

# Resource Reviews

## PROJECT MANAGEMENT

### **Project Management with the IBM Rational Unified Process**

Dennis Gibbs. 2007.

IBM Press (<http://www.redbooks.ibm.com/ibmpress>).  
312 pages.

ISBN-10: 0321336399

CSQE Body of Knowledge  
area: Project Management

*Reviewed by Robin F. Goldsmith  
robin@gopromanagement.com*

This book provides project management guidance, but not much of it is specific to the IBM Rational Unified Process (RUP). Also, many of the portions that do address RUP are broadly applicable to project management regardless of whether RUP is involved.

Drawing on his background in consulting, Gibbs mainly addresses management of software development projects performed by a contractor pursuant to a contractual outsourcing arrangement. In fact, I'd guess that some recent personal experience caused Gibbs to devote much of the "Introduction to Outsourcing" first chapter to a discourse on the need for consultants working on site to have adequate facilities.

Before returning to outsourcing and generic software development project topic chapters, Gibbs provides an essentially vanilla overview recounting of RUP. Here and at several subsequent points he falls into the common stereotype of assuming that if one is not using RUP, then he or she must be following a totally inflexible, never-looking-back

waterfall that goes on for eons, blindly generating documents before ever producing workable code, which of course turns out not to be right and must be redone when it finally does materialize.

One of the most common criticisms of RUP is that it's excessively burdensome in terms of procedures and paperwork. Gibbs doesn't address this issue but does briefly describe agile development and allows that RUP *can be* but may not be agile. The methods share a number of similarities, most notably emphasizing iterations that produce working deliverable software. Yet ironically, some "agilistas" are the most vocal critics with an almost fanatical obsession against RUP.

Outsourcing is definitely a project management topic, but it is not specifically the focus of RUP. Gibbs attributes outsourcing difficulties to use of the waterfall life cycle and says essentially that using RUP and selecting a contractor proficient in RUP will lead to outsourcing success. While these contentions are highly questionable, Gibbs' suggestion that RUP's iterations with executable deliverables can serve as control points for outsourced work does make sense.

It indeed is true that many outsourcing contracts are doomed from the start because the contractor is asked to commit to a full-project estimate with only a negligible definition of the deliverables and work to be done. To address this, Gibbs advocates splitting procurement into two procurements, which is relatively common. Typically, in such a format, one contract is responsible for defining requirements and design, which then are the basis for a separate implementation contract.

Gibbs suggests instead making one contract for the RUP inception and elaboration phases and a second contract for RUP's construction and transition phases. While seemingly similar to the traditional split, several issues emerge because RUP anticipates some construction during the elaboration phase. Such early coding either commits the construction contractor to work products it may not want to be bound by or it generates extra elaboration work that will be discarded.

It's not until Chapter 7 that the book finally explicitly gets into describing the RUP phases, starting with the RUP inception phase, which is the strongest chapter in the book. Gibbs says insightfully on p. 123, "Most projects fail during the inception phase, but the participants fail to recognize it." Inception involves identifying key requirements, a candidate architecture, and significant risks along with mitigation plans. Based on these, the project vision and plans for software development and iterations are developed, which in turn become the basis for cost and schedule estimates that all parties agree upon.

While some projects may involve similarly planned phasing or other subproject strategies rather than iterations, the conception information described previously is essential to successfully managing any project regardless of methodology. I'd further suggest that this upfront planning, especially the planning of iterations, is the part of development that most often is shortchanged, usually through lack of awareness, although agile development takes pride in consciously and intentionally skipping it.

---

The elaboration phase emphasizes requirements definition, which in the RUP world means use cases. Unlike many RUP authors, Gibbs recognizes that business and system use cases differ. But, like most authors, he focuses only on the system use cases that often are called user requirements but actually are usage requirements of the system as designed.

Project managers probably will find the chapter on construction iterations most wanting. While it does include checklists to assess readiness for construction, only general help is offered with respect to the really big challenge of defining what the iterations will be and what each will cover. And, the book fails to address how to accurately estimate each iteration's needed resources, effort, and duration.

Unfortunately, I fear most project managers share, and therefore wouldn't recognize, Gibbs' several critical misunderstandings about testing. For instance, on p. 17, he says testing "is one of the few software development disciplines that can be sent entirely offshore." Such low opinions of testing are predictable when one realizes that RUP indeed includes as a key activity in each iteration the execution of tests of new functionality plus regression tests of previous iterations' functionality, but RUP doesn't include test planning, design, or management anywhere.

Gibbs shows little awareness of the importance of the developer's doing suitable unit testing to find his or her own errors when they are easiest and cheapest to fix, which of course is one of the strengths of agile development. He implies that automation of test execution suffices, largely because it relieves what he considers tedium. Furthermore, Gibbs thinks a demonstration of the system for the users constitutes suitable user acceptance testing (UAT), whereas effective UAT involves independently and

rigorously planned execution by the users in a realistic environment.

Discussion of the transition phase on the one hand perpetuates the poor practice of writing documentation after the programs are working, rather than writing it as part of design and then building a system that functions as the user instructions say it will. On the other hand, again clearly drawing on his extensive personal experience, Gibbs raises key issues that ordinarily are overlooked, such as "destaffing" with minimal disruption as the project winds down.

