TRIED & TRUE OBJECT DEVELOPMENT

MANAGING OBJECT TECHNOLOGY SERIES

1. What Every Software Manager Must Know to Succeed with Object Technology • John D. Williams

2. Managing Your Move To Object Technology: Guidelines and Strategies for a Smooth Transition • Barry McGibbon

3. The Object Primer: Application Developer's Guide to Object Orientation • Scott W. Ambler

4. Getting Results with the Object-Oriented Enterprise Model • Thornton Gale and James Eldred

5. Deploying Distributed Business Software • Ted Lewis, Ph.D.

6. The Blueprint for Business Objects • Peter Fingar

7. Object Technology Strategies and Tactics • Gilbert Singer

8. Upgrading Relational Databases with Objects • Robert Vermeulen

9. Building Object Applications That Work: Your Step-by-Step Handbook for Developing Robust Systems with Object Technology • Scott W. Ambler

10. CUC96: Component-Based Software Engineering • Collected and Introduced by Thomas Jell

11. Developing Business Objects • Edited by Andy Carmichael

12. Component-Based Development for Enterprise Systems: Applying The SELECT Perspective™ • Paul Allen and Stuart Frost

13. Developing Business Systems with CORBA • Waqar Sadiq and Fred Cummins, J.D.

14. Cognitive Patterns: Problem-Solving Frameworks for Object Technology • Karen Gardner, Alex Rush, Michael Crist, Robert Konitzer, and Bobbin Teegarden.

15. Process Patterns: Building Large-Scale Systems Using Object Technology • Scott W. Ambler

16. Tried & True Object Development: Practical Approaches with UML • Ari Jaaksi, Juha-Markus Aalto, Ari Aalto, and Kimmo Vättö

Additional Volumes in Preparation

TRIED & TRUE OBJECT DEVELOPMENT

Practical Approaches with UML

Ari Jaaksi
Juha-Markus Aalto
Ari Aalto
Kimmo Vättö

CAMBRIDGE UNIVERSITY PRESS

SIGS BOOKS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge CB2 1RP, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
http: //www.cup.cam.ac.uk
40 West 20th Streeet, New York, NY 10011-4211, USA
http: //www.cup.org
10 Stamford Road, Oakleigh, Melbourne 3166, Australia

Published in association with SIGS Books

First published in 1999

Design and composition by Susan A. Ahlquist

Cover design by Yin Moy and Tom Jezek

Printed in the United States of America

A catalog record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data is on record with the publisher.

ISBN 0-521-64530-1 paperback

TABLE OF CONTENTS

Foreword

This is a great book about using object technology. Like all the new books in this area, it uses the Unified Modeling Language (UML), which is the standard notation for expressing object-oriented designs. For me, however, *Tried & True Object Development* really stands out from the recent flood of books on UML. Why? Because, this book is about the realities of object-oriented software engineering written from a position of authority. That authority is based on the authors' many years of experience developing complex software in Nokia, one of the world's most successful hi-tech corporations. The authors know the secrets of successful object-oriented development from the inside.

However, this book is much more than an experience report, it is truly about how to engineer software. Thus, unlike many other books, it is not written from the perspective of the single engineer working on a small new application. Instead it focuses on how real-world teams, large and small, can use UML and object-oriented methodology. It covers all the phases of software development from requirements through testing. It shows how the fundamental techniques for each of these phases can be molded into an incremental evolutionary life-cycle that can deal with project risk.

Real projects have to handle many problems that lie outside the tidy theories of object-oriented methodologies. This book faces up to those issues. For

example, it covers how to make objects persistent using a relational database, and how object-oriented design interfaces with GUI design.

If you want to know how to apply the latest ideas in object technology to product development, then *Tried & True Object Development* is for you. It is a unique contribution and not just another UML methods book.

Derek Coleman
Hewlett-Packard Software Initiative
Palo Alto

About the Authors

DR. ARI JAAKSI heads the Nokia Wireless Software Solutions development unit in Boston, Massachusetts. During the preparation of this book, he was on leave from Nokia acting as a professor of computer science at the University of Tampere, Finland. Dr. Jaaksi has over ten years' experience developing commercial software systems including CAD/CAM, production management, and telecommunication systems. He is a frequent lecturer in conferences and seminars, provides training and consulting services, and is the author of several publications about object-oriented software development. Dr. Jaaksi is a member of IEEE and ACM.

