ISSN 0975 3303
MJS, Vol.1, No. 2, Dec.-2002 - April 2003 pp. 69-81
https://doi.org/10.12725/mjs.2.7

Rational Unified Process Part - II

Best Practices for Software Development Team

*Manoj Tharian**

Business Modeling

One of the major problems with most business engineering efforts, is that the software engineering and the business engineering community do not communicate properly with each other. This leads to that the output from business engineering is not used properly as input to the software development effort, and viceversa. The Rational Unified Process addresses this by providing a common language and process for both communities, as well as showing how to create and maintain direct traceability between business and software models.

In Business Modeling we document business processes using so called business use cases. This assures a common understanding among all stakeholders of what business process needs to be supported in the organization. The business use cases are analyzed to understand how the business should support the 10 business processes. This is documented in a business object-model.
Many projects may choose not to do business modeling.

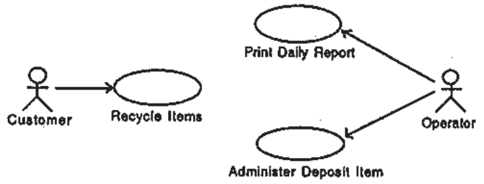
Requirements

The goal of the Requirements workflow is to describe what the system should do and allows the developers and the customer to agree on that description. To achieve

* *Director Micro Genesis Tech Soft, 20, 1 st Cross, Vasanthnagar, Bangalore- 560 052, INDIA, email: manoj.tharian@mgenindia.com*

this, we elicit, organize, and document required functionality and constraints; track and document tradeoffs and decisions.

A Vision document is created, and stakeholder needs are elicited. Actors are identified, representing the users, and any other system that may interact with the system being developed. Use cases are identified, representing the behavior of the system. Because use cases are developed according to the actor's needs, the system is more likely to be relevant to the users. The following figure shows an example of a use-case model for a recycling-machine system.



An example of a use-case model with actors and use cases.

Each use case is described in detail. The *use-case description* shows how the system interacts step by step with the actors and what the system does. Non-functional requirements are described in *Supplementary Specifications*.

The use cases function as a unifying thread throughout the system's development cycle. The same use-case model is used during requirements capture, analysis & design, and test.

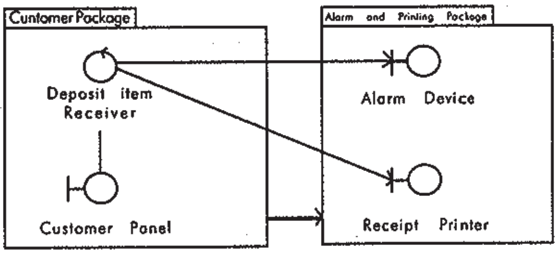
Analysis and Design

The goal of the Analysis and Design workflow is to show *how* the system will be *realized* in the implementation phase. You want to build a system that:

- Performs—in a specific implementation environment—the tasks and functions specified in the usecase descriptions.
- Fulfills all its requirements.
- Is structured to be robust (easy to change if and when its functional requirements change).

Analysis and Design results in a *design model* and optionally an *analysis model*. The design model serves as an abstraction of the source code; that is, the design model acts as a 'blueprint' of how the source code is structured and written.

The design model consists of design classes structured into design packages and design subsystems with well-defined interfaces, representing what will become components in the implementation. It also contains descriptions of how objects of these design classes collaborate to perform use cases. The next figure shows part of a sample design model for the recycling-machine system in the use-case model shown in the previous figure.



Part of a design model with communicating design classes, and package group design classes.

The design activities are centered around the notion of architecture. The production and validation of this architecture is the main focus of early design iterations. Architecture is represented by a number of architectural views [9]. These views capture the major structural design decisions. In essence, architectural views are abstractions or simplifications of the entire design, in which important characteristics are made more visible by leaving details aside. The architecture is an important vehicle not only for developing a good design model, but also for increasing the quality of any model built during system development.

Implementation

The purpose of implementation are:

- To define the organization of the code, in terms of implementation subsystems organized in layers.
- To implement classes and objects in terms of components (source files, binaries, executables, and others).
- To test the developed components as units.
- To integrate the results produced by individual implementers (or teams), into an executable system.

