

Domain Specific Languages Martin Fowler

“For software developers of all experience levels looking to improve their results, and design and implement domain-driven enterprise applications consistently with the best current state of professional practice, Implementing Domain-Driven Design will impart a treasure trove of knowledge hard won within the DDD and enterprise application architecture communities over the last couple decades.”
–Randy Stafford, Architect At-Large, Oracle Coherence Product Development
“This book is a must-read for anybody looking to put DDD into practice.” –Udi Dahan, Founder of NServiceBus

Implementing Domain-Driven Design presents a top-down approach to understanding domain-driven design (DDD) in a way that fluently connects strategic patterns to fundamental tactical programming tools. Vaughn Vernon couples guided approaches to implementation with modern architectures, highlighting the importance and value of focusing on the business domain while balancing technical considerations. Building on Eric Evans’ seminal book, Domain-Driven Design, the author presents practical DDD techniques through examples from familiar domains. Each principle is backed up by realistic Java examples—all applicable to C# developers—and all content is tied together by a single case study: the delivery of a large-scale Scrum-based SaaS system for a multitenant environment. The author takes you far beyond “DDD-lite” approaches that embrace DDD solely as a technical toolset, and shows you how to fully leverage DDD’s “strategic design patterns” using Bounded Context, Context Maps, and the Ubiquitous Language. Using these techniques and examples, you can reduce time to market and improve quality, as you build software that is more flexible, more scalable, and more tightly aligned to business goals. Coverage includes Getting started the right way with DDD, so you can rapidly gain value from it Using DDD within diverse architectures, including Hexagonal, SOA, REST, CQRS, Event-Driven, and Fabric/Grid-Based Appropriately designing and applying Entities—and learning when to use Value Objects instead Mastering DDD’s powerful new Domain Events technique Designing Repositories for ORM, NoSQL, and other databases

This book discusses how model-based approaches can improve the daily practice of software professionals. This is known as Model-Driven Software Engineering (MDSE) or, simply, Model-Driven Engineering (MDE). MDSE practices have proved to increase efficiency and effectiveness in software development, as demonstrated by various quantitative and qualitative studies. MDSE adoption in the software industry is foreseen to grow exponentially in the near future, e.g., due to the convergence of software development and business analysis. The aim of this book is to provide you with an agile and flexible tool to introduce you to the MDSE world, thus allowing you to quickly understand its basic principles and techniques and to choose the right set of MDSE instruments for your needs so that you can start to benefit from MDSE right away. The book is organized into two main parts. The first part discusses the foundations of

MDSE in terms of basic concepts (i.e., models and transformations), driving principles, application scenarios, and current standards, like the well-known MDA initiative proposed by OMG (Object Management Group) as well as the practices on how to integrate MDSE in existing development processes. The second part deals with the technical aspects of MDSE, spanning from the basics on when and how to build a domain-specific modeling language, to the description of Model-to-Text and Model-to-Model transformations, and the tools that support the management of MDSE projects. The second edition of the book features: a set of completely new topics, including: full example of the creation of a new modeling language (IFML), discussion of modeling issues and approaches in specific domains, like business process modeling, user interaction modeling, and enterprise architecture complete revision of examples, figures, and text, for improving readability, understandability, and coherence better formulation of definitions, dependencies between concepts and ideas addition of a complete index of book content In addition to the contents of the book, more resources are provided on the book's website <http://www.mdse-book.com>, including the examples presented in the book.

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.

Take advantage of Sinatra, the Ruby-based web application library and domain-

specific language used by Heroku, GitHub, Apple, Engine Yard, and other prominent organizations. With this concise book, you will quickly gain working knowledge of Sinatra and its minimalist approach to building both standalone and modular web applications. Sinatra serves as a lightweight wrapper around Rack middleware, with syntax that maps closely to functions exposed by HTTP verbs, which makes it ideal for web services and APIs. If you have experience building applications with Ruby, you'll quickly learn language fundamentals and see under-the-hood techniques, with the help of several practical examples. Then you'll get hands-on experience with Sinatra by building your own blog engine. Learn Sinatra's core concepts, and get started by building a simple application. Create views, manage sessions, and work with Sinatra route definitions. Become familiar with the language's internals, and take a closer look at Rack. Use different subclass methods for building flexible and robust architectures. Put Sinatra to work: build a blog that takes advantage of service hooks provided by the GitHub API.

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.

Covers methods and tools for designing and implementing the right components for a system family and automating component assembly. Methods are applicable to all commercial development, from the level of classes and procedures to developing families of large systems. Coverage encompasses domain engineering, feature modeling, and generic programming. Includes case studies in the programming, business, and scientific computing domains.

