

Object Oriented Software Engineering Practical Software Development Using Uml And Java

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). Software Engineering A Practical ApproachBy Laxmidhar V. Gaopandeln this book the author has covered almost all the topics in software engineering which includes types of software projects, their execution models, software development life cycles (SDLC), different development models like Waterfall, Iterative, Incremental, Spiral, Agile and Test Driven Development (TDD). He has covered in depth software requirements including business requirement documents (BRD), functional requirement documents (FRD), software requirement specifications (SRS), what makes a good specifications, software analysis, design and architecture covering structured system analysis and design method (SSADM), object oriented analysis and design (OOAD) methodology, unified modelling language (UML) and UML diagrams, design patterns, software architecture types like layered, microservices, serverless, even driven architecture. Usability and user experience (UX) chapter covers all important aspects of usability engineering and steps in usability. Chapters on quality and quality systems describe attributes of quality and quality systems like ISO 9001, SEI CMMI. Software testing chapter covers details of software testing, types of testing, testing models etc. Details of configuration management, release management, risk management, software support, project management and methodologies are covered in detail. Details on what makes a good project manager and project management organization are also covered in detail. Chapter on software estimation is very detailed and covers various estimation techniques, like Agile estimation, class based simplified estimation for OOAD systems, function point analysis, Mark II, COCOMO etc. Templates for various artifacts are also listed and will be useful for the software engineering work. The book covers five interesting case studies and learnings from them from author own practical experience while executing software projects and product development. The author has also given interesting eighteen exercises for developing a new software system covering all the topics in software engineering. Lot of useful data is also shared which will be very useful for students, teachers and practitioner.

"This book manages to convey the practical use of UML 2 in clear and understandable terms with many examples and guidelines. Even for people not working with the Unified Process, the book is still of great use. UML 2 and the Unified Process, Second Edition is a must-read for every UML 2 beginner and a helpful guide and reference for the experienced practitioner." --Roland Leibundgut, Technical Director, Zuehlke Engineering Ltd. "This book is a good starting point for organizations and individuals who are adopting UP and need to understand how to provide visualization of the different aspects needed to satisfy it. " --Eric Naiburg, Market Manager, Desktop Products, IBM Rational Software This thoroughly revised edition provides an indispensable and practical guide to the complex process of object-oriented analysis and design using UML 2. It describes how the process of OO analysis and design fits into the software development lifecycle as defined by the Unified Process (UP). UML 2 and the Unified Process contains a wealth of practical, powerful, and useful techniques that you can apply immediately. As you progress through the text, you will learn OO analysis and design techniques, UML syntax and semantics, and the relevant aspects of the UP. The book provides you with an accurate and succinct summary of both UML and UP from the point of view of the OO analyst and designer. This book provides Chapter roadmaps, detailed diagrams, and margin notes allowing you to focus on your needs Outline summaries for each chapter, making it ideal for revision, and a comprehensive index that can be used as a reference New to this edition: Completely revised and updated for UML 2 syntax Easy to understand explanations of the new UML 2 semantics More real-world examples A new section on the Object Constraint Language (OCL) Introductory material on the OMG's Model Driven Architecture (MDA) The accompanying website provides A complete example of a simple e-commerce system Open source tools for requirements engineering and use case modeling Industrial-strength UML course materials based on the book

Practical OO development tips for the C++ and Java programmer Practical Object-Oriented Development in C++ and Java offers advice on real-world ways to use these powerful programming languages and techniques. Using the Unified Modeling Language (UML) methodology, expert Cay S. Horstmann gives you clear, concise explanations of object-oriented design, C++, and Java in a way that makes these potentially daunting operations more accessible than they've ever been before. Horstmann compares and contrasts features of C++ and Java to give you a deeper understanding of OO design. He separates the genuinely useful C++, Java, and UML features from the less effective and potentially harmful ones. Horstmann shows you how to determine the best programming practice for whatever application you're in; provides the kind of eye-opening design tips and style rules that can only come from experience; and demystifies advanced topics like frameworks and object persistence. Dozens of illuminating programming examples are readily accessible through the accompanying Web site. Useful code is available for smart pointers, easy output formatting in C++ and Java, a set of classes that makes STL safe to use, and a nifty utility that automatically extracts header files. This unique book: * Offers over 100 practical design hints for good class design * Covers the essential OO features of Java 1.1-like serialization and reflection * Uses the C++ Standard Template Library (STL) throughout * Covers CRC cards in addition to UML

