Applying Domain Driven Design And Patterns With Examples In C And Net

Solve complex business problems by understanding users better, finding the right problem to solve, and building lean event-driven systems to give your customers what they really want Key Features Apply DDD principles using modern tools such as EventStorming, Event Sourcing, and CQRS Learn how DDD applies directly to various architectural styles such as REST, reactive systems, and microservices Empower teams to work flexibly with improved services and decoupled interactions Book Description Developers across the world are rapidly adopting DDD principles to deliver powerful results when writing software that deals with complex business requirements. This book will guide you in involving business stakeholders when choosing the software you are planning to build for them. By figuring out the temporal nature of behavior-driven domain models, you will be able to build leaner, more agile, and modular systems. You'll begin by uncovering domain complexity and learn how to capture the behavioral aspects of the domain language. You will then learn about EventStorming and advance to creating a new project in .NET Core 2.1; you'll also and write some code to transfer your events from sticky notes to C#. The book will show you how to use aggregates to handle commands and produce events. As you progress, you'll get to grips with Bounded Contexts, Context Map, Event Sourcing, and CQRS. After translating domain models into executable C# code, you will create a frontend for your application using Vue.js. In addition to this, you'll learn how to refactor your code and cover event versioning and migration essentials. By the end of this DDD book, you will have gained the confidence to implement the DDD approach in your organization and be able to explore new techniques that complement what you've learned from the book. What you will learn Discover and resolve domain complexity together with business stakeholders Avoid common pitfalls when creating the domain model Study the concept of Bounded Context and aggregate Design and build temporal models based on behavior and not only data Explore benefits and drawbacks of Event Sourcing Get acquainted with CQRS and to-the-point read models with projections Practice building one-way flow UI with Vue.js Understand how a task-based UI conforms to DDD principles Who this book is for This book is for .NET developers who have an intermediate level understanding of C#, and for those who seek to deliver value, not just write code. Intermediate level of competence in JavaScript will be helpful to follow the UI chapters. "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 Real examples written in PHP showcasing DDD Architectural Styles, Tactical Design, and Bounded Context Integration About This Book Focuses on practical code rather than theory Full of real-world examples that you can apply to your own projects Shows how to build PHP apps using DDD principles Who This Book Is For This book is for PHP developers who want to apply a DDD mindset to their code. You should have a good understanding of PHP and some knowledge of DDD. This book doesn't dwell on the theory, but instead gives you the code that you need. What You Will Learn Correctly design all design elements of Domain-Driven Design with PHP Learn all tactical patterns to achieve a fully worked-out Domain-Driven Design Apply hexagonal architecture within your application Integrate bounded contexts in your applications Use REST and Messaging approaches In Detail Domain-Driven Design (DDD) has arrived in the PHP community, but for all the talk, there is very little real code. Without being in a training session and with no PHP real examples, learning DDD can be challenging. This book changes all that. It details how to implement tactical DDD patterns and gives full examples of topics such as integrating Bounded Contexts with REST, and DDD messaging strategies. In this book, the authors show you, with tons of details and examples, how to properly design Entities, Value Objects, Services, Domain Events, Aggregates, Factories, Repositories, Services, and Application Services with PHP. They show how to apply Hexagonal Architecture within your application whether you use an open source framework or your own. Style and approach This highly practical book shows developers how to apply domain-driven design principles to PHP. It is full of solid code examples to work through.

Map concepts and ideas in domain-driven design (DDD) and transpose them into clean, testable, and quality code that is effective alongside the Laravel framework. This book teaches you how to implement the concepts and patterns present in DDD in the real world as a complete web application. With these tactics and concepts in place, you'll engage in a variety of example applications, built from the ground up, and taken directly from real-world domains. Begin by reviewing foundational stepping stones (with small, manageable examples to show proof of concepts as well as illustrations to conceptualize the more complex topics) of both DDD and Laravel. Specifically, such topics as entities, value objects, developing an ubiquitous language, DTOs, and knowledge discovery. Next, you will dive into some more advanced topics of DDD and use these concepts as a guide to make customizations to the default Laravel installation, giving you an understanding of why these alterations are vital to the DDD and Laravel platform. Finally, you will cover the very powerful Eloquent ORM that comes stock with Laravel and understand how it can be utilized to represent entities, handle repositories, and support domain events. Although there is a basic coverage chapter and a setup tutorial for Laravel (along with a high level intro about the components used within it), Domain-Driven Laravel is best suited to readers who have been at least exposed to the framework and have had the opportunity to tinker

