Object Oriented Software Engineering Ivar Jacobson

This introductory volume to Alexander's other works, A Pattern of Language and The Oregon Experiment, explains concepts fundamental to his original approaches to the theory and application of architecture

The Practical, Start-to-Finish Guide to Planning and Leading Iterative Software Projects Iterative processes have gained widespread acceptance because they help software developers reduce risk and cost, manage change, improve productivity, and deliver more effective, timely solutions. But conventional project management techniques don't work well in iterative projects, and newer iterative management techniques have been poorly documented. Managing Iterative Software Development Projects is the solution: a relentlessly practical guide to planning, organizing, estimating, staffing, and managing any iterative project, from start to finish. Leading iterative development experts Kurt Bittner and Ian Spence introduce a proven, scalable approach that improves both agility and control at the same time, satisfying the needs of developers, managers, and the business alike. Their techniques are easy to understand, and easy to use with any iterative methodology, from Rational Unified Process to Extreme Programming to the Microsoft Solutions Framework. Whatever your role–team leader, program manager, project manager, developer, sponsor, or user representative–this book will help you Understand the key drivers of success in iterative projects Leverage "time boxing" to define project lifecycles and measure results Use Unified Process phases to facilitate controlled iterative development Master core concepts of iterative project management, including layering and evolution Create project roadmaps, including release plans Discover key patterns of risk management, estimation, organization, and iteration planning Understand what must be controlled centrally, and what you can safely delegate Transition smoothly to iterative processes Scale iterative project management from the smallest to the largest projects Align software investments with the needs of the business Whether you are interested in software development using RUP, OpenUP, or other agile processes, this book will help you reduce the anxiety and cost associated with software improvement by providing an easy, non-intrusive path toward improved results–without overwhelming you and your team.

Provides information on analyzing, designing, and writing object-oriented software.

Addresses critical software engineering issues, showing how an object - oriented approach can provide much improved solutions over other methods. Designed as a technology tool.

Growing out of a historic workshop sponsored by IBM, this book brings together contributions from many of the leading figures in the field of human-computer interaction and object-oriented software engineering. The first book-length work devoted entirely to the subject of use-oriented design representations—or scenarios—it discusses an array of scenario-

based design approaches and demonstrates their practical applications across the system development life cycle, from requirements analysis and software design, to documentation, training, and prototype evaluation. Translates the latest research findings into techniques that readers can immediately use to enhance the effectiveness of user-interface design and object-oriented software engineering design Features contributions from the top names in the field, including Rebecca Wirfs-Brock, Ivar Jacobson, Jakob Nielsen, Tom Carey, Allan MacLean, Scott Robertson, Morten Kyng, Mary Beth Rosson, and others Assesses the effectiveness of various scenario-based design approaches in dealing with a wide range of design problems and in different types and sizes of organizations Packed with case studies and enlightening illustrations

Object-Oriented Software Engineering: An Agile Unified Methodology by David Kung presents a step-by-step methodology that integrates modeling and design, UML, patterns, test-driven development, quality assurance, configuration management, and agile principles throughout the life cycle. The overall approach is casual and easy to follow, with many practical examples that show the theory at work. The author uses his experiences as well as real-world stories to help the reader understand software design principles, patterns, and other software engineering concepts. The book also provides stimulating exercises that go far beyond the type of question that can be answered by simply copying portions of the text.

This book covers the essential knowledge and skills needed by a student who is specializing in software engineering. Readers will learn principles of object orientation, software development, software modeling, software design, requirements analysis, and testing. The use of the Unified Modelling Language to develop software is taught in depth. Many concepts are illustrated using complete examples, with code written in Java.

Introducing the reuse-driven software engineering business; Architectural style; Processes; Organizing a reuse business. Presents practical advice on the disciplines, techniques, tools, and practices of computer programming and how to approach software development with a sense of pride, honor, and self-respect.