EBOOK: OBJECT-ORIENTED SOFTWARE

Conquering Complex and Changing Systems

Practical Object-Oriented Design in Ruby

Software Engineering

A Use Case Driven Approach

Essays on Object-oriented Software Engineering

This book discusses how model-based approaches can improve the daily practice of software professionals. This is known as Model-Driven Software Engineering (MDSE) or, simply, Model-Driven Engineering (MDE). MDSE practices have proved to increase efficiency and effectiveness in software development, as demonstrated by various quantitative and qualitative studies. MDSE adoption in the software industry is foreseen to grow exponentially in the near future, e.g., due to the convergence of software development and business analysis. The aim of this book is to provide you with an agile and flexible tool to introduce you to the MDSE world, thus allowing you to quickly understand its basic principles and techniques and to choose the right set of MDSE instruments for your needs so that you can start to benefit from MDSE right away. The book is organized into two main parts. The first part discusses the foundations of MDSE in terms of basic concepts (i.e., models and transformations), driving principles, application scenarios, and current standards, like the well-known MDA initiative proposed by OMG (Object Management Group) as well as the practices on how to integrate MDSE in existing development processes. The second part deals with the technical aspects of MDSE, spanning from the basics on when and how to build a domain-specific modeling language, to the description of Model-to-Text and Model-to-Model transformations, and the tools that support the management of MDSE projects. The second edition of the book features: a set of completely new topics, including: full example of the creation of a new modeling language (FMML), discussion of modeling issues and approaches in specific domains, like business process modeling, user interaction modeling, and enterprise architecture complete revision of examples, figures, and text, for improving readability, understandability, and coherence better formulation of definitions, dependencies between concepts and ideas addition of a complete index of book content In addition to the contents of the book, more resources are provided on the book's website <http://www.mdse-book.com>, including the examples presented in the book.

Object-Oriented Software Engineering is written for both the traditional one-semester and the newer two-semester software engineering curriculum. Part I covers the underlying software engineering theory, while Part II presents the more practical life cycle, workflow by workflow. The text is intended for the substantial object-oriented segment of the software engineering market. It focuses exclusively on object-oriented approaches to the development of large software systems that are the most widely used. Text includes 2 running case studies, expanded coverage of agile processes and open-source development.

Test-Driven Development (TDD) is now an established technique for delivering better software faster. TDD is based on a simple idea: Write tests for your code before you write the code itself. However, this "simple" idea takes skill and judgment to do well. Now there's a practical guide to TDD that takes you beyond the basic concepts. Drawing on a decade of experience building real-world systems, two TDD pioneers show how to let tests guide your development and "grow" software that is coherent, reliable, and maintainable. Steve Freeman and Nat Pryce describe the processes they use, the design principles they strive to achieve, and some of the tools that help them get the job done. Through an extended worked example, you'll learn how TDD works at multiple levels, using tests to drive the features and the object-oriented structure of the code, and using Mock Objects to discover and then describe relationships between objects. Along the way, the book systematically addresses challenges that development teams encounter with TDD—from integrating TDD into your processes to testing your most difficult features. Coverage includes Implementing TDD effectively: getting started, and maintaining your momentum throughout the project Creating cleaner, more expressive, more sustainable code Using tests to stay relentlessly focused on sustaining quality Understanding how TDD, Mock Objects, and Object-Oriented Design come together in the context of a real software development project Using Mock Objects to guide object-oriented designs Succeeding where TDD is difficult: managing complex test data, and testing persistence and concurrency

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 short, intensive management courses. This textbook shows how to use both the principles of software engineering as well as 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, this book emphasizes practical experience: participants can apply the techniques learned in class by implementing a real-world software project.

