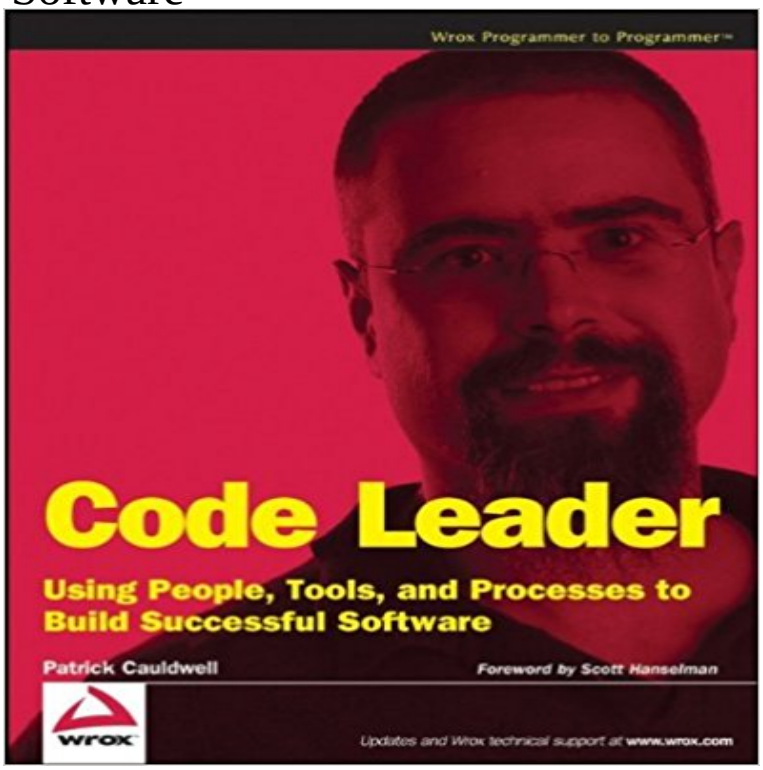
Code Leader: Using People, Tools, and Processes to Build Successful Software



This book is for the career developer who wants to take his or her skill set and/or project to the next level. If you are a professional software developer with 34 years of experience looking to bring a higher level of discipline to your project, or to learn the skills that will help you transition from software engineer to technical lead, then this book is for you. The topics covered in this book will help you focus on delivering software at a higher quality and lower cost. The book is about practical techniques and practices that will help you and your team realize those goals. This book is for the developer understands that the business of software is, first and foremost, business. Writing code is fun, but writing high-quality code on time and at the lowest possible cost is what makes a software project successful. A team lead or architect who wants to succeed must keep that in mind. Given that target audience, this book assumes a certain level of skill at reading code in one or more languages, and basic familiarity with building and testing software projects. It also assumes that you have at least a basic understanding of the software development lifecycle, and how requirements from customers become testable software projects. Who This Book Is Not For: This is not a book for the entry-level developer fresh out of college, or for those just getting started as professional coders. It isnt a book about writing code; its a book about how we write code together while keeping quality up and costs down. It is not for those who want to learn to write more efficient or literate code. There are plenty of other books available on those subjects, as mentioned previously. This is also not a book about project management or development methodology. All of the strategies and techniques presented here are just as applicable to waterfall projects as they are to those employing Agile

methodologies. While certain strategies such as Test-Driven Development and Continuous Integration have risen to popularity hand in hand with Agile development methodologies, there is no coupling between them. There are plenty of projects run using SCRUM that do not use TDD, and there are just as many waterfall projects that do. Philosophy versus Practicality: There are a lot of religious arguments in software development. Exceptions versus result codes, strongly typed versus dynamic languages, and where to put your curly braces are just a few examples. This book tried to steer clear of those arguments here. Most of the chapters in this book deal with practical steps that you as a developer can take to improve your skills and improve the state of your project. The author makes no claims that these practices represent the way to write software. They represent strategies that have worked well for the author and other developers that he have worked closely with. Philosophy certainly has its place in software development. Much of the current thinking in project management has been influenced by the Agile philosophy, for example. The next wave may be influenced by the Lean methodologies developed by Toyota for building automobiles. Because it represents a philosophy, the Lean process model can be applied to building software just as easily as to building cars. On the other hand, because they exist at the philosophical level, such methodologies can be difficult to conceptualize. The book tries to favor the practical over the philosophical, the concrete over the theoretical. This should be the kind of book that you can pick up, read one chapter of, and go away with some practical changes you can make to your software project that will make it better. That said, the first part of this book is entitled Philosophy because the strategies described in it represent ways of approaching a problem rather than a specific solution. There are just as many practical ways to do Test-Driven Development as there are ways to manage