around with it. What You'll Learn Utilize a blazing-fast rapid development pipeline built from DDD building blocks and facilitated with Laravel Implement value objects, repositories, entities, anti-corruption layers and others using Laravel as a web framework Apply enhanced techniques for quick prototyping of complex requirements and quality results using an iterative and focused approach Create a base framework (Laravel) that can serve as a template to start off any project Gain insight on which details are important to a project's success and how to acquire the necessary knowledge Who This Book Is For Ideal for for frontend/backend web developers, devops engineers, Laravel framework lovers and PHP developers hoping to learn more about either Domain Driven Design or the possibilities with the Laravel framework. Those with a working knowledge of plain PHP can also gain value from reading this book.

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.

As Python continues to grow in popularity, projects are becoming larger and more complex. Many Python developers are taking an interest in high-level software design patterns such as hexagonal/clean architecture, event-driven architecture, and the strategic patterns prescribed by domain-driven design (DDD). But translating those patterns into Python isn't always straightforward. With this hands-on guide, Harry Percival and Bob Gregory from MADE.com introduce proven architectural design patterns to help Python developers manage application complexity—and get the most value out of their test suites. Each pattern is illustrated with concrete examples in beautiful, idiomatic Python, avoiding some of the verbosity of Java and C# syntax. Patterns include: Dependency inversion and its links to ports and adapters (hexagonal/clean architecture) Domain-driven design's distinction between Entities, Value Objects, and Aggregates Repository and Unit of Work patterns for persistent storage Events, commands, and the message bus Command-query responsibility segregation (CQRS) Event-driven architecture and reactive microservices

Patterns, Domain-Driven Design (DDD), and Test-Driven Development (TDD) enable architects and developers to create systems that are powerful, robust, and maintainable. Now, there's a comprehensive, practical guide to leveraging all these techniques primarily in Microsoft .NET environments, but the discussions are just as useful for Java developers. Drawing on seminal work by Martin Fowler (Patterns of Enterprise Application Architecture) and Eric Evans (Domain-Driven Design), Jimmy Nilsson shows how to create real-world architectures for any .NET application. Nilsson illuminates each principle with clear, well-annotated code examples based on C# 1.1 and 2.0. His examples and discussions will be valuable both to C# developers and those working with other .NET languages and any databases–even with other platforms, such as J2EE. Coverage includes · Quick primers on patterns, TDD, and refactoring · Using architectural techniques to improve software quality · Using domain models to support business rules and validation · Applying enterprise patterns to provide persistence support via NHibernate · Planning effectively for the presentation layer and UI testing · Designing for Dependency Injection, Aspect Orientation, and other new paradigms

Domain-Driven Design (DDD) concept was introduced by first Eric Evans in 2003. The concept of microservices did not exist at that time. So basically DDD was introduced to solve the problem of a large monolithic code base. In the monolithic world, once the codebase starts growing with the growth of the business, it becomes difficult to maintain the code organized and structured as it was originally designed. Monolithic applications designed using MVC architecture have good separation between the business layer and the presentation layer. But in the absence of the strict architectural guidelines, the business layer does not provide specific rules to maintain responsibility boundaries between different modules and classes. That's why as the code base grows it increases the risk of logic breakdown, responsibility leakage between the different components of the application.

Handboek voor de uitvoering van ICT-projecten volgens een internationale, gezaghebbende standaard.

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

need a recent version of F# (4.0 or greater), and the appropriate .NET runtime for your platform.Full installation instructions for all platforms at fsharp.org.

