

O'REILLY®

Einführung in Domain-Driven Design

Von der Business-Strategie
zum technischen Design



Vlad Khononov

Übersetzung von Thomas Demmig

Über dieses Buch

Vladik Khononov ist ein scharfsinniger Denker, der seit Jahren DDD anwendet, um reale Business-Probleme zu lösen. Seine Ideen bringen die gesamte DDD-Community immer wieder voran, und dieses Buch wird DDD-Einsteiger inspirieren.

- Nick Tune, Technology Consultant

Was mir beim Lesen der Entwürfe dieses Buchs durch den Kopf ging und was mir besonders Spaß daran gemacht hat, ist, dass es das Versprechen seines Titels erfüllt! Es ist ein einladender und informativer Praxisratgeber, der das gesamte Feld des DDD abdeckt - von der Strategie bis hin zum technischen Design. Ich habe ein tieferes Verständnis von DDD und neue Einblicke in Bereiche erhalten, in denen ich schon Erfahrung besaß, und Konzepte und Praktiken kennengelernt, mit denen ich bisher weniger zu tun hatte. Vlad ist ein wundervoller Lehrer!

- Ruth Malan, Architecture Consultant bei Bredemeyer Consulting

Vlad hat bei seiner Arbeit an ausgesprochen komplexen Projekten Erfahrungen als DDD-Praktiker gesammelt und teilt dieses Wissen immer sehr gerne. In diesem Buch erzählt er die Geschichte des DDD auf eine einmalige Art und Weise und ermöglicht uns damit, sehr viel darüber zu lernen. Dieses Buch ist für Neueinsteigerinnen und Neueinsteiger gedacht, aber auch ich, die DDD schon lange praktiziert und über DDD schreibt und spricht, habe festgestellt, dass ich durch seine Sichtweise sehr viel gelernt habe.

- Julie Lerman, Software-Coach, O'Reilly-Autorin und engagierte DDD-Advokatin

Copyright und Urheberrechte:

Die durch die dpunkt.verlag GmbH vertriebenen digitalen Inhalte sind urheberrechtlich geschützt. Der Nutzer verpflichtet sich, die Urheberrechte anzuerkennen und einzuhalten. Es werden keine Urheber-, Nutzungs- und sonstigen Schutzrechte an den Inhalten auf den Nutzer übertragen. Der Nutzer ist nur berechtigt, den abgerufenen Inhalt zu eigenen Zwecken zu nutzen. Er ist nicht berechtigt, den Inhalt im Internet, in Intranets, in Extranets oder sonst wie Dritten zur Verwertung zur Verfügung zu stellen. Eine öffentliche Wiedergabe oder sonstige Weiterveröffentlichung und eine gewerbliche Vervielfältigung der Inhalte wird ausdrücklich ausgeschlossen. Der Nutzer darf Urheberrechtsvermerke, Markenzeichen und andere Rechtsvorbehalte im abgerufenen Inhalt nicht entfernen.

Einführung in Domain-Driven Design

*Von der Business-Strategie zum
technischen Design*

Vlad Khononov

*Deutsche Übersetzung von
Thomas Demmig*

O'REILLY®

Vlad Khononov

Lektorat: Ariane Hesse

Fachliche Unterstützung: Jörn Koch, Henning Schwentner

Übersetzung: Thomas Demmig

Korrektorat: Sibylle Feldmann, www.richtiger-text.de

Satz: III-Satz, www.drei-satz.de

Herstellung: Stefanie Weidner

Umschlaggestaltung: Karen Montgomery, Michael Oréal, www.oreal.de

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN:

Print 978-3-96009-195-0

PDF 978-3-96010-732-3

ePub 978-3-96010-733-0

mobi 978-3-96010-734-7

1. Auflage 2022

Translation Copyright für die deutschsprachige Ausgabe © 2022 dpunkt.verlag GmbH

Wieblinger Weg 17

69123 Heidelberg

Authorized German translation of the English edition of *Learning Domain-Driven Design*

ISBN 9781098100131 © 2022 Vladislav Khononov. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«. O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Hinweis:

Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.



Schreiben Sie uns:

Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: komentar@oreilly.de.

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag noch Übersetzer können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

5 4 3 2 1 0

Inhalt

Grußwort

Vorwort

Einleitung

Teil I Strategisches Design

1 Fachdomänen analysieren

Was ist eine Fachdomäne?

