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. Agile software development has become an umbrella term for a number of changes in how software developers plan and coordinate their work, how they communicate with customers and external stakeholders, and how software development is organized in small, medium, and large companies, from the telecom and healthcare sectors to games and interactive media. Still, after a decade of research, agile software development is the source of continued debate due to its multifaceted nature and insufficient synthesis of research results. Dingsøyr, Dybå, and Moe now present a comprehensive snapshot of the knowledge gained over many years of research by those working closely with or in the industry. It shows the current state of research on agile software development through an introduction and ten invited contributions on the main research fields, each written by renowned experts. These chapters cover three main issues: foundations and background of agile development, agile methods in practice, and principal challenges and new frontiers. They show
the important results in each subfield, and in addition they explain what these results mean to practitioners as well as for future research in the field. The book is aimed at reflective practitioners and researchers alike, and it also can serve as the basis for graduate courses at universities. For those considering Extreme Programming, this book provides no-nonsense advice on agile planning, development, delivery, and management taken from the authors' many years of experience. While plenty of books address the what and why of agile development, very few offer the information users can apply directly. 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. The Robert C. Martin Clean Code Collection consists of two bestselling eBooks: Clean Code: A Handbook of Agile Software Craftsmanship The Clean Coder: A Code of Conduct for Professional Programmers In Clean Code, legendary software expert Robert C. Martin has teamed up with his colleagues from Object Mentor to distill their best agile practice of cleaning code “on the fly” into a book that will instill within you the values of a software craftsman and make you a better programmer—but only if you work at it. You will be challenged to think about what’s right about that code and what’s wrong with it. More important, you will be challenged to reassess your professional values and your commitment to your craft. In The Clean Coder, Martin introduces the disciplines, techniques, tools, and practices of true software craftsmanship. This book is packed with practical advice—about everything from estimating and coding to refactoring and testing. It covers much more
than technique: It is about attitude. Martin shows how to approach software
development with honor, self-respect, and pride; work well and work clean;
communicate and estimate faithfully; face difficult decisions with clarity
and honesty; and understand that deep knowledge comes with a responsibility
to act. Readers of this collection will come away understanding How to tell
the difference between good and bad code How to write good code and how to
transform bad code into good code How to create good names, good functions,
good objects, and good classes How to format code for maximum readability
How to implement complete error handling without obscuring code logic How to
unit test and practice test-driven development What it means to behave as a
ttrue software craftsman How to deal with conflict, tight schedules, and
unreasonable managers How to get into the flow of coding and get past
writer’s block How to handle unrelenting pressure and avoid burnout How to
combine enduring attitudes with new development paradigms How to manage your
time and avoid blind alleys, marshes, bogs, and swamps How to foster
environments where programmers and teams can thrive When to say “No”--and
how to say it When to say “Yes”--and what yes really meansUsers can
dramatically improve the design, performance, and manageability of object-
oriented code without altering its interfaces or behavior. "Refactoring"
shows users exactly how to spot the best opportunities for refactoring and
exactly how to do it, step by step.Agile coding with design patterns and
SOLID principles As every developer knows, requirements are subject to
change. But when you build adaptability into your code, you can respond to
change more easily and avoid disruptive rework. Focusing on Agile
programming, this book describes the best practices, principles, and patterns that enable you to create flexible, adaptive code—and deliver better business value. Expert guidance to bridge the gap between theory and practice Get grounded in Scrum: artifacts, roles, metrics, phases Organize and manage architectural dependencies Review best practices for patterns and anti-patterns Master SOLID principles: single-responsibility, open/closed, Liskov substitution Manage the versatility of interfaces for adaptive code Perform unit testing and refactoring in tandem See how delegation and abstraction impact code adaptability Learn best ways to implement dependency interjection Apply what you learn to a pragmatic, agile coding project Get code samples at: http://github.com/garymclean/AdaptiveCode Agile Values and Principles for a New Generation “In the journey to all things Agile, Uncle Bob has been there, done that, and has the both the t-shirt and the scars to show for it. This delightful book is part history, part personal stories, and all wisdom. If you want to understand what Agile is and how it came to be, this is the book for you.” –Grady Booch “Bob’s frustration colors every sentence of Clean Agile, but it’s a justified frustration. What is in the world of Agile development is nothing compared to what could be. This book is Bob’s perspective on what to focus on to get to that ‘what could be.’ And he’s been there, so it’s worth listening.” –Kent Beck “It’s good to read Uncle Bob’s take on Agile. Whether just beginning, or a seasoned Agilista, you would do well to read this book. I agree with almost all of it. It’s just some of the parts make me realize my own shortcomings, dammit. It made me double-check our code coverage (85.09%).” –Jon Kern Nearly twenty years