Czarnecki is a research and consultant in the private sector. Eisenecker teaches computer science at the University of Applied Sciences Kaiserslautern at Zweibrücken. Annotation copyrighted by Book News, Inc., Portland, OR

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

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.

Model-Driven Software Development (MDSD) is currently a highly regarded development paradigm among developers and researchers. With the advent of OMG's MDA and Microsoft's Software Factories, the MDSD 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!

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.

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

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 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.

Even bad code can function. But if code isn't clean, it can bring a development organization to its knees. Every year, countless hours and significant resources are lost because of poorly written code. But it doesn't have to be that way. Noted software expert Robert C. Martin presents a revolutionary paradigm with *Clean Code: A Handbook of Agile Software Craftsmanship*. 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. What kind of work will you be doing? You'll be reading code—lots of code. And you will be challenged to think about what's right about that code, and what's wrong with it. More importantly, you will be challenged to reassess your professional values and your commitment to your craft. *Clean Code* is divided into three parts. The first describes the principles, patterns, and practices of writing clean code. The second part consists of several case studies of increasing complexity. Each case study is an exercise in cleaning up code—of transforming a code base that has some problems into one that is sound and efficient. The third part is the payoff: a single chapter containing a list of heuristics and "smells" gathered while creating the case studies. The result is a knowledge base that describes the way we think when we write, read, and clean code. Readers will come away from this book 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

This book is a must for any developer, software engineer, project manager, team lead, or systems analyst with an interest in producing better code.

An example-oriented approach to develop custom domain-specific languages. If you've already developed a few Clojure applications and wish to expand your knowledge on Clojure or domain-specific languages in general, then this book is for you. If you're an absolute Clojure beginner, then you may only find the detailed examples of the core Clojure components of value. If you've developed

DSLs in other languages, this Lisp and Java-based book might surprise you with the power of Clojure.

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.

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.

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.

A complete practitioner's catalog of proven domain services design solutions that can help any organization leverage SOA's full benefits * *Provides a vocabulary of proven SOA design solutions, with concrete examples and code that is easy for architects to adapt and implement.

*By Rob Daigneau, one of the industry's leading experts in complex systems integration.

*Helps architects and IT leaders accurately set stakeholder expectations for major SOA initiatives. Service-oriented architectures are typically called upon to deliver two general categories of services: enterprise services and domain services. Enterprise services are essentially composite services that typically leverage technologies such as message-oriented middleware. Domain services are the building blocks these composites depend upon. Each service category is best served by a distinct set of design solutions. This is the first book to systematically identify and explain best practice patterns for domain services. Rob Daigneau expands upon the Service Layer concept (covered expertly by Fowler in *Patterns of Enterprise Application Architecture*) domain services can be used with *Enterprise Integration Patterns* (made famous by Hohpe and Woolf). Daigneau begins by reviewing SOA concepts, illuminating the distinctions between enterprise and domain services, and identifying key relationships between domain services and other pattern groups. Next, he introduces each essential pattern for creating and delivering domain services, providing a vocabulary of design solutions that architects and other IT professionals can implement by referencing and adapting the concrete examples he supplies.

Using Agile methods, you can bring far greater innovation, value, and quality to any data warehousing (DW), business intelligence (BI), or analytics project. However, conventional Agile methods must be carefully adapted to address the unique characteristics of DW/BI projects. In *Agile Analytics*, Agile pioneer Ken Collier shows how to do just that. Collier introduces platform-agnostic Agile solutions for integrating infrastructures consisting of diverse operational, legacy, and specialty systems that mix commercial and custom code. Using working examples, he

shows how to manage analytics development teams with widely diverse skill sets and how to support enormous and fast-growing data volumes. Collier's techniques offer optimal value whether your projects involve "back-end" data management, "front-end" business analysis, or both. Part I focuses on Agile project management techniques and delivery team coordination, introducing core practices that shape the way your Agile DW/BI project community can collaborate toward success. Part II presents technical methods for enabling continuous delivery of business value at production-quality levels, including evolving superior designs; test-driven DW development; version control; and project automation. Collier brings together proven solutions you can apply right now--whether you're an IT decision-maker, data warehouse professional, database administrator, business intelligence specialist, or database developer. With his help, you can mitigate project risk, improve business alignment, achieve better results--and have fun along the way.