The system is realized through implementation of components. The Rational Unified Process describes how you reuse existing components, or implement new components with well defined responsibility, making the system easier to maintain, and increasing the possibilities to reuse.

Components are structured into Implementation Subsystems. Subsystems take the form of directories, with additional structural or management information. For example, a subsystem can be created as a directory or a folder in a file system, or a subsystem in Rational/Apex for C++ or Ada, or packages using Java.™

Test

- The purposes of testing are:
- To verify the interaction between objects.
- To verify the proper integration of all components of the software.
- To verify that all requirements have been correctly implemented.
- To identify and ensure defects are addressed prior to the deployment of the software.

The Rational Unified Process proposes an iterative approach, which means that you test throughout the project. This allows you to find defects as early as possible, which radically reduces the cost of fixing the defect. Test are carried out along three quality dimensions reliability, functionality, application performance and system performance. For each of these quality dimensions, the process describes how you go through the test lifecycle of planning, design, implementation, execution and evaluation.

Strategies for when and how to automate test are described. Test automation is especially important using an iterative approach, to allow regression testing at then end of each iteration, as well as for each new version of the product.

Deployment

The purpose of the deployment workflow is to successfully produce product releases, and deliver the software to its end users. It covers a wide range of activities including:

- Producing external releases of the software.

- Packaging the software.
- Distributing the software.
- Installing the software.
- Providing help and assistance to users.

In many cases, this also includes activities such as:

- Planning and conduct of beta tests.
- Migration of existing software or data.
- Formal acceptance.

Although deployment activities are mostly centered around the transition phase, many of the activities need to be included in earlier phases to prepare for deployment at the end of the construction phase.

The Deployment and Environment workflows of the Rational Unified Process contain less detail than other workflows.

Project Management

Software Project Management is the art of balancing competing objectives, managing risk, and overcoming constraints to deliver, successfully, a product which meets the needs of both customers (the payers of bills) and the users. The fact that so few projects are unarguably successful is comment enough on the difficulty of the task.

This workflow focuses mainly on the specific aspect of an iterative development process. Our goal with this section is to make the task easier by providing:

- A framework for managing software-intensive projects.
- Practical guidelines for planning, staffing, executing, and monitoring projects.
- A framework for managing risk.

It is not a recipe for success, but it presents an approach to managing the project that will markedly improve the odds of delivering successful software. [14]

Configuration and Change Management

In this workflow we describe how to control the numerous artifacts produced by the many people who work on a common project. Control helps avoid costly confusion, and ensures that resultant artifacts are not in conflict due to some of the following kinds of problems:

- Simultaneous Update ¾ When two or more workers work separately on the same artifact, the last one to make changes destroys the work of the former.
- Limited Notification ¾ When a problem is fixed in artifacts shared by several developers, and some of them are not notified of the change.
- Multiple Versions ¾ Most large programs are developed in evolutionary releases. One release could be in customer use, while another is in test, and the third is still in development. If problemsare found in any one of the versions, fixes need to be propagated between them. Confusion can arise leading to costly fixes and re-work unless changes are carefully controlled and monitored.

This workflow provides guidelines for managing multiple variants of evolving software systems, tracking which versions are used in given software builds, performing builds of individual programs or entire releases according to user-defined version specifications, and enforcing site-specific development policies.

We describe how you can manage parallel development, development done at multiple sites, and how to automate the build process. This is especially important in an iterative process where you may want to be able to do builds as often as daily, something that would become impossible without powerful automation. We also describe how you can keep an audit trail on why, when and by whom any artifact was changed.

This workflow also covers change request management, i.e. how to report defects, manage them through their lifecycle, and how to use defect data to track progress and trends.

Environment

The purpose of the environment workflow is to provide the software development organization with the software development environment—both processes and tools—that are needed to support the development team.

This workflow focuses on the activities to configure the process in the context of a project. It also focus on activities to develop the guidelines needed to support a project. A step-by-step procedure is provided describing how you implement a process in an organization.