after the Agile Manifesto was first presented, the legendary Robert C. Martin ("Uncle Bob") reintroduces Agile values and principles for a new generation—programmers and nonprogrammers alike. Martin, author of Clean Code and other highly influential software development guides, was there at Agile’s founding. Now, in Clean Agile: Back to Basics, he strips away misunderstandings and distractions that over the years have made it harder to use Agile than was originally intended. Martin describes what Agile is in no uncertain terms: a small discipline that helps small teams manage small projects . . . with huge implications because every big project is comprised of many small projects. Drawing on his fifty years’ experience with projects of every conceivable type, he shows how Agile can help you bring true professionalism to software development. Get back to the basics—what Agile is, was, and should always be Understand the origins, and proper practice, of SCRUM Master essential business-facing Agile practices, from small releases and acceptance tests to whole-team communication Explore Agile team members’ relationships with each other, and with their product Rediscover indispensable Agile technical practices: TDD, refactoring, simple design, and pair programming Understand the central roles values and craftsmanship play in your Agile team’s success If you want Agile’s true benefits, there are no shortcuts: You need to do Agile right. Clean Agile: Back to Basics will show you how, whether you’re a developer, tester, manager, project manager, or customer. Register your book for convenient access to downloads, updates, and/or corrections as they become available. See inside book for details. Agile techniques have demonstrated immense potential for developing
more effective, higher-quality software. However, scaling these techniques to the enterprise presents many challenges. The solution is to integrate the principles and practices of Lean Software Development with Agile’s ideology and methods. By doing so, software organizations leverage Lean’s powerful capabilities for “optimizing the whole” and managing complex enterprise projects. A combined “Lean-Agile” approach can dramatically improve both developer productivity and the software’s business value. In this book, three expert Lean software consultants draw from their unparalleled experience to gather all the insights, knowledge, and new skills you need to succeed with Lean-Agile development. Lean-Agile Software Development shows how to extend Scrum processes with an Enterprise view based on Lean principles. The authors present crucial technical insight into emergent design, and demonstrate how to apply it to make iterative development more effective. They also identify several common development “anti-patterns” that can work against your goals, and they offer actionable, proven alternatives. Lean-Agile Software Development shows how to Transition to Lean Software Development quickly and successfully. Manage the initiation of product enhancements. Help project managers work together to manage product portfolios more effectively. Manage dependencies across the software development organization and with its partners and colleagues. Integrate development and QA roles to improve quality and eliminate waste. Determine best practices for different software development teams. The book’s companion Web site, www.netobjectives.com/lassd, provides updates, links to related materials, and support for discussions of the book’s content. Understand the
big picture of the software development process. We use software every day – operating systems, applications, document editing programs, home banking – but have you ever wondered who creates software and how it's created? This book guides you through the entire process, from conception to the finished product with the aid of user-centric design theory and tools. Software Development: From A to Z provides an overview of backend development – from databases to communication protocols including practical programming skills in Java and of frontend development – from HTML and CSS to npm registry and Vue.js framework. You'll review quality assurance engineering, including the theory about different kind of tests and practicing end-to-end testing using Selenium. Dive into the devops world where authors discuss continuous integration and continuous delivery processes along with each topic's associated technologies. You'll then explore insightful product and project management coverage where authors talk about agile, scrum and other processes from their own experience. The topics that are covered do not require a deep knowledge of technology in general; anyone possessing basic computer and programming knowledge will be able to complete all the tasks and fully understand the concepts this book aims at delivering. You'll wear the hat of a project manager, product owner, designer, backend, frontend, QA and devops engineer, and find your favorite role. What You'll Learn
Understand the processes and roles involved in the creation of software
Organize your ideas when building the concept of a new product
Experience the work performed by stakeholders and other departments of expertise, their individual challenges, and how to overcome possible threats
Improve the ways
stakeholders and departments can work with each other. Gain ideas on how to improve communication and processes.

