Free Domain Specific Languages By Martin Fowler

Creating your own domain-specific languages (DSLs) is both challenging and exhilarating. DSLs give users a way to interact with your applications more effectively, and Kotlin is a fantastic language to serve as a host for internal DSLs, because it greatly reduces the pain and effort of design and development. But implementing DSLs on top of Kotlin requires understanding the key strengths of the language and knowing how to apply them appropriately. Learn to avoid the pitfalls and leverage the language while creating your own elegant, fluent, concise, and robust DSLs using Kotlin. Internal DSLs remove the burdens of implementing a full blown language compiler. The host language quickly becomes your ally to creating DSLs, but the syntax you can choose for your DSLs is limited to what the host language allows. You can work around the limitations by tactfully bending the rules and exploiting the language capabilities. Learn the power of Kotlin and ways to design with it, in the context of crafting internal DSLs Start by learning ways to exploit the flexibilities of Kotlin to make your DSLs fluent, expressive, and concise. Then pick up techniques to extend the language with domain specific properties and functions. Quickly move ahead to tie your DSL snippets into the runtime environment and context of execution of

your applications. Design to prevent any non-sensical syntax in your DSL that may otherwise be valid in the host language. Finally, learn techniques to gracefully handle errors. Practice using the multiple examples that are included in each chapter. Fire up your editor and follow along each example to become proficient in designing and implementing your own internal DSLs using Kotlin. What You Need: Kotlin version 1.3 or later and your favorite Kotlin IDE or code editor.

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.

Generic Tools, Specific Languages (GTSL) is an approach for developing tools and applications in a way that supports easier and more meaningful adaptation to specific domains. To achieve this goal, GTSL generalizes programming language IDEs to domains traditionally not addressed by languages and IDEs. At its core, GTSL represents applications as documents/programs/models expressed with

suitable languages. Application functionality is provided through an IDE that is aware of the languages and their semantics. The IDE provides editing support, and also directly integrates domain-specific analyses and execution services. Applications and their languages can be adapted to increasingly specific domains using language engineering; this includes developing incremental extensions to existing languages or creating additional, tightly integrated languages. Language workbenches act as the foundation on which such applications are built. mbeddr is an extensible set of integrated languages for embedded software development built using the Generic Tools, Specific Languages approach.

"[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.

Get up to speed on Scala, the JVM language that offers all the benefits of a modern object model, functional programming, and an advanced type system. Packed with code examples, this comprehensive book shows you how to be productive with the language and ecosystem right away, and explains why Scala is ideal for today's highly scalable, data-centric applications that support concurrency and distribution. This second edition covers recent language features, with new chapters on pattern matching, comprehensions, and advanced functional programming. You'll also learn about Scala's command-line tools,

third-party tools, libraries, and language-aware plugins for editors and IDEs. This book is ideal for beginning and advanced Scala developers alike. Program faster with Scala's succinct and flexible syntax Dive into basic and advanced functional programming (FP) techniques Build killer big-data apps, using Scala's functional combinators Use traits for mixin composition and pattern matching for data extraction Learn the sophisticated type system that combines FP and object-oriented programming concepts Explore Scala-specific concurrency tools, including Akka Understand how to develop rich domain-specific languages Learn good design techniques for building scalable and robust Scala applications

This book covers research into the most important practices in product line organization. Contributors offer experience-based knowledge on the domain and application engineering, the modeling and management of variability, and the design and use of tools to support the management of product line-related knowledge.

This book constitutes the proceedings of the 22nd International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2021, which was held virtually during January 17-19, 2021. The conference was planned to take place in Copenhagen, Denmark, but changed to an online event due to the COVID-19 pandemic. The 23 papers presented in this volume were carefully

reviewed from 48 submissions. VMCAI provides a forum for researchers working on verification, model checking, and abstract interpretation and facilitates interaction, cross-fertilization, and advancement of hybrid methods that combine these and related areas. The papers presented in this volume were organized in the following topical sections: hyperproperties and infinite-state systems; concurrent and distributed systems; checking; synthesis and repair; applications; and decision procedures. .

