Skinnier RUP

By Scott W. Ambler, May 01, 2004

In the Rational Unified Process, there are four management phases: inception, elaboration, construction and transition. Each of these phases is iterative. However, from the developer's standpoint, "there's no difference between the phases," as Robert Martin observes in *UML for Java Programmers* (Prentice Hall, 2003). "Each is simply composed of iterations, all of roughly the same structure. Each is a matter of identifying stories, estimating them, selecting them for implementation and then implementing them." There you have it: An important figure in the agile community tweaking RUP to fit the practices of Extreme Programming.

Why would you want to do such a thing? Perhaps you're in an organization that wants to use RUP on all projects, but understands that some teams need an agile process, whereas others require something more prescriptive. Other organizations want to take a prescriptive software process and streamline it—for example, improving RUP's project management discipline with Scrum concepts. Read on for my ideas on "agilizing" RUP.

The Right Mindset

When it comes to making RUP agile, the critical issues are philosophical: Encompassing agility isn't about adopting the right tool or technique, it's about the mindset. For example, if you're pair programming, that doesn't mean that you're agile; it implies only that you're pair programming. Similarly, adopting JUnit or VBUnit to implement your unit tests suggests only that you've got a good unit-testing tool. The essence of agility isn't a technique—it's a willingness to focus on people, the development of working software, the pursuit of active stakeholder participation and the embrace of change.

Collaboration is the foremost hallmark of agility. How do you achieve this within RUP? First, build a team of people who want to work together, learn from each other and succeed together. Seems obvious, but how many projects have you been on that included people simply because they were available? Second, support everyone's learning efforts with training, mentoring and coaching. Third, break down communication barriers: Colocate the team and get your project stakeholders actively involved in development. If this isn't possible, make sure you compensate with video conferencing, shared repositories and the like. Fourth, recognize common "process smells" that indicate serious communication problems. For example, the belief that you need detailed documentation is one such smell; as Alistair Cockburn points out in his book *Agile Software Development* (Addison-Wesley, 2002), documentation is your worst communication option, whereas a conversation in front of a whiteboard is your best.

Another process smell is the need to hold artifact reviews —which attempt to compensate for a team that wasn't working together as closely as it should. For example, the Design the Database activity of the RUP's Analysis and Design discipline suggests that you review the database design model once per iteration in the Elaboration and Construction phases. But consider the alternative: If your

DBAs are working side-by-side with your programmers and other DBAs as needed, if they have the requisite database skills, if they follow agreed-upon standards, if they take a test-first approach, and if they adopt collective ownership, there's little opportunity for the database design to get out of hand. High-communication environments significantly reduce your need for documentation and reviews, thereby reducing cost and time to delivery.

Another key aspect is the periodic delivery of working software—XP recommends an interval of every one to two weeks. RUP supports this concept through the internal deployment of an interim working system at the end of each construction iteration. You can improve on this by actively questioning anything that you do that isn't directly related to creating software. For example, why invest time writing use cases in full English prose when point/bullet form will do? Why use a prototyping tool to design a screen when a quick whiteboard sketch is sufficient? Why write a project status report when the person you're reporting to could attend your team's daily stand-up meetings?

A good rule of thumb is the shorter the iteration, the less opportunity you have for useless bureaucracy—iterations of one or two weeks don't leave time to waste entire days in meetings. In my experience, many RUP teams flounder when they tolerate iterations of more than four weeks in length. The longer the iteration, the easier it is to put off critical activities such as writing and testing code. Instead, you can pretend to develop software by writing plans and documentation describing how you think you'll develop software.

To further increase your effectiveness, tailor the principles and practices of Agile Modeling into RUP's Business Modeling, Requirements, Analysis and Design, and Implementation disciplines. This will help you trim your modeling and documentation efforts, enabling you to invest more effort in the meat of the matter: the creation of working software. For example, a software architecture document could be a 100-page, comprehensive description of how you intend to build your system, created by one or two bright architects and then presented to the programmers to implement. Or it could be a simple whiteboard sketch, created during a short modeling session by the development team, providing just enough guidance for the team to proceed with development. I'd leave this diagram on a publicly visible (at least to the team) whiteboard that the team would update as the project progresses. If the diagram proves valuable over time, consider transcribing it into a software drawing tool (better yet, simply take a digital snapshot of the whiteboard). Back it up with a few point-form notes describing critical design decisions. This agile approach won't produce a huge document, but it will produce practical data.

You can take a similar approach to virtually every single management or model-oriented artifact in RUP—use simple tools to capture the critical information that your team requires. I've seen effective project timelines sketched on whiteboards or created using paper, string and thumb tacks: You don't need to use Microsoft Project to create complex Gantt charts when simpler artifacts will do. I've seen the requirements for a huge system captured on index cards and a system successfully built from that model, and I've seen similar project teams insist on writing and reviewing a huge requirements document.

Talking Teams

Ensuring that you have active stakeholder participation is critical to making RUP agile; as the Agile Manifesto states, we value customer collaboration over contract negotiation. With simple tools such as paper and whiteboards, and techniques such as user stories, essential user interface prototypes and Class Responsibility Collaborator (CRC) cards, your stakeholders can become active participants in the development process. You can teach people how to apply these tools and techniques very quickly—something that you usually can't achieve with software tools or UML diagrams. Agilists will explicitly include business stakeholders in the Analysis and Design discipline, and not just the Business Modeling and Requirements disciplines as RUP currently does, to make it clear that stakeholders are valuable team members.

