wrox
A Wiley Brand

Patterns, Principles, and Practices of Domain-Driven Design

Scott Millett with Nick Tune

Patterns, Principles, and Practices of Domain-Driven Design

Scott Millett

Nick Tune

wrox™
A Wiley Brand

Patterns, Principles, and Practices of Domain-Driven Design

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
`www.wiley.com`

Published simultaneously in Canada

ISBN: 978-1-118-71470-6
ISBN: 978-1-118-71465-2 (ebk)
ISBN: 978-1-118-71469-0 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2014951018

Patterns, Principles, and Practices of Domain-Driven Design

For my darling buds, Primrose and Albert.

—SCOTT MILLETT

ABOUT THE AUTHOR

SCOTT MILLETT is the Director of IT for Iglu.com and has been working with .NET since version 1.0. He was awarded the ASP.NET MVP in 2010 and 2011. He is also the author of *Professional ASP.NET Design Patterns* and *Professional Enterprise .NET.* If you would like to contact Scott about DDD or working at Iglu, feel free to write to him at `scott@elbandit.co.uk`, by giving him a tweet `@ScottMillett`, or becoming friends via `https://www.linkedin.com/in/scottmillett`.

ABOUT THE CONTRIBUTING AUTHOR

NICK TUNE is passionate about solving business problems, building ambitious products, and constantly learning. Being a software developer really is his dream job. His career highlight so far was working at 7digital, where he was part of self-organizing, business-focused teams that deployed to production up to 25 times per day. His future ambitions are to work on exciting new products, with passionate people, and continually become a more complete problem solver.

You can learn more about Nick and his views on software development, software delivery, and his favorite technologies on his website (`www.ntcoding.co.uk`) and Twitter (`@ntcoding`).

ABOUT THE TECHNICAL EDITOR

ANTONY DENYER works as a developer, consultant, and coach and has been developing software professionally since 2004. He has worked on various projects that have effectively used DDD concepts and practices. More recently, he has been advocating the use of CQRS and REST in the majority of his projects. You can reach him via e-mail at `antonydenyer.co.uk`, and he tweets from `@tonydenyer`.

CREDITS

PROJECT EDITOR
Rosemarie Graham

TECHNICAL EDITOR
Antony Denyer

PRODUCTION EDITOR
Christine O'Connor

COPY EDITOR
Karen Gill

MANAGER OF CONTENT DEVELOPMENT AND ASSEMBLY
Mary Beth Wakefield

MARKETING DIRECTOR
David Mayhew

MARKETING MANAGER
Carrie Sherrill

PROFESSIONAL TECHNOLOGY & STRATEGY DIRECTOR
Barry Pruett

BUSINESS MANAGER
Amy Knies

ASSOCIATE PUBLISHER
Jim Minatel

PROJECT COORDINATOR, COVER
Brent Savage

PROOFREADER
Jenn Bennett, Word One

INDEXER
Johnna VanHoose Dinse

COVER DESIGNER
Wiley

COVER IMAGE
@iStockphoto.com/andynwt

ACKNOWLEDGMENTS

FIRSTLY I WOULD LIKE to give a massive thanks to Nick Tune for agreeing to help me out with this project and contributing greatly to many of the chapters. I would also like to thank Rosemarie Graham, Jim Minatel, and all those at Wrox who have helped to create this book. Thanks as well to Antony Denyer who did a sterling job as the technical editor. Lastly, many thanks to Isabel Mack for the grammar pointers and early feedback of the Leanpub draft.

INTRODUCTION

WRITING SOFTWARE IS EASY— at least if it's greenfield software. When it comes to modifying code written by other developers or code you wrote six months ago, it can be a bit of a bore at best and a nightmare at worst. The software works, but you aren't sure exactly how. It contains all the right frameworks and patterns, and has been created using an agile approach, but introducing new features into the codebase is harder than it should be. Even business experts aren't helpful because the code bears no resemblance to the language they use. Working on such systems becomes a chore, leaving developers frustrated and devoid of any coding pleasure.

Domain-Driven Design (DDD) is a process that aligns your code with the reality of your problem domain. As your product evolves, adding new features becomes as easy as it was in the good old days of greenfield development. Although DDD understands the need for software patterns, principles, methodologies, and frameworks, it values developers and domain experts working together to understand domain concepts, policies, and logic equally. With a greater knowledge of the problem domain and a synergy with the business, developers are more likely to build software that is more readable and easier to adapt for future enhancement.