"This book presents current research on all aspects of domain-specific language for scholars and practitioners in the software engineering fields, providing new results and answers to open problems in DSL research"--

Domain engineering is a set of activities intended to develop, maintain, and manage the creation and evolution of an area of knowledge suitable for processing by a range of software systems. It is of considerable practical significance, as it provides methods and techniques that help reduce time-to-market, development costs, and project risks on one hand, and helps improve system quality and performance on a consistent basis on the other. In this book, the editors present a collection of invited chapters from various fields related to domain engineering. The individual chapters present state-of-the-art research and are organized in three parts. The first part focuses on results that deal with domain engineering in software product lines. The second part describes how domain-specific languages are used to support the construction and deployment of

domains. Finally, the third part presents contributions dealing with domain engineering within the field of conceptual modeling. All chapters utilize a similar terminology, which will help readers to understand and relate to the chapters content. The book will be especially rewarding for researchers and students of software engineering methodologies in general and of domain engineering and its related fields in particular, as it contains the most comprehensive and up-to-date information on this topic. Annotation Contains abstracts of 440 papers presented at the January 2001 conference in nine major tracks. The session topics are collaboration systems and technology, complex systems, decision technologies for management, digital documents, emerging technologies, information technology in health care, Internet and the digital economy, organizational systems and technology, and software technology. Subject headings within the sessions include asynchronous learning networks, intelligent systems in traffic and transportation, e- government, telemedicine, web engineering, community informatics, trends in outsourcing of information systems, mobile computing and wireless networks, and domain-specific languages for software engineering. No subject index. The CD-ROM contains the complete papers in Abode Acrobat format. The disc is of a hybrid structure that allows access from PCs, Macintosh, and UNIX. Annotation copyrighted by Book News, Inc., Portland, OR.
This tutorial book presents an augmented selection of material presented at the International Summer School on Generative and Transformational Techniques in

Software Engineering, GTTSE 2005. The book comprises 7 tutorial lectures presented together with 8 technology presentations and 6 contributions to the participants workshop. The tutorials combine foundations, methods, examples, and tool support. Subjects covered include feature-oriented programming and the AHEAD tool suite; program transformation with reflection and aspect-oriented programming, and more. Domain-Specific LanguagesPearson Education 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.

Dijkstra once wrote that computer science is no more about computers than astronomy is about telescopes. Despite the many incredible advances in c- puter science from times that predate practical mechanical computing, there is still a myriad of fundamental questions in understanding the interface between computers and the rest of the world. Why is it still hard to mechanize many tasks that seem to be fundamentally routine, even as we see ever-increasing - pacity for raw mechanical computing? The disciplined study of domain-speci?c languages (DSLs) is an emerging area in computer science, and is one which has the potential to revolutionize the ?eld, and bring us closer to answering this question. DSLs are formalisms that have four general characteristics. – They relate to a well-de?ned domain of discourse, be it controlling tra?c lights or space ships. – They have well-de?ned notation, such as the ones that exist for prescribing

music, dance routines, or strategy in a football game. – The informal or intuitive meaning of the notation is clear. This can easily be overlooked, especially since intuitive meaning can be expressed by many di?erent notations that may be received very di?erently by users. – The formal meaning is clear and mechanizable, as is, hopefully, the case for the instructions we give to our bank or to a merchant online. This volume was published in honor of Stefania Gnesi's 65th birthday. The Festschrift volume contains 32 papers written by close collaborators and friends of Stefania and was presented to her on October 8, 2019 one-day colloquium held in Porto, Portugal, The Festschrift consists of eight sections, seven of which reflect the main research areas to which Stefania has contributed. Following a survey of Stefania's legacy in research and a homage by her thesis supervisor, these seven sections are ordered according to Stefania's life cycle in research, from software engineering to formal methods and tools, and back: Software Engineering; Formal Methods and Tools; Requirements Engineering; Natural Language Processing; Software Product Lines; Formal Verification; and Applications.
A general-purpose language like C# is designed to handle all programming tasks. By contrast, the structure and syntax of a Domain-Specific Language are designed to match a particular applications area. A DSL is designed for readability and easy programming of repeating problems. Using the innovative Boo language, it's a breeze to create a DSL for your application domain that works on .NET and does not sacrifice

performance. DSLs in Boo shows you how to design, extend, and evolve DSLs for .NET by focusing on approaches and patterns. You learn to define an app in terms that match the domain, and to use Boo to build DSLs that generate efficient executables. And you won't deal with the awkward XML-laden syntax many DSLs require. The book concentrates on writing internal (textual) DSLs that allow easy extensibility of the application and framework. And if you don't know Boo, don't worry-you'll learn right here all the techniques you need. 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. Software engineering has advanced rapidly in recent years in parallel with the complexity and scale of software systems. New requirements in software systems yield innovative approaches that are developed either through introducing new paradigms or extending the capabilities of well-established approaches. Modern Software Engineering Concepts and Practices: Advanced Approaches provides emerging theoretical approaches and their practices. This book includes case studies and real-world practices and presents a range of advanced approaches to reflect various perspectives in the discipline.

