagram. One possible area of confusion when using Rose is that the program doesn't impose any kind of sequence in the way the system is modeled. Although this flexibility is generally a good thing, there tends to be no clear path in reading a Rose file. For our project, the first diagrams you should navigate to (through the top-left pane) are those in the Use Case View folder, in its Business Use-Case Model subfolder. There you'll find these uses cases:

- Global View, which outlines the higher-level features offered by the system
- Account Manager, which captures the functions related to creating and maintaining customer accounts

Moving On: The Elaboration Phase

We're ready to collect additional requirements during the **elaboration** phase. Whereas the focus is on high-level requirements during the inception phase, the focus during the elaboration phase is on more detailed, lower-level requirements.

Like the business-process analyst, the system analyst is responsible for gathering requirements and capturing them using models like use cases. This analyst captures the needs of customers and shapes the system accordingly, and also captures interactions between use cases — not necessarily doing all this work personally, but at least ensuring that it gets done.

The system analyst contributes to two earlier documents, the vision document and the business glossary, and also produces two additional documents: the requirements management plan and the stakeholder requests document. These documents are essential to setting the stage before going out to elicit and capture requirements. Let's take a closer look at each of these artifacts, and then at the actual requirements collection and creation.

The Requirements Management Plan

The **requirements management plan** (RMP) is sort of a "meta-requirements" document, in that it describes what type of requirements must be collected. In our sample, the system analyst decides to collect three types of requirements: features, needs, and change requests. Each requirement can be further described by properties such as importance, cost, and benefit. These properties are captured as **attributes**, and the RMP lists and describes each attribute for every type of requirement that will be collected.

Phase	Artifact(s)	Role(s)	Program used
Elaboration	Requirements management plan	System analyst	Rational RequisitePro

Table 4: Requirements management plan

Once the requirement types have been established, you must enter them (along with their attributes) into RequisitePro. The first step is to add the requirement types themselves. You do this by choosing Properties from the File menu and then clicking the Requirements tab. After entering the requirement types, you can add their attributes by clicking the Attributes tab.

Ideally you'd be able to simply enter the information about requirements into RequisitePro and have it generate the RMP for you; however, since that's not the case, you'll need to make sure the information is consistent in both places: the RMP document and RequisitePro.

The Stakeholder Requests Document

Along with the RMP, the system analyst creates the **stakeholder requests document**, which is used in collecting the actual requirements (as we'll see in the next section). The concept behind it is simple: you establish a set of questions, to be asked during a stakeholder interview, that will yield all the information needed to characterize a set of requirements. The stakeholder requests document lends structure to the process of requirements collection and analysis, thereby enabling the requirements to be categorized, prioritized, and documented consistently.

Phase	Artifact(s)	Role(s)	Program used
Elaboration	Stakeholder requests document	System analyst	Rational RequisitePro

Table 5: Stakeholder requests document

Requirements Collection and Creation

Armed with the above documents, the team can go about collecting the requirements. A second iteration of the inception phase can occur at this time to collect high-level requirements. The requirements collected during the elaboration phase must be sufficiently detailed to enable the system architect and designer to construct the system in the next phase.

The requirements are created in RequisitePro, where the attributes of each requirement can now be assigned values. These requirements can then be used to generate the **software requirements plan** (SRP). Our sample shows two iterations of an SRP document: one generated during the inception phase and another during the elaboration phase.

The software requirements plan is essential input into the **software requirements specification** (SRS). In our sample, we begin the SRS in the elaboration phase, recording the high-level requirements for what the service will do and how it will perform. (We'll finish the SRS in the construction phase, specifying more detailed requirements.)

Phase	Artifact(s)	Role(s)	Program used
Elaboration	Software requirements plan Software requirements specification	System analyst	Rational RequisitePro

Table 6: Software requirements documents

Also at this time, the business-process analyst can create the **business object model** (captured under that name in the Logical View of the system in Rose). This object model describes the realization of business use cases. In our sample, we use collaboration diagrams to model the three key functions to be provided by the system: show currency exchange rates, buy currency, and sell currency.