Practical Object-Oriented Design

Practical Software Development Using UML and Java

Second Edition

Theory and Practice

Practical Object-oriented Design in Ruby

Object-oriented Software Engineering

The Complete Guide to Writing Maintainable, Manageable, Pleasing, and Powerful Object-Oriented Applications Object-oriented programming languages exist to help you create beautiful, straightforward applications that are easy to change and simple to extend. Unfortunately, the world is awash with object-oriented (OO) applications that are difficult to understand and expensive to change. Practical Object-Oriented Design, Second Edition, immerses you in an OO mindset and teaches you powerful, real-world, object-oriented design techniques with simple and practical examples. Sandi Metz demonstrates how to build new applications that can "survive success" and repair existing applications that have become impossible to change. Each technique is illustrated with extended examples in the easy-to-understand Ruby programming language, all downloadable from the companion website, poodr.com. Fully updated for Ruby 2.5, this guide shows how to Decide what belongs in a single class Avoid entangling objects that should be kept separate Define flexible interfaces among objects Reduce programming overhead costs with duck typing Successfully apply inheritance Build objects via composition Whatever your previous object-oriented experience, this concise guide will help you achieve the superior outcomes you're looking for. Register your book for convenient access to downloads, updates, and/or corrections as they become available. See inside book for details.

This textbook provides a progressive approach to the teaching of software engineering. First, readers are introduced to the core concepts of the object-oriented methodology, which is used throughout the book to act as the foundation for software engineering and programming practices, and partly for the software engineering process itself. Then, the processes involved in software engineering are explained in more detail, especially methods and their applications in design, implementation, testing, and measurement, as they relate to software engineering projects. At last, readers are given the chance to practice these concepts by applying commonly used skills and tasks to a hands-on project. The impact of such a format is the potential for quicker and deeper understanding. Readers will master concepts and skills at the most basic levels before continuing to expand on and apply these lessons in later chapters.

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.

Presents a novel metrics-based approach for detecting design problems in object-oriented software. Introduces an important suite of detection strategies for the identification of different well-known design flaws as well as some rarely mentioned ones.

Model-Driven Software Engineering in Practice

A Practical Guide to Testing Object-oriented Software

Project-based Software Engineering

Object-oriented and Classical Software Engineering

Object-oriented Analysis and Design in Practice

Classical and Object-Oriented Software Engineering is designed for an introductory software engineering course. This book provides an excellent introduction to software engineering fundamentals, covering both traditional and object-oriented techniques. Schach's unique organization and style makes it excellent for use in a classroom setting. It presents the underlying software engineering theory in Part I and follows it up with the more practical life-cycle material in Part II. Many software engineering books are more like reference books, which do not provide the appropriate fundamentals before inundating students with implementation details. In this edition, more practical material has been added to help students understand how to use what they are learning. This has been done through the use of "How To" boxes and greater implementation detail in the case study. Additionally, the new edition contains the references to the most current literature and includes an overview of extreme programming. The website in this edition will be more extensive. It will include Solutions, PowerPoints that incorporate lecture notes, newly developed self-quiz questions, and source code for the term project and case study.

Project-Based Software Engineering is the first book to provide hands-on process and practice in software engineering essentials for the beginner. The book presents steps through the software development life cycle and two running case studies that develop as the steps are presented. Running parallel to the process presentation and case studies, the book supports a semester-long software development project. This book focuses on object-oriented software development, and supports the conceptualization, analysis, design and implementation of an object-oriented project. It is mostly language-independent, with necessary code examples in Java. A subset of UML is used, with the notation explained as needed to support the readers' work. Two running case studies a video game and a library check out system show the development of a software project. Both have sample deliverables and thus provide the reader with examples of the type of work readers are to create. This book is appropriate for readers looking to gain experience in project analysis, design implementation, and testing.

Examines object-oriented methods, practices, terminology, and concepts

Object-oriented Software EngineeringPractical Software Development Using UML and JavaMcGraw-Hill College

Object-oriented software engineering

Developing Software with UML

Practical Object-oriented Design with UML

UML 2 and the Unified Process

Object-Oriented Software Engineering: Practical Software Development using UML and Java

Software Engineering: A Hands-On Approach

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.

