软件工程实践丛书

统一软件开发过程（影印版）

（美） Ivar Jacobson，Grady Booch
James Rumbaugh 著

PEARSON
Addison Wesley

THE UNIFIED SOFTWARE DEVELOPMENT PROCESS

IVAR JACOBSON
GRADY BOOCH
JAMES RUMBAUGH

UML

The complete guide to the Unified Process from the original designers

OBJECT TECHNOLOGY SERIES
BOOCH
JACOBSON
RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY

Pearson Education

清华大学出版社

软件工程实践丛书

统一软件开发过程

（影印版）

（美） Ivar Jacobson
Grady Booch
James Rumbaugh 著

清华大学出版社
北京

图书在版编目(CIP)数据

统一软件开发过程=The Unified Software Development Process / (美) 雅各布逊 (Jacobson, I), (美) 布奇 (Booch, G), (美) 朗鲍 (Rumbaugh, J.) 著. —影印版. —北京：清华大学出版社，2005.1
(软件工程实践丛书)
ISBN 7-302-09917-0
Ⅰ. 统… Ⅱ. ①雅…②布…③朗… Ⅲ. 软件开发—英文 Ⅳ. TP311.52

中国版本图书馆 CIP 数据核字（2004）第 119651 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969
文稿编辑：文开棋
封面设计：陈刘源
印 刷 者：北京市世界知识印刷厂
装 订 者：三河市李旗庄少明装订厂
发 行 者：新华书店总店北京发行所
开　　本：148×210　**印张**：15.5　**字数**：395 千字
版　　次：2005 年 1 月第 1 版　2005 年 1 月第1 次印刷
书　　号：ISBN 7-302-09917-0/TP・6820
印　　数：1～3000
定　　价：29.00 元

Preface

There is a belief held by some that professional enterprises should be organized around the skills of highly trained individuals. They know the work to be done and just do it! They hardly need guidance in policy and procedure from the organization for which they work.

This belief is mistaken in most cases, and badly mistaken in the case of software development. Certainly, software developers are highly trained, but the profession is still young. Consequently, developers need organizational guidance, which, in this book, we refer to as the "software development process." Moreover, because the process we set forth in this book represents the bringing together of previously separate methodologies, we feel justified in calling it the "Unified Process." It not only unifies the work of the three authors but incorporates the numerous contributions of other individuals and companies that contributed to the UML, as well as a significant number of key contributors at Rational Software Corporation. It draws significantly from the on-the-spot experience of hundreds of user organizations working with early versions of the process at customer sites.

A symphony orchestra conductor, for example, does little more during a performance than tell the players when to start and help them stay together. He or she can do so little because the conductor has guided the orchestra during rehearsals and preparation of the score, and because each musician is highly skilled on his own

instrument and actually plays it independently of the other orchestra members. More importantly for our purpose, each musician follows a "process" laid out long ago by the composer. It is the musical score that provides the bulk of the "policy and procedure" that guides the performance. In contrast, software developers do not play independently. They interact with each other and the users. They have no score to follow—until they have a *process*.

The need for process promises to become more critical, particularly in companies or organizations in which the software systems are "mission-critical," such as financial, air traffic control, defense, and telecommunications systems. By this we mean that the successful conduct of the business or execution of the public mission depends upon the software that supports it. These software systems are becoming more complex, their time to market needs to shrink, and their development, in turn, is becoming more difficult. For reasons such as these, the software industry needs a process to guide developers, just as an orchestra needs a composer's score to guide a performance.

What Is a Software Development Process?

A process defines *who* is doing *what when* and *how* to reach a certain goal. In software engineering the goal is to build a software product or to enhance an existing one. An effective process provides guidelines for the efficient development of quality software. It captures and presents the best practices that the current state of the art permits. In consequence, it reduces risk and increases predictability. The overall effect is to promote a common vision and culture.

We need such a process to serve as a guide for all the participants—customers, users, developers, and executive managers. Any old process will not do; we need one that will be the *best* process the industry is capable of putting together at this point in its history. Finally, we need a process that will be widely available so that all the stakeholders can understand its role in the development under consideration.