Contract negotiation is another field that's ripe for agilizing. In subcontracting situations, it's clear that a contract is involved, but in other situations, it's not so obvious. For example, a requirements document that's reviewed and accepted by your stakeholders is effectively

a contract that you've negotiated—a contract that defines what the stakeholders expect to receive (the requirements). Similarly, reviewed and accepted design models are effectively a contract that defines what your team will deliver. To make RUP more agile, focus on finding ways to empower your stakeholders; when you do this, their trust in you will increase and they'll place less emphasis on review and acceptance procedures. You empower your stakeholders when you make it clear that they, and not IT professionals, are responsible for defining and prioritizing requirements; when you adopt inclusive modeling techniques and then actually include them; and when you make them equal partners in the decision-making process.

Become Change-Friendly

The next strategy for agilizing RUP is to accept that it's natural for your stakeholders' priorities and comprehension of their requirements to evolve. Manage your schedule openly and honestly. If you can deliver only 50 points' worth of functionality in an iteration and your stakeholders want to add 10 points of high-priority functionality, 10 points of lower-priority functionality must be pushed to a future iteration. If your organization struggles with this concept, treat your stakeholders like adults and explain to them why they can't have everything today. A wonderful side effect of regularly delivering working software—that is, at the end of each iteration—is that stakeholders quickly realize that they will eventually get the functionality that they want. With this approach, you probably won't need a change-control board as suggested in RUP: If the stakeholders can prioritize their requirements and then let the developers work on them in that order, you've got all the change control you need. This assumes, of course, that you and your business stakeholders are empowered to do so.

Rethinking Things

The term Agile RUP isn't an oxymoron—you can streamline RUP if you want to. That said, RUP is far better suited as a base from which to tailor a heavyweight process because it provides a wealth of detailed advice. If you really want an agile process, you may be better served with Extreme Programming, Feature-Driven Development or Dynamic System Development Method; it's much easier to determine that something's missing and then add it to a lightweight process than it is to identify procedures (or portions thereof) of a heavyweight process that you don't need. But if your organization isn't ready to plunge into agility, merge RUP with agile techniques. You'll get a process that's effective and responsive to your stakeholders' needs—and isn't that what you really want?

Recommended Resources
These five articles give more specifics on agile RUP.

Palm-Sized Process by Gary Evans, *Software Development*, Sept. 2001. This article describes a real-world case study of how RUP was streamlined to be more agile through the adoption of four-week iterations, by adopting Agile Modeling practices that enabled them to model just enough to develop software, and by commandeering a large "war room" with ample whiteboard space.

Agile Modeling and the Unified Process by Scott Ambler. This essay describes how to incorporate the principles and practices of Agile Modeling into RUP.

Xtreme RUP: Lightening Up the Rational Unified Process by Allan Shalloway. This presentation overviews many of the issues you'll face when trying to agilize RUP. He suggests simplifying project management by adopting XP's planning game (users define and prioritize requirements, developers estimate them), short iterations, keeping your use cases simple, embracing change, and including stakeholders through development.

RUP vs. XP by Robert Martin. This article compares and contrasts RUP and XP, and then describes dX—an agile tailoring of RUP (also the letters XP upside down). This paper formed the basis for Chapter 7, "dX: The Practices," of Martin's latest book, *UML for Java Programmers*.

Making RUP Agile by Michael Hirsch. This paper describes the author's experiences with applying RUP on two small projects as well as advice for making RUP more agile. He suggests that you maintain only the artifacts that you really need and to scale down those that you do create. He also suggests that you build a team of experienced people to greatly simplify planning, adopt short (four-week) iterations, provide rapid feedback to stakeholders, and to simplify status reporting by holding meetings instead of writing reports.

Scott Ambler *is a senior consultant with Ronin International Inc. His latest book is* Agile Database Techniques *from Wiley Publishing.*

Related Reading

- News
- Commentary

- **Java Plumbr Unlocks Threads**
- **Xamarin Editions of IP*Works! & Integrator**
- **Jelastic Docker Integration For Orchestrated Delivery**
- **Mac OS Installer Platform From installCore**
  More News»

- Slideshow
- Video

- **Jolt Awards 2014: The Best Programmer Libraries**
- **NoSQL Options Compared**
- **Developer Reading List: The Must-Have Books for JavaScript**
- **Jolt Awards: Coding Tools**
  More Slideshows»

- Most Popular

- **iOS Data Storage: Core Data vs. SQLite**
- **The C++14 Standard: What You Need to Know**
- **Logging In C++**
- **Building GUI Applications in PowerShell**
  More Popular»

2014
Dr. Dobb's Journal
November - **Mobile Development**
August - **Web Development**
May - **Testing**
February - **Languages**

Dr. Dobb's Tech Digest
DevOps
Open Source
Windows and .NET programming
The Design of Messaging Middleware and 10 Tips from Tech Writers
Parallel Array Operations in Java 8 and Android on x86: Java Native Interface and the Android Native Development Kit

2013
January - **Mobile Development**
February - **Parallel Programming**
March - **Windows Programming**
April - **Programming Languages**
May - **Web Development**
June - **Database Development**
July - **Testing**
August - **Debugging and Defect Management**
September - **Version Control**
October - **DevOps**
November- **Really Big Data**
December - **Design**

2012
January - **C & C++**
February - **Parallel Programming**
March - **Microsoft Technologies**
April - **Mobile Development**
May - **Database Programming**
June - **Web Development**
July - **Security**
August - **ALM & Development Tools**
September - **Cloud & Web Development**
October - **JVM Languages**
November - **Testing**
December - **DevOps**

2011