Was ist eine Subdomain?

 Arten von Subdomains

 Subdomains vergleichen

 Grenzen von Subdomains ermitteln

Beispiele für eine Analyse der Fachdomäne

 Gigmaster

 BusVNext

Wer sind die Domänenexperten?

Zusammenfassung

Übungen

2 Domänenwissen ermitteln

Fachliche Probleme

Erlernen von Fachwissen

Kommunikation

Was ist eine Ubiquitous Language?

Sprache der Domänenexperten

Szenarien

Konsistenz

Modell der Fachdomäne

Was ist ein Modell?

Effektives Modellieren

Die Fachdomäne modellieren

Kontinuierliche Arbeit

Werkzeuge

Herausforderungen

Zusammenfassung

Übungen

3 Die Komplexität einer Domain im Griff behalten

Inkonsistente Modelle

Was ist ein Bounded Context?

Modellgrenzen

Verbesserte Ubiquitous Language

Gültigkeitsbereich eines Bounded Context

Bounded Contexts versus Subdomains

Subdomains

Bounded Contexts

Die Verbindung zwischen Subdomains und Bounded Contexts

Grenzen

Physische Grenzen

- Zuständigkeitsgrenzen
- Bounded Contexts in der Realität
- Semantic Domains
- Wissenschaft
- Einen Kühlschrank kaufen
- Zusammenfassung
- Übungen

4 Bounded Contexts integrieren

- Kooperation
 - Partnership
 - Shared Kernel
- Customer/Supplier
 - Conformist
 - Anticorruption Layer
 - Open-Host Service
- Separate Ways
 - Kommunikationsprobleme
 - Generic Subdomains
 - Modellunterschiede
- Context Map
 - Wartung
 - Einschränkungen
- Zusammenfassung
- Übungen

Teil II Taktisches Design

5 Einfache Business-Logik implementieren

- Transaction Script
 - Implementierung
 - So einfach ist es nicht!
 - Wann man ein Transaction Script verwendet

Active Record

Implementierung

Wann sollte Active Record genutzt werden?

Seien Sie pragmatisch

Zusammenfassung

Übungen

6 Komplexe Business-Logik angehen

Geschichte

Domain Model

Implementierung

Bausteine

Komplexität managen

Zusammenfassung

Übungen

7 Die zeitliche Dimension modellieren

Event Sourcing

Suche

Analyse

Source of Truth

Event Store

Event-Sourced Domain Model

Vorteile

Nachteile

Häufige Fragen

Performance

Daten löschen

Warum kann ich nicht einfach ...?

Zusammenfassung

Übungen

8 Architektur-Patterns

Business-Logik versus Architektur-Patterns

Schichtenarchitektur

- Presentation Layer

- Business Logic Layer

- Data Access Layer

- Kommunikation zwischen den Schichten

- Variation

- Wann Schichtenarchitektur genutzt werden sollte

Ports & Adapters

- Terminologie

- Dependency Inversion Principle

- Integration von Infrastrukturkomponenten

- Varianten

- Wann man Ports & Adapters verwendet

Command-Query Responsibility Segregation

- Polyglot Modeling

- Implementierung

- Read Models projizieren

- Herausforderungen

- Model Segregation

- Wann CQRS verwendet wird

Scope

Zusammenfassung

Übungen

9 Kommunikations-Patterns

Model Translation

- Stateless Model Translation

- Stateful Model Translation

Aggregates integrieren

- Outbox

- Saga

- Process Manager

Zusammenfassung

Übungen

Teil III Domain-Driven Design in der Praxis umsetzen

10 Designheuristiken

Heuristik

Bounded Contexts

Implementierungs-Patterns für Business-Logik

Architektur-Patterns

Teststrategie

 Testpyramide

 Testdiamant

 Umgekehrte Testpyramide

Entscheidungsbaum für taktisches Design

Zusammenfassung

Übungen

11 Evolution von Designentscheidungen

Änderungen an Domains

 Core nach Generic

 Generic nach Core

 Supporting nach Generic

 Supporting nach Core

 Core nach Supporting

 Generic nach Supporting

Strategische Designaspekte

Taktische Designaspekte

 Transaction Script nach Active Record

 Active Record nach Domain Model

 Domain Model nach Event-Sourced Domain Model

 Alte Übergänge generieren

 Migrationsevents modellieren

Organisationsänderungen

 Partnership nach Customer/Supplier

Customer/Supplier nach Separate Ways
Domänenwissen
Wachstum
Subdomains
Bounded Contexts
Aggregates
Zusammenfassung
Übungen

