What Is the Rational Unified Process?

by Philippe Kruchten
Rational Fellow
Rational Software Canada

What exactly is the Rational Unified Process, or RUP as many call it now? I can give several answers to this question, from different perspectives:

- ***What is the purpose of the RUP?*** *It is a software engineering process, aimed at guiding software development organizations in their endeavors.*
- ***How is the RUP designed and delivered?*** *It is a process product, designed like any software product, and integrated with the Rational suites of software development tools.*
- ***What is the structure of the RUP; how is it organized internally?*** *The RUP has a very well-defined and regular structure, using an object-oriented approach for its description.*
- ***How would an organization proceed to adopt the RUP?*** *The RUP is a process framework that allows a software development organization to tailor or extend the RUP to match its specific needs.*
- ***What will I find in the RUP?*** *It captures many of modern software development's best practices harvested by Rational over the years, in a form suitable for a wide range of projects and organizations.*

The RUP Is a Software Engineering Process

Many organizations have slowly become aware of just how important a well-defined and well-documented software development process is to the success of their software projects. The development of the CMM (Capability Maturity Model) by the Software Engineering Institute (SEI) has become a beacon, a standard to which many organizations look, when they aim at attaining level 2, 3, or higher. Over the years, these organizations have collected their knowledge and shared it with their developers. This collective know-how often grows out of design methods, published textbooks, training programs, and small how-to notes amassed internally over several projects. Unfortunately, in practice, these internally developed processes often end up gathering dust in nice binders on a developer's shelf -- rarely updated, rapidly becoming obsolete, and almost never followed. Other software development organizations have no process at all, and need a starting point, an initial process to jump-start them on the path of faster development of better quality software products.

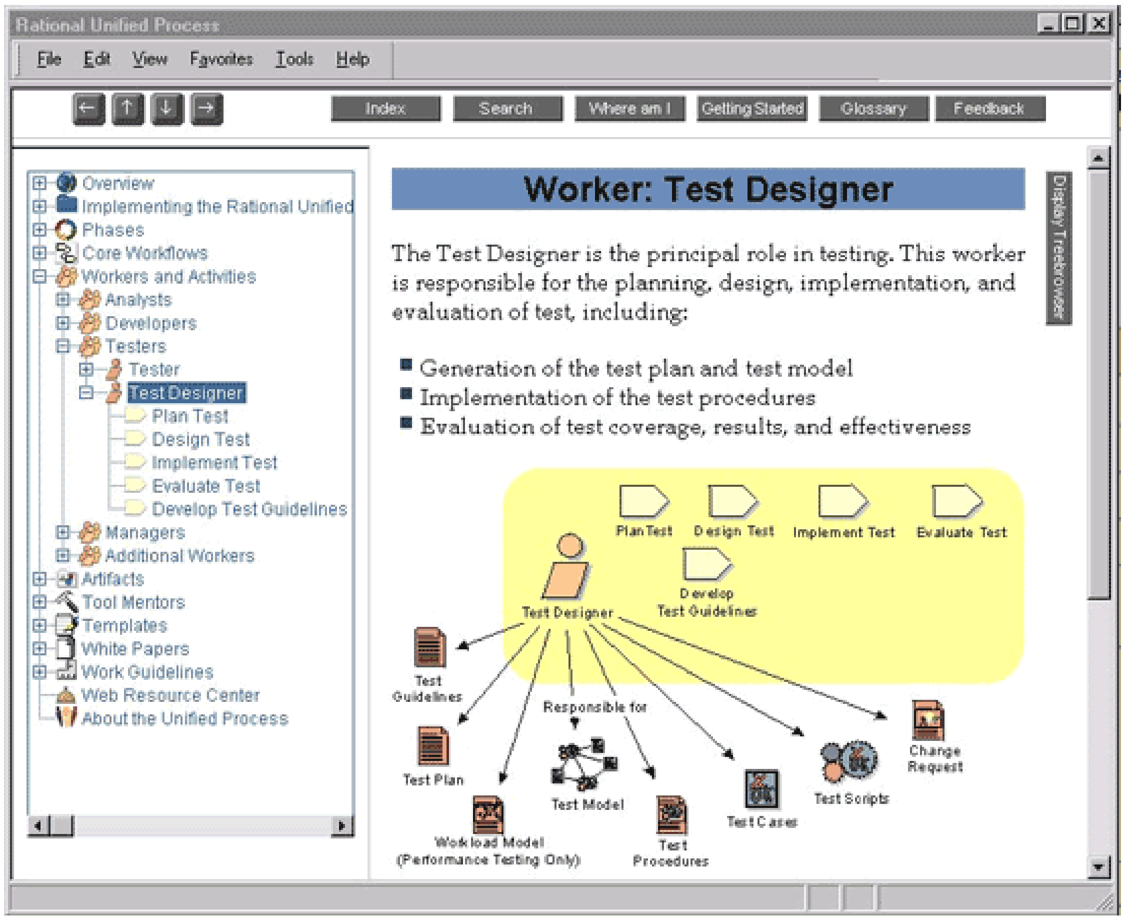
The RUP can help both kinds of organizations, by providing them with a mature, rigorous, and flexible software engineering process.