This book constitutes revised selected papers from the five workshops collocated with the 15th International Conference on Software Engineering and Formal Methods, SEFM 2017. The 38 papers presented in this volume were carefully reviewed and selected from a total of 55 submissions. They stem from the following workshops: DataMod 2017 -- 6th International Symposium "From Data to Models and Back"; FAACS 2017 -- 1st Workshop on Formal Approaches for Advanced Computing Systems; MSE 2017 -- 1st Workshop on Microservices: Science and Engineering; CoSim-CPS 2017 -- 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems; FOCLASA 2017 -- 15th International Workshop on Foundations Of Coordination Languages and Self-Adaptive Systems?.

See how Domain-Driven Design (DDD) combines with Jakarta EE MicroProfile or Spring Boot to offer a complete suite for building enterprise-grade applications. In this book you will see how these all come together in one of the most efficient ways to develop complex software. Practical Domain-Driven Design in Enterprise Java starts by building out the Cargo Tracker reference application as a monolithic application using the Jakarta EE platform. By doing so, you will map concepts of DDD (bounded contexts, language, and aggregates) to the corresponding available tools (CDI, JAX-RS, and JPA) within the Jakarta EE platform. Once you have completed the monolithic application, you will walk through the complete conversion of the monolith to a microservices-based architecture, again mapping the concepts of DDD and the corresponding available tools within the MicroProfile platform (config, discovery, and fault tolerance). To finish this section, you will examine the same microservices architecture on the Spring Boot platform. The final set of chapters looks at what the application would be like if you used the CQRS and event sourcing patterns. Here you'll use the Axon framework as the base framework. What You Will Learn Discover the DDD architectural principles and use the DDD design patterns Use the new Eclipse Jakarta EE platform Work with the Spring Boot framework Implement microservices design patterns, including context mapping, logic design, entities, integration, testing, and security Carry out event sourcing Apply CQRS Who This Book Is For Junior developers intending to start working on enterprise Java; senior developers transitioning from monolithic- to microservices-based architectures; and architects transitioning to a DDD philosophy of building applications.

Let's address the critical question right off the bat: why do you have to read this book? If you have a knack for software development, please do not throw this opportunity away. Now is your chance to become an expert.When reliable approaches function without domain driven design, such denial of this technology or market environment become costly. Even medium-sized mobile apps benefit immensely from the structure of the application of this amazing architecture. Too often, developers only chuck lines of code at problems that can be fixed with vital structural changes.Domain-Driven Design does a great job in incorporating industry conditions into aspects of software development. For example, this book focuses on how the accuracy of the model driven design involves constant communication in multiple occasions, and developers separated by team/locations do not participate in continual contact. Recommendations are provided on segmenting the software as a consequence of the market reality. This will enable efficient modeling across independent teams. Such approaches also take political problems within groups into consideration, as well as the collaboration of overburdened departments and legacy systems. In fact, the book points out a claim that many developers are protesting against, but this is particularly true: not all developers in a group need to pursue the same approach.The claim does not mean that developers are expected to use arbitrary solutions; it implies that programmers are not allowed to tie each other to a unique solution if they can address fundamentally different problems. Two teams working on your device may have a "User" category and may have a Consumer Category. But perhaps Team A wants a customer as part of the payment process, or Team B needs a customer as part of a support system. Should we use all departments in the same Customer Class? Perhaps not. Perhaps they should have Consumer-Grade billing and support. Then each Consumer includes only the actions they need for the job they have to do. Nevertheless, you will find considerable resistance to solutions like this-- critics are complaining of "unnecessary duplication," but in fact, it is not replication, and it is needed. Of similar reasons, the book tends to support the possibility of locking "Bounded Context."Furthermore, this beginner's guide is right on the money when it comes to structuring code in a manner that allows for business structure. The book also emphasizes concentration and project management in a sense that also helps teams to operate independently without the dictator and the design.

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

This book shows how to apply pattern ideas in business applications. It presents more than 20 structural and behavioral business patterns that use the REA (resources, events, agents) pattern as a common backbone. The developer working on business frameworks can use the patterns to derive the right abstractions and to design and ensure that the meta-rules are followed by the developers of the actual applications. The application developer can use these patterns to design a business application, to ensure that it does not violate the domain rules, and to adapt the application to changing requirements without the need to change the overall architecture.

