

Domain Specific Languages Martin Fowler

When somebody should go to the books stores, search establishment by shop, shelf by shelf, it is in fact problematic. This is why we provide the book compilations in this website. It will unquestionably ease you to see guide **Domain Specific Languages Martin Fowler** as you such as.

By searching the title, publisher, or authors of guide you essentially want, you can discover them rapidly. In the house, workplace, or perhaps in your method can be every best place within net connections. If you try to download and install the Domain Specific Languages Martin Fowler , it is unconditionally simple then, before currently we extend the connect to purchase and make bargains to download and install Domain Specific Languages Martin Fowler thus simple!

Domain-Driven Design Reference - Eric Evans 2014-09-22
Domain-Driven Design (DDD) is an approach to software development for complex businesses and other domains. DDD tackles that complexity by focusing the team's attention on knowledge of the domain, picking apart the most tricky, intricate problems with models, and shaping the software around those models. Easier said than done! The techniques of DDD help us approach this systematically. This reference gives a quick and authoritative summary of the key concepts of DDD. It is not meant as a learning introduction to the subject. Eric Evans' original book and a handful of others explain DDD in depth from different perspectives. On the other hand, we often need to scan a topic quickly or get the gist of a particular pattern. That is the purpose of this reference. It is complementary to the more discursive books. The starting point of this text was a set of excerpts from the original book by Eric Evans, *Domain-Driven-Design: Tackling Complexity in the Heart of Software*, 2004 - in particular, the pattern summaries, which were placed in the Creative Commons by Evans and the publisher, Pearson Education. In this reference, those original summaries have been updated and expanded with new content. The practice and

understanding of DDD has not stood still over the past decade, and Evans has taken this chance to document some important refinements. Some of the patterns and definitions have been edited or rewritten by Evans to clarify the original intent. Three patterns have been added, describing concepts whose usefulness and importance has emerged in the intervening years. Also, the sequence and grouping of the topics has been changed significantly to better emphasize the core principles. This is an up-to-date, quick reference to DDD.

Pragmatic Thinking and Learning - Andy Hunt 2008-10-28
Printed in full color. Software development happens in your head. Not in an editor, IDE, or designtool. You're well educated on how to work with software and hardware, but what about wetware--our own brains? Learning new skills and new technology is critical to your career, and it's all in your head. In this book by Andy Hunt, you'll learn how our brains are wired, and how to take advantage of your brain's architecture. You'll learn new tricks and tipsto learn more, faster, and retain more of what you learn. You need a pragmatic approach to thinking and learning. You need to Refactor Your Wetware. Programmers have to learn constantly; not just the stereotypical new technologies, but also the problem

domain of the application, the whims of the user community, the quirks of your teammates, the shifting sands of the industry, and the evolving characteristics of the project itself as it is built. We'll journey together through bits of cognitive and neuroscience, learning and behavioral theory. You'll see some surprising aspects of how our brains work, and how you can take advantage of the system to improve your own learning and thinking skills. In this book you'll learn how to: Use the Dreyfus Model of Skill Acquisition to become more expert Leverage the architecture of the brain to strengthen different thinking modes Avoid common "known bugs" in your mind Learn more deliberately and more effectively Manage knowledge more efficiently

xUnit Test Patterns - Gerard Meszaros 2007-05-21

Automated testing is a cornerstone of agile development. An effective testing strategy will deliver new functionality more aggressively, accelerate user feedback, and improve quality. However, for many developers, creating effective automated tests is a unique and unfamiliar challenge. *xUnit Test Patterns* is the definitive guide to writing automated tests using xUnit, the most popular unit testing framework in use today. Agile coach and test automation expert Gerard Meszaros describes 68 proven patterns for making tests easier to write, understand, and maintain. He then shows you how to make them more robust and repeatable--and far more cost-effective. Loaded with information, this book feels like three books in one. The first part is a detailed tutorial on test automation that covers everything from test strategy to in-depth test coding. The second part, a catalog of 18 frequently encountered "test smells," provides trouble-shooting guidelines to help you determine the root cause of problems and the most applicable patterns. The third part contains detailed descriptions of each pattern, including refactoring instructions illustrated by extensive code samples in multiple programming languages.

The Thoughtworks Anthology - Roy Singham 2008
ThoughtWorks is a well-known global consulting firm;

ThoughtWorkers are leaders in areas of design, architecture, SOA, testing, and agile methodologies. This collection of essays brings together contributions from well-known ThoughtWorkers such as Martin Fowler, along with other authors you may not know yet. While ThoughtWorks is perhaps best known for their work in the Agile community, this anthology confronts issues throughout the software development life cycle. From technology issues that transcend methodology, to issues of realizing business value from applications, you'll find it here.

Refactoring to Patterns - Joshua Kerievsky 2004-08-05

In 1994, *Design Patterns* changed the landscape of object-oriented development by introducing classic solutions to recurring design problems. In 1999, *Refactoring* revolutionized design by introducing an effective process for improving code. With the highly anticipated *Refactoring to Patterns*, Joshua Kerievsky has changed our approach to design by forever uniting patterns with the evolutionary process of refactoring. This book introduces the theory and practice of pattern-directed refactorings: sequences of low-level refactorings that allow designers to safely move designs to, towards, or away from pattern implementations. Using code from real-world projects, Kerievsky documents the thinking and steps underlying over two dozen pattern-based design transformations. Along the way he offers insights into pattern differences and how to implement patterns in the simplest possible ways. Coverage includes: A catalog of twenty-seven pattern-directed refactorings, featuring real-world code examples Descriptions of twelve design smells that indicate the need for this book's refactorings General information and new insights about patterns and refactoring Detailed implementation mechanics: how low-level refactorings are combined to implement high-level patterns Multiple ways to implement the same pattern--and when to use each Practical ways to get started even if you have little experience with patterns or refactoring *Refactoring to Patterns* reflects three years of refinement and the

insights of more than sixty software engineering thought leaders in the global patterns, refactoring, and agile development communities. Whether you're focused on legacy or "greenfield" development, this book will make you a better software designer by helping you learn how to make important design changes safely and effectively.

