Domain-Driven Design

Komplexe Software auf Basis fachlicher Modelle entwickeln

Domain-Driven Design stellt die fachliche Essenz der entstehenden Software in den Vordergrund, um die Qualität, Langlebigkeit, Erweiterbarkeit und Skalierbarkeit zu verbessern. Hierzu wird die Fachlichkeit mit klaren Begriffsdefinitionen und stringenter Struktur modelliert.

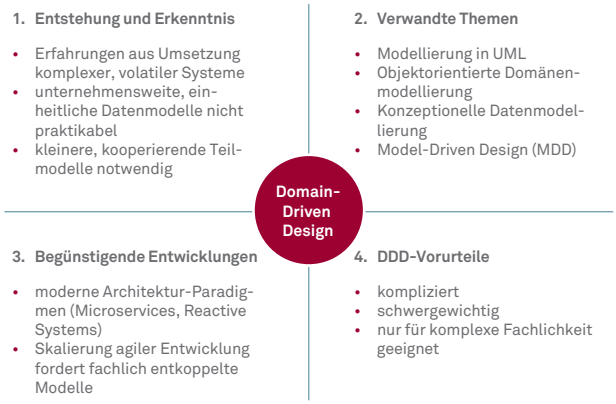
Definition

Eric Evans veröffentlichte 2003 ein Buch, das aus seinen Erfahrungen bei der Entwicklung komplexer und sich häufig ändernder Anwendungen entstanden ist. Diese Sammlung verschiedener Architekturmuster nannte er Domain-Driven Design (kurz DDD). Die Muster unterteilen sich in solche für strategisches Design, sprich dem Entwurf der Makroarchitektur, und taktisches Design, also der Verfeinerung der Mikroarchitektur.

Ausgehend von den konkreten Anforderungen beschreiben Fachexperten zusammen mit dem Entwicklungsteam die Kernkonzepte der Lösung in der Ubiquitous Language. Sie setzen die Kernkonzepte zueinander in Beziehung und gestalten sie in einer grafischen Modellierungssprache.

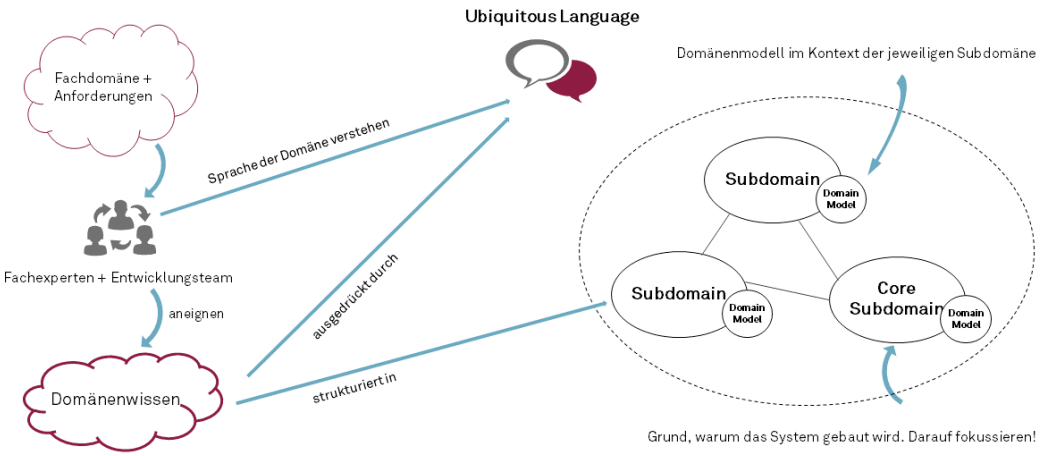
Im Unterschied zu herkömmlichen, schwergewichtigen oder gar unternehmensweiten Datenmodellen legt Domain-Driven Design Wert auf kleinere, klar voneinander abgegrenzte Domänenmodelle. Diese Subdomains wiederum unterteilen sich in eine oder wenige Coredomains und weitere Supporting Subdomains mit klaren Schnittstellen untereinander.

Die Coredomain ist das Herzstück der fachlichen Lösung und stellt den eigentlichen Mehrwert der Anwendung dar. Sie wird auf jeden Fall selbst entwickelt und muss daher gut verstanden, strukturiert und dokumentiert sein. Die anderen Subdomains sind meist generischer und für viele Anwendungen gleichermaßen nutzbar. Sie lassen sich oft als Open-Source- oder kommerzielle Komponenten respektive Dienste hinzufügen und müssen in ihrer inneren Struktur nicht im Detail bekannt sein. Subdomains können mittels Bounded Contexts weiter strukturiert werden. Die so entstehende fachliche Architektur der Anwendung wird durch Context-Maps visualisiert.



Referenzszenario

Domain-Driven Design eignet sich besonders zur Modellierung komplexer, hoch modularer und verteilter Informationssysteme.



Für diese Systeme stößt eine herkömmliche, dokumentenzentrierte Spezifikation sehr schnell an ihre Grenzen. Ebenso ist die Entwicklung eines einheitlichen, großen Datenmodells für die gesamte Anwendung wenig erfolgversprechend. Heute sollen solche Systeme zudem meist agil mit mehreren Teams realisiert werden, Stichwort skalierte Agilität. Damit diese Teams weitgehend autonom arbeiten können, ist die fachliche Entkopplung von Komponenten und Diensten zwingende Voraussetzung. DDD kann hier wertvolle Hilfestellung leisten. Wenn die technische Umsetzung mit einem modernen Architekturstil wie Microservices erfolgen soll, ist insbesondere die Anwendung der DDD-Konzepte Bounded Contexts und Anti-Corruption-Layer sinnvoll.