More than 300,000 developers have benefited from past editions of UML Distilled . This third edition is the best resource for quick, no-nonsense insights into understanding and using UML 2.0 and prior versions of the UML. Some readers will want to quickly get up to speed with the UML 2.0 and learn the essentials of the UML. Others will use this book as a handy, quick reference to the most common parts of the UML. The author delivers on both of these promises in a short, concise, and focused presentation. This book describes all the major UML diagram types, what they're used for, and the basic notation involved in creating and deciphering them. These diagrams include class, sequence, object, package, deployment, use case, state machine, activity, communication, composite structure, component, interaction overview,

and timing diagrams. The examples are clear and the explanations cut to the fundamental design logic. Includes a quick reference to the most useful parts of the UML notation and a useful summary of diagram types that were added to the UML 2.0. If you are like most developers, you don't have time to keep up with all the new innovations in software engineering. This new edition of Fowler's classic work gets you acquainted with some of the best thinking about efficient object-oriented software design using the UML--in a convenient format that will be essential to anyone who designs software professionally.

This book walks developers through every step of the object-oriented development process, showing how to tailor and document the development process that is ideal for their organizations.This book shows how to tailor your own object-oriented development process -- a process that delivers software more effectively and virtually documents itself. It presents new techniques for requirements gathering, performing initial object-oriented analysis, transitioning to object-oriented design from procedural environments, implementing a design, and validating the results. It includes comprehensive templates and examples for each phase of the lifecycle. It also presents a detailed case study of a complete project, with example workbook and work products.All object-oriented developers, regardless of the languages and environments they utilize.

This guide will help readers learn how to employ the significant power of use cases to their software development efforts. It provides a practical methodology, presenting key use case concepts.

Discusses how to define and organize use cases that model the user requirements of a software application. The approach focuses on identifying all the parties who will be using the system, then writing detailed use case descriptions and structuring the use case model. An ATM example runs throughout the book. The authors work at Rational Software. Annotation copyrighted by Book News, Inc., Portland, OR

Information systems in tourism, such as reservation systems, yield management systems, and tourism-marketing systems, have been among the pioneers of leading-edge technology applications and are among the most successful users of e-commerce. From both a technological and a management point of view, this book focuses on interorganizational processes and information systems, providing a coherent picture of current trends and future possibilities.

For courses in Software Engineering, Software Development, or Object-Oriented Design and Analysis at the Junior/Senior or Graduate level. This text can also be utilized in short technical courses or in short, intensive management courses. Shows students how to use both the principles of software engineering and the practices of various object-oriented tools, processes, and products. Using a step-by-step case study to illustrate the concepts and

topics in each chapter, Bruegge and Dutoit emphasize learning object-oriented software engineer through practical experience: students can apply the techniques learned in class by implementing a real-world software project. The third edition addresses new trends, in particular agile project management (Chapter 14 Project Management) and agile methodologies (Chapter 16 Methodologies).

Larman covers how to investigate requirements, create solutions and then translate designs into code, showing developers how to make practical use of the most significant recent developments. A summary of UML notation is included

Software architecture—the conceptual glue that holds every phase of a project together for its many stakeholders—is widely recognized as a critical element in modern software development. Practitioners have increasingly discovered that close attention to a software system's architecture pays valuable dividends. Without an architecture that is appropriate for the problem being solved, a project will stumble along or, most likely, fail. Even with a superb architecture, if that architecture is not well understood or well communicated the project is unlikely to succeed. Documenting Software Architectures, Second Edition, provides the most complete and current guidance, independent of language or notation, on how to capture an architecture in a commonly understandable form. Drawing on their extensive experience, the authors first help you decide what information to document, and then, with guidelines and examples (in various notations, including UML), show you how to express an architecture so that others can successfully build, use, and maintain a system from it. The book features rules for sound documentation, the goals and strategies of documentation, architectural views and styles, documentation for software interfaces and software behavior, and templates for capturing and organizing information to generate a coherent package. New and improved in this second edition: Coverage of architectural styles such as service-oriented architectures, multi-tier architectures, and data models Guidance for documentation in an Agile development environment Deeper treatment of documentation of rationale, reflecting best industrial practices Improved templates, reflecting years of use and feedback, and more documentation layout options A new, comprehensive example (available online), featuring documentation of a Web-based service-oriented system Reference guides for three important architecture documentation languages: UML, AADL, and SySML