Design and Implementation: The Construction Phase

Once a sufficient set of requirements has been captured, the design and implementation of the system may begin, in the **construction** phase of the development lifecycle. During the design, we match our customers' requirements with existing technology to see how we can best build the system. In our sample, we use the Microsoft .NET Framework to produce a Web service. A **Web service** is a component that runs on a server and can be invoked using the Simple Object Access Protocol (SOAP), an XML-based, operating system–independent protocol. Microsoft .NET offers an implementation of SOAP over HTML (hence the name *Web* service). Results from the Web service are returned in XML.

Catapulse Pacific Bank will (in addition to moving account management online) provide a Web service to allow online currency exchange. It's expected that third-party banks or financial institutions will use this Web service and offer it to their clients as part of their services. In such cases, the client that consumes the Web service may be a script that's embedded in a vendor's Web page.

Each customer of the system must have previously set up an account, which involves creating a debit account and a credit account. (These accounts can be the same but are internally tracked as two separate accounts.) When a currency exchange is made, the money is moved from the debit account, converted to the desired currency, and moved into the credit account.

Clients of the Web service can browse a list of available currencies and query the exchange rate of each one. When the client requests a currency exchange, the Web service returns a transaction ID. Receipt of the transaction ID doesn't guarantee a transfer of funds. If approved, the actual transaction (transfer of funds) will occur later, but no later than in three hours.

As we'll see in the following sections, we make heavy use of Rational Rose when designing and implementing the system. We also finish the SRS during this phase, adding details to it that specify the data the service will use. By now the SRS should be at a level of detail that clearly spells out how system design will be executed.

Design

Rational Rose allows you to choose the most appropriate UML diagram for illustrating what you're trying to convey. From the top-left pane in Rose, select the Design Model folder. It has four subfolders, each one representing a different layer in the system and illustrated using the most appropriate diagram for that layer (see Table 7). As you can see, separating the design model into layers conveys an excellent breakdown of the system architecture.

Layer	Contents of layer	Diagram used in Rose
System software layer	Components such as operating systems, databases, and interfaces to specific hardware	Class diagram
Middleware layer	Components such as GUI builders, interfaces to database management systems, platform-independent operating system services, and OLE components (such as spreadsheets and diagram editors)	Class diagram

Business-specific layer	Business-specific components, used in several applications	None for our sample
Application layer	Application-specific services and value-added software developed by the organization	Sequence diagram

Table 7: Design model layers

A critical design artifact for our sample is the class diagram for the Web service, which can be found in the subsystem folder by that name in the design model's Application layer. This diagram shows the Web service interface as defined in the business object model and then breaks down the various classes and their methods and properties. Elements can be copied in Rose between different folders (that is, phases of implementation), in which case Rose will cross-reference the element by tagging it with its origin; you can see this in the Web service class diagram. The diagram also shows the various system modules that the Web service uses, allowing us to map dependencies between our modules and system services — dependencies that intimately affect development and deployment.

Phase	Artifact(s)	Role(s)	Program used
Construction	Web service class diagram	Business-process and system analysts, developers, testers, managers	Rational Rose

Table 8: Web service class diagram

Implementation

Our implementation is written in C# in Microsoft Visual Studio .NET Beta 1. (Pronounced "C sharp," this language is based on C++ but was designed specifically with Web services in mind.) If you're using a later beta version or the final release, you may encounter some differences in the implementation.

Phase	Artifact(s)	Role(s)	Program used
Construction	Source code	Developers, testers, managers	Microsoft Visual Studio .NET Beta 1

Table 9: Source code

The roles mentioned in Table 9 are really categories of roles. The RUP specifies the different roles within each category. Loading the source code file referred to here into

Visual Studio .NET will show the implementation of the Web service. Explaining how a Web service is written and executed is beyond the scope of this article, but you can consult some of the references provided under "Related Resources" to gain a thorough understanding.