If you are an experienced JavaScript developer who wants to improve the design of his or her applications, or find yourself in a situation to implement an application in an unfamiliar domain, this book is for you. Prior knowledge of JavaScript is required and prior experience with Node.js will also be helpful.

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.

I want to thank you for checking out the book, "Domain Driven Design: How to Easily Implement Domain Driven Design - A Quick & Simple Guide". This book contains proven steps and strategies on how you can implement the domain-driven design approach in your projects to bring out better results. Through the domain-driven design approach, you and your project team will better understand the domain that you aim to serve and communicate in a common language that can ensure harmony and team work with your group. You will be able to finish the whole design and development process focused on what is truly essential. Thanks again and I hope you enjoy it!

"Demystifies object-oriented programming, and lays out how to use it to design truly secure and performant applications." —Charles Soetan, Plum.io Key Features Dozens of techniques for writing object-oriented code that's easy to read, reuse, and maintain Write code that other programmers will instantly understand Design rules for constructing objects, changing and exposing state, and more Examples written in an instantly familiar pseudocode that's easy to apply to Java, Python, C#, and any object-oriented language Purchase of the print book includes a free eBook in PDF, Kindle, and ePub formats from Manning Publications. About The Book Well-written object-oriented code is easy to read, modify, and debug. Elevate your coding style by mastering the universal best practices for object design presented in this book. These clearly presented rules, which apply to any OO language, maximize the clarity and durability of your codebase and increase productivity for you and your team. In Object Design Style Guide, veteran developer Matthias Noback lays out design rules for constructing objects, defining methods, and much more. All examples use instantly familiar pseudocode, so you can follow along in the language you prefer. You'll go case by case through important scenarios and challenges for object design and then walk through a simple web application that demonstrates how different types of objects can work together effectively. What You Will Learn Universal design rules for a wide range of objects Best practices for testing objects A catalog of common object types Changing and exposing state Test your object design skills with exercises This Book Is Written For For readers familiar with an object-oriented language and basic application architecture. About the Author Matthias Noback is a professional web developer with nearly two decades of experience. He runs his own web development, training, and consultancy company called "Noback's Office." Table of Contents: 1 ¦ Programming with objects: A primer 2 ¦ Creating services 3 ¦ Creating other objects 4 ¦ Manipulating objects 5 ¦ Using objects 6 ¦ Retrieving information 7 ¦ Performing tasks 8 ¦ Dividing responsibilities 9 ¦ Changing the behavior of services 10 ¦ A field guide to objects 11 ¦ Epilogue

Applied WPF 4 in Context sets the standard for leveraging the latest Windows user interface technology in your business applications. Using this book, you'll learn how to implement world-class Windows Professional Foundation (WPF) solutions in a real-world line of business applications, developing the code from the ground up, and understand how to apply best development practices and related .NET products and technologies to your solutions. You will cover designing and developing the application, testing and debugging, data access, reporting, and applying styles and themes to enhance the look of the user interface—all using WPF in a very practical, eminently useful context. You'll create asynchronous and parallel code, and learn how to distribute the application's components using Windows Communication Foundation (WCF). You'll also apply the Model-View-ViewModel pattern, again in a real-world WPF application. Elegant and functional WPF applications are easier to create than ever before with Applied WPF 4 in Context.

A step-by-step tutorial to get acquainted with the ASP.NET MVC4 Framework and its features in order to discover how to develop web applications using them.This book is targeted at people who are familiar with C# development on the .NET platform and are interested in web development with the ASP.NET development framework. No prior web or mobile development experience is required

In the classic style of Manning's "In Action" series, NHibernate in Action shows.NET developers how to use the NHibernate Object/Relational Mapping tool.This book is a translation from Java to .NET, as well as an expansion, ofManning's bestselling Hibernate in Action. All traces of Java have been carefullyreplaced by their .NET equivalents. The book shows how to implementcomplex business objects, and later teaches advanced techniques like cachingand session management. Readers will discover how to implement persistence ina .NET application, and how to configure NHibernate to specify the mappinginformation between business objects and database tables. Readers will also beintroduced to the internal architecture of NHibernate by progressively buildinga complete sample application using Agile methodologies. 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.