12 EventStorming

Was ist EventStorming?
Wer sollte am EventStorming teilnehmen?
Was brauchen Sie zum EventStorming?
Der EventStorming-Prozess
Schritt 1: Unstrukturiertes Erforschen
Schritt 2: Zeitachsen
Schritt 3: Pain Points
Schritt 4: Pivotal Events
Schritt 5: Commands
Schritt 6: Policies
Schritt 7: Read Models
Schritt 8: Externe Systeme
Schritt 9: Aggregates
Schritt 10: Bounded Contexts
Varianten
Wann Sie EventStorming einsetzen
Tipps für das EventStorming
Achten Sie auf die Dynamik
Remote EventStorming
Zusammenfassung
Übungen

13 Domain-Driven Design in der Praxis

Strategische Analyse

- Die Fachdomäne verstehen
- Das aktuelle Design untersuchen
- Modernisierungsstrategie
 - Strategisches Modernisieren
 - Taktisches Modernisieren
- Eine Ubiquitous Language kultivieren
- Pragmatisches Domain-Driven Design
- Domain-Driven Design schmackhaft machen
 - Undercover Domain-Driven Design
- Zusammenfassung
- Übungen

Teil IV DDD und andere Methodiken und Patterns

14 Microservices

- Was ist ein Service?
- Was ist ein Microservice?
 - Method as a Service: Perfekte Microservices?
 - Designziel
 - Systemkomplexität
 - Microservices als tiefe Services
 - Microservices als tiefe Module
- Domain-Driven Design und die Grenzen von Microservices
 - Bounded Contexts
 - Aggregates
 - Subdomains
- Die öffentlichen Schnittstellen von Microservices komprimieren
 - Open-Host Service
 - Anticorruption Layer
- Zusammenfassung

Übungen

15 Event-Driven Architecture

Event-Driven Architecture

Events

- Events, Commands und Messages

- Struktur

- Event-Typen

Design einer Event-gesteuerte Integration

- Verteilter Big Ball of Mud

- Zeitliche Kopplung

- Funktionelle Kopplung

- Implementierungskopplung

- Die Event-gesteuerte Integration refaktorisieren

- Event-gesteuerte Designheuristiken

Zusammenfassung

Übungen

16 Data Mesh

Analytisches Datenmodell versus transaktionales

Datenmodell

- Fact-Tabelle

- Dimension-Tabelle

- Analytische Modelle

Plattformen zum Managen analytischer Daten

- Data Warehouse

- Data Lake

- Herausforderungen von Data Warehouse und Data

- Lake Architectures

Data Mesh

- Daten anhand von Domains unterteilen

- Daten als Produkt

- Autonomie ermöglichen

- Ein Ökosystem schaffen

Data Mesh und Domain-Driven Design kombinieren
Zusammenfassung
Übungen

Abschließende Worte

Anhang A DDD anwenden: eine Fallstudie

Anhang B Antworten auf die Übungsfragen

Literatur

Index

Grußwort

Domain-Driven Design stellt eine Reihe von Praktiken bereit, um gemeinsam Software aus der Sicht des Business zu bauen – genau die Domäne und die Probleme, die Sie angehen wollen. Der Begriff wurde ursprünglich von Eric Evans im Jahr 2003 mit der Veröffentlichung des Buchs *Domain-Driven Design: Tackling Complexity in the Heart of Software* geprägt, das in der DDD-Community als »Blue Book« bezeichnet wird.

Zwar besteht das Ziel von Domain-Driven Design darin, die Komplexität zu reduzieren und einen klaren Pfad aufzuzeigen, aber es gibt dabei auch viele tolle Ideen, die ebenso auf weniger komplizierte Projekte angewendet werden können. DDD erinnert uns daran, dass Entwickler und Entwicklerinnen nicht die einzigen am Bau von Software beteiligten Personen sind. Die Domänenexperten oder Fachexperten (engl. Domain Experts), für die die Software gebaut wird, bringen ein entscheidendes Verständnis für die zu lösenden Probleme mit. Wir schaffen während der verschiedenen Stadien eine Partnerschaft, indem wir zunächst »strategisches Design« anwenden, um

das Business-Problem – also die Domäne – zu verstehen. Dann teilen wir das Problem in kleinere, lösbare und miteinander verbundene Probleme auf. Die Partnerschaft mit den Domänenexperten hilft uns auch dabei, in der Sprache der Domäne zu kommunizieren, statt die Business-Seite dazu zu zwingen, die technische Sprache der Software erlernen zu müssen.