The environment workflow also contains a Development Kit providing you with the guidelines, templates and tools necessary to customize the process. The Development Kit is described in more detail in the section " Development Kit for Process Customization" found later in this paper.

Certain aspects of the Environment workflow are not covered in the process such as selecting, acquiring, and making the tools work, and maintaining the development environment.

Rational Unified Process - The Product

The Rational Unified Process product consists of:

- A web-enabled searchable knowledge base providing all team members with guidelines, templates, and tool mentors for all critical development activities. The knowledge base can further be broken down to:
- Extensive guidelines for all team members, and all portions of the software lifecycle. Guidance is provided for both the high-level thought process, as well as for the more tedious day-to-day activities. The guidance is published in HTML form for easy platform-independent access on your desktop.
- Tool mentors providing hands-on guidance for tools covering the full lifecycle. The tool mentors are published in HTML form for easy platform-independent access on your desktop. See section "Integration with Tools" for more details.
- Rational Rose® examples and templates providing guidance for how tostructure the information in Rational Rose when following the Rational Unified Process (Rational Rose is Rational's tool for visual modeling)
- SoDA® templates ¾ more than 10 SoDA templates that helps automate software documentation (SoDA is Rational's Document Automation Tool)
- Microsoft® Word templates ¾ more than 30 Word templates assisting documentation in all workflows and all portions of the lifecycle

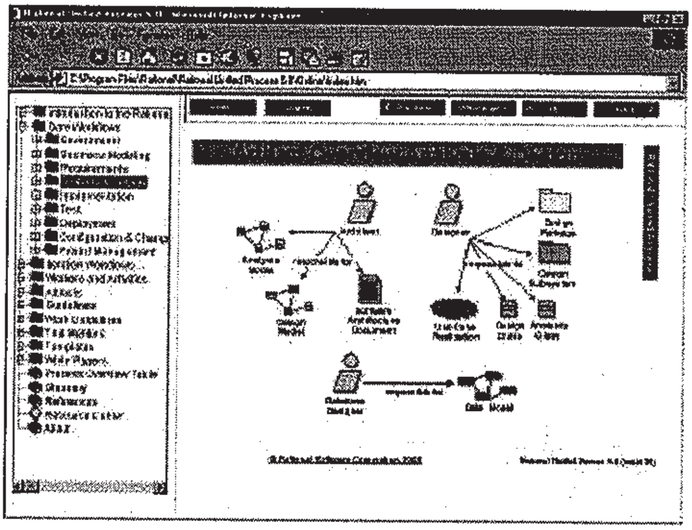
- Microsoft Project Plans ¾ Many managers find it difficult to create project plans that reflects an iterative development approach. Our templates jump start the creation of project plans for iterative development, according to the Rational Unified Process.
- Development Kit ¾ describes how to customize and extend the Rational Unified Process to the specific needs of the adopting organization or project, as well as provides tools and templates to assist the effort. This development kit is described in more detail later in this section.
- Access to Resource Center containing the latest white papers, updates, hints, and techniques, as well as references to add-on products and services.
- A book "Rational Unified Process — An Introduction", by Philippe Kruchten, published by Addison-Wesley. The book is on 277 pages and provides a good introduction and overview to the process and the knowledgebase.

Navigating the Knowledge Base

The Rational Unified Process knowledge allows you to access the content with any of the popular web browsers, such as Microsoft™ Internet Explorer and Netscape Navigator.

With the Rational Unified Process, you're never more than a few mouse clicks away from the information you want. The knowledge base contains a lot of hypertext links, and overviews of the various process elements are presented through interactive images, making it easy to find relevant information in an intuitive fashion. The powerful search engine, the index, and the "explorer looking" tree browser make it easy to use the process. Navigational buttons allow you to move to the next or previous page as if reading a book.

Information is presented in many different views, allowing you to look at information relevant to your role, to a specific activity, or to a workflow. Guided tours for easy learning of the process are provided for key project roles.



Interactive images and navigational buttons make it easy to find the specific information you are looking for.