a software project. You will have to pick the way that fits your chosen programming language, environment, and team structure. The book has tried to describe some tangible ways of realizing TDD, but it remains an abstract ideal rather than a one-size-fits-all technical solution. The same applies to Continuous Integration. There are numerous ways of thinking about and achieving a Continuous Integration solution, and this book presents only a few. Continuous Integration represents a way of thinking about your development process rather than a concrete or specific technique. The second and third parts represent more concrete process and construction techniques that can improve your code and your project. They focus on the pragmatic rather than the philosophical. Every Little Bit Helps: You do not have to sit down and read this book from cover to cover. While there are interrelationships between the chapters, each chapter can also stand on its own. If you know that you have a particular problem such as error handling with your current project, read that chapter and try to implement some of the suggestions in it. Dont feel that you have to overhaul your entire software project at once. The various techniques described in this book can all incrementally improve a project one at a time. If you are starting a brand new project and have an opportunity to define its structure, then by all means read the whole book and see how it influences the way you design your project. If you have to work within an existing project structure, you might have more success applying a few improvements at a time. In terms of personal career growth, the same applies. Every new technique you learn makes you a better developer, so take them one at a time as your schedule and projects allow. Examples: Most of the examples in this book are written in C#. However, the techniques described in this book apply just as well to any other modern programming language with a little translation. Even if you are unfamiliar with the inner workings or details of C# as a language, the examples are very small and

simple to understand. Again, this is not a book about how to write code, and the examples in it are all intended to illustrate a specific point, not to become a part of your software project in any literal sense. This book is organized into three sections, Philosophy, Process and Code Construction. The following is a short summary of what you will find in each section and chapter. Part I (Philosophy) contains chapters that focus on abstract ideas about how to approach a software project. Each chapter contains practical examples of how to realize those ideas. Chapter 1 (Buy, not Build) describes how to go about deciding which parts of your software project you need to write yourself and which parts you may be able to purchase or otherwise leverage from someplace else. In order to keep costs down and focus on your real competitive advantage, it is necessary to write only those parts of your application that you really need to. Chapter 2 (Test-Driven Development) examines the Test-Driven Development (or Test-Driven Design) philosophy and some practical ways of applying it to your development lifecycle to produce higher-quality code in less time. Chapter 3 (Continuous Integration) explores the Continuous Integration philosophy and how you can apply it to your project. CI involves automating your build and unit testing processes to give developers a shorter feedback cycle about changes that they make to the project. A shorter feedback cycle makes it easier for developers to work together as a team and at a higher level of productivity. The chapters in Part II (Process) explore processes and tools that you can use as a team to improve the quality of your source code and make it easier to understand and to maintain. Chapter 4 (Done Is Done) contains suggestions for defining what it means for a developer to finish a development task. Creating a done is done policy for your team can make it easier for developers to work together, and easier for developers and testers to work together. If everyone on your team follows the same

set of steps to complete each task, then development will be more predictable and of a higher quality. Chapter 5 (Testing) presents some concrete suggestions for how to create tests, how to run them, and how to organize them to make them easier to run, easier to measure, and more useful to developers and to testers. Included are sections on what code coverage means and how to measure it effectively, how to organize your tests by type, and how to automate your testing processes to get the most benefit from them. Chapter 6 (Source Control) explains techniques for using your source control system more effectively so that it is easier for developers to work together on the same project, and easier to correlate changes in source control with physical software binaries and with defect or issue reports in your tracking system. Chapter 7 (Static Analysis) examines what static analysis is, what information it can provide, and how it can improve the quality and maintainability of your projects. Part III (Code Construction) includes chapters on specific coding techniques that can improve the quality and maintainability of your software projects. Chapter 8 (Contract, Contract, Contract!) tackles programming by contract and how that can make your code easier for developers to understand and to use. Programming by contract can also make your application easier (and therefore less expensive) to maintain and support. Chapter 9 (Limiting Dependencies) focuses on techniques for limiting how dependent each part of your application is upon the others. Limiting dependencies can lead to software that is easier to make changes to and cheaper to maintain as well as easier to deploy and test. Chapter 10 (The Model-View-Presenter Model) offers a brief description of the MVP model and explains how following the MVP model will make your application easier to test. Chapter 11 (Tracing) describes ways to make the most of tracing in your application. Defining and following a solid tracing policy makes your application easier to debug and easier for your support

personnel and/or your customers to support. Chapter 12 (Error Handing) presents some techniques for handling errors in your code that if followed consistently make your application easier to debug and to support. Part IV (Putting It All Together) is simply a chapter that describes a day in the life of a developer who is following the guiding principles and using the techniques described in the rest of the book. Chapter 13 (Calculator Project: A Case Study) shows many of this books principles and techniques in actual use.