Provides information on domain-driven design to guild application software for enterprise applications.

Master the art of implementing scalable microservices in your production environment with ease About This Book Use domain-driven design to build microservices Use Spring Cloud to use Service Discovery and Registeration Use Kafka, Avro and Spring Streams for implementing event based microservices Who This Book Is For This book is for Java developers who are familiar with the microservices architecture and now wants to take a deeper dive into effectively implementing microservices at an enterprise level. A reasonable knowledge level and understanding of core microservice elements and applications is expected. What You Will Learn Use domain-driven design to design and implement microservices Secure microservices using Spring Security Learn to develop REST service development Deploy and test microservices Troubleshoot and debug the issues faced during development Learning best practices and common principals about microservices In Detail Microservices are the next big thing in designing scalable, easy-to-maintain applications. It not only makes app development easier, but also offers great flexibility to utilize various resources optimally. If you want to build an enterprise-ready implementation of the microservices architecture, then this is the book for you! Starting off by understanding the core concepts and framework, you will then focus on the high-level design of large software projects. You will gradually move on to setting up the development environment and configuring it before implementing continuous integration to deploy your microservice architecture. Using Spring security, you will secure microservices and test them effectively using REST Java clients and other tools like RxJava 2.0. We'll show you the best patterns, practices and common principals of microservice design and you'll learn to troubleshoot and debug the issues faced during development. We'll show you how to design and implement reactive microservices. Finally, we'll show you how to migrate a monolithic application to microservices based application. By the end of the book, you will know how to build smaller, lighter, and faster services that can be implemented easily in a production environment. Style and approach This book starts from the basics, including environment setup and provides easy-to-follow steps to implement the sample project using microservices.

Understand the key challenges and solutions around building microservices in the enterprise application environment. This book provides a comprehensive understanding of microservices architectural principles and how to use microservices in real-world scenarios. Architectural challenges using microservices with service integration and API management are presented and you learn how to eliminate the use of centralized integration products such as the enterprise service bus (ESB) through the use of composite/integration microservices. Concepts in the book are supported with use cases, and emphasis is put on the reality that most of you are implementing in a "brownfield" environment in which you must implement microservices alongside legacy applications with minimal disruption to your business. Microservices for the Enterprise covers state-of-the-art techniques around microservices messaging, service development and description, service discovery, governance, and data management technologies and guides you through the microservices design process. Also included is the importance of organizing services as core versus atomic, composite versus integration, and API versus edge, and how such organization helps to eliminate the use of a central ESB and expose services through an API gateway. What You'll Learn Design and develop microservices architectures with confidence Put into practice the

most modern techniques around messaging technologies Apply the Service Mesh pattern to overcome inter-service communication challenges Apply battle-tested microservices security patterns to address real-world scenarios Handle API management, decentralized data management, and observability Who This Book Is For Developers and DevOps engineers responsible for implementing applications around a microservices architecture, and architects and analysts who are designing such systems

Describes ways to incorporate domain modeling into software development.

As the first technical book of its kind, this unique resource walks you through the process of building a real-world application using Domain-Driven Design implemented in C#. Based on a real application for an existing company, each chapter is broken down into specific modules so that you can identify the problem, decide what solution will provide the best results, and then execute that design to solve the problem. With each chapter, you'll build a complete project from beginning to end.