A software development process should also be capable of evolving over many years. During this evolution it should limit its reach at any given point in time to the realities that technologies, tools, people, and organizational patterns permit.

- *Technologies.* Process must be built on technologies—programming languages, operating systems, computer systems, network capabilities, development environments, and so on—that are usable at the time the process is to be used. For example, twenty years ago visual modeling was not really mainstream. It was too expensive. At that time, a process builder almost had to assume that hand-drawn diagrams would be used. That assumption greatly limited the degree to which a process originator could build modeling into the process.

- *Tools.* Process and tools must develop in parallel. Tools are integral to process. To put it another way, a widely used process can support the investment that creates the tools that support it.

- *People.* A process builder must limit the skill set needed to operate the process to the skills that current developers possess or target ones that developers can be quickly trained to use. In many areas it is now possible to embed techniques that once required extensive skill, such as checking model drawings for consistency, in computer-based tools.

- *Organizational patterns.* While software developers may not be as independently expert as symphony musicians, they are far from the automaton workers on whom Frederick W. Taylor based "scientific management" one hundred years ago. The process builder has to adapt the process to today's realities—the facts of virtual organization; working at a distance through high-speed lines; the mix of partial owners (in small start-ups), salaried employees, contract workers, and outsourcing subcontractors; and the continuing shortage of software developers.

Process engineers need to balance these four sets of circumstances. Moreover, the balance must exist not just now but into the future. The process builder must design the process so it can evolve, just as a software developer tries to develop a system that not only works this year but evolves successfully for years to come. A process needs to mature for several years before it achieves the level of stability and maturity that will enable it to stand up to the rigors of commercial product development while holding the risk of its use to a reasonable level. Developing a new product is risky enough by itself without adding to it the risk of a process insufficiently tested by actual experience. Under these circumstances, a process can be stable. Without this balance of technologies, tools, people, and organization, using the process would be quite risky.

Goals of the Book

This book presents the software process that was constantly on our minds when we developed the Unified Modeling Language. While UML gives us a standard way to visualize, specify, construct, document, and communicate the artifacts of a software-intensive system, we of course recognize that such a language must be used within the context of an end-to-end software process. UML is a means, not an end. The ultimate end is a robust, resilient, scaleable software application. It takes both a process and a language to get there, and illustrating the process portion is the goal of this book. While we provide a brief appendix on the UML, it is not intended to be comprehensive or detailed. For a detailed UML tutorial refer to *The Unified Modeling Language User Guide* [11]. For a comprehensive UML reference refer to *The Unified Modeling Language Reference Manual* [12].

Audience

The Unified Software Development Process can be used by anyone involved in the development of software. It is primarily addressed to members of the development

team who deal with the life-cycle activities requirements, analysis, design, implementation, and testing—that is, in work that results in UML models. Thus, for instance, this book is relevant to analysts and end users (who specify the required structure and behavior of a system), application developers (who design systems that satisfy those requirements), programmers (who turn those designs into executable code), testers (who verify and validate the system's structure and behavior), component developers (who create and catalogue components), and project and product managers.

This book assumes a basic grasp of object-oriented concepts. Experience in software development and in an object-oriented programming language is also helpful, but not required.

Approach of the Book

We give the most space in this book to those activities—requirements, analysis, and design—on which UML places primary emphasis. It is in these fields of emphasis that the process develops the *architecture* of complex software systems. We do, however, treat the entire process, although in less detail. Still, it is the executable program that finally runs. To get there, a project depends on the efforts of every member of its team as well as the support of the stakeholders. As you will see, the process rests on a tremendous variety of activities. Many artifacts must be produced and kept track of. All the activities must be managed.

A complete account of a comprehensive, full life-cycle process is beyond the scope of any one book. Such a book would have to cover design guidelines, templates for artifacts, quality indicators, project management, configuration management, metrics, and more—much more! With the development of on-line access, that "more" is now available, and it can be updated as new developments dictate. For this, we refer you to the Rational Unified Process, a new Web-enabled software product that guides software development teams to more effective software development practices. (See http://www.rational.com for more information.) In covering the full software life cycle, the Rational Unified Process extends the Unified Process beyond areas described in this book and provides additional workflows that are not covered in this book or are mentioned only in passing, such as business modeling, project management, and configuration management.

