Chapter IX

Agile RUP:
Taming the Rational Unified Process®

Gary K. Evans
Evanetics, USA

Abstract

The Rational Unified Process® (RUP) is the de facto iterative software development process in use today. But it is huge (over 3,200 files), prescriptive, and generic rather than concise, agile, and specific. Organizations moving to RUP are often confused about how to apply it to their culture, unsure how much of it to adopt, and wary of how they can adapt it to their specific software projects. This chapter starts with a brief summary of the traditional waterfall development process, then offers an overview of RUP, its philosophy and distinctive features. Then the general philosophy of agile development is discussed. The body of the chapter defines a small set of activities that have been successfully applied in commercial “Agile” RUP projects by the author. The chapter then discusses how some of the major stakeholder groups on a software project are affected by moving to an agile process.
Introduction

The Rational Unified Process® (RUP) and the Unified Modeling Language (UML) are present in some way on many object-oriented projects today. UML is a visual notation for concisely expressing object-oriented concepts. RUP is a process description that uses UML to express the content of its own artifacts and tasks. RUP is not so much a specific process, as it is a description of a process framework. It is large and complex (over 3,500 files in more than 200 folders) because it is a generic framework. But despite its girth, RUP incorporates just a few very basic principles and advocates the best practices of [RUP]:

- Develop Iteratively
- Manage Requirements
- Use Component Architectures
- Model Visually
- Continuously Verify Quality
- Manage Change

These practices are predicated on the principles that an effective process should be:

- **Iterative:** Do the same activities in small pieces, over and over.
- **Incremental:** Gain a bit more understanding of the problem, and add a bit more solution, at each iteration, building on what was done previously.
- **Risk-Focused:** Address risk early and often, focusing on the most architecturally significant properties of the system, and the highest risk areas before developing the easy, “low hanging fruit” of the system.
- **Controlled:** Control the process to meet your needs, do not allow the process to blindly control you.
- **Use Case (i.e., requirements) Driven:** The goal is established by the total requirements, and the operational requirements are captured in a form known as use cases.
- **Architecture-Centric:** Architectural integrity and stability are emphasized over ad hoc software design details.