Following the DDD philosophy will give developers the knowledge and skills they need to tackle large or complex business systems effectively. Future enhancement requests won't be met with an air of dread, and developers will no longer have stigma attached to the legacy application. In fact, the term *legacy* will be recategorized in a developer's mind as meaning this: a system that continues to give value for the business.

OVERVIEW OF THE BOOK AND TECHNOLOGY

This book provides a thorough understanding of how you can apply the patterns and practices of DDD on your own projects, but before delving into the details, it's good to take a bird's-eye view of the philosophy so you can get a sense of what DDD is really all about.

The Problem Space

Before you can develop a solution, you must understand the problem. DDD emphasizes the need to focus on the business problem domain: its terminology, the core reasons behind why the software is being developed, and what success means to the business. The need for the development team to value domain knowledge just as much as technical expertise is vital to gain a deeper insight into the problem domain and to decompose large domains into smaller subdomains.

Figure I-1 shows a high-level overview of the problem space of DDD that will be introduced in the first part of this book.

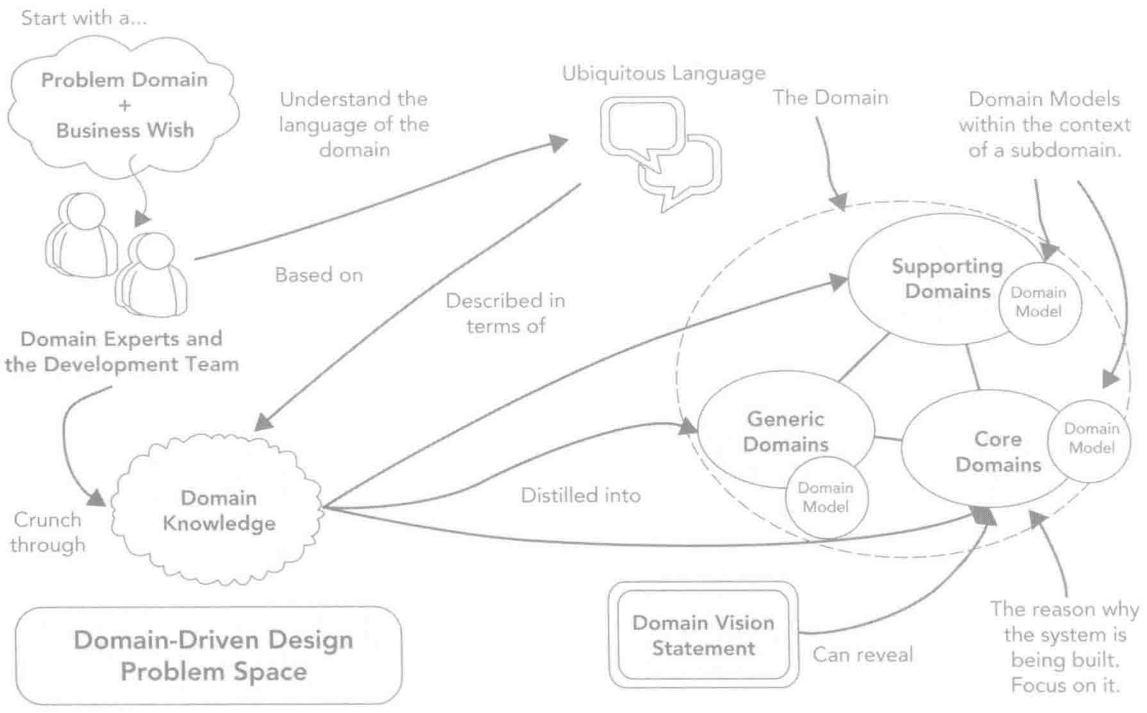
Start with a...
Problem Domain + Business Wish
Understand the language of the domain
Ubiquitous Language
The Domain
Domain Models within the context of a subdomain.
Domain Experts and the Development Team
Based on
Described in terms of
Supporting Domains
Domain Model
Generic Domains
Domain Model
Core Domains
Domain Model
Crunch through
Domain Knowledge
Distilled into
Domain Vision Statement
Can reveal
The reason why the system is being built. Focus on it.
Domain-Driven Design Problem Space

FIGURE I-1: A blueprint of the problem space of DDD.