History of the Unified Process

The Unified Process is balanced because it is the end product of three decades of development and practical use. Its development as a product follows a path (see Figure P.1) from the Objectory Process (first released in 1987) via the Rational Objectory Process (released in 1997) to the Rational Unified Process (released in 1998). Its development has been influenced by many sources. We don't intend to try to identify them all (we actually don't know what they all are) and leave that for software archeologists to research. However, we will describe the impact of the Ericsson and Rational approaches to the product as well as several other sources.

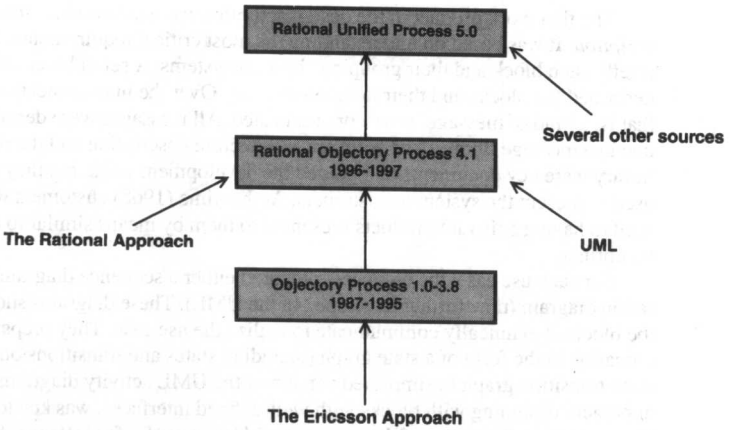

Figure P.1 The development of the Unified Process. (Versions of the product are denoted in gray rectangles.)

The Ericsson Approach

The Unified Process has deep roots. To employ the language of Peter F. Drucker, it is a "knowledge-based innovation." "There is a protracted span between the emergence of new knowledge and its distillation into usable technology," he advises us. "Then there is another long period before this new technology appears in the marketplace in products, processes, or services." [1]

One reason for this long lead-time is that knowledge-based innovation is grounded on the bringing together of many kinds of knowledge, and this takes time. Another reason is that the people who have to make the new idea effective need time to digest it and spread it to others.

As a first step toward illuminating the development of the Unified Process, let us go back to 1967 and outline more specifically what Ericsson achieved [14], [15], [16]. Ericsson modeled the whole system as a set of interconnected blocks (in UML these are known as "subsystems" and implemented as "components"). They assembled the lowest-level blocks into higher-level subsystems to make the system more manageable. They found the blocks by working through the previously specified traffic cases—now called "use cases." For each use case, they identified the blocks that cooperate to realize it. Knowing the responsibilities of each block, they prepared a specification for it. Their design activities resulted in a set of static block diagrams with their interfaces, and groupings into subsystems. These block diagrams correspond directly to a simplified version of UML class or package diagrams—simplified in that it only showed associations used for communications.

The first work product in the design activities was a software *architecture description.* It was based on understanding the most critical requirements. It described briefly each block and their groupings into subsystems. A set of block diagrams described the blocks and their interconnections. Over the interconnections, signals, that is, a kind of message, were communicated. All messages were described one by one in a message library. The software architecture description and the message library were key documents that guided the development work, but they were also used to present the system to customers. At that time (1968) customers were not used to having software products presented to them by means similar to engineering blueprints.

For each use case, the engineers prepared either a sequence diagram or a collaboration diagram (now further developed in the UML). These diagrams showed how the blocks dynamically communicate to realize the use case. They prepared a specification in the form of a state graph (including states and transitions only) and a state-transition graph (a simplified version of the UML activity diagrams). This approach, designing with blocks with well-defined interfaces, was key to the success. Now a new configuration of the system could be created—for instance, for a new customer—by exchanging a block with another one that provided the same interfaces.