Who This Book Is For

Anyone who is on a team that creates software and is curious to learn more about other stakeholders or departments involved. Those interested in a career change and want to learn about how software gets created. Those who want to build technical startups and wonder what roles might be involved in the process.

The rules of battle for tracking down -- and eliminating -- hardware and software bugs. When the pressure is on to root out an elusive software or hardware glitch, what's needed is a cool head courtesy of a set of rules guaranteed to work on any system, in any circumstance. Written in a frank but engaging style, Debugging provides simple, foolproof principles guaranteed to help find any bug quickly. This book makes those shelves of application-specific debugging books (on C++, Perl, Java, etc.) obsolete. It changes the way readers think about debugging, making those pesky problems suddenly much easier to find and fix. Illustrating the rules with real-life bug-detection war stories, the book shows readers how to:

* Understand the system: how perceiving the ""roadmap"" can hasten your journey
* Quit thinking and look: when hands-on investigation can't be avoided
* Isolate critical factors: why changing one element at a time can be an essential tool
* Keep an audit trail: how keeping a record of the debugging process can win the day

The rules of battle for tracking down -- and eliminating -- hardware and software bugs. When the pressure is on to root out an elusive software or hardware glitch, what's needed is a cool head courtesy of a set of rules guaranteed to work on any system, in any circumstance. Written in a
Read PDF Agile Software Development Principles Patterns Practices

frank but engaging style, Debugging provides simple, foolproof principles guaranteed to help find any bug quickly. This book makes those shelves of application-specific debugging books (on C++, Perl, Java, etc.) obsolete. It changes the way readers think about debugging, making those pesky problems suddenly much easier to find and fix. Illustrating the rules with real-life bug-detection war stories, the book shows readers how to: * Understand the system: how perceiving the "roadmap" can hasten your journey * Quit thinking and look: when hands-on investigation can't be avoided * Isolate critical factors: why changing one element at a time can be an essential tool * Keep an audit trail: how keeping a record of the debugging process can win the day The rules of battle for tracking down -- and eliminating -- hardware and software bugs. When the pressure is on to root out an elusive software or hardware glitch, what's needed is a cool head courtesy of a set of rules guaranteed to work on any system, in any circumstance. Written in a frank but engaging style, Debugging provides simple, foolproof principles guaranteed to help find any bug quickly. This book makes those shelves of application-specific debugging books (on C++, Perl, Java, etc.) obsolete. It changes the way readers think about debugging, making those pesky problems suddenly much easier to find and fix. Illustrating the rules with real-life bug-detection war stories, the book shows readers how to: * Understand the system: how perceiving the "roadmap" can hasten your journey * Quit thinking and look: when hands-on investigation can't be avoided * Isolate critical factors: why changing one element at a time can be an essential tool * Keep an audit trail: how keeping a record of the debugging process
can win the day. For courses in Object-Oriented Design, C++ Intermediate Programming, and Object-Oriented Programming. Written for software engineers in the trenches, this text focuses on the technology—the principles, patterns, and process—that help software engineers effectively manage increasingly complex operating systems and applications. There is also a strong emphasis on the people behind the technology. This text will prepare students for a career in software engineering and serve as an on-going education for software engineers. We're losing tens of billions of dollars a year on broken software, and great new ideas such as agile development and Scrum don't always pay off. But there's hope. The nine software development practices in Beyond Legacy Code are designed to solve the problems facing our industry. Discover why these practices work, not just how they work, and dramatically increase the quality and maintainability of any software project. These nine practices could save the software industry. Beyond Legacy Code is filled with practical, hands-on advice and a common-sense exploration of why technical practices such as refactoring and test-first development are critical to building maintainable software. Discover how to avoid the pitfalls teams encounter when adopting these practices, and how to dramatically reduce the risk associated with building software—realizing significant savings in both the short and long term. With a deeper understanding of the principles behind the practices, you'll build software that's easier and less costly to maintain and extend. By adopting these nine key technical practices, you'll learn to say what, why, and for whom before how; build in small batches; integrate continuously; collaborate; create