Das zweite Stadium eines DDD-basierten Projekts ist das »taktische Design«, bei dem wir die Erkenntnisse des strategischen Designs in Softwarearchitektur und -Implementierung umsetzen. Auch hier liefert das DDD wieder Ratschläge und Patterns für das Organisieren dieser Domänen und das Vermeiden zusätzlicher Komplexität. Im taktischen Design geht die Partnerschaft weiter, wenn die Domänenexperten beim Blick auf den von den Softwareteams gebauten Code ihre Domänensprache wiedererkennen.

Seit der Veröffentlichung des »Blue Book« haben nicht nur viele Organisationen von den Ideen profitiert, es hat sich auch eine Gemeinschaft aus erfahrenen DDD-Praktikern gebildet. Und die kollaborative Natur von DDD sorgt dafür, dass diese Gemeinschaft ihre Erfahrungen und Perspektiven gerne weitergibt und Tools hervorbringt, die Teams dabei helfen, diese Ideen anzunehmen und Nutzen aus ihnen zu ziehen. In einer Keynote zur Explore DDD 2019 ermutigte Eric Evans die Gemeinschaft darin, DDD weiterhin voranzubringen – und nicht nur deren Praktiken zu entwickeln, sondern auch Wege zu finden, die Ideen von DDD effektiver zu verbreiten.

Und das bringt mich dazu, warum ich ein so großer Fan von *Einführung in Domain-Driven Design* bin. Ich war von Vlad schon aufgrund seiner Vorträge auf Konferenzen und seiner anderen Texte begeistert. Er hat bei seiner Arbeit an

ausgesprochen komplexen Projekten Erfahrungen als DDD-Praktiker gesammelt und teilt dieses Wissen immer sehr gerne. In diesem Buch erzählt er die Geschichte des DDD (nicht die Historie, sondern die Konzepte) auf eine einmalige Art und Weise, und er ermöglicht uns damit, sehr viel darüber zu lernen. Dieses Buch ist für Neueinsteigerinnen und Neueinsteiger gedacht, aber auch ich, die DDD schon lange praktiziert und über DDD schreibt und spricht, habe festgestellt, dass ich durch seine Sichtweise sehr viel gelernt habe. Ich habe auf das Buch in meinem DDD-Fundamentals-Kurs bei Pluralsight schon hingewiesen, bevor es überhaupt veröffentlicht wurde, und habe einige seiner Sichtweisen in Gesprächen mit Kunden weitergegeben.

Der Einstieg in DDD kann verwirrend sein. Wo wir DDD doch einsetzen, um die Komplexität von Projekten zu reduzieren, stellt Vlad DDD konsequenterweise auf eine Art und Weise vor, die die Komplexität des Themas an sich verringert. Und er macht mehr, als nur die Prinzipien von DDD vorzustellen. Im hinteren Teil des Buchs sind ein paar wichtige Praktiken beschrieben, die sich aus DDD entwickelt haben, wie zum Beispiel das EventStorming, der Umgang mit der Weiterentwicklung des Business-Fokus oder der Organisation an sich und deren Einfluss auf die Software, und es wird besprochen, wie DDD und Microservices zusammenpassen und wie Sie beides mit wohlbekannten Software-Patterns integrieren können. Ich denke, dass *Einführung in Domain-Driven Design* ein ausgezeichneter Einstieg für Neulinge sein wird, aber auch erfahrene Praktiker und Praktikerinnen davon sehr profitieren werden.

- Julie Lerman,
Software-Coach, O'Reilly-Autorin

und schon lange überzeugte DDD-Advokatin

Vorwort

Ich erinnere mich noch lebhaft an den Tag, an dem ich meinen ersten Job in der Softwareentwicklung antrat. Ich war gleichzeitig euphorisch und eingeschüchtert. Nachdem ich während meiner Zeit auf dem Gymnasium Software für lokale Firmen zusammengehackt hatte, wollte ich nun unbedingt ein »richtiger Programmierer« werden und Code für eine der größten Outsourcing-Firmen des Landes schreiben.