The RUP Is a Process Product

The RUP is not just a book, a development method developed and published once and for all in paper form. "Software processes are software, too," wrote Lee Osterweil, Professor of Computer Science at the University of Massachusetts. In contrast with the dusty binder approach, the Rational Unified Process is designed, developed, delivered, and maintained like any software tool. The Rational Unified Process shares many characteristics with software products:

- Like a software product, the Rational Unified Process is designed and documented using the Unified Modeling Language (UML). An underlying object model, the Unified Software Process Model (USPM) provides a very coherent backbone to the process.
- It is delivered online using Web technology, not in books or binders, so it's literally at the developers' fingertips.
- Regular software upgrades are released by Rational Software approximately twice a year. So the process is never obsolete, and its users benefit from the latest development. All team members access the same version of the process.
- Because it is modular and in electronic form, it can be tailored and configured to suit the specific needs of a development organization, something that's hard to do with a book or a binder.
- It is integrated with the many software development tools in the Rational Suites, so developers can access process guidance within the tool they are using.

Figure 1 shows a page from the RUP.



Worker: Test Designer

The Test Designer is the principal role in testing. This worker is responsible for the planning, design, implementation, and evaluation of test, including:

- Generation of the test plan and test model
- Implementation of the test procedures
- Evaluation of test coverage, results, and effectiveness

Figure 1: A Page from the RUP
(View full size graphic in new window)

The Architecture of the RUP

The process itself has been designed using techniques similar to those for software design. In particular, it has an underlying object-oriented model, using UML. Figure 2 shows the overall architecture of the Rational Unified Process. The process has two structures or, if you prefer, two dimensions:

- The horizontal dimension represents time and shows the lifecycle aspects of the process as it unfolds.
- The vertical dimension represents core process disciplines (or workflows), which logically group software engineering activities by their nature.

The first (horizontal) dimension represents the *dynamic aspect* of the process expressed in terms of cycles, phases, iterations, and milestones. In the RUP, a software product is designed and built in a succession of incremental iterations. This allows testing and validation of design ideas, as well as risk mitigation, to occur earlier in the lifecycle. The second (vertical) dimension represents the *static aspect* of the process described in terms of process components: activities, disciplines, artifacts, and roles.