Now, the blocks were not just subsystems and source-code components; they were compiled into executable blocks, they were installed in the target machine one by one, and it was established that they worked with all the rest of the executable blocks. Beyond that, they must be able to install each new or changed executable block in the field in a running system while it was executing calls for telephone systems operating 100% of the time. One does not shut down such a system just to make changes. It is like changing tires while a car is moving 60 miles an hour.

In essence, the approach used was what we today call *component-based development.* Ivar Jacobson was the originator of this development method. He guided its evolution into a software development process over the many years of the pre-Objectory period.

The Specification and Description Language

A significant development during this period was the issuance in 1976 by CCITT, the international body for standardization in the telecommunications field, of the *Specification and Description Language* (SDL) for the functional behavior of telecommunications systems. This standard, significantly influenced by the Ericsson approach, specified a system as a set of interconnected blocks that communicated with each other solely through messages (called "signals" in the standard). Each block owned a set of "processes," which was the SDL term for active classes. A process had instances much as classes do in object-oriented terms. Process instances interacted by messages. It recommended diagrams that were specializations of what UML now calls class diagrams, activity diagrams, collaboration diagrams, and sequence diagrams.

Thus, SDL was a specialized object-modeling standard. Periodically updated, it is still in use by more than 10,000 developers and supported by several tool vendors.

Developed originally more than 20 years ago, it was far ahead of its time. However, it was developed at a time when object modeling had not matured. SDL will likely be supplanted by the Unified Modeling Language, which was standardized in November 1997.

Objectory

In 1987 Ivar Jacobson left Ericsson and established Objectory AB in Stockholm. During the next eight years he and his associates developed a process product called Objectory ("Objectory" is an abbreviation of "Object Factory."). They extended it to industries outside of telecommunications and to countries beyond Sweden.

Although the concept of the *use case* had been present in the work at Ericsson, it now had a name (which was introduced at the OOPSLA conference in 1987), a diagramming technique was developed, and the idea was extended to embrace a variety of applications. That it is use cases that drive development became more clear. That it is architecture that guides the developers and informs the stakeholders came to the fore.

The successive workflows were represented in a series of models: requirements-use cases, analysis, design, implementation, and test. A model is a perspective of a system. The relationships between the models in this series were important to developers as a way of following a feature from one end of the model series to the other end. In fact, traceability became a prerequisite of use-case-driven development. Developers could trace a use case through the model sequence to the source code or, when problems arose, back again.

Development of the Objectory process proceeded in a series of releases, from Objectory 1.0 in 1988 to the first on-line version, Objectory 3.8, in 1995 (an overview of Objectory is presented in [2]).

It is important to note that the Objectory product itself came to be viewed as a system. This way of describing the process—as a system product—provided a better way to develop a new version of Objectory from an earlier one. This way of designing Objectory made it easier to tailor it to meet the particular needs of different development organizations. The fact that the Objectory software development process itself was engineered was a unique feature.

The experience in developing Objectory also provided insights into how to engineer the processes on which a business in general operates. The same principles were applicable and were incorporated in a 1995 book [3].

The Rational Approach

Rational Software Corporation acquired Objectory AB in the fall of 1995 and the task of unifying the basic principles underlying the existing software development processes gained new urgency. Rational had developed a number of software development practices, many of which were complementary to those embodied in Objectory.

For example, "In 1981, Rational set out to produce an interactive environment that would improve productivity for the development of large software systems,"

James E. Archer Jr. and Michael T. Devlin recalled in 1986 [4]. In this effort, object-oriented design, abstraction, information hiding, reusability, and prototyping were important, they went on to say.

Scores of books, papers, and internal documents detail Rational developments since 1981, but perhaps the two most important contributions to *process* were the emphases on architecture and iterative development. For instance, in 1990 Mike Devlin wrote a vision paper on an architecture-driven iterative development process. Philippe Kruchten, in charge of the Architecture Practice within Rational, authored papers on iteration and architecture.