CLEAN code; write the test first; specify behaviors with tests; implement the design last; and refactor legacy code. Software developers will find hands-on, pragmatic advice for writing higher quality, more maintainable, and bug-free code. Managers, customers, and product owners will gain deeper insight into vital processes. By moving beyond the old-fashioned procedural thinking of the Industrial Revolution, and working together to embrace standards and practices that will advance software development, we can turn the legacy code crisis into a true Information Revolution. With the award-winning book Agile Software Development: Principles, Patterns, and Practices, Robert C. Martin helped bring Agile principles to tens of thousands of Java and C++ programmers. Now .NET programmers have a definitive guide to agile methods with this completely updated volume from Robert C. Martin and Micah Martin, Agile Principles, Patterns, and Practices in C#. This book presents a series of case studies illustrating the fundamentals of Agile development and Agile design, and moves quickly from UML models to real C# code. The introductory chapters lay out the basics of the agile movement, while the later chapters show proven techniques in action. The book includes many source code examples that are also available for download from the authors’ Web site. Readers will come away from this book understanding Agile principles, and the fourteen practices of Extreme Programming—spiking, splitting, velocity, and planning iterations and releases—test-driven development, test-first design, and acceptance testing. Refactoring with unit testing—pair programming—Agile design and design smells—The five types of UML diagrams and how to use them effectively—Object-
oriented package design and design patterns. How to put all of it together for a real-world project. Whether you are a C# programmer or a Visual Basic or Java programmer learning C#, a software development manager, or a business analyst, Agile Principles, Patterns, and Practices in C# is the first book you should read to understand agile software and how it applies to programming in the .NET Framework. For courses in Advanced Software Engineering or Object-Oriented Design. This book covers the human and organizational dimension of the software improvement process and software project management—whether based on the CMM or ISO 9000 or the Rational Unified Process. Drawn from a decade of research, it emphasizes common-sense practices. Its principles are general but concrete; every pattern is its own built-in example. Historical supporting material from other disciplines is provided. Though even pattern experts will appreciate the depth and currency of the material, it is self-contained and well-suited for the layperson. Section 1 Agile development Section 2 Agile design Section 3 The payroll case study Section 4 Packaging the payroll system Section 5 The weather station case study Section 6 The ETS case study Looks at the principles and clean code, includes case studies showcasing the practices of writing clean code, and contains a list of heuristics and "smells" accumulated from the process of writing clean code. For software to consistently deliver promised results, software development must mature into a true profession. Emergent Design points the way. As software continues to evolve and mature, software development processes become more complicated, relying on a variety of methodologies and approaches. This book illuminates
the path to building the next generation of software. Author Scott L. Bain integrates the best of today's most important development disciplines into a unified, streamlined, realistic, and fully actionable approach to developing software. Drawing on patterns, refactoring, and test-driven development, Bain offers a blueprint for moving efficiently through the entire software lifecycle, smoothly managing change, and consistently delivering systems that are robust, reliable, and cost-effective. Reflecting a deep understanding of the natural flow of system development, Emergent Design helps developers work with the flow, instead of against it. Bain introduces the principles and practices of emergent design one step at a time, showing how to promote the natural evolution of software systems over time, making systems work better and provide greater value. To illuminate his approach, Bain presents code examples wherever necessary and concludes with a complete project case study. This book provides developers, project leads, and testers powerful new ways to collaborate, achieve immediate goals, and build systems that improve in quality with each iteration. Coverage includes How to design software in a more natural, evolutionary, and professional way How to use the “open-closed” principle to mitigate risks and eliminate waste How and when to test your design throughout the development process How to translate design principles into practices that actually lead to better code How to determine how much design is enough How refactoring can help you reduce over-design and manage change more effectively The book's companion Web site, www.netobjectives.com/resources, provides updates, links to related materials, and support for discussions of the book's content.