Refactoring - Paul Becker 1999

Refactoring is gaining momentum amongst the object oriented programming community. It can transform the internal dynamics of applications and has the capacity to transform bad code into good code. This book offers an introduction to refactoring.

Implementing Domain-driven Design - Vaughn Vernon 2013

Vaughn Vernon presents concrete and realistic domain-driven design (DDD) techniques through examples from familiar domains, such as a Scrum-based project management application that integrates with a collaboration suite and security provider. Each principle is backed up by realistic Java examples, and all content is tied together by a single case study of a company charged with delivering a set of advanced software systems with DDD.

Fowler - Martin Fowler 2012-03-09

The practice of enterprise application development has benefited from the emergence of many new enabling technologies. Multi-tiered object-oriented platforms, such as Java and .NET, have become commonplace. These new tools and technologies are capable of building powerful applications, but they are not easily implemented. Common failures in enterprise applications often occur because their developers do not understand the architectural lessons that experienced object developers have learned. Patterns of Enterprise Application Architecture is written in direct response to the stiff challenges that face enterprise application developers. The author, noted object-oriented designer Martin Fowler, noticed that despite changes in technology--from Smalltalk to CORBA to Java to .NET--the same basic design ideas can be adapted and applied to solve common problems. With the help of an expert group

of contributors, Martin distills over forty recurring solutions into patterns. The result is an indispensable handbook of solutions that are applicable to any enterprise application platform. This book is actually two books in one. The first section is a short tutorial on developing enterprise applications, which you can read from start to finish to understand the scope of the book's lessons. The next section, the bulk of the book, is a detailed reference to the patterns themselves. Each pattern provides usage and implementation information, as well as detailed code examples in Java or C#. The entire book is also richly illustrated with UML diagrams to further explain the concepts. Armed with this book, you will have the knowledge necessary to make important architectural decisions about building an enterprise application and the proven patterns for use when building them. The topics covered include · Dividing an enterprise application into layers · The major approaches to organizing business logic · An in-depth treatment of mapping between objects and relational databases · Using Model-View-Controller to organize a Web presentation · Handling concurrency for data that spans multiple transactions · Designing distributed object interfaces

Managing Software Debt - Chris Sterling 2010-12-10

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.

Fundamentals of Object-oriented Design in UML - Meilir Page-Jones 2000

Fundamentals of Object-Oriented Design in UML shows aspiring and experienced programmers alike how to apply design concepts, the UML, and the best practices in OO development to improve both their code and their success rates with object-based projects.

Clean Architecture - Robert C. Martin 2017-09-12
Practical Software Architecture Solutions from the Legendary Robert C. Martin ("Uncle Bob") By applying universal rules of software architecture, you can dramatically improve developer productivity throughout

the life of any software system. Now, building upon the success of his best-selling books Clean Code and The Clean Coder, legendary software craftsman Robert C. Martin ("Uncle Bob") reveals those rules and helps you apply them. Martin's Clean Architecture doesn't merely present options. Drawing on over a half-century of experience in software environments of every imaginable type, Martin tells you what choices to make and why they are critical to your success. As you've come to expect from Uncle Bob, this book is packed with direct, no-nonsense solutions for the real challenges you'll face—the ones that will make or break your projects. Learn what software architects need to achieve—and core disciplines and practices for achieving it Master essential software design principles for addressing function, component separation, and data management See how programming paradigms impose discipline by restricting what developers can do Understand what's critically important and what's merely a "detail" Implement optimal, high-level structures for web, database, thick-client, console, and embedded applications Define appropriate boundaries and layers, and organize components and services See why designs and architectures go wrong, and how to prevent (or fix) these failures Clean Architecture is essential reading for every current or aspiring software architect, systems analyst, system designer, and software manager—and for every programmer who must execute someone else's designs. Register your product for convenient access to downloads, updates, and/or corrections as they become available.

DSLs in Action - Debasish Ghosh 2010-11-30

Your success—and sanity—are closer at hand when you work at a higher level of abstraction, allowing your attention to be on the business problem rather than the details of the programming platform. Domain Specific Languages—"little languages" implemented on top of conventional programming languages—give you a way to do this because they model the domain of your business problem. DSLs in Action introduces the concepts and

definitions a developer needs to build high-quality domain specific languages. It provides a solid foundation to the usage as well as implementation aspects of a DSL, focusing on the necessity of applications speaking the language of the domain. After reading this book, a programmer will be able to design APIs that make better domain models. For experienced developers, the book addresses the intricacies of domain language design without the pain of writing parsers by hand. The book discusses DSL usage and implementations in the real world based on a suite of JVM languages like Java, Ruby, Scala, and Groovy. It contains code snippets that implement real world DSL designs and discusses the pros and cons of each implementation. Purchase of the print book comes with an offer of a free PDF, ePub, and Kindle eBook from Manning. Also available is all code from the book. What's Inside Tested, real-world examples How to find the right level of abstraction Using language features to build internal DSLs Designing parser/combinator-based little languages

NoSQL Distilled - Pramod J. Sadalage 2013

'NoSQL Distilled' is designed to provide you with enough background on how NoSQL databases work, so that you can choose the right data store without having to trawl the whole web to do it. It won't answer your questions definitively, but it should narrow down the range of options you have to consider.