The Solution Space

When you have a sound understanding of the problem domain, strategic patterns of DDD can help you implement a technical solution in synergy with the problem space. Patterns enable core parts of your system that are crucial to the success of the product to be protected from the generic areas. Isolating integral components allows them to be modified without having a rippling effect throughout the system.

Core parts of your product that are sufficiently complex or will frequently change should be based on a model. The tactical patterns of DDD along with Model-Driven Design will help you create a useful model of your domain in code. A model is the home to all of the domain logic that enables your application to fulfill business use cases. A model is kept separate from technical complexities to enable business rules and policies to evolve. A model that is in synergy with the problem domain will enable your software to be adaptable and understood by other developers and business experts.

Figure I-2 shows a high-level overview of the solution space of DDD that is introduced in the first part of this book.

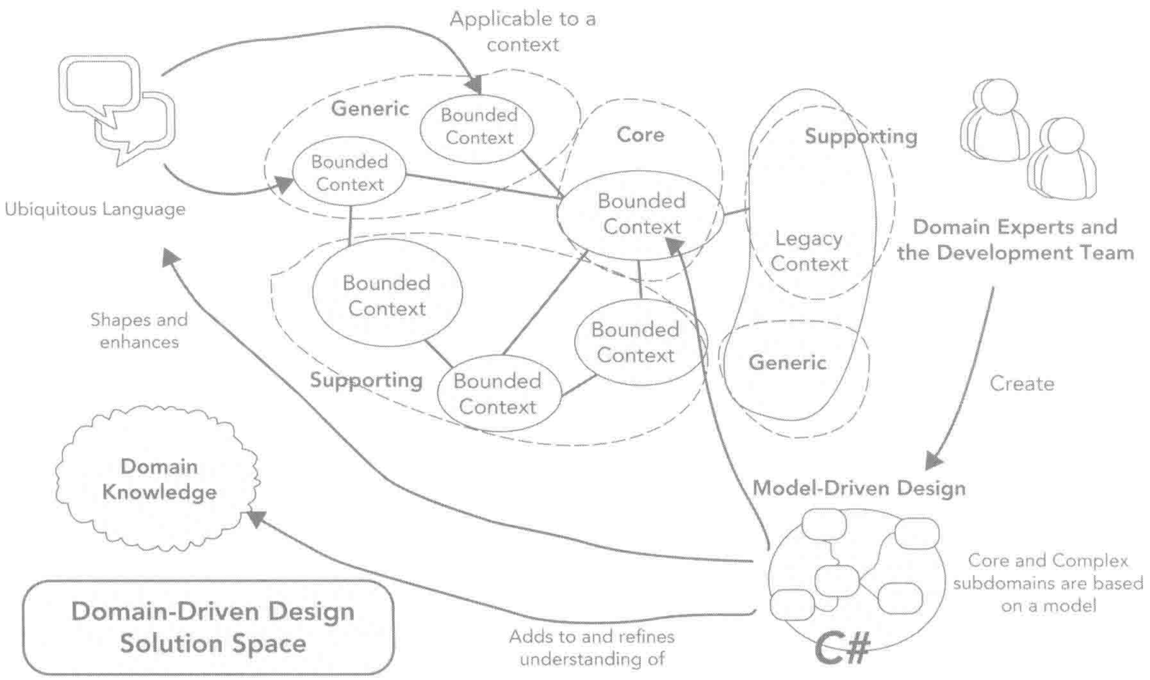

FIGURE I-2: A blueprint of the solution space of Domain-Driven Design.

HOW THIS BOOK IS ORGANIZED

This book is divided into four parts. Part I focuses on the philosophy, principles, and practices of DDD. Part II details the strategic patterns of integrating bounded contexts. Part III covers tactical patterns for creating effective domain models. Part IV delves into design patterns you can apply to utilize the domain model and build effective applications.

Part I: The Principles and Practices of Domain-Driven Design

Part I introduces you to the principles and practices of DDD.

Chapter 1: What Is Domain-Driven Design?

DDD is a philosophy to help with the challenges of building software for complex domains. This chapter introduces the philosophy and explains why language, collaboration, and context are the most important facets of DDD and why it is much more than a collection of coding patterns.