Diagramming and process are important topics in today's software development world, as the UML diagramming language has come to be almost universally accepted. Yet process is necessary; by themselves, diagrams are of little use. Use Case Driven Object Modeling with UML - Theory and Practice combines the notation of UML with a lightweight but effective process - the ICONIX process - for designing and developing software systems. ICONIX has developed a growing following over the years. Sitting between the free-for-all of Extreme Programming and overly rigid processes such as RUP, ICONIX offers just enough structure to be successful.

For nearly ten years, the Unified Modeling Language (UML) has been the industry standard for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. As the de facto standard modeling language, the UML facilitates communication and reduces confusion among project stakeholders. The recent standardization of UML 2.0 has further extended the language's scope and viability. Its inherent expressiveness allows users to model everything from enterprise information systems and distributed Web-based applications to real-time embedded systems. In this eagerly anticipated revision of the best-selling and definitive guide to the use of the UML,

the creators of the language provide a tutorial to its core aspects in a two-color format designed to facilitate learning. Starting with an overview of the UML, the book explains the language gradually by introducing a few concepts and notations in each chapter. It also illustrates the application of the UML to complex modeling problems across a variety of application domains. The in-depth coverage and example-driven approach that made the first edition of The Unified Modeling Language User Guide an indispensable resource remain unchanged. However, content has been thoroughly updated to reflect changes to notation and usage required by UML 2.0. Highlights include: A new chapter on components and internal structure, including significant new capabilities for building encapsulated designs New details and updated coverage of provided and required interfaces, collaborations, and UML profiles Additions and changes to discussions of sequence diagrams, activity diagrams, and more Coverage of many other changes introduced by the UML 2.0 specification With this essential guide, you will quickly get up to speed on the latest features of the industry standard modeling language and be able to apply them to your next software project.

This new edition continues its unique approach to teaching all aspects of object-oriented programming, bringing it right up to date with the latest advances in technology. It requires no extensive knowledge of programming languages. It is divided into four parts, each presenting the issues involved in object-oriented programming from a different perspective: software engineering and design, languages and system development, abstract data types and polymorphism, and applications and frameworks. Software engineers who want to understand the theory behind modern object-oriented technology while learning about such new topics as patterns, UML, and Java.

Object-Oriented Design with Applications has long been the essential reference to object-oriented technology, which, in turn, has evolved to join the mainstream of industrial-strength software development. In this third edition--the first revision in 13 years--readers can learn to apply object-oriented methods using new paradigms such as Java, the Unified Modeling Language (UML) 2.0, and .NET. The authors draw upon their rich and varied experience to offer improved methods for object development and numerous examples that tackle the complex problems faced by software engineers, including systems architecture, data acquisition, cryptoanalysis, control systems, and Web development. They illustrate essential concepts, explain the method, and show successful applications in a variety of fields. You'll also find pragmatic advice on a host of issues, including classification, implementation strategies, and cost-effective project management. New to this new edition are An introduction to the new UML 2.0, from the notation's most fundamental and advanced elements with an emphasis on key changes New domains and contexts A greatly enhanced focus on modeling--as eagerly requested by readers--with five chapters that each delve into one phase of the overall development lifecycle. Fresh approaches to reasoning about complex systems An examination of the conceptual foundation of the widely misunderstood fundamental elements of the object model, such as abstraction, encapsulation, modularity, and hierarchy How to allocate the resources of a team of developers and mange the risks associated with developing complex software systems An appendix on object-oriented programming languages This is the seminal text for anyone who wishes to use object-oriented technology to manage the complexity inherent in many kinds of systems. Sidebars Preface Acknowledgments About the Authors Section I: Concepts Chapter 1: Complexity Chapter 2: The Object Model Chapter 3: Classes and Objects Chapter 4: Classification Section II: Method Chapter 5: Notation Chapter 6: Process Chapter 7: Pragmatics Chapter 8: System Architecture: Satellite-Based Navigation Chapter 9: Control System: Traffic Management Chapter 10: Artificial Intelligence: Cryptanalysis Chapter 11: Data Acquisition: Weather Monitoring Station Chapter 12: Web Application: Vacation Tracking System Appendix A: Object-Oriented Programming Languages Appendix B: Further Reading Notes Glossary Classified Bibliography Index