Software Language Engineering - Mark van den Brand 2010-03-25

This book constitutes the thoroughly refereed post-conference proceedings of the Second International Conference on Software Language Engineering, SLE 2009, held in Denver, CO, USA, in October 2009. The 15 revised full papers and 6 revised short paper presented together with 2 tool demonstration papers were carefully reviewed and selected from 75 initial submissions. The papers are organized in topical sections on language and model evolution, variability and product lines, parsing, compilation, and demo, modularity in languages, and metamodeling and demo.

Fit for Developing Software - Rick Mugridge 2005-06-29
The Fit open source testing framework brings unprecedented agility to the entire development process. Fit for Developing Software shows you how to use Fit to clarify business rules, express them with concrete examples, and organize the examples into test tables that drive testing throughout the software lifecycle. Using a realistic case study, Rick Mugridge and Ward Cunningham--the creator of Fit--introduce each of Fit's underlying concepts and techniques, and explain how you can put Fit to work incrementally, with the lowest possible risk. Highlights include Integrating Fit into your development processes Using Fit to promote effective communication between businesspeople, testers, and developers Expressing business rules that define calculations, decisions, and business processes Connecting Fit tables to the system with "fixtures" that check whether tests are actually satisfied Constructing tests for code evolution, restructuring, and other changes to legacy systems Managing the quality and evolution of tests A companion Web site (<http://fit.c2.com/>) that offers additional resources and source code

Domain Specific Languages - Martin Fowler 2010

When carefully selected and used, Domain-Specific Languages (DSLs) may simplify complex code, promote effective communication with customers, improve productivity, and unclog development bottlenecks. In *Domain-Specific Languages*, noted software development expert Martin Fowler first provides the information software professionals need to decide if and when to utilize DSLs. Then, where DSLs prove suitable, Fowler presents effective techniques for building them, and guides software engineers in choosing the right approaches for their applications. This book's techniques may be utilized with most modern object-oriented languages; the author provides numerous examples in Java and C#, as well as selected examples in Ruby. Wherever possible, chapters are organized to be self-standing, and most reference topics are presented

in a familiar patterns format. Armed with this wide-ranging book, developers will have the knowledge they need to make important decisions about DSLs--and, where appropriate, gain the significant technical and business benefits they offer. The topics covered include: " How DSLs compare to frameworks and libraries, and when those alternatives are sufficient " Using parsers and parser generators, and parsing external DSLs " Understanding, comparing, and choosing DSL language constructs " Determining whether to use code generation, and comparing code generation strategies " Previewing new language workbench tools for creating DSLs.

Model-Driven Software Development - Markus Völter
2013-06-26

Model-Driven Software Development (MDS) is currently a highly regarded development paradigm among developers and researchers. With the advent of OMG's MDA and Microsoft's Software Factories, the MDS approach has moved to the centre of the programmer's attention, becoming the focus of conferences such as OOPSLA, JAOO and OOP. MDS is about using domain-specific languages to create models that express application structure or behaviour in an efficient and domain-specific way. These models are subsequently transformed into executable code by a sequence of model transformations. This practical guide for software architects and developers is peppered with practical examples and extensive case studies. International experts deliver: * A comprehensive overview of MDS and how it relates to industry standards such as MDA and Software Factories. * Technical details on meta modeling, DSL construction, model-to-model and model-to-code transformations, and software architecture. * Invaluable insight into the software development process, plus engineering issues such as versioning, testing and product line engineering. * Essential management knowledge covering economic and organizational topics, from a global perspective. Get started and benefit from some practical support along the way!

Analysis Patterns - Martin Fowler 1996-10-09

This innovative book recognizes the need within the

object-oriented community for a book that goes beyond the tools and techniques of the typical methodology book. In *Analysis Patterns: Reusable Object Models*, Martin Fowler focuses on the end result of object-oriented analysis and design--the models themselves. He shares with you his wealth of object modeling experience and his keen eye for identifying repeating problems and transforming them into reusable models. *Analysis Patterns* provides a catalogue of patterns that have emerged in a wide range of domains including trading, measurement, accounting and organizational relationships. Recognizing that conceptual patterns cannot exist in isolation, the author also presents a series of "support patterns" that discuss how to turn conceptual models into software that in turn fits into an architecture for a large information system. Included in each pattern is the reasoning behind their design, rules for when they should and should not be used, and tips for implementation. The examples presented in this book comprise a cookbook of useful models and insight into the skill of reuse that will improve analysis, modeling and implementation.