JUHA-MARKUS AALTO, M.Sc. in software engineering, works as the Head of Quality and Processes of the Network Management Systems business unit at Nokia Telecommunications, Tampere, Finland. He has over ten years' of experience in software development, process improvement, and quality management. Mr. Aalto has work experience as a programmer in the domains of health care, software development tools, and telecommunications. He has also worked as the software engineering manager at Nokia Telecommunications. Mr. Aalto has given several lectures and courses, and consulted in object-oriented analysis and large-scale software development.

ARI AALTO, M.Sc., Lic. Tech. in software engineering, heads an R & D Department at Nokia Telecommunications, Tampere, Finland. Mr. Aalto has held various positions in software product and customer project development in several Finnish software companies. He worked as a programmer, project manager, technical consultant, and product manager from 1986–1993. He joined Nokia Network Management Systems R & D in 1993, and has since then been involved in the development of core functionality of the Nokia NMS product family. His area of specialization is data management, and he has consulted and given numerous seminars on object data management in the Finnish IT industry.

KIMMO VÄTTÖ, M.Sc., Ph.L in software engineering, is currently acting as a project manager in Data Warehousing product development in Düsseldorf, Germany. He has experience in building software systems for healthcare, finance, and telecommunications. He joined Nokia Network Management Systems R & D in 1993, and has since then been involved in the development of the Nokia NMS product family. He specializes in data management, an area in which he has provided training and consulting. He has given numerous lectures at conferences and seminars, and he is the author of several publications about object data management.

Preface

What This Book Is About

This book is about software development. It presents approaches and practices for the construction of software systems based on features, objects, components, and architectural patterns. This book presents the key phases of software development, as well as the artifacts produced during these phases. Above all, this book communicates our practical experiences in developing large software systems.

The approaches presented here were developed at Nokia. We authors were involved in the development of the presented practices, which are currently used within Nokia and in other organizations of different sizes.

The approaches are blended from several commercial object-oriented methods and our own process improvement, and they use the *Unified Modeling Language* (UML) to visualize their phase products and blueprints. We have tuned the processes for our own needs for the development of large interactive software products. In addition, we have scaled the practices down to be applicable for projects of a few persons.

This book concentrates on the key elements of software development—the elements that we have found most important. First, we present our industry proven process model, its phases and deliverables. To be useful in real-life software projects, the process framework must be understandable and clear. We discus how to utilize the process model in different kinds of software projects.

Second, we present our practices for the development of the user interface and the data management. These elements exist in the majority of software systems, and are often the most difficult parts; still, the majority of publications and process models seem to ignore them. We discuss what kind of practices, software tools, and architectural solutions support the development of these parts. Third, we present our practices for dealing with software architectures starting with the architecture of entire product families and going all the way down to the architecture of individual software components. For us, architecture is the most concrete and essential element of any software system. Fourth, we present how to scale our practices in both directions, up and down. We present how to organize the development of large software systems, develop closely related products as product lines, support large-scale reuse, and organize the incremental development of big products. In addition, we present a downscaled version of our process model suitable for small projects of just a few persons or for companies just starting object development.

This book does not introduce a new all-purpose software development paradigm or method. Instead, it provides practical guidelines and process models for the development of software systems. The book utilizes our experiences; how we at Nokia have used the theories of object-orientation in practice. This book concentrates on the development of industrial-sized interactive software systems. The emphasis is placed on phases, phase products, and guidelines that, based on our experiences, work in practice.

The Structure of This Book

This book has four chapters. The chapters are closely related and reveal different views of software development. However, the chapters can also be studied separately.

Chapter 1, *Developing Interactive Software Systems,* presents our method framework, its phases and phase products. It concentrates on the development of interactive systems with graphical user interfaces and distributed architectures. This chapter shows how to use the UML to visualize specifications produced during the process. In addition, it gives examples of how to arrange software development projects around the presented process model.

Chapter 2, *Object-Oriented Data Management,* presents our guidelines for the development of OO data management solutions. It provides patterns of

how to implement persistence by using different database approaches. This chapter also discusses the strategies on selecting proper data management technologies.