The implementation details of the system are captured in the Component View in Rose. This view is used to set up a description of the subsystems that make up in the implementation. Rose is more restrictive here in terms of the diagrams that can be used: the only one allowed is a component diagram, because the others don't make sense in this context. The Currency Exchange Service diagram maps out the implementation of the sample.

For a step-by-step guide of how to build and test the Web service, see the accompanying document "Building and Testing the Web Service Sample."

Documentation

At this point, since you're probably getting familiar with Rose, let's explore another of its key features related to design. Rose allows a lot of documentation about the design to be embedded in the models themselves. Each element in Rose has associated documentation. To view the documentation, you first need to locate the element in a diagram. Each element is listed in the same view in which the diagram appears. For example, in the Implementation View under the component diagram, you'll see each of the elements in the form of packages and components used in the diagram. Double-clicking any of these elements in this view will bring up the element's specification, which contains additional documentation for that element. You can also bring up this specification by double-clicking the element from within the diagram itself.

If documentation is embedded throughout the Rose diagrams, you may wonder whether you have to navigate to every element to view all of it. Here the Rational Development Suite comes to the rescue with a report-generation utility called Rational SoDA. SoDA allows contents of the Rose file to be selectively saved to a Microsoft Word document. SoDA also works with RequisitePro; you can select the type of information you want from RequisitePro and use SoDA to save it to a Word file.

Phase	Artifact(s)	Role(s)	Program used
Inception	Use-case model survey (report containing use-case diagrams and documentation from Rose)	Business-process and system analysts	Rational SoDA
Elaboration	Requirements summary (report containing description field of each requirement in RequisitePro)	System analyst	Rational SoDA
	Requirements details (report containing requirements in RequisitePro, with description and attributes of each)		
Construction	Design document (report containing analysis and design views from Rose)	System analyst, system architect, developers, testers, managers	Rational SoDA

Table 10: SoDA reports

Summary

We've seen how the RUP can be applied to a project that migrates a bank's currency exchange services onto the Web. We've looked at the first three RUP phases — inception, elaboration, and construction — and the related roles, activities, and artifacts. Table 11 summarizes (by phase) the artifacts that entered into our sample project.

Phase	Artifact(s)	Role(s)	Program used
Inception	Vision document	Business-process and system analysts	Rational RequisitePro
	Business use-case model	Business-process analyst	Rational Rose
	Use-case model survey	Business-process and system analysts	Rational SoDA

Elaboration	Requirements management plan Stakeholder requests document Software requirements plan Software requirements specification (started)	System analyst	Rational RequisitePro
	Requirements summary Requirements details	System analyst	Rational SoDA
Construction	Web service class diagram	Business-process and system analysts, system architect, developers, testers, managers	Rational Rose
	Source code	Developers, testers, managers	Microsoft Visual Studio .NET Beta 1
	Software requirements specification (completed)	System analyst	Rational RequisitePro
	Design document	System analyst, system architect, developers, testers, managers	Rational SoDA
All	Business glossary	Business-process and system analysts	Rational RequisitePro

Table 11: Summary of sample artifacts

In the fourth and final phase, transition, the software product is moved to the user community; for the sake of brevity, we've omitted exploring this phase of our sample project, but you can learn more about this phase and any other aspects of the RUP from the introductory RUP book or the RUP product itself.

Related Resources

- *The Rational Unified Process: An Introduction, Second Edition* by Philippe Kruchten (Addison-Wesley, 2000)

- Articles from Microsoft: “Develop a Web Service: Up and Running with the SOAP Toolkit for Visual Studio” by Rob Caron, and “Web Services with ASP.NET” by Rob Howard

- .NET sample Web services from GotDotNet

About the Author

Aspi Havewala is an independent software consultant with extensive experience developing products for various markets (including managing software projects, consulting, and designing and developing device drivers and both desktop and Web applications). He also writes technical articles on programming, process, and management.