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>

The second instance of the international summer school on Generative and Transformational Techniques in Software Engineering (GTTSE 2007) was held in Braga, Portugal, during July 2–7, 2007. This volume contains an augmented selection of the material presented at the school, including full tutorials, short tutorials, and contributions to the participants workshop. The GTTSE summer school series brings together PhD students, lecturers, technology presenters, as well as other researchers and practitioners who are interested in the generation and the transformation of programs, data, models, metamodels, documentation, and entire software systems. This concerns many areas of software engineering: software reverse and re-engineering, model-driven engineering, automated software engineering, generic language technology, to name a few. These areas differ with regard to the specific sorts of metamodels (or grammars, schemas, formats etc.) that underlie the involved artifacts, and with regard to the specific techniques that are employed for the generation and the transformation of the artifacts. The first instance of the school was held in 2005 and its proceedings appeared as volume 4143 in the LNCS series.

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. D is a programming language built to help programmers address the challenges of modern

software development. It does so by fostering modules interconnected through precise interfaces, a federation of tightly integrated programming paradigms, language-enforced thread isolation, modular type safety, an efficient memory model, and more. The D Programming Language is an authoritative and comprehensive introduction to D. Reflecting the author's signature style, the writing is casual and conversational, but never at the expense of focus and precision. It covers all aspects of the language (such as expressions, statements, types, functions, contracts, and modules), but it is much more than an enumeration of features. Inside the book you will find In-depth explanations, with idiomatic examples, for all language features How feature groups support major programming paradigms Rationale and best-use advice for each major feature Discussion of cross-cutting issues, such as error handling, contract programming, and concurrency Tables, figures, and "cheat sheets" that serve as a handy quick reference for day-to-day problem solving with D Written for the working programmer, The D Programming Language not only introduces the D language—it presents a compendium of good practices and idioms to help both your coding with D and your coding in general. Users 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.

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.

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

What others in the trenches say about *The Pragmatic Programmer*... “The cool thing about this book is that it’s great for keeping the programming process fresh. The book helps you to continue to grow and clearly comes from people who have been there.” —Kent Beck, author of *Extreme Programming Explained: Embrace Change* “I found this book to be a great mix of solid advice and wonderful analogies!” —Martin Fowler, author of *Refactoring* and *UML Distilled* “I would buy a copy, read it twice, then tell all my colleagues to run out and grab a copy. This is a book I would never loan because I would worry about it being lost.” —Kevin Ruland, Management Science, MSG-Logistics “The wisdom and practical experience of the authors is obvious. The topics presented are relevant and useful.... By far its greatest strength for me has been the outstanding analogies—tracer bullets, broken windows, and the fabulous helicopter-based explanation of the need for orthogonality, especially in a crisis situation. I have little doubt that this book will eventually become an excellent source of useful information for journeymen programmers and expert mentors alike.” —John Lakos, author of *Large-Scale C++ Software Design* “This is the sort of book I will buy a dozen copies of when it comes out so I can give it to my clients.” —Eric Vought, Software Engineer “Most modern books on software development fail to cover the basics of what makes a great software developer, instead spending their time on syntax or technology where in reality the greatest leverage possible for any software team is in having talented developers who really know their craft well. An excellent book.” —Pete McBreen, Independent Consultant “Since reading this book, I have implemented many of the practical suggestions and tips it contains. Across the board, they have saved my company time and money while helping me get my job done quicker! This should be a desktop reference for everyone who works with code for a living.” —Jared Richardson, Senior Software Developer, iRenaissance, Inc. “I would like to see this issued to every new employee at my company....” —Chris Cleeland, Senior Software Engineer, Object Computing, Inc. “If I’m putting together a project, it’s the authors of this book that I want. . . . And failing that I’d settle for people who’ve read their book.” —Ward Cunningham

Straight from the programming trenches, *The Pragmatic Programmer* cuts through the increasing specialization and technicalities of modern software development to examine the core process--taking a requirement and producing working, maintainable code that delights its users. It

covers topics ranging from personal responsibility and career development to architectural techniques for keeping your code flexible and easy to adapt and reuse. Read this book, and you'll learn how to Fight software rot; Avoid the trap of duplicating knowledge; Write flexible, dynamic, and adaptable code; Avoid programming by coincidence; Bullet-proof your code with contracts, assertions, and exceptions; Capture real requirements; Test ruthlessly and effectively; Delight your users; Build teams of pragmatic programmers; and Make your developments more precise with automation. Written as a series of self-contained sections and filled with entertaining anecdotes, thoughtful examples, and interesting analogies, *The Pragmatic Programmer* illustrates the best practices and major pitfalls of many different aspects of software development. Whether you're a new coder, an experienced programmer, or a manager responsible for software projects, use these lessons daily, and you'll quickly see improvements in personal productivity, accuracy, and job satisfaction. You'll learn skills and develop habits and attitudes that form the foundation for long-term success in your career. You'll become a Pragmatic Programmer.