We cite one, an article on an architectural representation in four views: the logical view; the process view; the physical view; and the development view, plus an additional view that illustrates the first four views with uses cases or scenarios [6]. The value of having a set of views, rather than trying to cram everything into one type of diagram, grew out of Kruchten's experience on several large projects. Multiple views enabled both stakeholders and developers to find what they needed for their diverse purposes in the appropriate view.

Some have perceived iterative development as somewhat chaotic or anarchic. The four-phase approach (inception, elaboration, construction, and transition) was devised to better structure and control progress while iterating. The phases impose order on the iterations. The detailed planning of the phases and ordering of the iterations within phases was a team effort with Walker Royce and Rich Reitman, as well as the continuing participation of Grady Booch and Philippe Kruchten.

Booch was on the scene from the very beginning of Rational, and in 1996 in one of his books he cited two "first principles" that bear upon architecture and iteration:

- "An architecture-driven style of development is usually the best approach for the creation of most complex software-intensive projects."
- "A successful object-oriented project must apply an incremental and iterative process." [7]

Rational Objectory Process: 1995–1997

At the time of the merger, Objectory 3.8 had shown how a software development process as a product could be developed and modeled. It had designed the original architecture of a software development process. It had identified a set of models that recorded the outcome of the process. In areas such as use-case modeling, analysis, and design, it was well developed. In other areas—requirements management other than use cases, implementation, and test—it was less well developed. Moreover, it contained little on project management, configuration management, deployment, and the preparation of the development environment (tools and process procurement).

Now Rational's experience and practices were added to form the Rational Objectory Process 4.1. The phases and the *controlled* iterative approach, in particular, were added. Architecture was made explicit in the form of an architecture description—the "bible" of the software development organization. A precise definition of architecture

was developed. It treated architecture as being the significant parts of the organization of the system. It depicted the architecture as architectural views of the models. Iterative development was advanced from a relatively general concept to a risk-driven approach that put architecture first.

At this time UML was in development and was used as the modeling language of the Rational Objectory Process (ROP). The authors of this book contributed as the original developers of UML. The process development team, headed by Philippe Kruchten, alleviated some of the weaknesses in ROP by strengthening project management, for instance, based on contributions from Royce [8].

Unified Modeling Language

The need for a uniform and consistent visual language in which to express the results of the rather numerous object-oriented methodologies extant in the early 1990s had been evident for some time.

During this period Grady Booch, for example, was the author of the Booch method [9]. James Rumbaugh was the principal developer at the General Electric Research and Development Center of OMT (Object Modeling Technique) [10] When he joined Rational in October 1994, the two began an effort, in concert with many of Rational's customers, to unify their methods. They released version 0.8 of the Unified Method in October 1995, about the same time that Ivar Jacobson joined Rational.

The three, working together, released Unified Modeling Language 0.9. The effort was expanded to include other methodologists and a variety of companies, including IBM, HP, and Microsoft, each of which contributed to the evolving standard. In November 1997 after going through the standardization process, the Unified Modeling Language version 1.1, was promulgated as a standard by the Object Management Group. For detailed information refer to the *User Guide* [11] and the *Reference Manual* [12].

UML was used for all models in the Rational Objectory Process.

Rational Unified Process

During this period Rational acquired or merged with other software tool companies. Each brought to the mix expertise in process areas that further expanded the Rational Objectory process:

- Requisite Inc. brought experience in requirements management.
- SQA Inc. had developed a test process to go with its test product, adding to Rational's long experience in this field.
- Pure-Atria added its experience in configuration management to that of Rational.
- Performance Awareness added performance testing and load testing
- Vigortech added expertise in data engineering.

The process was also expanded with a new workflow for business modeling, based on [3], that is used to derive requirements from the business processes the software was to serve. It also was extended to design user interfaces driven by use cases (based on work done at Objectory AB).

By mid-1998 the Rational Objectory Process had become a full-fledged process able to support the entire software development life cycle. In so doing, it unified a wide variety of contributions, not only by the three present authors but by the many sources from which Rational and the UML were able to draw upon. In June, Rational released a new version of the product, the Rational Unified Process 5.0 [13]. Many elements of this proprietary process now become available to the general public for this first time in the form of this book.

