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 falschlag. 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 vermittle.

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.
- In Kapitel 16 wechseln wir schließlich noch von operationalen Systemen hin zu analytischen Datenmanagementsystemen und sprechen über das Zusammenspiel von Domain-Driven Design und der Data Mesh Architecture.

Alle diese Kapitel enden mit einer Reihe von Übungsfragen, die Sie beim Lernen unterstützen sollen. Manche der Fragen beziehen sich auf die fiktive Firma »WolfDesk«, um die verschiedenen Aspekte von Domain-Driven Design aufzuzeigen. Bitte lesen Sie sich die folgende Beschreibung von WolfDesk durch und werfen Sie auch gelegentlich erneut einen Blick darauf, wenn Sie relevante Übungsaufgaben beantworten wollen.

Beispieldomäne: WolfDesk

WolfDesk bietet ein Ticket-Management-System für Help Desks als Service an. Muss Ihr Start-up Ihren Kundinnen und Kunden Unterstützung bieten können, ist es in Nullkommanichts möglich, mithilfe von WolfDesk loszulegen.

WolfDesk nutzt ein anderes Bezahlmodell als seine Mitbewerber. Statt einen Betrag pro User zu verlangen, erlaubt es seinen Tenants (dt. Mandanten bzw. Unternehmenskunden, die die Software für den Support ihrer Kunden einsetzen), so viele Tickets wie nötig anzulegen. Die Kosten sind dann abhängig von der Anzahl der Support-Tickets, die pro Berechnungszeitraum geöffnet werden. Es gibt keinen Minimalbetrag, und es gibt automatische Volumenrabatte ab bestimmten Mengen an Tickets pro Monat: 10 % für das Öffnen von mehr als 500 Tickets, 20 % für mehr als 750 Tickets und 30 % für mehr als 1.000 Tickets.

Um zu verhindern, dass die Tenants das Geschäftsmodell missbrauchen, stellt der Algorithmus für den Ticket-Lebenszyklus sicher, dass inaktive Tickets automatisch geschlossen werden, sodass Kunden, die den Service nutzen, dazu angehalten sind, neue Tickets zu öffnen, wenn weitere Hilfe notwendig ist. Zudem implementiert WolfDesk ein Fraud Detection System, das Nachrichten analysiert und Fälle aufdeckt, in denen nicht miteinander in Zusammenhang stehende Themen im selben Ticket diskutiert werden.

Damit die Tenants bei ihrer Support-Arbeit unterstützt werden, hat WolfDesk ein »Support Autopilot«-Feature implementiert. Der Autopilot analysiert neue Tickets und versucht automatisch, eine passende Lösung in der Ticket-Historie zu finden. Die Funktionalität erlaubt es, die Nutzungsdauer der Tickets weiter zu verringern, und die Kunden werden dazu angehalten, neue Tickets für weitere Fragen zu öffnen.

WolfDesk implementiert alle denkbaren Sicherheitsstandards und -maßnahmen, um die User ihrer Tenants zu authentifizieren und zu autorisieren. Zudem können die Tenants ein Single Sign-On (SSO) auf Basis ihrer bestehenden Benutzerverwaltung konfigurieren.

Die Administrationsoberfläche erlaubt den Tenants, die möglichen Werte für die Ticket-Kategorien und eine Liste der Produkte zu konfigurieren.

Um neue Tickets nur während der Arbeitszeiten an die Support Agents des Tenants zu routen, kann man deren Arbeitszeiten in WolfDesk erfassen.

Da WolfDesk seine Services ohne Mindestgebühr anbietet, muss es seine Infrastruktur so optimieren, dass die Kosten beim Anlegen neuer Tenants minimiert werden. Dazu setzt WolfDesk auf Serverless Computing, sodass es seine Rechenressourcen flexibel an die Menge an aktiven Tickets anpassen kann.

In diesem Buch verwendete Konventionen

Die folgenden typografischen Konventionen werden in diesem Buch eingesetzt:

Kursiv
: Kennzeichnet neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateiendungen.

`Nichtproportionalschrift`
: Für Programmlistings, aber auch für Codefragmente in Absätzen, wie zum Beispiel Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.

Dieses Symbol steht für eine allgemeine Anmerkung.

Der Einsatz von Codebeispielen

Zusätzliches Material (Codebeispiele, Übungen und so weiter) finden Sie (auf Englisch) zum Herunterladen auf *https://learning-ddd.com*.

Alle Beispiele in diesem Buch sind in C# implementiert. Im Allgemeinen handelt es sich bei den Codebeispielen in den Kapiteln um Auszüge, die die besprochenen Konzepte aufzeigen sollen.

Natürlich sind die in diesem Buch behandelten Konzepte und Techniken nicht auf C# oder einen objektorientierten Ansatz beschränkt. Alles ist auch für andere Sprachen und andere Programmierparadigmen relevant. Sie dürfen daher gerne die Beispiele aus dem Buch in Ihrer Lieblingssprache implementieren und mir zukommen lassen. Ich werde sie dann auf der Website zum Buch ergänzen.