This ground-breaking book presents a complete methodology for adaptive programming in any object-oriented programming language.

Lieberherr's adaptive method signals a new approach to object-oriented program design that goes beyond object encapsulation and hard-coded navigation paths to achieve more flexible interactions among objects. Programmers using this method work at a higher, schematic level of abstraction; graph notation represents the class structure and a "propagation pattern" language tells how to distribute meaningful methods - including navigation - across the structure. Using this method, programmers can easily adapt and modify programs as they evolve. This book can be used with any object-oriented programming environment, or with the Demeter Tools Version 5.5, a complete, professional software system for creating and maintaining adaptive programs.

"This thoroughly updated text teaches students or industry R & D practitioners to successfully negotiate the terrain for building and maintaining large, complex software systems. The authors introduce the basic skills needed for a developer to apply software engineering techniques. Next, they focus on methods and technologies that enable developers to specify, design, and implement complex systems. Finally, the authors show how to support the system changes throughout the software life cycle."--BOOK JACKET.Title Summary field provided by Blackwell North America, Inc. All Rights Reserved

A guide on how to reverse engineer legacy systems to understand their problems, and then reengineer those systems to meet new demands. It uses patterns to clarify and explain the process of understanding large code bases, hence transforming them to meet new requirements.

This book provides an excellent overview of Ivar Jacobson's work on the Unified Software Development Process.

Venturing beyond C++ programming, this text shows how to engineer software products using object-oriented principles. It covers gathering requirements, specifying objects, object verification, defining relations between objects, translating object design into code, object testing, and software maintenance.

An authoritative source about methods, languages, methodologies and supporting tools for constructing information systems that also provides examples for references models. Its strength is the careful selection of each of the above mentioned components, based on technical merit. The second edition completely revises all articles and features new material on the latest developments in XML & UML. The structure follows the definition of the major components of Enterprise Integration as defined by GERAM (Generalised Enterprise Reference Architecture and Methodology). 1st edition sold about 600 copies since January 2003.

The first course in software engineering is the most critical. Education must start from an understanding of the heart of software development, from familiar ground that is common to all software development endeavors. This book is an in-depth introduction to software engineering that uses a systematic, universal kernel to teach the essential elements of all software engineering methods. This kernel, Essence, is a vocabulary for defining methods and practices. Essence was envisioned and originally created by Ivar Jacobson and his colleagues, developed by Software Engineering Method and Theory (SEMAT) and approved by The Object Management Group (OMG) as a standard in 2014. Essence is a practice-independent framework for thinking and reasoning about the practices we have and the practices we need. Essence establishes a shared and standard understanding of what is at the heart of software development. Essence is agnostic to any particular method, lifecycle independent, programming language independent, concise, scalable, extensible, and formally specified. Essence frees the practices from their method prisons. The first part of the book describes Essence, the essential elements to work with, the essential things to do and the essential competencies

you need when developing software. The other three parts describe more and more advanced use cases of Essence. Using real but manageable examples, it covers the fundamentals of Essence and the innovative use of serious games to support software engineering. It also explains how current practices such as user stories, use cases, Scrum, and micro-services can be described using Essence, and illustrates how their activities can be represented using the Essence notions of cards and checklists. The fourth part of the book offers a vision how Essence can be scaled to support large, complex systems engineering. Essence is supported by an ecosystem developed and maintained by a community of experienced people worldwide. From this ecosystem, professors and students can select what they need and create their own way of working, thus learning how to create ONE way of working that matches the particular situation and needs.