The name change reflects the fact that unification had taken place in many dimensions: unification of development approaches, using the Unified Modeling Language, and unification of the work of many methodologists—not just at Rational but also at the hundreds of customer sites that had been using the process over many years.

Acknowledgments

A project of this magnitude is the work of many people, and we would like to acknowledge as many as is practical by name.

For Contributions to This Book

Birgitte Lønvig prepared the Interbank system example and worked it through all models. This is the primary example used throughout the book.

Patrik Jonsson extracted material from the Rational Objectory Process documentation and arranged it in the order of the proposed chapters. He also assisted in the preparation of the examples. In doing so, he contributed many ideas on how best to present the Unified Process.

Ware Myers participated in the development of this book from the initial outlines forward. He took first drafts prepared by the lead author and turned them into more readable English prose.

Of the reviewers we particularly thank Kurt Bittner, Cris Kobryn, and Earl Ecklund, Jr. In addition, we are most appreciative of the reviews by Walker Royce, Philippe Kruchten, Dean Leffingwell, Martin Griss, Maria Ericsson, and Bruce Katz. The reviewers also include Pete McBreen, Glenn Jones, Johan Galle, N. Venu Gopal, David Rine, Mary Loomis, Marie Lenzi, Janet Gardner, and some anonymous reviewers, all of whom we want to thank.

Terry Quatrani of Rational improved the English in Chapters 1-5. Karen Tongish copyedited the whole book. We thank both of them.

In particular, we want to thank Stefan Bylund who reviewed the drafts extensively and suggested improvements in detail, many of which we incorporated. His contributions have substantially increased the quality of the book.

Over The Years

We also want to thank several people who have over the years helped us to "get the process right" and supported the work in its various forms. Specifically, we would like to thank the following individuals: Stefan Ahlquist, Ali Ali, Gunilla Andersson, Kjell S. Andersson, Sten-Erik Bergner, Dave Bernstein, Kurt Bittner, Per Bjork, Hans Brandtberg, Mark Broms, Stefan Bylund, Ann Carlbrand, Ingemar Carlsson, Margaret Chan, Magnus Christerson, Geoff Clemm, Catherine Connor, Hakan Dahl, Stephane Desjardins, Mike Devlin, Hakan Dyrhage, Susanne Dyrhage, Staffan Ehnebom, Christian Ehrenborg, Maria Ericsson, Gunnar M. Eriksson, Iain Gavin, Carlo Goti, Sam Guckenheimer, Bjorn Gullbrand, Sunny Gupta, Marten Gustafsson, Bjorn Gustafsson, Lars Hallmarken, David Hanslip, Per Hedfors, Barbara Hedlund, Jorgen Hellberg, Joachim Herzog, Kelli Houston, Agneta Jacobson, Sten Jacobson, Paer Jansson, Hakan Jansson, Christer Johansson, Ingemar Johnsson, Patrik Jonsson, Dan Jonsson, Bruce Katz, Kurt Katzeff, Kevin Kelly, Anthony Kesterton, Per Kilgren, Rudi Koster, Per Kroll, Ron Krubeck, Mikael Larsson, Bud Lawson, Dean Leffingwell, Rolf Leidhammar, Hakan Lidstrom, Lars Lindroos, Fredrik Lindstrom, Chris Littlejohns, Andrew Lyons, Jas Madhur, Bruce Malasky, Chris McClenaghan, Christian Meck, Sue Mickel, Jorma Mobrin, Christer Nilsson, Rune Nilsson, Anders Nordin, Jan-Erik Nordin, Roger Oberg, Benny Odenteg, Erik Ornulf, Gunnar Overgaard, Karin Palmkvist, Fabio Peruzzi, Janne Pettersson, Gary Pollice, Tonya Prince, Leslee Probasco, Terry Quatrani, Anders Rockstrom, Walker Royce, Goran Schefte, Jeff Schuster, John Smith, John Smith, Kjell Sorme, Ian Spence, Birgitta Spiridon, Fredrik Stromberg, Goran Sundelof, Per Sundquist, Per-Olof Thysselius, Mike Tudball, Karin Villers, Ctirad Vrana, Stefan Wallin, Roland Wester, Lars Wetterborg, Brian White, Lars Wiktorin, Charlotte Wranne, and Jan Wunsche..