Domain-Specific Languages in Practice - Antonio Bucchiarone 2021-06-24

This book covers several topics related to domain-specific language (DSL) engineering in general and how they can be handled by means of the JetBrains Meta Programming System (MPS), an open source language workbench developed by JetBrains over the last 15 years. The book begins with an overview of the domain of language workbenches, which provides perspectives and motivations underpinning the creation of MPS. Moreover, technical details of the language underneath MPS together with the definition of the tool's main features are discussed. The remaining ten chapters are then organized in three parts, each dedicated to a specific aspect of the topic. Part I "MPS in Industrial Applications" deals with the challenges and inadequacies of general-purpose languages used in companies, as opposed to the reasons why DSLs are essential, together

with their benefits and efficiency, and summarizes lessons learnt by using MPS. Part II about "MPS in Research Projects" covers the benefits of text-based languages, the design and development of gamification applications, and research fields with generally low expertise in language engineering. Eventually, Part III focuses on "Teaching and Learning with MPS" by discussing the organization of both commercial and academic courses on MPS. MPS is used to implement languages for real-world use. Its distinguishing feature is projectional editing, which supports practically unlimited language extension and composition possibilities as well as a flexible mix of a wide range of textual, tabular, mathematical and graphical notations. The number and diversity of the presented use-cases demonstrate the strength and malleability of the DSLs defined using MPS. The selected contributions represent the current state of the art and practice in using JetBrains MPS to implement languages for real-world applications.

DSL Engineering - Markus Voelter 2013

The definitive resource on domain-specific languages: based on years of real-world experience, relying on modern language workbenches and full of examples. Domain-Specific Languages are programming languages specialized for a particular application domain. By incorporating knowledge about that domain, DSLs can lead to more concise and more analyzable programs, better code quality and increased development speed. This book provides a thorough introduction to DSL, relying on today's state of the art language workbenches. The book has four parts: introduction, DSL design, DSL implementation as well as the role of DSLs in various aspects of software engineering. Part I Introduction: This part introduces DSLs in general and discusses their advantages and drawbacks. It also defines important terms and concepts and introduces the case studies used in the most of the remainder of the book. Part II DSL Design: This part discusses the design of DSLs - independent of implementation techniques. It reviews

seven design dimensions, explains a number of reusable language paradigms and points out a number of process-related issues. Part III DSL Implementation: This part provides details about the implementation of DSLs with lots of code. It uses three state-of-the-art but quite different language workbenches: JetBrains MPS, Eclipse Xtext and TU Delft's Spoofox. Part IV DSLs and Software Engineering: This part discusses the use of DSLs for requirements, architecture, implementation and product line engineering, as well as their roles as a developer utility and for implementing business logic. The book is available as a printed version (the one you are looking at) and as a PDF. For details see the book's companion website at <http://dslbook.org>

Domain-Driven Design Quickly - Floyd Marinescu
2007-12-01

Domain Driven Design is a vision and approach for dealing with highly complex domains that is based on making the domain itself the main focus of the project, and maintaining a software model that reflects a deep understanding of the domain. This book is a short, quickly-readable summary and introduction to the fundamentals of DDD; it does not introduce any new concepts; it attempts to concisely summarize the essence of what DDD is, drawing mostly Eric Evans' original book, as well other sources since published such as Jimmy Nilsson's Applying Domain Driven Design, and various DDD discussion forums. The main topics covered in the book include: Building Domain Knowledge, The Ubiquitous Language, Model Driven Design, Refactoring Toward Deeper Insight, and Preserving Model Integrity. Also included is an interview with Eric Evans on Domain Driven Design today.

Service Design Patterns - Robert Daigneau 2011-10-25
Web services have been used for many years. In this time, developers and architects have encountered a number of recurring design challenges related to their usage, and have learned that certain service design approaches work better than others to solve certain problems. In Service Design Patterns, Rob Daigneau

codifies proven design solutions for web services that follow the REST architectural style or leverage the SOAP/WSDL specifications. This catalogue identifies the fundamental topics in web service design and lists the common design patterns for each topic. All patterns identify the context in which they may be used, explain the constituent design elements, and explore the relative strengths and trade-offs. Code examples are provided to help you better understand how the patterns work but are kept general so that you can see how the solutions may be applied to disparate technologies that will inevitably change in the years to come. This book will help readers answer the following questions: How do you create a web service API, what are the common API styles, and when should a particular style be used? How can clients and web services communicate, and what are the foundations for creating complex conversations in which multiple parties exchange data over extended periods of time? What are the options for implementing web service logic, and when should a particular approach be used? How can clients become less coupled to the underlying systems used by a service? How can information about a web service be discovered? How can generic functions like authentication, validation, caching, and logging be supported on the client or service? What changes to a service cause clients to break? What are the common ways to version a service? How can web services be designed to support the continuing evolution of business logic without forcing clients to constantly upgrade? This book is an invaluable resource for enterprise architects, solution architects, and developers who use web services to create enterprise IT applications, commercial or open source products, and Software as a Service (SaaS) products that leverage emerging Cloud platforms.

Language Implementation Patterns - Terence Parr
2009-12-31

Learn to build configuration file readers, data readers, model-driven code generators, source-to-source translators, source analyzers, and interpreters. You

don't need a background in computer science--ANTLR creator Terence Parr demystifies language implementation by breaking it down into the most common design patterns. Pattern by pattern, you'll learn the key skills you need to implement your own computer languages. Knowing how to create domain-specific languages (DSLs) can give you a huge productivity boost. Instead of writing code in a general-purpose programming language, you can first build a custom language tailored to make you efficient in a particular domain. The key is understanding the common patterns found across language implementations. Language Design Patterns identifies and condenses the most common design patterns, providing sample implementations of each. The pattern implementations use Java, but the patterns themselves are completely general. Some of the implementations use the well-known ANTLR parser generator, so readers will find this book an excellent source of ANTLR examples as well. But this book will benefit anyone interested in implementing languages, regardless of their tool of choice. Other language implementation books focus on compilers, which you rarely need in your daily life. Instead, Language Design Patterns shows you patterns you can use for all kinds of language applications. You'll learn to create configuration file readers, data readers, model-driven code generators, source-to-source translators, source analyzers, and interpreters. Each chapter groups related design patterns and, in each pattern, you'll get hands-on experience by building a complete sample implementation. By the time you finish the book, you'll know how to solve most common language implementation problems.