Chapter 3, *Large-Scale Development,* presents our way of managing large software systems. It discusses different levels of architecture and evolutionary software processes. This chapter scales the process model up to manage product families, features, and large development teams.

Chapter 4, *Small-Scale Development,* goes to another extreme: It gives an example of a very simple process model suitable for companies just starting object development.

Ari Jaaksi is the author of Chapters 1 and 4 and the supportive parts of the book. Juha-Markus Aalto is the author of Chapter 3, and Ari Aalto and Kimmo Vättö are the authors of Chapter 2. However, we have worked in close cooperation with each other to review and enhance each part.

Who Should Read This Book

We wrote this book for heterogeneous groups of people. Individual software developers can study our method of constructing object-oriented software systems all the way from requirements and product ideas to the delivered systems. They may also benefit from our architectural patterns, especially when developing data management or user interface parts of systems. Project managers can study our way of arranging software development according to different life cycle models. Product managers and system analysts can learn from our experiences on screening product concepts, collecting and analyzing requirements, and forming features, products, and product lines. Those managers responsible for organizing software development organizations, outlining product architectures, or managing product development that lasts years may find our practices for the development of large software products beneficial. Finally, those parts of software development organizations that select and develop tools and methods can hopefully learn from our experiences and suggestions.

The reader should know the basics of software engineering and object-orientation. Concepts, such as object, class, inheritance, and late binding, should be familiar to the reader. Also, the reader should know the basics of an object-oriented programming language, such as C++ or Java. In addition to these requirements, experience in real-life software projects would benefit the reader.

Acknowledgments

Thanks are due to many people in preparing this book. Writing a book was definitively more laborious and time-consuming than we ever expected. We would like to express our gratitude to our families for their endless patience and support.

We thank our numerous colleagues at Nokia Telecommunications and University of Tampere. In addition, the book has profited greatly from the comments of Prof. Ilkka Haikala from Tampere University of Technology; Dr. Akmal B. Chaudhri from The City University of London, UK, and M.Sc. Markku Ruonavaara from Nokia.

Introduction

Software systems are becoming larger and more complex. Large groups of designers with different skill levels participate in the development of these systems. On one hand, the systems are becoming more critical and the requirements for their reliability and usefulness are growing. Thus, implementation of software systems is becoming more and more complicated and time-consuming. Because of this, major failures happen: software projects overshoot their schedules, systems do not function as required, or systems are not taken into use at all.

Because of the growing complexity and demanding requirements of software systems, software development cannot just depend on the exceptional but arbitrary achievements of talented designers. Instead, software development is becoming a more disciplined work, having both a scientific and an engineering basis. According to many researchers and practitioners, object-orientation together with the latest component-based paradigms is the most promising approach for systematic software development.

The Context of This Book

This book is about how to use objects and components in the development of large and complex software systems. It presents some new ideas but also introduces a synthesis of old ideas. These ideas and their combinations are currently in everyday use in various projects inside and outside Nokia. Thus, we have proof that the development of large software systems can be based on the presented approach. We have also experience with small start-up projects to add to these experiences, so the practices scale down as well as we will show.

This book presents practices that aim at the production of high-quality software systems. The writing of the actual code is only a part of the software developer's job. This book, therefore, presents a path for a software developer to develop a software system from requirements to a ready system. In addition, this book concentrates on some elements we have found especially demanding; for example, the specification and implementation of interactive applications, data management, and particularly the development of large systems.

When talking with our colleagues at Nokia, we call our process framework OMT++. The name originates from the most widely used object-oriented method, *Object Modeling Technique* (OMT), which was the starting point of our approach. However, not much is left from the original OMT. The name OMT++ itself illustrates the origin of our approach, the fact that we have modified and improved it for our own purposes, and that we use mainly it in the context of the C++ programming language. However, because we do not want to introduce a new, one-size-fits-all method, we would rather talk about "our process" instead of "OMT++."

Our process is a collection of practices, selected notations, and phase-product templates that assist software developers in their everyday work. It also forms the backbone of software project management and quality assurance. All the software development work is planned, managed, and performed according to its phases. One might call our process model a complete method, because it covers software development from the requirements to the tested systems and provides a means to monitor and manage this development. However, it is not a complete method in the sense that it would be fully documented and supported by some training organization, for example. After all, we are not interested in developing methods. Our main goal is to produce high-quality software products. We want to share a collection of practices that we have used to reach our goal.