In meinen ersten Tagen dort zeigten mir meine neuen Kolleginnen und Kollegen die wichtigsten Dinge. Nach dem Einrichten des Firmen-E-Mail-Kontos und dem Vertrautmachen mit dem Zeiterfassungssystem ging es endlich um die interessanten Dinge: die Coding-Richtlinien und -Standards des Unternehmens. Mir wurde gesagt: »Hier schreiben wir immer gut designten Code, und wir verwenden Schichtenarchitektur.« Wir gingen die Definition jeder der drei Schichten durch – Datenzugriffs-, Business-Logik- und Präsentationsschicht – und sprachen dann über die Technologien und Frameworks für die jeweiligen Anforderungen der Schichten. Damals wurden

Daten mit dem Microsoft SQL Server 2000 gespeichert, der mithilfe von [ADO.NET](#) in die Datenzugriffsschicht integriert wurde. Die Präsentationsschicht setzte wahlweise auf WinForms für Desktopanwendungen oder auf [ASP.NET](#) WebForms für das Web. Wir verbrachten recht viel Zeit mit diesen beiden Schichten, und es verwirrte mich, dass die Business-Logik-Schicht kaum Aufmerksamkeit erhielt:

»Aber was ist mit der Business-Logik-Schicht?«

»Die ist ganz einfach. Da implementierst du die Business-Logik.«

»Aber was ist Business-Logik?«

»Ach, Business-Logik sind all die Schleifen und if-else-Anweisungen, die du brauchst, um die Anforderungen zu erfüllen.«

An diesem Tag begann meine Forschungsreise, auf der ich herausfinden wollte, was genau Business-Logik ist und wie sie eigentlich in gut designtem Code implementiert werden sollte. Es dauerte mehr als drei Jahre, bis ich endlich die Antwort fand.

Sie lag in Eric Evans bahnbrechendem Buch *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Es stellte sich heraus, dass ich nicht falsch lag. Business-Logik ist tatsächlich wichtig – sie ist das Herz der Software! Leider dauerte es aber weitere drei Jahre, bis ich die Weisheiten verstand, die Eric mir mitteilte. Das Buch ist sehr fortgeschritten, und die Tatsache, dass Englisch nur meine Drittsprache ist, hat dabei nicht geholfen.

Schließlich fügte sich aber alles zusammen, und ich schloss meinen Frieden mit Domain-Driven Design (DDD). Ich lernte die Prinzipien und Patterns von DDD kennen sowie die Feinheiten des Modellierens und Implementierens der Business-Logik und wie man die Komplexität im Herzen der Software angeht, die ich gerade baute. Trotz der Schwierigkeiten war es das wert. Meine Karriere verlief durch Domain-Driven Design deutlich anders.

Warum ich dieses Buch geschrieben habe

In den letzten zehn Jahren habe ich meinen Kolleginnen und Kollegen bei den unterschiedlichsten Firmen Domain-Driven Design vorgestellt und Kurse online wie offline darüber gehalten. Die Perspektive des Lehrers hat mir nicht nur dabei geholfen, mein Wissen zu vertiefen, sondern auch, die Art und Weise zu optimieren, wie ich die Prinzipien und Patterns von Domain-Driven Design vermittele.

Wie so oft ist Unterrichten noch herausfordernder als Lernen. Ich bin ein großer Fan der Arbeit und der Vorträge von Eliyahu M. Goldratt (https://de.wikipedia.org/wiki/Eliyahu_M._Goldratt). Eliyahu hat gerne erzählt, dass selbst die komplexesten Systeme inhärent einfach sind, wenn man sie aus dem richtigen Winkel betrachtet. In all den Jahren, in denen ich DDD lehre, war ich auf der Suche nach einem Modell der Methodik, die die inhärente Einfachheit von Domain-Driven Design aufzeigt.

Dieses Buch ist das Ergebnis meiner Bemühungen. Sein Ziel ist es, Domain-Driven Design zu demokratisieren – es leichter zu verstehen und es besser einsetzen zu können. Ich glaube, dass die DDD-Methodik von unschätzbarem Wert ist, insbesondere wenn Sie moderne Softwaresysteme entwerfen. Dieses Buch wird Ihnen genug Werkzeuge an die Hand geben, um bei Ihrer tagtäglichen Arbeit mit dem Anwenden von Domain-Driven Design beginnen zu können.