Implementing Domain-Specific Languages with Xtext and Xtend - Lorenzo Bettini 2016-08-31

Learn how to implement a DSL with Xtext and Xtend using easy-to-understand examples and best practices About This Book Leverage the latest features of Xtext and Xtend to develop a domain-specific language. Integrate Xtext with popular third party IDEs and get the best out of both worlds. Discover how to test a DSL

implementation and how to customize runtime and IDE aspects of the DSL Who This Book Is For This book is targeted at programmers and developers who want to create a domain-specific language with Xtext. They should have a basic familiarity with Eclipse and its functionality. Previous experience with compiler implementation can be helpful but is not necessary since this book will explain all the development stages of a DSL. What You Will Learn Write Xtext grammar for a DSL; Use Xtend as an alternative to Java to write cleaner, easier-to-read, and more maintainable code; Build your Xtext DSLs easily with Maven/Tycho and Gradle; Write a code generator and an interpreter for a DSL; Explore the Xtext scoping mechanism for symbol resolution; Test most aspects of the DSL implementation with JUnit; Understand best practices in DSL implementations with Xtext and Xtend; Develop your Xtext DSLs using Continuous Integration mechanisms; Use an Xtext editor in a web application In Detail Xtext is an open source Eclipse framework for implementing domain-specific languages together with IDE functionalities. It lets you implement languages really quickly; most of all, it covers all aspects of a complete language infrastructure, including the parser, code generator, interpreter, and more. This book will enable you to implement Domain Specific Languages (DSL) efficiently, together with their IDE tooling, with Xtext and Xtend. Opening with brief coverage of Xtext features involved in DSL implementation, including integration in an IDE, the book will then introduce you to Xtend as this language will be used in all the examples throughout the book. You will then explore the typical programming development workflow with Xtext when we modify the grammar of the DSL. Further, the Xtend programming language (a fully-featured Java-like language tightly integrated with Java) will be introduced. We then explain the main concepts of Xtext, such as validation, code generation, and customizations of runtime and UI aspects. You will have learned how to test a DSL implemented in Xtext with JUnit and will progress to

advanced concepts such as type checking and scoping. You will then integrate the typical Continuous Integration systems built in to Xtext DSLs and familiarize yourself with Xbase. By the end of the book, you will manually maintain the EMF model for an Xtext DSL and will see how an Xtext DSL can also be used in IntelliJ. Style and approach A step-by step-tutorial with illustrative examples that will let you master using Xtext and implementing DSLs with its custom language, Xtend. *Modern Java in Action* - Raoul-Gabriel Urma 2018-09-26 Summary Manning's bestselling Java 8 book has been revised for Java 9! In *Modern Java in Action*, you'll build on your existing Java language skills with the newest features and techniques. Purchase of the print book includes a free eBook in PDF, Kindle, and ePub formats from Manning Publications. About the Technology Modern applications take advantage of innovative designs, including microservices, reactive architectures, and streaming data. Modern Java features like lambdas, streams, and the long-awaited Java Module System make implementing these designs significantly easier. It's time to upgrade your skills and meet these challenges head on! About the Book *Modern Java in Action* connects new features of the Java language with their practical applications. Using crystal-clear examples and careful attention to detail, this book respects your time. It will help you expand your existing knowledge of core Java as you master modern additions like the Streams API and the Java Module System, explore new approaches to concurrency, and learn how functional concepts can help you write code that's easier to read and maintain. What's inside Thoroughly revised edition of Manning's bestselling *Java 8 in Action* New features in Java 8, Java 9, and beyond Streaming data and reactive programming The Java Module System About the Reader Written for developers familiar with core Java features. About the Author Raoul-Gabriel Urma is CEO of Cambridge Spark. Mario Fusco is a senior software engineer at Red Hat. Alan Mycroft is a University of Cambridge computer science professor; he cofounded the

Raspberry Pi Foundation. Table of Contents PART 1 - FUNDAMENTALS Java 8, 9, 10, and 11: what's happening? Passing code with behavior parameterization Lambda expressions PART 2 - FUNCTIONAL-STYLE DATA PROCESSING WITH STREAMS Introducing streams Working with streams Collecting data with streams Parallel data processing and performance PART 3 - EFFECTIVE PROGRAMMING WITH STREAMS AND LAMBDA Collection API enhancements Refactoring, testing, and debugging Domain-specific languages using lambdas PART 4 - EVERYDAY JAVA Using Optional as a better alternative to null New Date and Time API Default methods The Java Module System PART 5 - ENHANCED JAVA CONCURRENCY Concepts behind CompletableFuture and reactive programming CompletableFuture: composable asynchronous programming Reactive programming PART 6 - FUNCTIONAL PROGRAMMING AND FUTURE JAVA EVOLUTION Thinking functionally Functional programming techniques Blending OOP and FP: Comparing Java and Scala Conclusions and where next for Java *Software Languages* - Ralf Lämmel 2018-05-17 This book identifies, defines and illustrates the fundamental concepts and engineering techniques relevant to applications of software languages in software development. It presents software languages primarily from a software engineering perspective, i.e., it addresses how to parse, analyze, transform, generate, format, and otherwise process software artifacts in different software languages, as they appear in software development. To this end, it covers a wide range of software languages - most notably programming languages, domain-specific languages, modeling languages, exchange formats, and specifically also language definition languages. Further, different languages are leveraged to illustrate software language engineering concepts and techniques. The functional programming language Haskell dominates the book, while the mainstream programming languages Python and Java are additionally used for illustration. By doing this, the book collects and organizes scattered knowledge from software language engineering, focusing on application areas such as