Haben Sie eine technische Frage oder ein Problem beim Einsatz der Codebeispiele, schreiben Sie bitte eine Mail an *bookquestions@oreilly.com*.

Dieses Buch soll Ihnen bei der Arbeit helfen. Generell können Sie die Codebeispiele in diesem Buch in Ihren Programmen und Ihrer Dokumentation nutzen. Sie müssen uns nicht um Erlaubnis fragen, sofern Sie nicht einen signifikanten Anteil des Codes veröffentlichen. So erfordert zum Beispiel das Schreiben eines Programms, das diverse Codeschnipsel aus diesem Buch nutzt, keine Erlaubnis. Das Verkaufen oder Bereitstellen von Beispielen aus diesem Buch benötigt hingegen eine Erlaubnis. Beantworten Sie eine Frage, indem Sie dieses Buch zitieren und Beispielcode daraus nutzen, benötigen Sie keine Erlaubnis. Übernehmen Sie einen deutlichen Teil des Beispielcodes für Ihre Produktdokumentation, müssen Sie fragen.

Wir freuen uns über eine Quellenangabe, verlangen sie aber nicht unbedingt. Zu einer Quellenangabe gehören normalerweise Titel, Autor, Verlag und ISBN – zum Beispiel: »Learning Domain-Driven Design by Vlad Khononov (O'Reilly). Copyright 2022 Vladislav Khononov, 978-1-098-10013-1.«

Wenn Sie das Gefühl haben, dass Sie die Codebeispiele über die Grenzen des Erlaubten hinaus einsetzen, können Sie uns über *permissions@oreilly.com* erreichen.

Danksagung

Ursprünglich hatte dieses Buch den Titel »What Is Domain-Driven Design?« und wurde 2019 als Report veröffentlicht. *Einführung in Domain-Driven Design* hätte das Licht der Welt nie ohne diesen Report erblickt, und ich bin denjenigen zu großem Dank verpflichtet, die »What Is Domain-Driven Design?« möglich gemacht haben: Chris Guzikowski, Ryan Shaw und Alicia Yound.[1]

Dieses Buch wäre durch den Content Director und Diversity Talent Lead Melissa Duffield ebenso wenig möglich gewesen, da sie für dieses Projekt gekämpft und es realisiert hat. Vielen Dank für all die Hilfe, Melissa!

Jill Leonard war Development Editor, Project Manager und Head Coach für dieses Buch. Ihre Rolle an diesem Buch kann gar nicht hoch genug gelobt werden. Jill – vielen Dank für all die harte Arbeit und deine Hilfe! Ein besonderer Dank dafür, dass du meine Motivation hochgehalten hast, auch als ich mit dem Gedanken spielte, meinen Namen zu ändern und ins Ausland abzutauchen.

Ein großer Dank gebührt dem Produktionsteam, durch die das Buch nicht nur geschrieben, sondern auch gelesen werden konnte: Kristen Brown, Audrey Doyle, Kate Dullea, Robert Romano und Katherine Tozer. Ich möchte hier auch dem gesamten Team von O'Reilly für seine tolle Arbeit danken. Die Arbeit mit euch ist ein Traum!

Vielen Dank an all die Menschen, die ich interviewt und mit denen ich mich beraten habe: Zófia Herendi, Scott Hirleman, Trond Hjorteland, Mark Lisker, Chris Richardson, Vaughn Vernon und Ivan Zakrevsky. Vielen Dank für eure Weisheit und dafür, dass ihr da wart, als ich Hilfe brauchte!

Ein besonderer Dank geht an das Team der Reviewer, die die frühen Entwürfe gelesen und mir dabei geholfen haben, die finale Version zu schaffen: Julie Lerman, Ruth Malan, Diana Montalion, Andrew Padilla, Rodion Promyshlennikov, Viktor Pshenitsyn, Alexei Torunov, Nick Tune, Vasiliy Vasilyuk und Rebecca Wirfs-Brock. Eure Unterstützung, euer Feedback und eure Kritik haben unfassbar viel geholfen. Danke sehr!

Auch möchte ich Kenny Baas-Schwegler, Alberto Brandolini, Eric Evans, Marco Heimeshoff, Paul Rayner, Mathias Verraes und dem Rest der tollen Domain-Driven Design Community danken. Ihr wisst, wer ihr seid. Ihr seid meine Lehrer und Mentorinnen. Vielen Dank, dass ihr euer Wissen per Social Media, Blogs und Konferenzen weitergebt!

Den größten Dank schulde ich meiner lieben Frau Vera, die mich immer bei meinen verrückten Projekten unterstützt und versucht, mich von Dingen fernzuhalten, die mich vom Schreiben abhalten. Ich verspreche, jetzt auch endlich den Keller aufzuräumen. Ganz bestimmt!

1 Bei jeder Erwähnung einer Gruppe von Personen ist die Liste alphabetisch nach Nachnamen sortiert.

Schließlich möchte ich dieses Buch unserer lieben Galina Ivanovna Tyumentseva widmen, die mich in diesem Projekt so sehr unterstützt hat und die leider während des Schreibens des Buchs von uns gegangen ist. Wir werden immer an dich denken.

#AdoptDontShop