“I’m dancing! By god I’m dancing on the walls. I’m dancing on the ceiling. I’m ecstatic. I’m overjoyed. I’m really, really pleased.” –From the Foreword by Robert C. Martin (a.k.a. Uncle Bob) This isn’t the first book on Java application architecture. No doubt it won’t be the last. But rest assured, this title is different. The way we develop Java applications is about to change, and this title explores the new way of Java application architecture. Over the past several years, module frameworks have been gaining traction on the Java platform, and upcoming versions of Java will include a module system that allows you to leverage the power of modularity to build more resilient and flexible software systems. Modularity isn’t a new concept. But modularity will change the way we develop Java applications, and you’ll only be able to realize the benefits if you understand how to design more modular software systems. Java Application Architecture will help you Design modular software that is extensible, reusable, maintainable, and adaptable Design modular software today, in anticipation of future platform support for modularity Break large software systems into a flexible composite of collaborating modules Understand where to place your architectural focus Migrate large-scale monolithic applications to applications with a modular architecture Articulate the advantages of modular software to your team Java Application Architecture lays the foundation you’ll need to incorporate modular design thinking into your development initiatives. Before it walks you through eighteen patterns that will help you architect modular software, it lays a solid foundation
that shows you why modularity is a critical weapon in your arsenal of design tools. Throughout, you’ll find examples that illustrate the concepts. By designing modular applications today, you are positioning yourself for the platform and architecture of tomorrow. That’s why Uncle Bob is dancing.

More C++ Gems picks up where the first book left off, presenting tips, tricks, proven strategies, easy-to-follow techniques, and usable source code. Provides information on successful software development, covering such topics as customer requirements, task estimates, principles of good design, dealing with source code, system testing, and handling bugs. Write code that can adapt to changes. By applying this book’s principles, you can create code that accommodates new requirements and unforeseen scenarios without significant rewrites. Gary McLean Hall describes Agile best practices, principles, and patterns for designing and writing code that can evolve more quickly and easily, with fewer errors, because it doesn’t impede change.

Now revised, updated, and expanded, Adaptive Code, Second Edition adds indispensable practical insights on Kanban, dependency inversion, and creating reusable abstractions. Drawing on over a decade of Agile consulting and development experience, McLean Hall has updated his best-seller with deeper coverage of unit testing, refactoring, pure dependency injection, and more. Master powerful new ways to:

- Write code that enables and complements Scrum, Kanban, or any other Agile framework
- Develop code that can survive major changes in requirements
- Plan for adaptability by using dependencies, layering, interfaces, and design patterns
- Perform unit testing and refactoring in tandem, gaining more value from both
- Use the “golden
master” technique to make legacy code adaptive • Build SOLID code with single-responsibility, open/closed, and Liskov substitution principles • Create smaller interfaces to support more-diverse client and architectural needs • Leverage dependency injection best practices to improve code adaptability • Apply dependency inversion with the Stairway pattern, and avoid related anti-patterns About You This book is for programmers of all skill levels seeking more-practical insight into design patterns, SOLID principles, unit testing, refactoring, and related topics. Most readers will have programmed in C#, Java, C++, or similar object-oriented languages, and will be familiar with core procedural programming techniques.