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.

The need to handle increasingly larger data volumes is one factor driving the adoption of a new class of nonrelational "NoSQL" databases. Advocates of

NoSQL databases claim they can be used to build systems that are more performant, scale better, and are easier to program. NoSQL Distilled is a concise but thorough introduction to this rapidly emerging technology. Pramod J. Sadalage and Martin Fowler explain how NoSQL databases work and the ways that they may be a superior alternative to a traditional RDBMS. The authors provide a fast-paced guide to the concepts you need to know in order to evaluate whether NoSQL databases are right for your needs and, if so, which technologies you should explore further. The first part of the book concentrates on core concepts, including schemaless data models, aggregates, new distribution models, the CAP theorem, and map-reduce. In the second part, the authors explore architectural and design issues associated with implementing NoSQL. They also present realistic use cases that demonstrate NoSQL databases at work and feature representative examples using Riak, MongoDB, Cassandra, and Neo4j. In addition, by drawing on Pramod Sadalage's pioneering work, NoSQL Distilled shows how to implement evolutionary design with schema migration: an essential technique for applying NoSQL databases. The book concludes by describing how NoSQL is ushering in a new age of Polyglot Persistence, where multiple data-storage worlds coexist, and architects can choose the technology best optimized for each type of data access.

This is a detailed summary of research on design rationale providing researchers in software engineering with an excellent overview of the subject. Professional software engineers will find many examples, resources and incentives to enhance their ability to make decisions during all phases of the software lifecycle. Software engineering is still primarily a human-based activity and rationale management is concerned with making design and development decisions explicit to all stakeholders involved.

Describes ways to incorporate domain modeling into software development. 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, as 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 well-informed 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.

This book is a practical tutorial, walking the reader through examples of building DSLs with Groovy covering meta-programming with Groovy. Some complex concepts are covered in the book but we go through these in a clear and methodical way so that readers will gain a good working knowledge of the techniques involved. This book is for Java software developers who have an interest in building domain scripting into their Java applications. No knowledge of Groovy is required, although it will be helpful. The book does not teach Groovy, but quickly introduces the basic ideas of Groovy. An experienced Java developer should have no problems with these and move quickly on to the more involved aspects of creating DSLs with Groovy. No experience of creating a DSL is required. The book should also be useful for experienced Groovy developers who have so far only used Groovy DSLs such as Groovy builders and would like to start building their own Groovy-based DSLs.

Programmers run into parsing problems all the time. Whether it's a data format like JSON, a network protocol like SMTP, a server configuration file for Apache, a PostScript/PDF file, or a simple spreadsheet macro language--ANTLR v4 and this book will demystify the process. ANTLR v4 has been rewritten from scratch to make it easier than ever to build parsers and the language applications built on top. This completely rewritten new edition of the bestselling Definitive ANTLR Reference shows you how to take advantage of these new features. Build your own languages with ANTLR v4, using ANTLR's new advanced parsing technology. In this book, you'll learn how ANTLR automatically builds a data structure representing the input (parse tree) and generates code that can walk the tree (visitor). You can use that combination to implement data readers, language interpreters, and translators. You'll start by learning how to identify grammar patterns in language reference manuals and then slowly start building increasingly complex grammars. Next, you'll build applications based upon those grammars by walking the automatically generated parse trees. Then you'll tackle some nasty language problems by parsing files containing more than one language (such as XML, Java, and Javadoc). You'll also see how to take absolute control over parsing by embedding Java actions into the grammar. You'll learn directly from well-known parsing expert Terence Parr, the ANTLR creator and project lead. You'll master ANTLR grammar construction and learn how to build language tools using the built-in parse tree visitor mechanism. The book teaches using real-world examples and shows you how to use ANTLR to build such things as a data file reader, a JSON to XML translator, an R parser, and a Java class->interface extractor. This book is your ticket to becoming a parsing guru! What You Need: ANTLR 4.0 and above. Java development tools. Ant build system optional(needed for building ANTLR from source)