Object-oriented Software EngineeringA Use Case Driven ApproachAddison-Wesley

This book presents the collected writings of OMT guru Dr James Rumbaugh. These articles encompass the development, refinement, and current state of OMT.

More than ever, mission-critical and business-critical applications depend on object-oriented (OO) software. Testing techniques tailored to the unique challenges of OO technology are necessary to achieve high reliability and quality. "Testing Object-Oriented Systems: Models, Patterns, and Tools" is an authoritative guide to designing and automating test suites for OO applications. This comprehensive book explains why testing must be model-based and provides in-depth coverage of techniques to develop testable models from state machines, combinational logic, and the Unified Modeling Language (UML). It introduces the test design pattern and presents 37 patterns that explain how to design responsibility-based test suites, how to tailor integration and regression testing for OO code, how to test reusable components and frameworks, and how to develop highly effective test suites from use cases. Effective testing must be automated and must leverage object technology. The author describes how to design and code specification-based assertions to offset testability losses due to inheritance and polymorphism. Fifteen micro-patterns present oracle strategies--practical solutions for one of the hardest problems in test design. Seventeen design patterns explain how to automate your test suites with a coherent OO test harness framework. The author provides thorough coverage of testing issues such as: The bug hazards of OO programming and differences from testing procedural code How to design responsibility-based tests for classes, clusters, and subsystems using class invariants, interface data flow models, hierarchic state machines, class associations, and scenario analysis How to support reuse by effective testing of abstract classes, generic classes, components, and frameworks How to choose an integration strategy that supports iterative and incremental development How to achieve comprehensive system testing with testable use cases How to choose a regression test approach How to develop expected test results and evaluate the post-test state of an object How to automate testing with assertions, OO test drivers, stubs, and test frameworks Real-world experience, world-class best practices, and the latest research in object-oriented testing are included. Practical examples illustrate test design and test automation for Ada 95, C++, Eiffel, Java, Objective-C, and Smalltalk. The UML is used throughout, but the test design patterns apply to systems developed with any OO language or methodology.

0201809389B04062001

Based on Objectory which is the first commercially avilable comprehensive object-orientd process for developing large scale industrial systems.

The Unified Modeling Language has become the industry standard for the expression of software designs. The Java programming language continues to grow in popularity as the language of choice for the serious application developer. Using UML and Java together would appear to be a natural marriage, one that can produce considerable benefit. However, there are nuances that the seasoned developer needs to keep in mind when using UML and Java together. Software expert Robert Martin presents a concise guide, with numerous examples, that will help the programmer leverage the power of both development concepts. The author ignores features of UML that do not apply to java programmers, saving the reader time and effort. He provides direct guidance and points the reader to real-world usage scenarios. The overall practical approach of this book brings key information related to Java to the many presentations. The result is an highly practical guide to using the UML with Java.