Lean Software Development: An Agile Toolkit Adapting agile practices to your development organization Uncovering and eradicating waste throughout the software development lifecycle Practical techniques for every development manager, project manager, and technical leader Lean software development: applying agile principles to your organization In Lean Software Development, Mary and Tom Poppendieck identify seven fundamental "lean" principles, adapt them for the world of software development, and show how they can serve as the foundation for agile development approaches that work. Along the way, they introduce 22 "thinking tools" that can help you customize the right agile practices for any environment. Better, cheaper, faster software development. You can have all three—if you adopt the same lean principles that have already revolutionized manufacturing, logistics and product development. Iterating towards excellence: software development as an exercise in discovery Managing uncertainty: "decide as late as possible" by building
change into the system. Compressing the value stream: rapid development, feedback, and improvement Empowering teams and individuals without compromising coordination Software with integrity: promoting coherence, usability, fitness, maintainability, and adaptability How to "see the whole"—even when your developers are scattered across multiple locations and contractors Simply put, Lean Software Development helps you refocus development on value, flow, and people—so you can achieve breakthrough quality, savings, speed, and business alignment. Agile software development approaches have had significant impact on industrial software development practices. Today, agile software development has penetrated to most IT companies across the globe, with an intention to increase quality, productivity, and profitability. Comprehensive knowledge is needed to understand the architectural challenges involved in adopting and using agile approaches and industrial practices to deal with the development of large, architecturally challenging systems in an agile way. Agile Software Architecture focuses on gaps in the requirements of applying architecture-centric approaches and principles of agile software development and demystifies the agile architecture paradox. Readers will learn how agile and architectural cultures can co-exist and support each other according to the context. Moreover, this book will also provide useful leads for future research in architecture and agile to bridge such gaps by developing appropriate approaches that incorporate architecturally sound practices in agile methods. Presents a consolidated view of the state-of-art and state-of-practice as well as the newest research findings Identifies gaps in the
requirements of applying architecture-centric approaches and principles of agile software development and demystifies the agile architecture paradox. Explains whether or not and how agile and architectural cultures can co-exist and support each other depending upon the context. Provides useful leads for future research in both architecture and agile to bridge such gaps by developing appropriate approaches, which incorporate architecturally sound practices in agile methods. The Object of Data Abstraction and Structures Using Java is the perfect book for your data structures course. It presents traditional data structures topics with a distinct object-oriented flavor that offers students useful approaches for data structure design and implementation. For courses in Object-Oriented Design, C++ Intermediate Programming, and Object-Oriented Programming. Written for software engineers “in the trenches,” this text focuses on the technology—the principles, patterns, and process—that help software engineers effectively manage increasingly complex operating systems and applications. There is also a strong emphasis on the people behind the technology. This text will prepare students for a career in software engineering and serve as an on-going education for software engineers. The Java Virtual Machine (JVM) is the underlying technology behind Java's most distinctive features including size, security and cross-platform delivery. This guide shows programmers how to write programs for the Java Virtual Machine. Software development continues to be an ever-evolving field as organizations require new and innovative programs that can be implemented to make processes more efficient, productive, and cost-effective. Agile