Use Roslyn as a service to write powerful extensions and tools and use them in Visual Studio to improve code quality and maintain your source code more effectively. About This Book Use Roslyn extensions and tools in Visual Studio to enforce "house rules" on code and fix security and performance vulnerabilities in your code. Write Roslyn

extensions using the Roslyn service API to help developers enforce conventions and design idioms. Improve developer productivity by using Roslyn-based agile development features in Visual Studio, such as live unit testing, C# interactive and scripting. Contribute to the C# language and compiler tool chain to analyze and edit code. Who This Book Is For .NET Developers and architects, who are interested in taking full advantage of the Roslyn based extensions and tools to improve the development processes, will find this book useful. Roslyn contributors, i.e. the producers and C# community developers, will also find this book useful What You Will Learn Write extensions to analyze source code and report warnings and errors. Edit C# source code to fix compiler/analyzer diagnostics or refactor source code. Improve code maintenance and readability by using analyzers and code fixes. Catch security and performance issues by using PUMA scan analyzers and FxCop analyzers. Perform Live Unit tests in Visual Studio. Use C# interactive and scripting in Visual Studio. Design a new C# language feature and implement various compiler phases for a new language feature. Write command line tools to analyze and edit C# code. In Detail Open-sourcing the C# and Visual Basic compilers is one of the most appreciated things by the .NET community, especially as it exposes rich code analysis APIs to analyze and edit code. If you want to use Roslyn API to write powerful extensions and contribute to the C# developer tool chain, then this book is for you. Additionally, if you are just a .NET developer and want to use this rich Roslyn-based functionality in Visual Studio to improve the code quality and maintenance of your code base, then this book is also for you. This book is divided into the following broad modules: Writing and consuming analyzers/fixers (Chapters 1 - 5): You will learn to write different categories of Roslyn analyzers and harness and configure analyzers in your C# projects to catch quality, security and performance issues. Moving ahead, you will learn how to improve code maintenance and readability by using code fixes and refactorings and also learn how to write them. Using Roslyn-based agile development features (Chapters 6 and 7): You will learn how to improve developer productivity in Visual Studio by using features such as live unit testing, C# interactive and scripting. Contributing to the C# language and compiler tool chain (Chapters 8 - 10): You will see the power of open-sourcing the Roslyn compiler via the simple steps this book provides; thus, you will contribute a completely new C# language feature and implement it in the Roslyn compiler codebase. Finally, you will write simple command line tools based on the Roslyn service API to analyze and edit C# code. Style and approach This book takes a recipe-based approach, teaching you how to perform various hacks with the Compiler API in your hands.

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

Domain-Specific Languages Pearson Education

This book is Open Access under a CC BY licence. This book constitutes the proceedings of the 22nd International Conference on Fundamental Approaches to Software Engineering, FASE 2019, which took place in Prague, Czech Republic in April 2019, held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019. The 24 papers presented in this volume were carefully reviewed and selected from 94 submissions. The papers are organized in topical sections named: software verification; model-driven development and model transformation; software evolution and requirements engineering; specification, design, and implementation of particular classes of systems; and software testing.

You can test just about anything with Cucumber. We certainly have, and in Cucumber Recipes we'll show you how to apply our hard-won field experience to your own projects. Once you've mastered the basics, this book will show you how to get the most out of Cucumber--from specific situations to advanced test-writing advice. With over forty practical recipes, you'll test desktop, web, mobile, and server applications across a variety of platforms. This book gives you tools that you can use today to automate any system that you encounter, and do it well. The Cucumber Book showed you how your team can work together to write executable specifications--documents that tell a clear story and also happen to be working test code. We'll arm you with ready-rolled solutions to real-world problems: your tests will run faster, read more clearly, and work in any environment. Our first tips will help you fit Cucumber into your workflow. Powerful filters will tame tables full of test data, transforming them into the format your application needs. Custom output formatters will generate reports for any occasion. Continuous Integration servers will run your Cucumber tests every time the code changes. Next, you'll find recipes tailored to the platform you're running on. Ever wanted to know how to test a Grails app from Cucumber? Need to put a Windows program through its paces? How about a mobile app running on Android or iOS? We'll show you how to do all of these. Throughout the book, you'll see how to make Cucumber sing as you interoperate with different platforms, languages, and environments. From embedded circuits to Python and PHP web apps, Cucumber has something for you. What You Need: You'll need basic working knowledge of Cucumber and Ruby. Individual recipes may have additional requirements; for example, a recipe on Windows automation might pull in an open source GUI driver. We've written the recipes for compatibility with Ruby 1.9.3 and 1.8.7, plus Cucumber 1.1.4. Other versions may work as well, but these are the ones we test with.

[Copyright: 438884c6ac25523e1e78f8bc1fbbc11d](#)