The Complete Guide to Writing More Maintainable, Manageable, Pleasing, and Powerful Ruby Applications Ruby's widely admired ease of use has a downside: Too many Ruby applications have been created without concern for their long-term maintenance or evolution. The Web is awash in Ruby code that is now virtually impossible to change or extend. This text helps you solve that problem by using powerful real-world object-oriented design techniques, which it thoroughly explains using simple and practical Ruby examples. Sandi Metz has distilled a lifetime of conversations and presentations about object-oriented design into a set of Ruby-focused practices for crafting manageable, extensible, and pleasing code. She shows you how to build new applications that can survive success and repair existing applications that have become impossible to change. Each technique is illustrated with extended examples, all downloadable from the companion Web site, poodr.info. The first title to focus squarely on object-oriented Ruby application design, Practical Object-Oriented Design in Ruby will guide you to superior outcomes, whatever your previous Ruby experience. Novice Ruby programmers will find specific rules to live by; intermediate Ruby programmers will find valuable principles they can flexibly interpret and apply; and advanced Ruby programmers will find a common language they can use to lead development and guide their colleagues. This guide will help you Understand how object-oriented programming can help you craft Ruby code that is easier to maintain and upgrade Decide what belongs in a single Ruby class Avoid entangling objects that should be kept separate Define flexible interfaces among objects Reduce programming overhead costs with duck typing Successfully apply inheritance Build objects via composition Design cost-effective tests Solve common problems associated with poorly designed Ruby code

David A. Sykes is a member of Wofford College's faculty.

EBOOK: OBJECT-ORIENTED SOFTWARE

An Object-oriented Approach

Practical API Design

Object-Oriented Software Engineering: Practical Software Development

Object-Oriented and Classical Software Engineering

Object-Oriented Metrics in Practice

Object-Oriented Software Engineering: An Agile Unified Methodology

Software -- Software Engineering.

This content helps in preparing yourself to face the real world of object-oriented software engineering challenges. This book presents the fundamental concepts of object-oriented software engineering, including analysis, design, implementation and testing in reader friendly way. All the contents are presented via comprehensive descriptions, with well-structured figures and examples to make the concept crystal clear. This book presents a solid comprehensive self-study guide in the field of object-oriented software engineering for both students and Software-developers. The core part of this book is UML Diagrams and Software Architecture that is ready to build a concrete concept of object-oriented software engineering with a practical approach. This book is written with the aim to provide compressive contents in the hand of readers that enables them to understand & build concepts in minimal time.

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.

The Complete Guide to Writing More Maintainable, Manageable, Pleasing, and Powerful Ruby Applications Ruby's widely admired ease of use has a downside: Too many Ruby and Rails applications have been created without concern for their long-term maintenance or evolution. The Web is awash in Ruby code that is now virtually impossible to change or extend. This text helps you solve that problem by using powerful real-world object-oriented design techniques, which it thoroughly explains using simple and practical Ruby examples. Sandi Metz has distilled a lifetime of conversations and presentations about object-oriented design into a set of Ruby-focused practices for crafting manageable, extensible, and pleasing code. She shows you how to build new applications that can survive success and repair existing applications that have become impossible to change. Each technique is illustrated with extended examples, all downloadable from the companion Web site,poodr.info. The first title to focus squarely on object-oriented Ruby application design, Practical Object-Oriented Design in Ruby will guide you to superior outcomes, whatever your previous Ruby experience. Novice Ruby programmers will find specific rules to live by; intermediate Ruby programmers will find valuable principles they can flexibly interpret and apply; and advanced Ruby programmers will find a common language they can use to lead development and guide their colleagues. This guide will help you Understand how object-oriented programming can help you craft Ruby code that is easier to maintain and upgrade Decide what belongs in a single Ruby class Avoid entangling objects that should be kept separate Define flexible interfaces among objects Reduce programming overhead costs with duck typing Successfully apply inheritance Build objects via composition Design cost-effective tests Solve common problems associated with poorly designed Ruby code

Practical Object-Oriented Analysis and Design

Practical Object-Oriented Development in C++ and Java

Growing Object-Oriented Software, Guided by Tests

Object-Oriented Software Engineering Using UML, Patterns, and Java

Object-Oriented Software Engineering

Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems

This book shows us how to use UML and apply it in object-oriented software development. Part 1 of the book guides the reader step-by-step through the development process while part 2 explains the basics of UML in detail.