software analysis (software reverse engineering), software transformation (software re-engineering), software composition (modularity), and domain-specific languages. It is designed as a textbook for independent study as well as for bachelor's (advanced level) or master's university courses in Computer Science. An additional website provides complementary material, for example, lecture slides and videos. This book is a valuable resource for anyone wanting to understand the fundamental concepts and important engineering principles underlying software languages, allowing them to acquire much of the operational intelligence needed for dealing with software languages in software development practice. This is an important skill set for software engineers, as languages are increasingly permeating software development.

Generative and Transformational Techniques in Software Engineering IV - Ralf Lämmel 2013-01-03

This tutorial volume includes revised and extended lecture notes of six long tutorials, five short tutorials, and one peer-reviewed participant contribution held at the 4th International Summer School on Generative and Transformational Techniques in Software Engineering, GTTSE 2011. The school presents the state of the art in software language engineering and generative and transformational techniques in software engineering with coverage of foundations, methods, tools, and case studies.

.NET Design Patterns - Praseed Pai 2017-01-31

Explore the world of .NET design patterns and bring the benefits that the right patterns can offer to your toolkit today About This Book Dive into the powerful fundamentals of .NET framework for software development The code is explained piece by piece and the application of the pattern is also showcased. This fast-paced guide shows you how to implement the patterns into your existing applications Who This Book Is For This book is for those with familiarity with .NET development who would like to take their skills to the next level and be in the driver's seat when it comes to modern development

techniques. Basic object-oriented C# programming experience and an elementary familiarity with the .NET framework library is required. What You Will Learn Put patterns and pattern catalogs into the right perspective Apply patterns for software development under C#/.NET Use GoF and other patterns in real-life development scenarios Be able to enrich your design vocabulary and well articulate your design thoughts Leverage object/functional programming by mixing OOP and FP Understand the reactive programming model using Rx and RxJs Writing compositional code using C# LINQ constructs Be able to implement concurrent/parallel programming techniques using idioms under .NET Avoiding pitfalls when creating compositional, readable, and maintainable code using imperative, functional, and reactive code. In Detail Knowing about design patterns enables developers to improve their code base, promoting code reuse and making their design more robust. This book focuses on the practical aspects of programming in .NET. You will learn about some of the relevant design patterns (and their application) that are most widely used. We start with classic object-oriented programming (OOP) techniques, evaluate parallel programming and concurrency models, enhance implementations by mixing OOP and functional programming, and finally to the reactive programming model where functional programming and OOP are used in synergy to write better code. Throughout this book, we'll show you how to deal with architecture/design techniques, GoF patterns, relevant patterns from other catalogs, functional programming, and reactive programming techniques. After reading this book, you will be able to convincingly leverage these design patterns (factory pattern, builder pattern, prototype pattern, adapter pattern, facade pattern, decorator pattern, observer pattern and so on) for your programs. You will also be able to write fluid functional code in .NET that would leverage concurrency and parallelism! Style and approach This tutorial-based book takes a step-by-step approach. It covers the major patterns and explains them in a detailed manner along

with code examples.

Domain-specific Languages - Martin Fowler 2011
Martin Fowler's breakthrough practitioner-oriented book on Domain Specific Languages - will do for DSLs what Fowler did for refactoring! * *Fowler's highly anticipated introduction to DSLs: a category-defining book by one of the software world's most influential authors. *Two books in one: a concise narrative that introduces DSLs, and a larger reference that shows how to plan and develop them. *Helps software professionals reduce the cost and complexity of building DSLs - so they can take full advantage of them. Domain Specific Languages (DSLs) offer immense promise for software engineers who need better, faster ways to solve problems of specific types, or in specific areas or industries. DSLs have been around for several years, and have begun to grow in popularity. Now, Martin Fowler - one of the world's most influential software engineering authors - has written the first practitioner-oriented book about them. Fowler's legendary book, Refactoring, made software refactoring a crucial tool for software engineers worldwide; this book will do the same for DSLs. Fowler has designed Domain Specific Languages as two books in one. The first --a narrative designed to be read from 'cover to cover' - offers a concise introduction to DSLs, how they are implemented, and what are useful for. Next, Fowler thoroughly introduces today's most effective techniques for building DSLs. Fowler covers both 'external' and 'internal' DSLs, a well as alternative computational models, code generation, common parser topics, and much more. He provides extensive Java and C# examples throughout, as well as selected Ruby examples for concepts that can best be explained using a dynamic language. Together, both sections enable readers to make wellinformed choices about whether to use a DSL in their work, and which techniques to employ in order to build DSLs more quickly and cost-effectively.