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 Program generation holds the promise of helping to bridge the gap between application-level problem solutions and efficient implementations at the level of today's source programs as written in C or Java. Thus, program generation can substantially contribute to reducing production cost and time-to-market in future software production, while improving the quality and stability of the product. This book is about domain-specific program generation; it is the outcome of a Dagstuhl seminar on the topic held in March

2003. After an introductory preface by the volume editors, the 18 carefully reviewed revised full papers presented are organized into topical sections on - surveys of domain-specific programming technologies - domain-specific programming languages - tool support for program generation - domain-specific techniques for program optimization Gain an accelerated introduction to domain-specific languages in R, including coverage of regular expressions. This compact, in-depth book shows you how DSLs are programming languages specialized for a particular purpose, as opposed to general purpose programming languages. Along the way, you'll learn to specify tasks you want to do in a precise way and achieve programming goals within a domain-specific context. Domain-Specific Languages in R includes examples of DSLs including large data sets or matrix multiplication; pattern matching DSLs for application in computer vision; and DSLs for continuous time Markov chains and their applications in data science. After reading and using this book, you'll understand how to write DSLs in R and have skills you can extrapolate to other programming languages. What You'll Learn Program with domain-specific languages using R Discover the components of DSLs Carry out large matrix expressions and multiplications Implement metaprogramming with DSLs Parse and manipulate expressions Who This Book Is For Those with prior programming experience. R knowledge is helpful but not required.

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 methodically 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.

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.

Advances in Computers, Volume 116, presents innovations in computer hardware, software, theory, design, and applications, with this updated volume including new chapters on Teaching Graduate Students How to Review Research Articles and How to Respond to Reviewer Comments, ALGATOR - An Automatic Algorithm Evaluation System, Graph Grammar Induction, Asymmetric Windows in Digital Signal Processing, Intelligent Agents in Games: Review With an Open-Source Tool, Using Clickstream Data to Enhance Reverse Engineering of Web Applications, and more. Contains novel subject matter that is relevant to computer science Includes the expertise of contributing authors Presents an easy to comprehend writing style

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 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 Spoofax. 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 your are looking at) and as a PDF. For details see the book's companion website at http: //dslbook.org

This book details the conceptual foundations, design and implementation of the

domain-specific language (DSL) development system DjDSL. DjDSL facilitates design-decision-making on and implementation of reusable DSL and DSL-product lines, and represents the state-of-the-art in language-based and composition-based DSL development. As such, it unites elements at the crossroads between software-language engineering, model-driven software engineering, and feature-oriented software engineering. The book is divided into six chapters. Chapter 1 ("DSL as Variable Software") explains the notion of DSL as variable software in greater detail and introduces readers to the idea of software-product line engineering for DSL-based software systems. Chapter 2 ("Variability Support in DSL Development") sheds light on a number of interrelated dimensions of DSL variability: variable development processes, variable design-decisions, and variability-implementation techniques for DSL. The three subsequent chapters are devoted to the key conceptual and technical contributions of DjDSL: Chapter 3 ("Variable Language Models") explains how to design and implement the abstract syntax of a DSL in a variable manner. Chapter 4 ("Variable Context Conditions") then provides the means to refine an abstract syntax (language model) by using composable context conditions (invariants). Next, Chapter 5 ("Variable Textual Syntaxes") details solutions to implementing variable textual syntaxes for different types of DSL. In closing,

Chapter 6 ("A Story of a DSL Family") shows how to develop a mixed DSL in a step-by-step manner, demonstrating how the previously introduced techniques can be employed in an advanced example of developing a DSL family. The book is intended for readers interested in language-oriented as well as model-driven software development, including software-engineering researchers and advanced software developers alike. An understanding of software-engineering basics (architecture, design, implementation, testing) and software patterns is essential. Readers should especially be familiar with the basics of object-oriented modelling (UML, MOF, Ecore) and programming (e.g., Java).

For more than 40 years, Computerworld has been the leading source of technology news and information for IT influencers worldwide. Computerworld's award-winning Web site (Computerworld.com), twice-monthly publication, focused conference series and custom research form the hub of the world's largest global IT media network.