This book will explain how to apply domain-driven design concepts in a project with Spring Boot 2.0.6 and how to combine them with practices, such as unit testing (test driven development), relational databases and object relational mappers like JPA(Java Persistence API). We will see step by step how to grow an application from the very beginning to a full-fledged solution with DDD principles. Finally there will be two projects, one (static web project using jQuery & HTML) for user interface and another (Spring Boot + REST + JPA project) for API, logic and persistence.You will see the full process of building a software project using concepts such as entities, value objects, aggregates, repositories, bounded contexts, and domain events. In the way I will explain why we make one decision over another. You will learn what DDD concepts are applicable in which particular case and why it is so. We will see, how to apply the domain-driven design principles in a real world application.Book Outline and Prerequisites :IntroductionStarting with the First Bounded ContextIntroducing UI and Persistence LayersExtending the Bounded Context with AggregatesIntroducing RepositoriesIntroducing the Second Bounded ContextWorking with Domain EventsLooking Forward to Further EnhancementsBook Summary :Full application from scratchDomain modelingDDD concepts in practiceSpring BootDatabase and ORMUnit testingMVC

"[This] is a book about design in the .NET world, driven in an agile manner and infused with the products of the enterprise patterns community. [It] shows you how to begin applying such things as TDD, object relational mapping, and DDD to .NET projects ... techniques that many developers think are the key to future software development ... As the technology gets more capable and sophisticated, it becomes more important to understand how to use it well. This book is a valuable step toward advancing that understanding."--Martin Fowler, author of Refactoring and Patterns of Enterprise Application Architecture Patterns, Domain-Driven Design (DDD), and Test-Driven Development (TDD) enable architects and developers to create systems that are powerful, robust, and maintainable. Now, there's a comprehensive, practical guide to leveraging all these techniques primarily in Microsoft .NET environments, but the discussions are just as useful for Java developers. Drawing on seminal work by Martin Fowler (Patterns of Enterprise Application Architecture) and Eric Evans (Domain-Driven Design), Jimmy Nilsson shows how to create real-world architectures for any .NET application. Nilsson illuminates each principle with clear, well-annotated code examples based on C# 1.1 and 2.0. His examples and discussions will be valuable both to C# developers and those working with other .NET languages and any databases-even with other platforms, such as J2EE. Coverage includes · Quick primers on patterns, TDD, and refactoring · Using architectural techniques to improve software quality · Using domain models to support business rules and validation · Applying enterprise patterns to provide persistence support via NHibernate · Planning effectively for the presentation layer and UI testing · Designing for Dependency Injection, Aspect Orientation, and other new paradigms.

The Software Factory methodology is based on recognition of these similarities and a drive to extend the concept of "reusability" to the point where we achieve entirely automated product lines. Based on an analysis and understanding of the common features and techniques of a set of applications, a Software Factory defines a tailored, end-to-end methodology for building these applications. At the heart of the Software factory methodology is the concept of Domain Specific Languages (DSLs), which in essence are development environments specifically tailored to the set of applications in hand. It removes a certain degree of flexibility but greatly enhances productivity by removing a lot of the coding complexity (for an analogy, consider the use of the now ubiquitous drag-and-drop controls in Winforms or Visual Basic). Further, in the SF methodology, patterns, process advice, and best practices can be harvested and applied for all applications in the set. There are some good books on the theory of SF already on the market. Up until this point, a lot of these concepts were fairly theoretical and abstract.

Een fascinerend verhaal vol wijze levenslessen en een genot om te lezen. – Paulo Coelho, auteur van De alchemist De monnik die zijn Ferrari verkocht is het verhaal van Julian Mantle, een geslaagd advocaat, die door zijn enerverende maar onevenwichtige leven een bijna noodlottige hartaanval krijgt. Geconfronteerd met zijn fysieke broosheid maakt Julian een spirituele crisis door die hem noopt op zoek te gaan naar de antwoorden op de grote vragen van het leven. Hij besluit radicaal te breken met zijn luxe maar oppervlakkige bestaan en waagt zich op een buitengewone odyssee naar een eeuwenoude cultuur in de Himalaya. Daar vindt hij een krachtig systeem dat de mens in staat stelt om de potentie van geest, lichaam en ziel ten volle te ontplooien en intenser, gelukkiger en harmonieuzer te leven. De monnik die zijn Ferrari verkocht is een inspirerende vertelling, waarin de tijdloze spirituele wijsheid van het Oosten wordt vermengd met messcherpe zakelijkheid van het Westen. Een boek dat u stap voor stap de weg wijst naar een leven met meer moed, evenwicht, vreugde en innerlijke rijkdom.