Development Kit for Process Customization

The Rational Unified Process is general and complete enough to be used "as is" by some software development organizations. However in many circumstances, this software engineering process will need to be modified, adjusted, and tailored to accommodate the specific characteristics, constraints, and history of the adopting organization. In particular a process should not be followed blindly, generating useless work, producing artifacts that are of little added value. It must be made as lean as possible and still be able to fulfill its mission to produce rapidly and predictably high quality software.

The process contains a Development Kit, which contains guidelines for how you can customize the process to fit the specific needs of the adopting organization or project. Templates are also included for process authoring, as well as tools for generation or manipulation of search engine, index, site map, tree browser, etc. The Development Kit enables the customizing organization to maintain the look and feel of the Rational Unified Process.

The more the process is customized, the more difficult will it be to move over customizations to future releases of the process. The Development Kit describes strategies, tools and techniques to minimize the work associated with moving customizations to future releases.

Integration with Tools

A software-engineering process requires tools to support all activities in a system's lifecycle, especially to support the development, maintenance and bookkeeping of various artifacts—models in particular. An iterative development process puts special requirements on the tool set you use, such as better integration among tools and round-trip engineering between models and code. You also need tools to keep track of changes, to support requirements traceability, to automate documentation, as well as tools to automate tests to facilitate regression test. The Rational Unified Process can be used with a variety of tools, either from Rational or other vendors. However, Rational provides many well-integrated tools that efficiently support the Rational Unified Process.

Below you find a list of some of Rational's tools that support the Rational Unified Process.

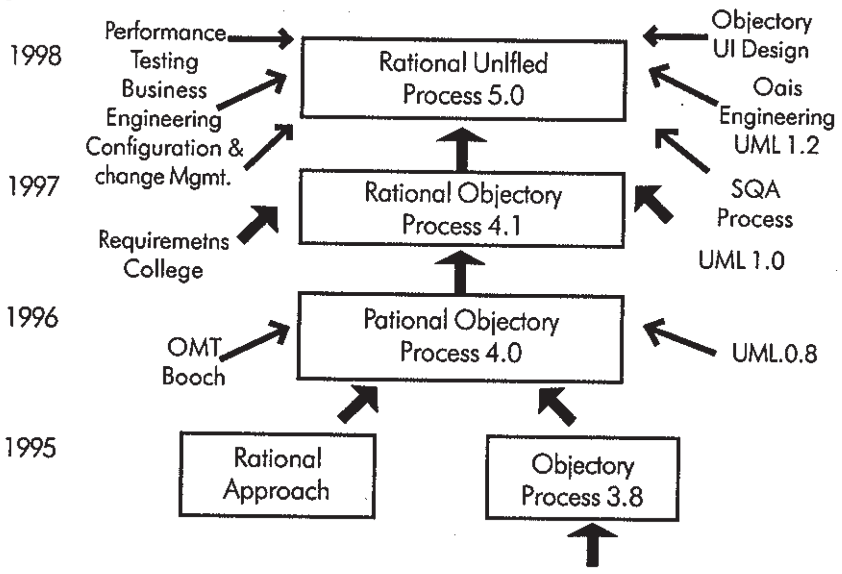
The Rational Unified Process contains Tool Mentors for almost all of these products. A Tool Mentor is a step-by-step guide describing in detail how to operate a tool, (i.e. what menus to launch, what information to enter into dialog boxes, and how to navigate a tool) to carry out an activity within the process. The Tool Mentors allow us to link the tool-independent process to the actual manipulation of the tools in your daily work.

- Rational Requisite®Pro ¾ Keeps the entire development team updated and on track throughout the application development process by making requirements easy to write, communicate and change.
- Rational ClearQuest™ — A Windows and Web-based change-request management product that enables project teams to track and manage all change activities that occur throughout the development lifecycle.
- Rational Rose® 98 — The world's leading visual modeling tool for business process modeling, requirements analysis, and component architecture design.
- Rational SoDA® ¾ Automates the production of documentation for the entire software development process, dramatically reducing documentation time and costs.
- Rational Purify® ¾ A run-time error checking tool for application and component software developers programming in C/C++; helps detect memory errors.