Design It! - Michael Keeling 2017-10-18

Don't engineer by coincidence-design it like you mean

it! Filled with practical techniques, Design It! is the perfect introduction to software architecture for programmers who are ready to grow their design skills. Lead your team as a software architect, ask the right stakeholders the right questions, explore design options, and help your team implement a system that promotes the right -ilities. Share your design decisions, facilitate collaborative design workshops that are fast, effective, and fun-and develop more awesome software! With dozens of design methods, examples, and practical know-how, Design It! shows you how to become a software architect. Walk through the core concepts every architect must know, discover how to apply them, and learn a variety of skills that will make you a better programmer, leader, and designer. Uncover the big ideas behind software architecture and gain confidence working on projects big and small. Plan, design, implement, and evaluate software architectures and collaborate with your team, stakeholders, and other architects. Identify the right stakeholders and understand their needs, dig for architecturally significant requirements, write amazing quality attribute scenarios, and make confident decisions. Choose technologies based on their architectural impact, facilitate architecture-centric design workshops, and evaluate architectures using lightweight, effective methods. Write lean architecture descriptions people love to read. Run an architecture design studio, implement the architecture you've designed, and grow your team's architectural knowledge. Good design requires good communication. Talk about your software architecture with stakeholders using whiteboards, documents, and code, and apply architecture-focused design methods in your day-to-day practice. Hands-on exercises, real-world scenarios, and practical team-based decision-making tools will get everyone on board and give you the experience you need to become a confident software architect.

Domain Modeling Made Functional - Scott Wlaschin
2018-01-25

You want increased customer satisfaction, faster development cycles, and less wasted work. Domain-driven design (DDD) combined with functional programming is the innovative combo that will get you there. In this pragmatic, down-to-earth guide, you'll see how applying the core principles of functional programming can result in software designs that model real-world requirements both elegantly and concisely - often more so than an object-oriented approach. Practical examples in the open-source F# functional language, and examples from familiar business domains, show you how to apply these techniques to build software that is business-focused, flexible, and high quality. Domain-driven design is a well-established approach to designing software that ensures that domain experts and developers work together effectively to create high-quality software. This book is the first to combine DDD with techniques from statically typed functional programming. This book is perfect for newcomers to DDD or functional programming - all the techniques you need will be introduced and explained. Model a complex domain accurately using the F# type system, creating compilable code that is also readable documentation---ensuring that the code and design never get out of sync. Encode business rules in the design so that you have "compile-time unit tests," and eliminate many potential bugs by making illegal states unrepresentable. Assemble a series of small, testable functions into a complete use case, and compose these individual scenarios into a large-scale design. Discover why the combination of functional programming and DDD leads naturally to service-oriented and hexagonal architectures. Finally, create a functional domain model that works with traditional databases, NoSQL, and event stores, and safely expose your domain via a website or API. Solve real problems by focusing on real-world requirements for your software. What You Need: The code in this book is designed to be run interactively on Windows, Mac and Linux. You will need a recent version of F# (4.0 or greater), and the appropriate .NET runtime for your platform. Full

installation instructions for all platforms at fsharp.org.

Continuous Integration - Paul M. Duvall 2007-06-29

For any software developer who has spent days in "integration hell," cobbling together myriad software components, *Continuous Integration: Improving Software Quality and Reducing Risk* illustrates how to transform integration from a necessary evil into an everyday part of the development process. The key, as the authors show, is to integrate regularly and often using continuous integration (CI) practices and techniques. The authors first examine the concept of CI and its practices from the ground up and then move on to explore other effective processes performed by CI systems, such as database integration, testing, inspection, deployment, and feedback. Through more than forty CI-related practices using application examples in different languages, readers learn that CI leads to more rapid software development, produces deployable software at every step in the development lifecycle, and reduces the time between defect introduction and detection, saving time and lowering costs. With successful implementation of CI, developers reduce risks and repetitive manual processes, and teams receive better project visibility. The book covers How to make integration a "non-event" on your software development projects How to reduce the amount of repetitive processes you perform when building your software Practices and techniques for using CI effectively with your teams Reducing the risks of late defect discovery, low-quality software, lack of visibility, and lack of deployable software Assessments of different CI servers and related tools on the market The book's companion Web site, www.integratebutton.com, provides updates and code examples.

Domain-Specific Modeling - Steven Kelly 2008-04-11

"[The authors] are pioneers. . . . Few in our industry have their breadth of knowledge and experience." -From the Foreword by Dave Thomas, Bedarra Labs *Domain-Specific Modeling (DSM)* is the latest approach to

software development, promising to greatly increase the speed and ease of software creation. Early adopters of DSM have been enjoying productivity increases of 500-1000% in production for over a decade. This book introduces DSM and offers examples from various fields to illustrate to experienced developers how DSM can improve software development in their teams. Two authorities in the field explain what DSM is, why it works, and how to successfully create and use a DSM solution to improve productivity and quality. Divided into four parts, the book covers: background and motivation; fundamentals; in-depth examples; and creating DSM solutions. There is an emphasis throughout the book on practical guidelines for implementing DSM, including how to identify the necessary language constructs, how to generate full code from models, and how to provide tool support for a new DSM language. The example cases described in the book are available the book's Website, www.dsmbook.com, along with, an evaluation copy of the MetaEdit+ tool (for Windows, Mac OS X, and Linux), which allows readers to examine and try out the modeling languages and code generators. *Domain-Specific Modeling* is an essential reference for lead developers, software engineers, architects, methodologists, and technical managers who want to learn how to create a DSM solution and successfully put it into practice.