Software practitioners are rapidly discovering the immense value of Domain-Specific Languages (DSLs) in solving problems within clearly definable problem domains. Developers are applying DSLs to improve productivity and quality in a wide range of areas, such as finance, combat simulation, macro scripting, image generation, and more. But until now, there have been few practical resources that

explain how DSLs work and how to construct them for optimal use. Software Language Engineering fills that need. Written by expert DSL consultant Anneke Kleppe, this is the first comprehensive guide to successful DSL design. Kleppe systematically introduces and explains every ingredient of an effective language specification, including its description of concepts, how those concepts are denoted, and what those concepts mean in relation to the problem domain. Kleppe carefully illuminates good design strategy, showing how to maximize the flexibility of the languages you create. She also demonstrates powerful techniques for creating new DSLs that cooperate well with general-purpose languages and leverage their power. Completely tool-independent, this book can serve as the primary resource for readers using Microsoft DSL tools, the Eclipse Modeling Framework, openArchitectureWare, or any other DSL toolset. It contains multiple examples, an illustrative running case study, and insights and background information drawn from Kleppe's leading-edge work as a DSL researcher. Specific topics covered include Discovering the types of problems that DSLs can solve, and when to use them Comparing DSLs with general-purpose languages, frameworks, APIs, and other approaches Understanding the roles and tools available to language users and engineers Creating each component of a DSL specification Modeling both concrete and abstract syntax

Understanding and describing language semantics Defining textual and visual languages based on object-oriented metamodeling and graph transformations Using metamodels and associated tools to generate grammars Integrating object-oriented modeling with graph theory Building code generators for new languages Supporting multilanguage models and programs This book provides software engineers with all the guidance they need to create DSLs that solve real problems more rapidly, and with higher-quality code.
Written for developers who need to create user-facing DSLs, Domain-Specific Languages Made Easy unlocks clear and practical methods to create DSLs with easy-to-use interfaces. Imagine if your non-technical clients could safely produce software without the need for anyone to manually write code. Domain-specific languages are purpose-built programming interfaces that make that possible—no programming experience required. Written for developers who need to create user-facing DSLs, Domain-Specific Languages Made Easy unlocks clear and practical methods to create DSLs with easy-to-use interfaces. Author Meinte Boersma lays out an iterative process for creating languages accessible to domain experts such as operations specialists, data analysts, and financial experts. You'll start with an overview of software language engineering before diving into the unique projectional editing paradigm that makes it easy to produce

DSLs for business. Purchase of the print book includes a free eBook in PDF, Kindle, and ePub formats from Manning Publications.

This book constitutes the refereed proceedings of the 8th International Conference, MLDM 2012, held in Berlin, Germany in July 2012. The 51 revised full papers presented were carefully reviewed and selected from 212 submissions. The topics range from theoretical topics for classification, clustering, association rule and pattern mining to specific data mining methods for the different multimedia data types such as image mining, text mining, video mining and web mining.

This book, complete with online files and updates, covers a hugely important area of study in computing. It constitutes the refereed proceedings of the 10th International Symposium on Practical Aspects of Declarative Languages, PADL 2008, held in San Francisco, CA, USA, in January 2008. The 20 revised full papers along with the abstract of 1 invited talk were carefully reviewed and selected from 44 submissions. The papers address all current aspects of declarative programming.

Achieve Breakthrough Productivity and Quality with MDD and Eclipse-Based DSLs Domain-specific languages (DSLs) and model-driven development (MDD) offer software engineers powerful new ways to improve productivity, enhance quality, and insulate systems from rapid

technological change. Now, there's a pragmatic, start-to-finish guide to creating DSLs and using MDD techniques with the powerful open source Eclipse platform. In Eclipse Modeling Project, Richard C. Gronback illuminates both the principles and techniques software professionals need to master, offering insights that will be invaluable to developers working with any tool or platform. As coleader of the Eclipse Modeling Project, Gronback is singularly well-positioned to demonstrate DSLs and MDD at work in Eclipse. Gronback systematically introduces each of the Eclipse technologies that can be used in DSL and MDD development. Throughout, he introduces key concepts and technologies in the context of a complete worked example and presents new best practices and never-before published techniques. He also covers Eclipse projects discussed in no other book, including Query/View/Transformation (QVT) and the Graphical Modeling Framework (GMF)—a project the author personally leads. Eclipse Modeling Project gives software practitioners all the knowledge they need to explore the remarkable potential of DSLs and MDD—and includes coverage of Why a model-based approach enables the rapid customization of high-quality solutions within the product line paradigm How the Eclipse Modeling Project's capabilities can be used to efficiently create new DSLs Powerful techniques for developing DSL abstract syntax, graphical notation, and textual syntax How to build Model-to-Model (M2M) and Model-to-Text (M2T) transformations—including a powerful new M2M implementation of the Object Management Group's QVT Operational Mapping Language (OML) Efficiently packaging and deploying DSLs with Eclipse Complete reference sections for the Graphical Editing Framework (GEF), GMF runtime and tooling, QVT OML, Xpand, and more