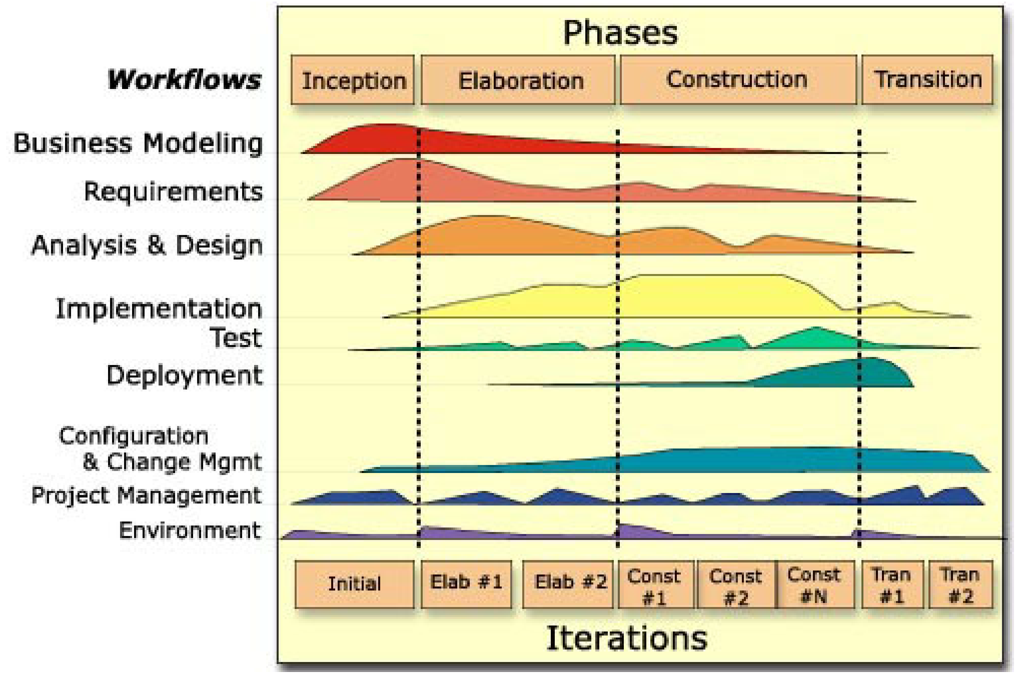


Figure 2 - Two Dimensions of the RUP

The RUP Is a Process Framework

The Rational Unified Process is also a *process framework* that can be adapted and extended to suit the needs of an adopting organization. It is general and comprehensive enough to be used "as is," i.e., out-of-the-box, by many small-to-medium software development organizations, especially those that do not have a very strong process culture. But the adopting organization can also modify, adjust, and expand the Rational Unified Process to accommodate the specific needs, characteristics, constraints, and history of its organization, culture, and domain. A process should not be followed blindly, generating useless work and producing artifacts that are of little added value. Instead, the process must be made as lean as possible while still fulfilling its mission to help developers rapidly produce predictably high-quality software. The best practices of the adopting organization, along with its specific rules and procedures, should complement the process.

The process elements that are likely to be modified, customized, added, or suppressed include artifacts, activities, workers, and workflows as well as guidelines and artifact templates. The Rational Unified Process itself contains the roles, activities, artifacts, guidelines, and examples necessary for its modification and configuration by the adopting organization. Moreover, these activities are also supported by the Rational Process Workbench™ (RPW) tool. This new tool uses a UML model of the Rational Unified Process to support process design and authoring activities, and the production of company-specific or project-specific RUP variants, called *development cases*.

Starting in 2000, the RUP contains several *variants*, or pre-packaged development cases for different types of software development organizations.

The RUP Captures Software Development Best Practices

The Rational Unified Process captures many of modern software development's *best practices* in a form suitable for a wide range of projects and organizations:

- Develop software iteratively.
- Manage requirements.
- Use component-based architectures.
- Visually model software.
- Continuously verify software quality.
- Control changes to software.

1. Develop Software Iteratively

Most software teams still use a *waterfall* process for development projects, completing in strict sequence the phases of requirement analysis, design, implementation/integration, and test. This inefficient approach idles key team members for extended periods and defers testing until the end of the project lifecycle, when problems tend to be tough and expensive to resolve, and pose a serious threat to release deadlines. By contrast, RUP represents an iterative approach that is superior for a number of reasons:

- It lets you take into account changing requirements. The truth is that requirements usually change. Requirements change and "requirements creep" -- the addition of requirements that are unnecessary and/or not customer-driven as a project progresses -- have always been primary sources of project trouble, leading to late delivery, missed schedules, dissatisfied customers, and frustrated developers.
- Integration is not one "big bang" at the end; instead, elements are integrated progressively -- almost continuously. With RUP, what used to be a lengthy time of uncertainty and pain -- taking up to 40% of the total effort at the end of a project -- is broken down into six to nine smaller integrations involving fewer elements.
- Risks are usually discovered or addressed during integration. With the iterative approach, you can mitigate risks earlier. As you unroll the early iterations, you test all process components, exercising many aspects of the project, such as tools, off-the-shelf software, people skills, and so on. You can quickly see whether perceived risks prove to be real and also uncover new, unsuspected risks

when they are easier and less costly to address.