**Robin F. Goldsmith** is president of Go Pro Management, Inc., consultancy in Needham, Mass. For 25 years he has lived by his project management results. He is the author of the Proactive Testing™ methodology and the recent Artech House book, *Discovering REAL Business Requirements for Software Project Success*.

## SYSTEMS AND SOFTWARE ENGINEERING PROCESSES

### Distributed Systems Architecture: A Middleware Approach

Arno Puder, Kay Römer,  
and Frank Pilhofer.  
2006. Elsevier  
(<http://www.elsevier.com>).  
323 pages.  
ISBN-10: 1-55860-648-3

CSQE Body of Knowledge  
area: Systems and Software  
Engineering Processes

Reviewed by Ray Schneider  
[rschneid@bridgewater.edu](mailto:rschneid@bridgewater.edu)

When reviewing a complex topic there is the question of whether it is better to have a reviewer who is an expert on the topic or a reviewer who is less of an expert and, thus,

closer to the reader who is learning the topic. The expert is likely to note all the book's failures to capture the fine points but may overlook just how effective the book is at communicating the basic elements.

I've been interested in multiprocessing and asynchronous systems since even before encountering the Ada rendezvous mechanism in the 1980s. I discovered *Distributed Systems Architecture* to be a fascinating book that uses common object request broker architecture (CORBA) to illustrate the role middleware plays in implementing distributed systems. I didn't know much about CORBA when I started the book, so I'm in the excellent position of being able to tell readers how much I learned, and it was a lot. The authors have done an excellent job of making a complex, multilayered, and intricate topic accessible.

The authors credit Andrew S. Tanenbaum's textbook on operating systems, which used as its centerpiece a small, illustrative operating system called Minix (see <http://www.minix3.org/>), as their inspiration. Tanenbaum built his textbook around the implementation of his small operating system. The authors have built their book on distributed systems around a small implementation of the CORBA specification they call "Mico." Originally, Mico stood for mini CORBA, but it expanded into a complete CORBA implementation, so they reinvented the acronym to "Mico Is CORBA," in honor of GNU's "Gnu is Not Unix."

Divided cleanly into 10 chapters, *Distributed Systems Architecture* builds smoothly outward from a quick five-page introduction that establishes middleware as a tool firmly planted between the system and the application, a black box to the application programmers and a white box to the system programmers. That's how the authors summarize it before launching off into basic concepts in Chapter 2.

# Reviews

---

Even if one knows nothing about distributed systems in general and middleware in particular, *Distributed Systems Architecture* provides a nice introduction. I particularly like the mix of illustrative diagrams and wide margins with italicized topic signals in the margins that explain where a particular idea can be found in the running narrative. That kind of help is good on the first read and absolutely invaluable when going back through the book looking for something.

Once the basic ideas of distributed systems, the object model, and the centrality of middleware are established, readers are ready for the grand adventure of a specific example—CORBA. Chapters 3 through 9 take readers through CORBA in such a nicely structured fashion that one hardly notices things are getting more and more complicated and involved. Each chapter builds on the one before. Code is judiciously inserted to illustrate the topics, but rarely more than about 20 lines, so it's very accessible. Linkage to the larger code body is retained by including line numbers in the excerpts so readers can always go back to the appendices and see the code in context. Code is presented in both C++ and Java.

Chapter 4 eases readers into CORBA implementation using  $\mu$ ORB, a miniature implementation of a subset of CORBA. This allows important basic ideas to be illustrated in preparation for a more comprehensive treatment in later chapters. Object request broker (ORB) design follows in Chapter 5. It's the functional building block of what follows—interoperability. One discovers a lot of alphabet soup along the way, including protocols such as General Inter-ORB Protocol (GIOP) and Internet Inter-ORB Protocol (IIOP). When finished with Chapter 6, readers are ready to adapt a few objects, and that's where Chapter 7 on portable object adapters (POA) comes in. Three more chapters follow. Chapter 8 on invocation adapters, which let clients invoke methods, and then Chapter 9 takes the cover off the interface definition language (IDL) compiler, which helps develop distributed applications and separate interfaces from implementations. Chapter 10 closes out the CORBA discussion with a brief treatment of the CORBA component model (CCM) and goes on to briefly discuss some other middleware technologies.

Readers are not off the hook yet. Now that they have been exposed to the basics, the authors have some homework for them to do. Appendix A guides readers through a MICO installation in either Unix or Windows and three other appendices provide an overview, implementation details, and a sample MICO application.

This may not be the only book one will need to learn about distributed systems and CORBA, but it's a great choice for those who are getting started. It is suitable for both managers and implementers. Managers will get a clear sense of the complexity of distributed systems and implementers will get a sense for both the big picture, as well as a clear and realistic introduction into an extremely important area of modern network-aware system design. I learned a lot.

**Ray Schneider** holds a bachelor's degree in physics, a master's degree in engineering science, and a doctorate in information technology. He is a licensed professional engineer in the state of Virginia. He has more than 35 years of product development and applied research and development experience working for government, defense industry, and small business. He is currently an assistant professor in the Mathematics and Computer Science Department of Bridgewater College in Bridgewater, Va.