Extend and enhance your Java applications with domain-specific scripting in Groovy About

This Book Build domain-specific mini languages in Groovy that integrate seamlessly with your Java apps with this hands-on guide Increase stakeholder participation in the development process with domain-specific scripting in Groovy Get up to speed with the newest features in Groovy using this second edition and integrate Groovy-based DSLs into your existing Java applications. Who This Book Is For 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. This 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. What You Will Learn Familiarize yourself with Groovy scripting and work with Groovy closures Use the meta-programming features in Groovy to build mini languages Employ Groovy mark-up and builders to simplify application development Familiarize yourself with Groovy mark-up and build your own Groovy builders Build effective DSLs with operator overloading, command chains, builders, and a host of other Groovy language features Integrate Groovy with your Java and JVM based applications In Detail The times when developing on the JVM meant you were a Java programmer have long passed. The JVM is now firmly established as a polyglot development environment with many projects opting for alternative development languages to Java such as Groovy, Scala, Clojure, and JRuby. In this pantheon of development languages, Groovy stands out for its excellent DSL enabling features which allows it to be manipulated to produce mini languages that are tailored to a project's needs. A comprehensive tutorial on designing and developing mini Groovy based Domain Specific Languages, this book will guide you through the development of several mini

DSLs that will help you gain all the skills needed to develop your own Groovy based DSLs with confidence and ease. Starting with the bare basics, this book will focus on how Groovy can be used to construct domain specific mini languages, and will go through the more complex meta-programming features of Groovy, including using the Abstract Syntax Tree (AST). Practical examples are used throughout this book to de-mystify these seemingly complex language features and to show how they can be used to create simple and elegant DSLs. Packed with examples, including several fully worked DSLs, this book will serve as a springboard for developing your own DSLs. Style and approach This book is a hands-on guide that will walk you through examples for building DSLs with Groovy rather than just talking about "metaprogramming with Groovy". The examples in this book have been designed to help you gain a good working knowledge of the techniques involved and apply these to producing your own Groovy based DSLs.

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.

The success of product line engineering techniques in the last 15 years has popularized the use of software variability as a key modeling approach for describing the commonality and variability of systems at all stages of the software lifecycle. Software product lines enable a family of products to share a common core platform, while allowing for product specific functionality being built on top of the platform. Many companies have exploited the concept of software product lines to increase the resources that focus on highly differentiating functionality and thus improve their competitiveness with higher quality and reusable products and decreasing the time-to-market condition. Many books on product line engineering either introduce specific product line techniques or include brief summaries of industrial cases. From these sources, it is difficult to gain a comprehensive understanding of the various dimensions

and aspects of software variability. Here the editors address this gap by providing a comprehensive reference on the notion of variability modeling in the context of software product line engineering, presenting an overview of the techniques proposed for variability modeling and giving a detailed perspective on software variability management. Their book is organized in four main parts, which guide the reader through the various aspects and dimensions of software variability. Part 1 which is mostly written by the editors themselves introduces the major topics related to software variability modeling, thus providing a multi-faceted view of both technological and management issues. Next, part 2 of the book comprises four separate chapters dedicated to research and commercial tools. Part 3 then continues with the most practical viewpoint of the book presenting three different industry cases on how variability is managed in real industry projects. Finally, part 4 concludes the book and encompasses six different chapters on emerging research topics in software variability like e.g. service-oriented or dynamic software product lines, or variability and aspect orientation. Each chapter briefly summarizes "What you will learn in this chapter", so both expert and novice readers can easily locate the topics dealt with. Overall, the book captures the current state of the art and best practices, and indicates important open research challenges as well as possible pitfalls. Thus it serves as a reference for researchers and practitioners in software variability management, allowing them to develop the next set of solutions, techniques and methods in this complicated and yet fascinating field of software engineering.
Describes ways to incorporate domain modeling into software development.
Copyright: b7e94eec0cfd606a2b27c9005b3e2345