Domain-Specific Languages - Martin Fowler 2010-09-23

When carefully selected and used, *Domain-Specific Languages (DSLs)* may simplify complex code, promote effective communication with customers, improve productivity, and unclog development bottlenecks. In *Domain-Specific Languages*, noted software development expert Martin Fowler first provides the information software professionals need to decide if and when to utilize DSLs. Then, where DSLs prove suitable, Fowler presents effective techniques for building them, and guides software engineers in choosing the right approaches for their applications. This book's techniques may be utilized with most modern object-

oriented languages; the author provides numerous examples in Java and C#, as well as selected examples in Ruby. Wherever possible, chapters are organized to be self-standing, and most reference topics are presented in a familiar patterns format. Armed with this wide-ranging book, developers will have the knowledge they need to make important decisions about DSLs—and, where appropriate, gain the significant technical and business benefits they offer. The topics covered include: How DSLs compare to frameworks and libraries, and when those alternatives are sufficient Using parsers and parser generators, and parsing external DSLs Understanding, comparing, and choosing DSL language constructs Determining whether to use code generation, and comparing code generation strategies Previewing new language workbench tools for creating DSLs

Refactoring - Jay Fields 2009-10

The Definitive Refactoring Guide, Fully Revamped for Ruby With refactoring, programmers can transform even the most chaotic software into well-designed systems that are far easier to evolve and maintain. What's more, they can do it one step at a time, through a series of simple, proven steps. Now, there's an authoritative and extensively updated version of Martin Fowler's classic refactoring book that utilizes Ruby examples and idioms throughout—not code adapted from Java or any other environment. The authors introduce a detailed catalog of more than 70 proven Ruby refactorings, with specific guidance on when to apply each of them, step-by-step instructions for using them, and example code illustrating how they work. Many of the authors' refactorings use powerful Ruby-specific features, and all code samples are available for download. Leveraging Fowler's original concepts, the authors show how to perform refactoring in a controlled, efficient, incremental manner, so you methodically improve your code's structure without introducing new bugs. Whatever your role in writing or maintaining Ruby code, this book will be an indispensable resource. This book will help you

- * Understand the core principles of refactoring and

the reasons for doing it

- * Recognize "bad smells" in your Ruby code
- * Rework bad designs into well-designed code, one step at a time
- * Build tests to make sure your refactorings work properly
- * Understand the challenges of refactoring and how they can be overcome
- * Compose methods to package code properly
- * Move features between objects to place responsibilities where they fit best
- * Organize data to make it easier to work with
- * Simplify conditional expressions and make more effective use of polymorphism
- * Create interfaces that are easier to understand and use
- * Generalize more effectively
- * Perform larger refactorings that transform entire software systems and may take months or years
- * Successfully refactor Ruby on Rails code

Domain-driven Design - Eric Evans 2004
Describes ways to incorporate domain modeling into software development.

UML Distilled - Martin Fowler 2018-08-30

More than 300,000 developers have benefited from past editions of *UML Distilled*. This third edition is the best resource for quick, no-nonsense insights into understanding and using UML 2.0 and prior versions of the UML. Some readers will want to quickly get up to speed with the UML 2.0 and learn the essentials of the UML. Others will use this book as a handy, quick reference to the most common parts of the UML. The author delivers on both of these promises in a short, concise, and focused presentation. This book describes all the major UML diagram types, what they're used for, and the basic notation involved in creating and deciphering them. These diagrams include class, sequence, object, package, deployment, use case, state machine, activity, communication, composite structure, component, interaction overview, and timing diagrams. The examples are clear and the explanations cut to the fundamental design logic. Includes a quick reference to the most useful parts of the UML notation and a useful summary of diagram types that were added to the UML 2.0. If you are like most developers, you don't have time to keep up with all the new innovations in software

engineering. This new edition of Fowler's classic work gets you acquainted with some of the best thinking about efficient object-oriented software design using the UML—in a convenient format that will be essential to anyone who designs software professionally.

Pattern-Oriented Software Architecture, A Pattern Language for Distributed Computing - Frank Buschmann
2007-04-04

The eagerly awaited Pattern-Oriented Software Architecture (POSA) Volume 4 is about a pattern language for distributed computing. The authors will guide you through the best practices and introduce you to key areas of building distributed software systems. POSA 4 connects many stand-alone patterns, pattern collections and pattern languages from the existing body of literature found in the POSA series. Such patterns relate to and are useful for distributed computing to a single language. The panel of experts provides you with a consistent and coherent holistic view on the craft of building distributed systems. Includes a foreword by Martin Fowler A must read for practitioners who want practical advice to develop a comprehensive language integrating patterns from key literature.

Building Maintainable Software, Java Edition - Joost Visser 2016-01-28

Have you ever felt frustrated working with someone else's code? Difficult-to-maintain source code is a big

problem in software development today, leading to costly delays and defects. Be part of the solution. With this practical book, you'll learn 10 easy-to-follow guidelines for delivering Java software that's easy to maintain and adapt. These guidelines have been derived from analyzing hundreds of real-world systems. Written by consultants from the Software Improvement Group (SIG), this book provides clear and concise explanations, with advice for turning the guidelines into practice. Examples for this edition are written in Java, while our companion C# book provides workable examples in that language. Write short units of code: limit the length of methods and constructors Write simple units of code: limit the number of branch points per method Write code once, rather than risk copying buggy code Keep unit interfaces small by extracting parameters into objects Separate concerns to avoid building large classes Couple architecture components loosely Balance the number and size of top-level components in your code Keep your codebase as small as possible Automate tests for your codebase Write clean code, avoiding "code smells" that indicate deeper problems

Clean Code - Robert C. Martin 2009

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.