practices particularly have shown great benefits for improving the effectiveness of software development and its maintenance due to their ability to adapt to change. It is integral to remain up to date with the most emerging tactics and techniques involved in the development of new and innovative software. The Research Anthology on Agile Software, Software Development, and Testing is a comprehensive resource on the emerging trends of software development and testing. This text discusses the newest developments in agile software and its usage spanning multiple industries. Featuring a collection of insights from diverse authors, this research anthology offers international perspectives on agile software. Covering topics such as global software engineering, knowledge management, and product development, this comprehensive resource is valuable to software developers, software engineers, computer engineers, IT directors, students, managers, faculty, researchers, and academicians. For senior/graduate level courses on Object Oriented Design using C++, and the Booch (BC) – OOD book. A practical, problem-solving approach to the fundamental concepts of Object Oriented Design and their application using C++. This book is written for the "engineer in the trenches". It is a serious guide for practitioners of Object-Oriented design. The style is narrative, and accessible for the beginner, and yet the topics are covered in enough depth to be relevant to the consumate designer. The principles of OOD explained, one by one, and then demonstrated with numerous examples and case studies. Write maintainable, extensible, and durable software with modern C++. This book is a must for every developer, software architect, or team leader who is
interested in good C++ code, and thus also wants to save development costs. If you want to teach yourself about writing clean C++, Clean C++ is exactly what you need. It is written to help C++ developers of all skill levels and shows by example how to write understandable, flexible, maintainable, and efficient C++ code. Even if you are a seasoned C++ developer, there are nuggets and data points in this book that you will find useful in your work. If you don't take care with your code, you can produce a large, messy, and unmaintainable beast in any programming language. However, C++ projects in particular are prone to be messy and tend to slip into bad habits. Lots of C++ code that is written today looks as if it was written in the 1980s. It seems that C++ developers have been forgotten by those who preach Software Craftsmanship and Clean Code principles. The Web is full of bad, but apparently very fast and highly optimized C++ code examples, with cruel syntax that completely ignores elementary principles of good design and well-written code. This book will explain how to avoid this scenario and how to get the most out of your C++ code. You'll find your coding becomes more efficient and, importantly, more fun. What You'll Learn Gain sound principles and rules for clean coding in C++ Carry out test driven development (TDD) Discover C++ design patterns and idioms Apply these design patterns Who This Book Is For Any C++ developer and software engineer with an interest in producing better code. This title focuses on the most critical aspects of software development: building robust, bug free systems, meeting deadlines, and coming in under budget. It includes artifacts, anecdotes, and actual code from an enterprise-class XP project.
Using Patterns and Agile Development guides students through the software development process. By describing practical stories, explaining the design and programming process in detail, and using projects as a learning context, the text helps readers understand why a given technique is required and why techniques must be combined to overcome the challenges facing software developers. The presentation is pedagogically organized as a realistic development story in which customer requests require introducing new techniques to combat ever-increasing software complexity. After an overview and introduction of basic terminology, the book presents the core practices, concepts, tools, and analytic skills for designing flexible and reliable software, including test-driven development, refactoring, design patterns, test doubles, and responsibility driven and compositional design. It then provides a collection of design patterns leading to a thorough discussion of frameworks, exemplified by a graphical user interface framework (MiniDraw). The author also discusses the important topics of configuration management and systematic testing. In the last chapter, projects lead students to design and implement their own frameworks, resulting in a reliable and usable implementation of a large and complex software system complete with a graphical user interface. This text teaches how to design, program, and maintain flexible and reliable software. Installation guides, source code for the examples, exercises, and projects can be found on the author’s website. More and more Agile projects are seeking architectural roots as they struggle with complexity and scale - and they’re seeking lightweight ways to do it. Still seeking? In this book the authors help you to find your own path.