Building software is harder than ever. As a developer, you not only have to chase ever-changing technological trends but also need to understand the business domains behind the software. This practical book provides you with a set of core patterns, principles, and practices for analyzing business domains, understanding business strategy, and, most importantly, aligning software design with its business needs. Author Vlad Khononov shows you how these practices lead to robust implementation of business logic and help to future-proof software design and architecture. You'll examine the relationship between domain-driven design (DDD) and other methodologies to ensure you make architectural decisions that meet business requirements. You'll also explore the real-life story of implementing DDD in a startup company. With this book, you'll learn how to: Analyze a company's business domain to learn how the system you're building fits its competitive strategy Use DDD's strategic and tactical tools to architect effective software solutions that address business needs Build a shared understanding of the business domains you encounter Decompose a system into bounded contexts Coordinate the work of multiple teams Gradually introduce DDD to brownfield projects

JavaScript backs some of the most advanced applications. It is time to adapt modern software development practices from JavaScript to model complex business needs. JavaScript Domain-Driven Design allows you to leverage your JavaScript skills to create advanced applications. You'll start with learning domain-driven concepts and working with UML diagrams. You'll follow this up with how to set up your projects and utilize the TDD tools. Different objects and prototypes will help you create model for your business process and see how DDD develops common

language for developers and domain experts. Context map will help you manage interactions in a system. By the end of the book, you will learn to use other design patterns such as DSLs to extend DDD with object-oriented design base, and then get an insight into how to select the right scenarios to implement DDD.

Methods for managing complex software construction following the practices, principles and patterns of Domain-Driven Design with code examples in C# This book presents the philosophy of Domain-Driven Design (DDD) in a down-to-earth and practical manner for experienced developers building applications for complex domains. A focus is placed on the principles and practices of decomposing a complex problem space as well as the implementation patterns and best practices for shaping a maintainable solution space. You will learn how to build effective domain models through the use of tactical patterns and how to retain their integrity by applying the strategic patterns of DDD. Full end-to-end coding examples demonstrate techniques for integrating a decomposed and distributed solution space while coding best practices and patterns advise you on how to architect applications for maintenance and scale. Offers a thorough introduction to the philosophy of DDD for professional developers Includes masses of code and examples of concept in action that other books have only covered theoretically Covers the patterns of CQRS, Messaging, REST, Event Sourcing and Event-Driven Architectures Also ideal for Java developers who want to better understand the implementation of DDD

Domain-Driven Design (DDD) software modeling delivers powerful results in practice, not just in theory, which is why developers worldwide are rapidly moving to adopt it. Now, for the first time, there's an accessible guide to the basics of DDD: What it is, what problems it solves, how it works, and how to quickly gain value from it. Concise, readable, and actionable, Domain-Driven Design Distilled never buries you in detail–it focuses on what you need to know to get results. Vaughn Vernon, author of the best-selling Implementing Domain-Driven Design, draws on his twenty years of experience applying DDD principles to real-world situations. He is uniquely well-qualified to demystify its complexities, illuminate its subtleties, and help you solve the problems you might encounter. Vernon guides you through each core DDD technique for building better software. You'll learn how to segregate domain models using the powerful Bounded Contexts pattern, to develop a Ubiquitous Language within an explicitly bounded context, and to help domain experts and developers work together to create that language. Vernon shows how to use Subdomains to handle legacy systems and to integrate multiple Bounded Contexts to define both team relationships and technical mechanisms. Domain-Driven Design Distilled brings DDD to life. Whether you're a developer, architect, analyst, consultant, or customer, Vernon helps you truly understand it so you can benefit from its remarkable power. Coverage includes What DDD can do for you and your organization–and why it's so important The cornerstones of strategic design with DDD: Bounded Contexts and Ubiquitous Language Strategic design with Subdomains Context Mapping: helping teams work together and integrate software more strategically Tactical design with Aggregates and Domain Events Using project acceleration and management tools to establish and maintain team cadence

Copyright: 316fb09a0de92d73e6176784dd04be91