Chapter 2: Distilling the Problem Domain

Making sense of a complex problem domain is essential to creating maintainable software. Knowledge crunching with domain experts is key to unlocking that knowledge. Chapter 2 details techniques to enable development teams to collaborate, experiment, and learn with domain experts to create an effective domain model.

Chapter 3: Focusing on the Core Domain

Chapter 3 explains how to distill large problem domains and identify the most important part of a problem: the core domain. It then explains why you should focus time and energy in the core domain and isolate it from the less important supporting and generic domains.

Chapter 4: Model-Driven Design

Business colleagues understand an analysis model based on the problem area you are working within. Development teams have their own code version of this model. In order for business and technical teams to collaborate a single model is needed. A ubiquitous language and a shared understanding of the problem space is what binds the analysis model to the code model. The idea of a shared language is core to DDD and underpins the philosophy. A language describing the terms and concepts of the domain, which is created by both the development team and the business experts, is vital to aid communication on complex systems.

Chapter 5: Domain Model Implementation Patterns

Chapter 5 expands on the role of the domain model within your application and the responsibilities it takes on. The chapter also presents the various patterns that can be used to implement a domain model and what situations they are most appropriate for.

Chapter 6: Maintaining the Integrity of Domain Models with Bounded Contexts

In large solutions more than a single model may exist. It is important to protect the integrity of each model to remove the chance of ambiguity in the language and concepts being reused inappropriately by different teams. The strategic pattern known as bounded context is designed to isolate and protect a model in a context while ensuring it can collaborate with other models.

Chapter 7: Context Mapping

Using a context map to understand the relationships between different models in an application and how they integrate is vital for strategic design. It is not only the technical integrations that context maps cover but also the political relationships between teams. Context maps provide a view of the landscape that can help teams understand their model in the context of the entire landscape.

Chapter 8: Application Architecture

An application needs to be able to utilize the domain model to satisfy business use cases. Chapter 8 introduces architectural patterns to structure your applications to retain the integrity of your domain model.

Chapter 9: Common Problems for Teams Starting Out with Domain-Driven Design

Chapter 9 describes the common issues teams face when applying DDD and why it's important to know when not to use it. The chapter also focuses on why applying DDD to simple problems can lead to overdesigned systems and needless complexity.

Chapter 10: Applying the Principles, Practices, and Patterns of DDD

Chapter 10 covers techniques to sell DDD and to start applying the principles and practices to your projects. It explains how exploration and experimentation are more useful to build great software than trying to create the perfect domain model.

Part II: Strategic Patterns: Communicating between Bounded Contexts

Part II shows you how to integrate bounded contexts, and offers details on the options open for architecting bounded contexts. Code examples are presented that detail how to integrate with legacy applications. Also included are techniques for communicating across bounded contexts.

Chapter 11: Introduction to Bounded Context Integration

Modern software applications are distributed systems that have scalability and reliability requirements. This chapter blends distributed systems theory with DDD so that you can have the best of both worlds.

Chapter 12: Integrating via Messaging

A sample application is built showing how to apply distributed systems principles synergistically with DDD using a message bus for asynchronous messaging.

Chapter 13: Integrating via HTTP with RPC and REST

Another sample application is built showing an alternative approach to building asynchronous distributed systems. This approach uses standard protocols like Hypertext Transport Protocol (HTTP), REST, and Atom instead of a message bus.

Part III: Tactical Patterns: Creating Effective Domain Models

Part III covers the design patterns you can use to build a domain model in code, along with patterns to persist your model and patterns to manage the lifecycles of the domain objects that form your model.

Chapter 14: Introducing the Domain Modeling Building Blocks

This chapter is an introduction to all the tactical patterns at your disposal that allow you to build an effective domain model. The chapter highlights some best practice guidelines that produce more manageable and expressive models in code.

Chapter 15: Value Objects

This is an introduction to the DDD modeling construct that represents identityless domain concepts like money.

Chapter 16: Entities

Entities are domain concepts that have an identity, such as customers, transactions, and hotels. This chapter covers a variety of examples and complementary implementation patterns.

Chapter 17: Domain Services

Some domain concepts are stateless operations that do not belong to a value object or an entity. They are known as domain services.