Wer dieses Buch lesen sollte

Ich denke, dass das Wissen um Prinzipien und Patterns von Domain-Driven Design für die Softwareentwicklung auf allen Ebenen nützlich ist – beim Einstieg in die Entwicklung, während man besser wird bis hin zur Arbeit auf Leitungsebene. DDD stellt nicht nur Werkzeuge und Techniken zum Modellieren und effektiven Implementieren von Software bereit, es beleuchtet zudem einen häufig übersehenen Aspekt der Softwareentwicklung – den Kontext. Ausgestattet mit dem Wissen über das Business-Problem des Systems, werden Sie beim Wählen einer passenden Lösung viel effektiver sein, einer Lösung, die nicht zu sehr vereinfacht noch zu »overengineert« ist, sondern die Bedürfnisse und Ziele des Business erfüllt.

Domain-Driven Design ist für die Softwarearchitektur sogar noch wichtiger – und insbesondere für angehende Softwarearchitektinnen und -architekten. Seine Tools für strategische Designentscheidungen werden Ihnen dabei helfen, ein großes System in Komponenten zu unterteilen – Services, Microservices oder Subsysteme – und zu designen, wie die Komponenten miteinander interagieren, um ein System zu bilden.

Und schließlich werden wir in diesem Buch darüber sprechen, wie Sie nicht nur Software designen, sondern auch, wie Sie das Design parallel zu Änderungen im Business-Umfeld weiterentwickeln. Dieser wichtige Aspekt der Softwareentwicklung wird Ihnen dabei helfen, das Systemdesign mit der Zeit »in Form« zu halten und zu verhindern, dass es sich in einen Big Ball of Mud verwandelt.

Übersicht über das Buch

Dieses Buch ist in vier Teile unterteilt: strategisches Design, taktisches Design, DDD in der Praxis und die Beziehung zwischen DDD und anderen Methodiken und Patterns. In [Teil I](#) stellen wir Werkzeuge und Techniken für Entscheidungen zu umfassenden Designfragen vor. In [Teil II](#) konzentrieren wir uns auf den Code – die verschiedenen Wege, die Business-Logik in ein System zu integrieren. [Teil III](#) behandelt Techniken und Strategien für das Anwenden von DDD auf reale Projekte. Und in [Teil IV](#) geht es ebenfalls um Domain-Driven Design, nun aber im Kontext anderer Methodiken und Patterns.

Dies werden Sie im jeweiligen Kapitel finden:

- [Kapitel 1](#) setzt den Rahmen für ein Softwareentwicklungsprojekt: die Fachdomäne, ihre Ziele und wie die Software sie unterstützen soll.
- [Kapitel 2](#) stellt die Idee einer »Ubiquitous Language« vor: die Praktik von Domain-Driven Design für eine effektive Kommunikation und Wissensvermittlung.
- [Kapitel 3](#) behandelt den Umgang mit der Komplexität von Fachdomänen und das Designen der High-Level-Architektur-Komponenten des Systems: Bounded Contexts.
- [Kapitel 4](#) untersucht die verschiedenen Patterns (Entwurfsmuster) zum Organisieren der Kommunikation und zur Integration der Bounded Contexts.
- In [Kapitel 5](#) beginnt der Einstieg in die Implementierungs-Patterns für die Business-Logik mit zwei Patterns, die für die einfache Business-Logik gedacht sind.
- In [Kapitel 6](#) geht es weiter zu komplexer Business-Logik und dem dazu passenden Domain Model Pattern.

- [Kapitel 7](#) ergänzt die Zeit-Perspektive und es führt einen noch fortgeschritteneren Weg zum Modellieren und Implementieren von Business-Logik ein: das Event-Sourced Domain Model.
- In [Kapitel 8](#) verschiebt sich der Fokus hin zu einer höheren Ebene, und es werden drei Architektur-Patterns zum Strukturieren von Komponenten beschrieben.
- [Kapitel 9](#) stellt die Patterns vor, die zum Orchestrieren der Arbeit der Systemkomponenten erforderlich sind.
- In [Kapitel 10](#) werden die bisher besprochenen Patterns zu einer Handvoll einfacher Faustregeln zusammengeführt, die das Treffen von Designentscheidungen vereinfachen.
- [Kapitel 11](#) schaut sich das Softwaredesign im zeitlichen Verlauf an und beschreibt, wie es sich im Rahmen seiner Lebensspanne verändern und weiterentwickeln sollte.
- [Kapitel 12](#) stellt das EventStorming vor: ein Low-Tech-Workshop für das effektive Teilen von Wissen, den Aufbau eines gemeinsamen Verständnisses und das Designen von Software.
- [Kapitel 13](#) führt die Schwierigkeiten auf, denen Sie sich eventuell gegenübersehen, wenn Sie Domain-Driven Design in bestehende Projekte einführen.
- In [Kapitel 14](#) geht es um die Beziehung zwischen dem Microservices-Architekturstil und Domain-Driven Design: wo die Unterschiede liegen und wo sie sich gegenseitig ergänzen.
- [Kapitel 15](#) schaut sich Patterns und Werkzeuge von Domain-Driven Design im Kontext der Event-Driven Architecture an.