Software Engineering: The Current Practice teaches students basic software engineering skills and helps practitioners refresh their knowledge and explore recent developments in the field, including software changes and iterative processes of software development. After a historical overview and an introduction to software technology and models, the book discusses the software change and its practices, including concept location, impact analysis, refactoring, actualization, and verification. It then covers the most common iterative processes: agile, directed, and centralized

processes. The text also journeys through the software life span from the initial development of software from scratch to the final stages that lead toward software closedown. For Professionals The book gives programmers and software managers a unified view of the contemporary practice of software engineering. It shows how various developments fit together and fit into the contemporary software engineering mosaic. The knowledge gained from the book allows practitioners to evaluate and improve the software engineering processes in their projects. For Instructors Instructors have several options for using this classroom-tested material. Designed to be run in conjunction with the lectures, ideas for student projects include open source programs that use Java or C++ and range in size from 50 to 500 thousand lines of code. These projects emphasize the role of developers in a classroom-tailored version of the directed iterative process (DIP). For Students Students gain a real understanding of software engineering processes through the lectures and projects. They acquire hands-on experience with software of the size and quality comparable to that of industrial software. As is the case in the industry, students work in teams but have individual assignments and accountability.

Taking a learn-by-doing approach, Software Engineering Design: Theory and Practice uses examples, review questions, chapter exercises, and case study assignments to provide students and practitioners with the understanding required to design complex software systems. Explaining the concepts that are immediately relevant to software designers, it begins with a review of software design fundamentals. The text presents a formal top-down design process that consists of several design activities with varied levels of detail, including the macro-, micro-, and construction-design levels. As part of the top-down approach, it provides in-depth coverage of applied architectural, creational, structural, and behavioral design patterns. For each design issue covered, it includes a step-by-step breakdown of the execution of the design solution, along with an evaluation, discussion, and justification for using that particular solution. The book outlines industry-proven software design practices for leading large-scale software design efforts, developing reusable and high-quality software systems, and producing technical and customer-driven design documentation. It also: Offers one-stop guidance for mastering the Software Design & Construction sections of the official Software Engineering Body of Knowledge (SWEBOK®) Details a collection of standards and guidelines for structuring high-quality code Describes techniques for analyzing and evaluating the quality of software designs Collectively, the text supplies comprehensive coverage of the software design concepts students will need to succeed as professional design leaders. The section on engineering leadership for software designers covers the necessary ethical and leadership skills required of software developers in the public domain. The section on creating software design documents (SDD) familiarizes students with the software design notations, structural descriptions, and behavioral models required for SDDs. Course notes, exercises with answers, online resources, and an instructor ' s manual are available upon qualified course adoption. Instructors can contact the author about these resources via the author's website: <http://softwareengineeringdesign.com/>

Classical and Object-Oriented Software Engineering, 5/e is designed for an introductory software engineering course. This book provides an excellent introduction to software engineering fundamentals, covering both traditional and object-oriented techniques.Schach's unique organization and style makes it excellent for use in a classroom setting. It presents the underlying software engineering theory in Part I and follows it up with the more practical life-cycle material in Part II. Many software engineering books are more like reference books, which do not provide the appropriate fundamentals before inundating students with implementation details.In this edition, more practical material has been added to help students understand how to use what they are learning. This has been done through the use of "How To" boxes and greater implementation detail in the case study. Additionally, the new edition contains the references to the most current literature and includes an overview of extreme programming.The website in this edition will be more extensive. It will include Solutions, PowerPoints that incorporate lecture notes, newly developed self-quiz questions, and source code for the term project and case study.

Software Engineering with Ada

Object-oriented Software Engineering with Eiffel

Software Engineering and Computer Games

Practical Approach To Software Engineering

Confessions of a Java Framework Architect

Pearson New International Edition

Provides complete coverage of the Ada language and Ada programming in general by recognized authorities in Ada software engineering. Demonstrates the power and performance of Ada in the management of large-scale object-oriented systems, and shows how to use Ada features such as generics, packages, and tasking.