One rigid approach is never enough. Even a single software company needs different process models for different types of projects. Large projects developing additional features on top of an established software product need different methods than do smaller projects developing the first versions of software entering new market segments. For the first project type, the key goals are probably to keep schedules and produce faultless outcome, whereas the latter project tries to enter the market as soon as possible, even with limited functionality and lower-quality standards.

Many elements of a method can be similar for different kinds of projects. The notation used is one of those elements. In this book, we use the *Unified Modeling Language* (UML) for the notation, because it provides all that is needed to support industrial software development. Actually, we need only a small subset of UML. One of the main problems of UML is that it is already too big and complicated. This problem seems to affect almost any "unified" approach.

Another widely accepted and almost unified approach that we use is the major phase structure of software development. According to it, we call the phases object-oriented analysis, design, programming, and testing. We make a clear distinction between analysis and design. Analysis deals with the concepts relevant to the end users. It analyses the requirements and produces the initial solution descriptions. After successful analysis, both the users and the developers should have a similar vision about the future system. Design deals with the concepts relevant to the programmers. Design specifies *how* the outlined and analyzed solution will be implemented. Design aims at programming, and the programming phase produces the software modules consisting of code. The programming phase makes the most detailed design decisions. Typically, testing the most elementary units of code, such as classes and functions, is considered to be programming. The testing phase integrates the modules together, builds the final software packages, and tests it against the requirements.

Art or Industry?

This book is about software development. However, it is not the only area of human activity where certain practices and commonly agreed-upon approaches are needed. As a comparison, let us examine a recording session in a recording studio as an analogy of software development.

Let us suppose that you are about to make a recording of one single musical tune. Although more and more music nowadays is recorded with computers, you will use real musicians. At first you want to record the tracks of a rhythm section: drums, bass, rhythm guitars, and keyboards. You have already made an arrangement of the tune beforehand, based on a composition that was available only on a cassette tape. This is typical, because many people play and sing their compositions on tape rather than write them down on a piece of paper. Based on that tape, you make the preliminary arrangements. In contrast to classical music or music played by larger bands, the details of the arrangements will be decided only while you are rehearsing the tune with the band.

At the beginning of the recording session, you listen to the original tape together with other musicians. After that, you briefly describe the basic ideas of your arrangement, including what kind of atmosphere and rhythmical feeling you have in mind. Then you give the arrangement to the musicians on a piece of paper, such as the one illustrated in Figure 1, and demonstrate your ideas on a piano.

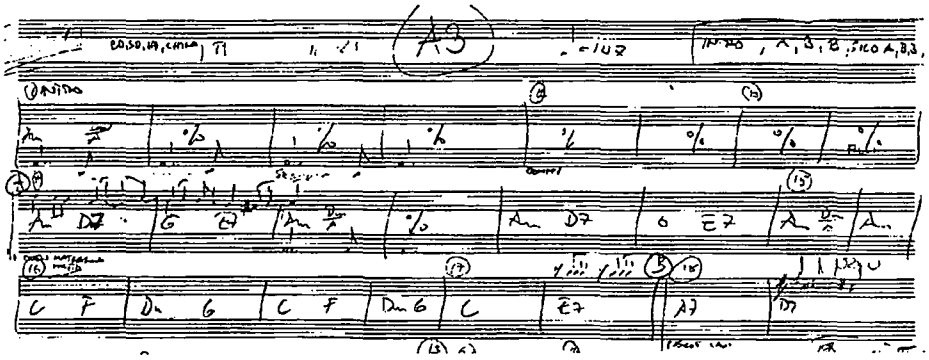

FIGURE 1. A rough arrangement, or specification, of a tune.

The arrangement has many levels of information, although many details of the tune have not been written down at all. You have illustrated the structure of a song with markings such as INTRO, A, B, Solo, Chorus, and Bridge, as depicted in the upper right-hand corner of Figure 1. These markings provide a high-level view of the tune by illustrating the basic structure of verses, choruses, and other such elements. Some might call this structure the architecture of the tune.