- historische Sichtweisen
 - alte Übergänge generieren [206](#)
 - retroaktives Debugging [139](#)
 - vergangene Status des Aggregate rekonstruieren [138](#)
- Logs in eine Log-Tabelle einfügen [143](#)
- Löschen von Daten [142](#)
- Skalierbarkeit [142](#)
- Snapshot der Events eines Aggregate [141](#)
- statusbasierte Source of Truth [143](#)
- Versionsfeld für Modifikationsstatus [132](#)
- Zusammenarbeit bei Bounded Contexts
 - Partnership nach Customer/Supplier [209](#)
- Zuständigkeitsgrenzen [70](#)
 - Data Lake und Warehouse Architecture [290](#)

Über den Autor

Vlad (Vladik) Khononov ist Softwareentwickler mit über 20 Jahren Branchenerfahrung, in denen er für große und kleine Firmen gearbeitet hat – in unterschiedlichsten Rollen vom Webmaster bis zum Chefarchitekten. Vlad ist zudem als Sprecher, Blogger und Autor unterwegs. Er ist überall auf der Welt aktiv, um zu beraten und über Domain-Driven Design, Microservices und Softwarearchitektur ganz allgemein zu sprechen. Vlad hilft Firmen dabei, aus ihren Fachdomänen Sinn zu ziehen, Legacy-Systeme zu entwirren und komplexe Architekturaufgaben anzugehen. Er lebt im Norden Israels zusammen mit seiner Frau und einer fast vernünftigen Anzahl an Katzen.

Kolophon

Das Tier auf dem Titelbild von *Einführung in Domain-Driven Design* ist eine Monameerkatze (*Cercopithecus mona*), die in den tropischen Wäldern Westafrikas und der karibischen Inseln zu finden ist, wo sie während des Sklavenhandels eingeschleppt wurde. Sie springt im mittleren bis oberen Bereich des Laubdachs von Baum zu Baum und nutzt dabei ihren langen Schwanz zum Ausbalancieren.

Monameerkatzen haben ein rötlich braunes Fell, das im Gesicht, an den Gliedmaßen und am Schwanz dunkler ist. Die Unterseite und die Innenseite der Beine sind weiß. Weibchen sind im Schnitt 40 cm groß, während Männchen auf 50 cm kommen – der Schwanz kann dazu noch eine Länge von 65 cm oder mehr haben. Lange Fellbüschel an den Wangen können gelb oder grau gefärbt sein, und die Nasen zeigen eine hellpinke Färbung. Die Wangen dienen als Vorratsbereich bei der Futtersuche, sie können genauso groß wie der Magen sein.

Monameerkatzen fressen Früchte, Samen, Insekten und Blätter und werden in freier Wildbahn etwa 30 Jahre alt. Jeden Tag suchen sie mehrfach in großen Gruppen nach Futter. Es wurden Scharen von über 40 Tieren dokumentiert – meist dominiert ein Männchen die Gruppe, das mehrere Weibchen begattet und gegen rivalisierende Männchen kämpft. Diese Gruppen können sehr viel Lärm machen.

Monameerkatzen haben durch die Aktivitäten des Menschen den Gefährdungsgrad »potenziell gefährdet«. Viele der Tiere auf den O'Reilly-Covern sind vom Aussterben bedroht; sie alle sind wichtig für unsere Welt.

Die Umschlagillustration stammt von Karen Montgomery und basiert auf einem Schwarz-Weiß-Stich aus *Lydekker's Royal Natural History*. Die Schriftarten für das Cover sind Gilroy Semibold und Guardian Sans.