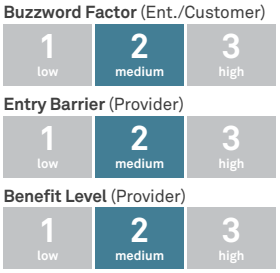
Potenzial

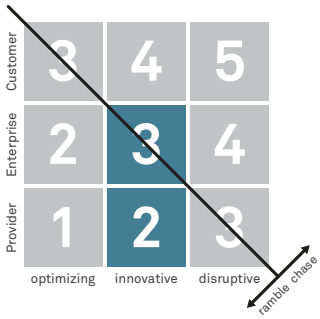
DDD erfährt nach Jahren des Schattendaseins derzeit verstärkte Aufmerksamkeit. Es setzt sich zunehmend die Erkenntnis durch, dass komplexe Systeme ohne fachliche Modellierung nicht zu beherrschen sind. Eine robuste, langlebige IT-Architektur benötigt als Basis eine durchdachte Facharchitektur (Coredomain und Subdomains in DDD), die wiederum auf wohldefinierten Kernkonzepten (Ubiquitous Language nach DDD) aufbaut.

Es erfordert einen nicht zu vernachlässigenden Einarbeitungsaufwand sowie Fähigkeiten an der Grenze zwischen Fachlichkeit und IT-Architektur, um Domain-Driven Design anwenden zu können. Insbesondere ist heute in vielen Projektteams nur mehr wenig Modellierunsg-Know-how hinsichtlich der Werkzeuge und Sprachen wie UML vorhanden. Dies ist jedoch eine notwendige Voraussetzung für DDD.

Reifegrad

Während viele Anwender die Vorteile von DDD derzeit neu entdecken, haben in den letzten Jahren einige Teams schlechte Erfahrungen mit der Anwendung von DDD gemacht. Selbst Eric Evans gibt freimütig zu, dass DDD zur Entwicklung großer Monolithen nicht sehr sinnvoll ist. Aktuelle IT-Trends, etwa Microservices oder Message-Driven Systems, lassen die Grundkonzepte von DDD jedoch wieder als attraktive Grundlage für die technische Umsetzung erscheinen. Zudem sind seit 2013 einfacher zugängliche Fachbücher und Schulungen zu DDD erschienen, die die Konzepte anschaulich erklären und den Bezug zu modernen Entwicklungsmethoden und Architekturstilen herstellen.





Marktübersicht

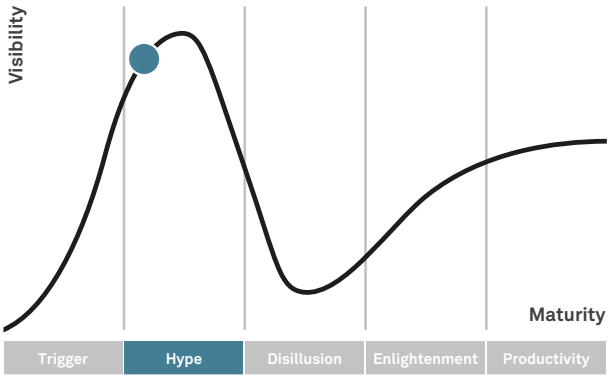
Einige Vorreiter und Evangelisten haben lesenswerte Bücher zu DDD verfasst und sind gefragte Redner auf Konferenzen. Das Originalbuch zu DDD stammt von Eric Evans. Weitere prominente Autoren und Redner sind Vaughn Vernon und Scott Millett.

Inzwischen gibt es jährliche Konferenzen, die sich ausschließlich mit dem Thema DDD befassen, namentlich *Explore DDD* in den USA und *DDD Europe*.

Neben lokalen Schulungen verschiedenster Anbieter gibt es unter anderem E-Learning-Angebote wie *Domain-Driven Design Distilled* von Addison-Wesley und *Event-Driven Microservices* von O'Reilly Media.

Alternativen

Zu den Kernkonzepten Ubiquitous Language, Coredomain, Bounded Context und Context-Maps gibt es keine Alternativen. Die übrigen, IT-näheren taktischen Muster lassen sich aber auf vielfältige Art durch andere architekturelle Lösungen ersetzen.



Pro	Contra
betont die Bedeutung von klaren und einheitlichen Definitionen in Teams	bisher nicht sehr populär, weil zu technisch für Business Analysts und zu abstrakt für Implementierer
Gegenentwurf zum unrealistischen Unternehmensdatenmodell	wenig Unterstützung durch Modellierungswerkzeuge
umfangreicher Werkzeugkasten von strategischen und taktischen Mustern	viele überflüssige Details verstellen den Blick auf die wenigen Kernkonzepte
	keine besonders aktive Community, jedoch einzelne Evangelisten

https://msg.direct/techrefresh

Stand: September 2018

msg systems ag

Robert-Bürkle-Straße 1 | 85737 Ismaning/München | Telefon: +49 89 96101-0 | Fax: +49 89 96101-1113 | www.msg.group | info@msg.group