- Rational Visual Quantify™ — An advanced performance profiling tool for application and component software developers programming in C++, Visual Basic, and Java; helps eliminate performance bottlenecks.
- Rational Visual PureCoverage™ — Automatically pinpoints areas of code not exercised in testing so developers can thoroughly, efficiently and effectively test their applications.
- Rational TeamTest — Creates, maintains and executes automated functional tests, allowing you to thoroughly test your code and determine if your software meets requirements and performs asexpected.
- Rational PerformanceStudio™ — An easy-to-use, accurate and scalable tool that measures and predicts the performance of client/server and Web systems.
- Rational ClearCase® — Market-leading software configuration management tool, giving project managers the power to track the evolution of every software development project.

A brief history of the Rational Unified Process

The Rational Unified Process has matured over many years and reflects the collective experience of the many people and companies that make up today Rational Software's rich heritage. Let us have a quick look at the process's ancestry, as illustrated in the figure below.



Genealogy of the Rational Unified Process

Going backwards in time, the Rational Unified Process is the direct successor to the Rational Objectory Process (version 4). The Rational Unified Process incorporates more material in the areas of data engineering, business modeling, project management, and configuration management, the latter as a result of the merger with Pure-Atria. It also brings a tighter integration to the Rational Software suite of tools.

The Rational Objectory Process was the result of the integration of the "Rational Approach" and the Objectory process (version 3), after the merger of Rational Software Corporation and Objectory AB in 1995. From its Objectory ancestry, the process has inherited its process structure and the central concept of use case. From its Rational background, it gained the current formulation of iterative development and architecture. This version also incorporated material on requirements management from Requisite, Inc. and a detailed test process inherited from SQA,® Inc., companies which also merged with Rational Software. Finally, this process was the first one to use the newly created Unified Modeling Language (UML 0.8).

The Objectory process was created in Sweden in 1987 by Ivar Jacobson as the result of his experience with Ericsson. This process became a product at his company, Objectory AB. Centered around the concept of use case and an object-oriented design method, it rapidly gained recognition in the software industry and has been adopted and integrated by many companies worldwide. A simplified version of the Objectory process was published as a text book in 1992.

The Rational Unified Process is a specific and detailed instance of a more generic process described by Ivar Jacobson, Grady Booch, and James Rumbaugh in the textbook, The Unified Software Development Process.

References

1. Barry W. Boehm, A Spiral Model of Software Development and Enhancement, Computer, May 1988, IEEE, pp.61-72

2. Barry W. Boehm, Anchoring the Software Process, IEEE Software, 13, 4, July 1996, pp. 73-82.

3. Grady Booch, Object Solutions, Addison-Wesley, 1995.

4. Grady Booch, Ivar Jacobson, and James Rumbaugh, Unified Modeling Language 1.3, White paper, Rational Software Corp., 1998.

5. Alan W. Brown (ed.), Component-Based Software Engineering, IEEE Computer Society, Los Alamitos, CA, 1996, pp.140.

6. Michael T. Devlin, and Walker E. Royce, Improving Software Economics in the Aerospace and Defense Industry, Technical paper TP-46, Santa Clara, CA, Rational Software Corp., 1995

7. Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard, Object-Oriented Software Engineering—A Use Case Driven Approach, Wokingham, England, Addison-Wesley, 1992, 582p.

8. Ivar Jacobson, M. Griss, and P. Jonsson, Software Reuse—Architecture, Process and Organization for Business Success, Harlow, England, AWL, 1997.

9. Philippe Kruchten, The 4+1 View Model of Architecture, IEEE Software, 12 (6), November 1995, IEEE, pp.42-50.

10. Philippe Kruchten, A Rational Development Process, CrossTalk, 9 (7), STSC, Hill AFB, UT, pp.11-16.

11. Ivar Jacobson, Grady Booch, and Jim Rumbaugh, Unified Software Development Process, Addison- Wesley, 1999.

12. Grady Booch, Jim Rumbaugh, and Ivar Jacobson, Unified Modeling Language—User's Guide, Addison-Wesley, 1999.

13. Philippe Kruchten, Rational Unified Process—An Introduction, Addison-Wesley, 1999.

14. Walker Royce, Software Project Management—A Unified Framework, Addison-Wesley, 1998.