In addition to the structural view, there is also a more detailed view of the arrangement. Chords, such as E7 and A7, illustrated in Figure 2, are the basic

notation for the harmonics of a tune. The chords give musicians their harmonic limits. Within these limits they can choose individual notes and rhythmic details based on their musical intuition and our common agreements. This is typically all that a talented musician needs to produce music with others.

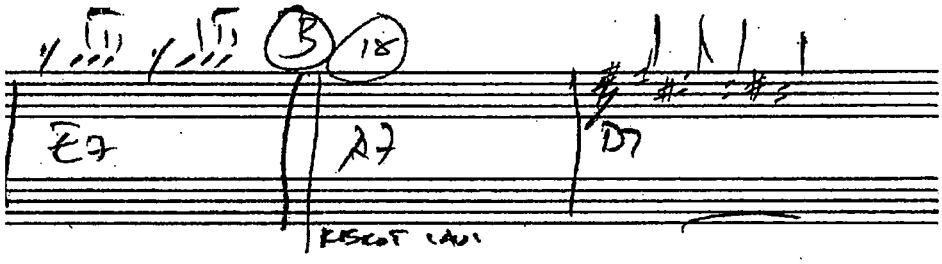

FIGURE 2. Some details of the arrangement.

In addition to the basic structure and harmonics of the tune, you have indicated some "hot spots" in the tune. It is of vital importance that every musician follows the arrangement during these hot spots. Such spots are typically needed, for example, when you move from a verse to a chorus and back. Also, certain riffs and other rhythmical patterns may need a more detailed description. These details are illustrated as notes, as depicted in the upper corners of Figure 2.

After you and other musicians have listened to the original composition and studied and discussed the arrangement, you start to practice for the recording. You play the tune several times and concentrate especially on the overall feeling, hot spots, and some tricky parts of the arrangement. After a while, you should try to play the tune on a tape and listen to it. After listening to the first take, you then typically make some minor changes and go for a final take.

After a few trials you and the other musicians manage to record the tune the way you wanted. The overall feeling is good, and there are no major errors. However, although the take is almost perfect, usually some minor things need corrections and overdubs. To make the corrections, you rewind the tape[1] and correct even a single note by rerecording it. After this, the recording of the rhythm section is done and the musicians deserve a break. After the break, you start to work on a different tune and after a couple of days, the horn section arrives and plays their parts. But that is another story.

This simple example of a recording session demonstrates the need for a method even when creating music. Much more than just an inspiration is needed, and not many people understand how much discipline and organized work is required to produce a piece of music. Without a method that includes a common working approach and notations, nobody would be able to create music with other musicians within a certain time frame. You must not forget the schedules, because recording studios can be extremely expensive to hire.

The recording pattern we have just explained is used all over the world. It has certain phases, such as making an arrangement, rehearsal for recording, doing the actual recording, and making corrections. It has also certain notations, as illustrated in Figures 1 and 2, which are known to all musicians. Without these elements, records would be hard to produce, and the larger the orchestra is and the more complicated the arrangement is, the more discipline is needed. Making a recording of a classical symphony, for example, requires much more detailed processes.

The process of software development is very similar to the process of music recording. Both tasks require the collaboration of people, must produce good quality products, and must be done within a certain time frame. In addition, adequate patterns for the construction the final products are required. For example, it is typical that the bass guitar follows the rhythmical pattern of the base drum in order to provide a nice groove. Also, if you use a horn section, you should arrange their part not to drown out the lead vocal. Similar patterns emerge in software development. You should design the data storage for efficiency and maintainability, distribute your application into executable processes without sacrificing the performance, and so forth. Such patterns increase the chances of quality outcome.

We claim that there are as much creative work involved in software development as in music. Nevertheless, both music and software development need a certain amount of discipline and rules that set the framework for the art itself. Only within frameworks that are flexible but tight enough can you produce something useful and creative. And although you need discipline, rules, and limits, no approach can render the skills and talents of a designer or musician useless. Software construction is a creative process, and sound methodology can empower and liberate the creative mind. However, a methodology cannot help individuals who lack the right skills, talents, and attitude.

[1]If you recorded directly on a hard disk, rewinding should be taken metaphorically.