Chapter 18: Domain Events

In many domains, focusing on events reveals greater insight than focusing on just entities. This chapter introduces the domain event design pattern that allows you to express events more clearly in your domain model.

Chapter 19: Aggregates

Aggregates are clusters of domain objects that represent domain concepts. Aggregates are a consistency boundary defined around invariants. They are the most powerful of the tactical patterns.

Chapter 20: Factories

Factories are a lifecycle pattern that separate use from construction for complex domain objects.

Chapter 21: Repositories

Repositories mediate between the domain model and the underlying data model. They ensure that the domain model is kept separate from any infrastructure concerns.

Chapter 22: Event Sourcing

Like domain events in Chapter 18, event sourcing is a useful technique for emphasizing, in code, events that occur in the problem domain. Event sourcing goes beyond domain events by storing the state of the domain model as events. This chapter provides a number of examples, including ones that use a purpose-built event store.

Part IV: Design Patterns for Effective Applications

Part IV showcases the design patterns for architecting applications that utilize and protect the integrity of your domain model.

Chapter 23: Architecting Application User Interfaces

For systems composed of many bounded contexts, the user interface often requires the composition of data from a number of them, especially when your bounded contexts form a distributed system.

Chapter 24: CQRS: An Architecture of a Bounded Context

CQRS is a design pattern that creates two models where there once was one. Instead of a single model to handle the two different contexts of reads and writes, two explicit models are created to handle commands or serve queries for reports.

Chapter 25: Commands: Application Service Patterns for Processing Business Use Cases

Learn the difference between application and domain logic to keep your model focused and your system maintainable.

Chapter 26: Queries: Domain Reporting

Business people need information to make informed business and product-development decisions. A range of techniques for building reports that empower the business is demonstrated in this chapter.

WHO SHOULD READ THIS BOOK

This book introduces the main themes behind DDD—its practices, patterns, and principles along with personal experiences and interpretation of the philosophy. It is intended to be used as a learning aid for those interested in or starting out with the philosophy. It is not a replacement for *Domain-Driven Design: Tackling Complexity in the Heart of Software* by Eric Evans (Addison-Wesley Professional, 2003). Instead, it takes the concepts introduced by Evans and distills them into simple straightforward prose, with practical examples so that any developer can get up to speed with the philosophy before going on to study the subject in more depth.

This book is based on the author's personal experiences with the subject matter. You may not always agree with it if you are a seasoned DDD practitioner, but you should still get something out of it.

SOURCE CODE

As you work through the examples in this book, you may choose either to type in all the code manually, or to use the source code files that accompany the book. All the source code used in this book is available for download at `www.wrox.com`. Specifically for this book, the code download is on the Download Code tab at: `www.wrox.com/go/domaindrivendesign`. Although code examples are presented in C# .NET. The concepts and practices can be applied to any programming language.

You can also search for the book at `www.wrox.com` by ISBN (the ISBN for this book is 978-1-1187-1470-6) to find the code. And a complete list of code downloads for all current Wrox books is available at `www.wrox.com/dynamic/books/download.aspx`.

ERRATA

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata, you may save another reader hours of frustration, and at the same time, you will be helping us provide even higher quality information.

To find the errata page for this book, go to `www.wrox.com/go/domaindrivendesign`.

And click the Errata link. On this page you can view all errata that has been submitted for this book and posted by Wrox editors.

If you don't spot "your" error on the Book Errata page, go to `www.wrox.com/contact/techsupport.shtml` and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

P2P.WROX.COM

For author and peer discussion, join the P2P forums at `http://p2p.wrox.com`. The forums are a web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At `http://p2p.wrox.com`, you will find a number of different forums that will help you, not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to `http://p2p.wrox.com` and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join, as well as any optional information you wish to provide, and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.

NOTE *You can read messages in the forums without joining P2P, but in order to post your own messages, you must join.*

Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works, as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

SUMMARY

The aim of this book is to present the philosophy of 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 by using tactical patterns and how to retain their integrity by applying the strategic patterns of DDD.

By the end of this book, you will have a thorough understanding of DDD. You will be able to communicate its value and when to use it. You will understand that even though the tactical patterns of DDD are useful, it is the principles, practices, and strategic patterns that will help you architect applications for maintenance and scale. With the information gained within this book, you will be in a better place to manage the construction and maintenance of complex software for large and complex problem domains.