- Iterative development provides management with a means of making tactical changes to the product -- to compete with existing products, for example. It allows you to release a product early with reduced functionality to counter a move by a competitor, or to adopt another vendor for a given technology.
- Iteration facilitates reuse; it is easier to identify common parts as they are partially designed or implemented than to recognize them during planning. Design reviews in early iterations allow architects to spot potential opportunities for reuse, and then develop and mature common code for these opportunities in subsequent iterations.
- When you can correct errors over several iterations, the result is a more robust architecture. As the product moves beyond inception into elaboration, flaws are detected even in early iterations rather than during a massive testing phase at the end. Performance bottlenecks are discovered at a time when they can still be addressed, instead of creating panic on the eve of delivery.
- Developers can learn along the way, and their various abilities and specialties are more fully employed during the entire lifecycle. Testers start testing early, technical writers begin writing early, and so on. In a non-iterative development, the same people would be waiting around to begin their work, making plan after plan but not making any concrete progress. What can a tester test when the product consists of only three feet of design documentation on a shelf? In addition, training needs, or the need for additional people, are spotted early, during assessment reviews.
- The development process itself can be improved and refined along the way. The assessment at the end of an iteration not only looks at the status of the project from a product or schedule perspective, but also analyzes what should be changed in the organization and in the process to make it perform better in the next iteration.

Project managers often resist the iterative approach, seeing it as a kind of endless and uncontrolled hacking. In the Rational Unified Process, the iterative approach is very controlled; the number, duration, and objectives of iterations are carefully planned, and the tasks and responsibilities of participants are well defined. In addition, objective measures of progress are captured. Some reworking takes place from one iteration to the next, but this, too, is carefully controlled.

2. Manage Requirements

Requirements management is a systematic approach to eliciting, organizing, communicating, and managing the changing requirements of a software-intensive system or application.

The benefits of effective requirements management are numerous:

- Better control of complex projects. This includes greater understanding of the intended system behavior as well as prevention of requirements creep.

- Improved software quality and customer satisfaction. The fundamental measure of quality is whether a system does what it is supposed to do. With the Rational Unified Process, this can be more easily assessed because all stakeholders have a common understanding of what must be built and tested.
- Reduced project costs and delays. Fixing errors in requirements is very expensive. With effective requirements management, you can decrease these errors early in the development, thereby cutting project costs and preventing delays.
- Improved team communication. Requirements management facilitates the involvement of users early in the process, helping to ensure that the application meets their needs. Well-managed requirements build a common understanding of the project needs and commitments among the stakeholders: users, customers, management, designers, and testers.

It is often difficult to look at a traditional object-oriented system model and tell how the system does what it is supposed to do. This difficulty stems from the lack of a consistent, visible thread through the system when it performs certain tasks. In the Rational Unified Process, *use cases* provide that thread by defining the behavior performed by a system.

Use cases are not required in object orientation, nor are they a compulsory vehicle in the Rational Unified Process. Where they are appropriate, however, they provide an important link between system requirements and other development artifacts, such as design and tests. Other object-oriented methods provide use-case-like representation but use different names for it, such as scenarios or threads.

The Rational Unified Process is a use-case-driven approach, which means that the use cases defined for the system can serve as the foundation for the rest of the development process. Use cases used for capturing requirements play a major role in several of the process workflows, especially design, test, user-interface design, and project management. They are also critical to business modeling.

3. Use Component-Based Architecture

Use cases drive the Rational Unified Process throughout the entire lifecycle, but design activities center on architecture -- either system architecture or, for software-intensive systems, software architecture. The main focus of early iterations is to produce and validate a software architecture. In the initial development cycle, this takes the form of an executable architectural prototype that gradually evolves, through subsequent iterations, into the final system.

The Rational Unified Process provides a methodical, systematic way to design, develop, and validate an architecture. It offers templates for describing an architecture based on the concept of multiple architectural views. It provides for the capture of architectural style, design rules, and constraints. The design process component contains specific activities aimed at identifying architectural constraints and architecturally significant elements, as well as guidelines on how to make architectural choices. The management process shows how planning the early iterations takes into account the design of an architecture and the resolution of major technical risks.

A *component* can be defined as a nontrivial piece of software: a module, package, or subsystem that fulfills a clear function, has a clear boundary, and can be integrated into a well-defined architecture. It is the physical realization of an abstraction in your design. Component-based development can proceed in several ways:

- In defining a modular architecture, you identify, isolate, design, develop, and test well-formed components. These components can be individually tested and gradually integrated to form the whole system.
- Furthermore, some of these components can be developed to be reusable, especially components that provide solutions to a wide range of common problems. Reusable components are typically larger than mere collections of utilities or class libraries. They form the basis of reuse within an organization, increasing overall software productivity and quality.
- More recently, the advent of commercially successful infrastructures supporting the concept of software components -- such as Common Object Request Broker Architecture (CORBA), the Internet, ActiveX, and JavaBeans -- has launched a whole industry of off-the-shelf components for various domains, allowing developers to buy and integrate components rather than develop them in-house.