Do modern programming languages, IDEs, and libraries make coding easy? Maybe, but coding is not design. Large-scale or expensive apps clearly require evaluation of design choices. Still, software design directly impacts code reuse and longevity even for small-scale apps with limited overhead. This text evaluates and contrasts common object-oriented designs. A given problem may have many solutions. A developer may employ different design techniques – composition, inheritance, dependency injection, delegation, etc. – to solve a particular problem. A skilled developer can determine the costs and benefits of different design responses, even amid competing concerns. A responsible developer documents design choices as a contract with the client, delineating external and internal responsibilities. To promote effective software design, this book examines contractual, object-oriented designs for immediate and sustained use as well as code reuse. The intent of identifying design variants is to recognize and manage conflicting goals such as short versus long-term utility, stability versus flexibility, and storage versus computation. Many examples are given to evaluate and contrast different solutions and to compare C# and C++ effects. No one has a crystal ball; however, deliberate design promotes software longevity. With the prominence of legacy OO code, a clear understanding of different object-oriented designs is essential. Design questions abound. Is code reuse better with inheritance or composition? Should composition rely on complete encapsulation? Design choices impact flexibility, efficiency, stability, longevity, and reuse, yet compilers do not enforce design and syntax does not necessarily illustrate design. Through deliberate design, or redesign when refactoring, developers construct sustainable, efficient code.

This book is based on object-oriented techniques applied to software engineering. Employing the latest technologies such as UML, Patterns, and Java, Bernd Bruegge and Allen H. Dutoit offer a cohesive, class-tested presentation of object-oriented software engineering in a step-by-step format based on ten years of teaching and real-world software engineering experience. This text teaches practical experience in developing complex software appropriate for software engineering project courses, as well as industry R & D practitioners. The reader benefits from timely exposure to state-of-the-art tools and methods. Unlike other texts based on the teaching premise of multiple classes or developing multiple systems, this book focuses on techniques and applications in a reasonably complex environment, such as multi-team development projects including 20 to 60 participants. The book is based on concrete examples from real applications such as accident management, emissions modeling, facility management, and centralized traffic control. Provides an integrated communication infrastructure for distributed development Shows the state of the art in Software Engineering: UML, Java, Design Patterns, Distributed Development, and Multiproject Management Illustrates how the reader learns to develop in a distributed team with hands-on experience on real system development problems Offers a CD-ROM containing the materials used in courses taught by the authors-problem statements, requirement analysis documents, system design documents, test manuals, prototypes, and all the artifacts produced during the development of a facility management system Presents Companion Website (www.prenhall.com/bruegge) withsupplemental material such as problem statements, requirement analysis documents, system design documents, test manuals, and solutions to exercises

This is a revised and updated edition of this title, which provides a practical introduction to the design of object-oriented programs using UML. It includes detailed coverage of modelling techniques and notation, with worked examples throughout. The book contains substantial code examples in Java. It clearly connects design concepts with code, and is useful for people with programming experience who wish to learn about design. It is also useful for computer science and software engineering undergraduates taking courses covering object-oriented techniques. The book provides explanations of UML and OCL notation emphasis on transitions from design to code, as well as including complete case studies with code, and many exercises.

Software Engineering Design

practical software development using UML and Java

With UML Diagram

The Current Practice

Design Patterns for Object-oriented Software Development

An Agile Primer Using Ruby

You might think more than enough design books exist in the programming world already. In fact, there are so many that it makes sense to ask why you would read yet another. Is there really a need for yet another design book? In fact, there is a greater need than ever before, and Practical API Design: Confessions of a Java Framework Architect fills that need! Teaches you how to write an API that will stand the test of time Written by the designer of the NetBeans API at Sun Technologies Based on best practices, scalability, and API design patterns

An indispensable resource for anyone working with Eiffel, this up-to-date guide provides full coverage of the most recent version of the language, focusing on Eiffel's practical use in the development of large, mission-critical software systems. In addition to a comprehensive description of Eiffel's syntax and semantics, you will find in-depth information on style guides, analysis and design, design patterns, and validation and testing. Descriptions and comparisons of available compilers and libraries will help you decide which Eiffel tools best fit your development needs. The book even includes an Eiffel resource guide. The book's most notable feature is its three large-scale case studies that demonstrate Eiffel in action, illustrating implementation techniques and showcasing Eiffel's power and effectiveness in three different realms: the MIS world, the embedded systems/telecommunications world, and the numeric world. By reading this book, you will not only obtain a knowledge of the mechanics of Eiffel programming, but you will also come away with an understanding of Eiffel's role in the field of object-oriented technology and a sense of the language's strong potential in large software development. 0201633817B04062001

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 modelling, 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.

This book solves the dilemma of wanting to learn Windows-based software engineering without knowing Windows programming. The basics in Windows programming are explained alongside ideas of object-oriented software engineering. (Midwest).

A Practical Approach

An Agile Primer

Object-Oriented Design Choices