In addition, the following people have given the lead author personal support over the years, for which he is highly appreciative: Dines Bjorner, Tore Bingefors, Dave Bulman, Larry Constantine, Goran Hemdal, Bo Hedfors, Tom Love, Nils Lennmarker, Lars-Olof Noren, Dave Thomas, and Lars-Erik Thorelli.

Finally, We Would Like to Thank in Particular

Mike Devlin, president of Rational Software Corporation, for his belief in the Objectory process as a product to help those developing software worldwide, and for his continued support in using effective software process as a driver for the development of software tools.

And lastly, we would like to thank Philippe Kruchten, director of the Rational Unified Process, and all the members of the Rational process team for integrating the best of Objectory with Rational's best practices and the UML, while preserving the values of each. In addition, we could not have reached this objective without Philippe's personal commitment and perseverance to the task of building, quite simply, the best software process the world has ever seen.

Process Breaks Through

Through this book and its related books, on-line versions, and tools, software development *process* comes of age. The Unified Process drew its inspiration from many sources. Already it is being widely used. It provides a common medium of process understanding from which management, developers, and stakeholders can draw.

Still, much work remains to be done. Developers have to learn unified ways of working. Stakeholders and management have to support them. For many software organizations, the breakthrough is only potential. You can make it actual.

Ivar Jacobson
Palo Alto, California
December 1998
ivar@rational.com

References

[1] Peter F. Drucker, "The Discipline of Innovation," *Harvard Business Review*, May-June, 1985; reprinted Nov.-Dec. 1998, pp. 149–157.

[2] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard, *Object-Oriented Software Engineering: A Use-Case Driven Approach*, Reading, MA: Addison-Wesley, 1992.

[3] Ivar Jacobson, Maria Ericsson, and Agneta Jacobson, *The Object Advantage: Business Process Reengineering with Object Technology*, Reading, MA: Addison-Wesley, 1995.

[4] James E. Archer Jr. and Michael T. Devlin, "Rational's Experience Using Ada for Very Large Systems," *Proceedings of the First International Conference on Ada Programming Language Applications for the NASA Space Station*, June, 1986.

[6] Philippe B. Kruchten, "The 4 + 1 View Model of Architecture," *IEEE Software*, November 1995, pp. 42–50.

[7] Grady Booch, *Object Solutions: Managing the Object-Oriented Project*, Reading, MA: Addison-Wesley, 1996.

[8] Walker Royce, *Software Project Management: A Unified Framework*, Reading, MA: Addison-Wesley, 1998.

[9] Grady Booch, *Object-Oriented Analysis and Design with Applications*, Redwood City, CA: Benjamin/Cummings, 1994.

[10] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object-Oriented Modeling and Design*, Englewood Cliffs, NJ: Prentice Hall, 1991.

[11] Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*, Reading, MA: Addison-Wesley, 1998.

[12] James Rumbaugh, Ivar Jacobson, and Grady Booch, *The Unified Modeling Language Reference Manual,* Reading, MA: Addison-Wesley, 1998.

[13] Philippe Kruchten, *The Rational Unified Process: An Introduction,* Reading, MA: Addison-Wesley, 1998.

[14] Ivar Jacobson, *Concepts for Modeling Large Real Time Systems,* Chapter 2, Dissertation, Department of Computer Systems, The Royal Institute of Technology, Stockholm, Sept. 1985.

[15] Ivar Jacobson, "Object-Orientation as a Competitive Advantage," *American Programmer,* Oct. 1992.

[16] Ivar Jacobson, "A Large Commercial Success Story with Objects, Succeeding with Objects," *Object Magazine,* May 1996.

Contents