Before Committing, Update - Code Leader: Using People, Tools make logging more useful to support personnel is to - Selection from Code Leader: Using People, Tools, and Processes to Build Successful Software [Book] **code leader: using people, tools, and processes to build successful** Inversion of control means that rather than code - Selection from Code Leader: Using People, Tools, and Processes to Build Successful Software [Book] **Code Leader: Using People, Tools, and Processes to Build - Wrox** Code leader : using people, tools, and processes to build successful software / Patrick. View the summary of this work. Bookmark: http:///work/ If you are a professional software developer - Selection from Code Leader: Using People, Tools, and Processes to Build Successful Software [Book] **Code Leader: Using People, Tools, and Processes to Build** To test an MVP application, you create a - Selection from Code Leader: Using People, Tools, and Processes to Build Successful Software [Book] **Code Leader Using People, Tools, and Processes to Build** Code Leader: Using People, Tools, and Processes to Build Successful Software The topics covered in this book will help you focus on delivering software at a **Book review: Code Leader - Using People, Tools and Processes to** Cauldwell P. Code Leader: Using People, Tools, and Processes to Build Successful Software. pdf 3,20 . **Cauldwell P. Code Leader: Using People, Tools, and Processes to** Code Leader Using People, Tools, and Processes to Build Successful Software Patrick is a pragmatist with a purists knowledge. He has a deep understanding **Making Messages Actionable - Code Leader: Using People, Tools** 5 days ago Code Leader Using People, Tools, and Processes to Build Successful Software by Patrick Cauldwell Download Code Leader Using People, **Code Leader: Using People, Tools, and Processes to - Goodreads** L?s om Code Leader - Using People, Tools, and Processes to Build Successful Software. Udgivet af John Wiley & Sons Inc. Bogens ISBN er 9780470259245, **Testing MVP Applications - Code Leader: Using People, Tools, and** Code Leader Using People Tools and Processes to Build Successful Software, Buy Now in India Rs. 254, You SAVE 15%, Code Leader Using People Tools **Book review: Code Leader Using People, Tools and Processes to Code Leader: Using People, Tools, and Processes to Build - Wrox** Jul 12, 2008 Book review: Code Leader Using People, Tools and Processes to Build Successful Software. Code Leader Like Karl Seguins most excellent **Images for Code Leader: Using People, Tools, and Processes to Build Successful**

Software One of the most common reasons that builds fail is because - Selection from Code Leader: Using People, Tools, and Processes to Build Successful Software **Code Leader: Using People, Tools, and Processes - Google Books** Book review: Code Leader - Using People, Tools and Processes to Build Successful Software. July 12, 2008 Reading. Code Leader. Like Karl Seguins most **Buy Code Leader: Using People, Tools, and Processes to Build** Read and Download Ebook R.e.a.d Code Leader: Using People, Tools, And Processes To Build Successful Software PDF. R.e.a.d Code Leader: Using People,. **R.e.a.d Code Leader: Using People, Tools, and Processes to Build** Ellibs E-kirjakauppa - E-kirja: Code Leader: Using People, Tools, and Processes to Build Successful Software - Tekija: Cauldwell, Patrick - Hinta: 39,70 **Study Free Programming Book by Patrick Cauldwell Code Leader** Code Leader: Using People, Tools, and Processes to Build Successful Software [Patrick Cauldwell, Scott Hanselman] on . *FREE* shipping on **Code Leader: Using People, Tools, and Processes to Build** 2013?11?8? TITLE : Code Leader: Using People, Tools, and Processes to Build Successful Software (Programmer to Programmer) **Inversion of Control - Code Leader: Using People, Tools, and** Code Leader: Using People, Tools, and Processes to Build Successful Software. Patrick Cauldwell, Scott Hanselman (Foreword by). ISBN: 978-0-470-25924-5. **Code leader : using people, tools, and processes to build successful** Apr 30, 2008 Code Leader: Using People, Tools, and Processes to Build Successful Software. Front Cover. Patrick Cauldwell. John Wiley & Sons, Apr 30, **Code Leader af Patrick Cauldwell (Bog) - kob hos Saxo Code Leader: Using People, Tools, and Processes to Build** Code Leader: Using People, Tools, and Processes to Build Successful Software transition from software engineer to technical lead, then this book is for you. **Limiting Surface Area - Code Leader: Using People, Tools, and** Study Free Programming Book by Patrick Cauldwell Code Leader: Using People, Tools, And Processes To Build Successful Software. Posted: 01. 02. **Code Leader - OReilly Media** This is the forum to discuss the Wrox book Code Leader: Using People, Tools, and Processes to Build Successful Software by Patrick Cauldwell, Scott **Code Leader: Using People, Tools, and Processes to Build** Code Leader: Using People, Tools, and Processes to Build Successful Software. Patrick Cauldwell, Scott Hanselman (Foreword by). ISBN: 978-0-470-25924-5. **Code Leader Using People Tools and Processes to Build** Limiting Surface Area Remember the if you build it, they will come from Code Leader: Using People, Tools, and Processes to Build Successful Software [Book]