"This book isn't just another introduction to use cases. The authors have used their wealth of experience to produce an excellent and insightful collection of detailed examples, explanations, and advice on how to work with use cases." –Maria Ericsson The toughest challenge in building a software system that meets the needs of your audience lies in clearly understanding the problems that the system must solve. Advanced Use Case Modeling presents a framework for discovering, identifying, and modeling the problem that the software system will ultimately solve. Software developers often employ use cases to specify what should be performed by the system they're constructing. Although use case-driven analysis, design, and testing of software systems has become increasingly popular, little has been written on the role of use cases in the complete software cycle. This book fills that need by describing how to create use case models for complex software development projects, using practical examples to explain conceptual information. The authors extend the work of software visionary Ivar Jacobson, using the Unified Modeling Language (UML) as the notation to describe the book's models. Aimed primarily at software professionals, Advanced Use Case Modeling also includes information that relates use case technique to business processes. This book presents a process for creating and maintaining use case models in a framework that can be fully customized for your organization. The authors, pioneers in the application of use cases in software development, bring their extensive experience to cover topics such as: A process model for applying a use case model How to keep your use case modeling effort on track Tips and pitfalls in use case modeling How to organize your use case model for large-system development Similarities between Advanced Use Case Modeling and the Rational Unified Process framework Effect of use cases on user interface design Guidelines for quality use case modeling This open access book includes contributions by leading researchers and industry thought leaders on various topics related to the essence of software engineering and their application in industrial projects. It offers a broad overview of research findings dealing with current practical software engineering issues and also pointers to potential future developments. Celebrating the 20th anniversary of adesso AG, adesso gathered some of the pioneers of software engineering including Manfred Broy, Ivar Jacobson

and Carlo Ghezzi at a special symposium, where they presented their thoughts about latest software engineering research and which are part of this book. This way it offers readers a concise overview of the essence of software engineering, providing valuable insights into the latest methodological research findings and adesso's experience applying these results in real-world projects.

For professionals involved in large software development projects with thousands or even millions of lines of code, this best-selling guide offers complete coverage of both classic Software Lifecycle -- requirements, specifications, design, implementation, testing, and maintenance -- and the latest Object-Oriented design approaches. Important new issues, such as object patterns and software architecture, are also included.

Object technology pioneer Wirfs-Brock teams with expert McKean to present a thoroughly updated, modern, and proven method for the design of software. The book is packed with practical design techniques that enable the practitioner to get the job done.

"If you are a serious user of UML, there is no other book quite like this one. I have been involved with the UML specification process for some time, but I still found myself learning things while reading through this book-especially on the changes and new capabilities that have come with UML." -Ed Seidewitz, Chief Architect, IntelliData Technologies Corporation The latest version of the Unified Modeling Language-UML 2.0-has increased its capabilities as the standard notation for modeling software-intensive systems. Like most standards documents, however, the official UML specification is difficult to read and navigate. In addition, UML 2.0 is far more complex than previous versions, making a thorough reference book more essential than ever. In this significantly updated and expanded edition of the definitive reference to the standard, James Rumbaugh, Ivar Jacobson, and Grady Booch-the UML's creators-clearly and completely describe UML concepts, including major revisions to sequence diagrams, activity models, state machines, components, internal structure of classes and components, and profiles. Whether you are capturing requirements, developing software architectures, designing implementations, or trying to understand existing systems, this is the book for you. Highlights include: Alphabetical dictionary of articles covering every UML concept Integrated summary of UML concepts by diagram type Two-color diagrams with extensive annotations in blue Thorough coverage of both semantics and notation, separated in each article for easy reference Further explanations of concepts whose meaning or purpose is obscure in the original specifications Discussion sections offering usage advice and additional insight into tricky concepts Notation summary, with references to individual articles An enhanced online index available on the book's web site allowing readers to quickly and easily search the entire text for specific topics The result is an indispensable resource for anyone who needs to understand the inner workings of the industry standard modeling language.

A catalog of solutions to commonly occurring design problems, presenting 23 patterns that allow designers to create flexible and reusable designs for object-oriented software. Describes the circumstances in which each pattern is applicable, and discusses the consequences and trade-offs of using the pattern within a larger design. Patterns are compiled from real systems, and include code for implementation in object-oriented programming languages like C++ and Smalltalk. Includes a bibliography. Annotation

copyright by Book News, Inc., Portland, OR

Copyright: 17f0a8918b66891df4409712e58759da