Taking cues from Lean development, they can help steer your project toward practices with longstanding track records. Up-front architecture? Sure. You can deliver an architecture as code that compiles and that concretely guides development without bogging it down in a mass of documents and guesses about the implementation. Documentation? Even a whiteboard diagram, or a CRC card, is documentation: the goal isn't to avoid documentation, but to document just the right things in just the right amount. Process? This all works within the frameworks of Scrum, XP, and other Agile approaches. You need to get value from your software project. You need it "free, now, and perfect." We can't get you there, but we can help you get to "cheaper, sooner, and better." This book leads you from the desire for value down to the specific activities that help good Agile projects deliver better software sooner, and at a lower cost. Using simple sketches and a few words, the author invites you to follow his path of learning and understanding from a half century of software development and from his engagement with Agile methods from their very beginning. The book describes software development, starting from our natural desire to get something of value. Each topic is described with a picture and a few paragraphs. You're invited to think about each topic; to take it in. You'll think about how each step into the process leads to the next. You'll begin to see why Agile methods ask for what they do, and you'll learn why a shallow implementation of Agile can lead to only limited improvement. This is not a detailed map, nor a step-by-step set of instructions for building the perfect project. There is no map or instructions that will do that for you. You need to build your own project,
making it a bit more perfect every day. To do that effectively, you need to build up an understanding of the whole process. This book points out the milestones on your journey of understanding the nature of software development done well. It takes you to a location, describes it briefly, and leaves you to explore and fill in your own understanding. What You Need: You'll need your Standard Issue Brain, a bit of curiosity, and a desire to build your own understanding rather than have someone else's detailed ideas poured into your head. Shipping imperfect software is like going into debt. When you incur debt, the illusion of doing things faster can lead to exponential growth in the cost of maintaining software. Software debt takes five major forms: technical, quality, configuration management, design, and platform experience. In today's rush to market, software debt is inevitable. And that's okay—if you're careful about the debt you incur, and if you quickly pay it back. In Managing Software Debt, leading Agile expert Chris Sterling shows how understanding software debt can help you move products to market faster, with a realistic plan for refactoring them based on experience. Writing for all Agile software professionals, Sterling explains why you're going into software debt whether you know it or not—and why the interest on that debt can bring projects to a standstill. Next, he thoroughly explains each form of software debt, showing how to plan for it intelligently and repay it successfully. You'll learn why accepting software debt is not the same as deliberate sloppiness, and you'll learn how to use the software debt concept to systematically improve architectural agility. Coverage includes Managing tensions between speed and perfection and
recognizing that you’ll inevitably ship some “not quite right” code Planning to minimize interest payments by paying debts quickly Building architectures that respond to change and help enterprises run more smoothly Incorporating emergent architecture concepts into daily activities, using Agile collaboration and refactoring techniques Delivering code and other software internals that reduce the friction of future change Using early, automated testing to move past the “break/fix” mentality Scripting and streamlining both deployment and rollback Implementing team configuration patterns and knowledge sharing approaches that make software debt easier to repay Clearing away technical impediments in existing architectures Using the YAGNI (“you ain’t gonna need it”) approach to strip away unnecessary complexity Using this book’s techniques, senior software leadership can deliver more business value; managers can organize and support development teams more effectively; and teams and team members can improve their performance throughout the development lifecycle. Head First Agile is a complete guide to learning real-world agile ideas, practices, principles. What will you learn from this book? In Head First Agile, you’ll learn all about the ideas behind agile and the straightforward practices that drive it. You'll take deep dives into Scrum, XP, Lean, and Kanban, the most common real-world agile approaches today. You'll learn how to use agile to help your teams plan better, work better together, write better code, and improve as a team—because agile not only leads to great results, but agile teams say they also have a much better time at work. Head First Agile will help you get agile into your brain and onto your team! Preparing for your PMI-ACP®
certification? This book also has everything you need to get certified, with 100% coverage of the PMI-ACP® exam. Luckily, the most effective way to prepare for the exam is to get agile into your brain—so instead of cramming, you're learning. Why does this book look so different? Based on the latest research in cognitive science and learning theory, Head First Agile uses a visually rich format to engage your mind, rather than a text-heavy approach that puts you to sleep. Why waste your time struggling with new concepts? This multi-sensory learning experience is designed for the way your brain really works.

Copyright code: 11328f870fe501f7411e743f0f9051cf