The first point above exploits the old concepts of modularity and encapsulation, bringing the concepts underlying object-oriented technology a step further. The final two points shift software development from programming software (one line at a time) to composing software (by assembling components).

The Rational Unified Process supports component-based development in several ways.

- The iterative approach allows developers to progressively identify components and decide which ones to develop, which ones to reuse, and which ones to buy.
- The focus on software architecture allows you to articulate the structure. The architecture enumerates the components and the ways they integrate, as well as the fundamental mechanisms and patterns by which they interact.
- Concepts such as packages, subsystems, and layers are used during analysis and design to organize components and specify interfaces.
- Testing is organized around single components first and then is gradually expanded to include larger sets of integrated components.

4. Visually Model Software

Models are simplifications of reality; they help us to understand and shape both a problem and its solution, and to comprehend large, complex systems that we could not otherwise understand as a whole. A large part of the Rational Unified Process is about developing and maintaining models of the system under development.

The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. It gives you a standard means of writing the system's blueprints, covering conceptual items such as business processes and system functions, as well as concrete items such as classes written in a specific programming language, database schemas, and reusable software components. While it provides the vocabulary to express various models, the UML does not tell you how to develop software. That is why Rational developed the Rational Unified Process, a guide to the effective use of the UML for modeling. It describes the models you need, why you need them, and how to construct them. RUP2000 uses UML version 1.4.

5. Continuously Verify Quality

Often people ask why there is no worker in charge of quality in the Rational Unified Process. The answer is that quality is not added to a product by a few people. Instead, quality is the responsibility of every member of the development organization. In software development, our concern about quality is focused on two areas: product quality and process quality.

- **Product quality** -- The quality of the principal product being produced (the software or system) and all the elements it comprises (for example, components, subsystems, architecture, and so on).
- **Process quality** -- The degree to which an acceptable process (including measurements and criteria for quality) was implemented and adhered to during the manufacturing of the product.

  Additionally, process quality is concerned with the quality of the artifacts (such as iteration plans, test plans, use-case realizations, design model, and so on) produced in support of the principal product.

6. Control Changes to Software

Particularly in an iterative development, many work products are often modified. By allowing flexibility in the planning and execution of the development and by allowing the requirements to evolve, iterative development emphasizes the vital issues of keeping track of changes and ensuring that everything and everyone is in sync. Focused closely on the needs of the development organization, change management is a systematic approach to managing changes in requirements, design, and implementation. It also covers the important activities of keeping track of defects, misunderstandings, and project commitments as well as associating these activities with specific artifacts and releases. Change management is tied to configuration management and measurements.

Who Is Using the Rational Unified Process?

More than a thousand companies were using the Rational Unified Process at the end of 2000. They use it in various application domains, for both large and small projects. This shows the versatility and wide applicability of the Rational Unified Process. Here are examples of the various industry sectors around the world that use it:

- Telecommunications
- Transportation, aerospace, defense
- Manufacturing
- Financial services
- Systems integrators

More than 50% of these users are either using the Rational Unified Process for e-business or planning to do so in the near future. This is a sign of change in our industry: as the time-to-market pressure increases, as well as the demand for quality, companies are looking at learning from others' experience, and are ready to adopt proven best practices. The way these organizations use the Rational Unified Process also varies greatly: some use it very formally; they have evolved their own company process from the Rational Unified Process, which they follow with great care. Other organizations have a more informal usage, taking the Rational Unified Process as a repository of advice, templates, and guidance that they use as they go along -- as a sort of "electronic coach" on software engineering. By working with these customers, observing how they use the RUP, listening to their feedback, looking at the additions they make to the process to address specific concerns, the RUP development team at Rational continues to refine the process for the benefit of all.

To Learn More

- *Rational Unified Process 2000*, Rational Software, Cupertino, CA (2000) http://www.rational.com/rup/
- Philippe Kruchten, *The Rational Unified Process -- An Introduction*, 2nd ed., Addison-Wesley-Longman, Reading, MA (2000).
- Grady Booch *et al.*, UML Users' Guide, Addison-Wesley-Longman, Reading, MA (2000)
- Ivar Jacobson et al., *The Unified Software Development Process*, Addison-Wesley-Longman, Reading, MA (1999).

For more information on the products or services discussed in this article, please click here and follow the instructions provided.

Thank you!