CLOUD NATIVE COMPUTING

CLOUD-NATIVE

Unit:

Domain Driven Design

(1) Was ist das?

creative commons

CLOUD NATIVE COMPUTING

Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.

KAPITEL 14

Domain Driven Design

14.1 Fachlichkeit, Fachlichkeit, Fachlichkeit

14.2 Strategisches Design

- Subdomänen
- Ubiquitous Language
- Bounded Context
- Context Mapping

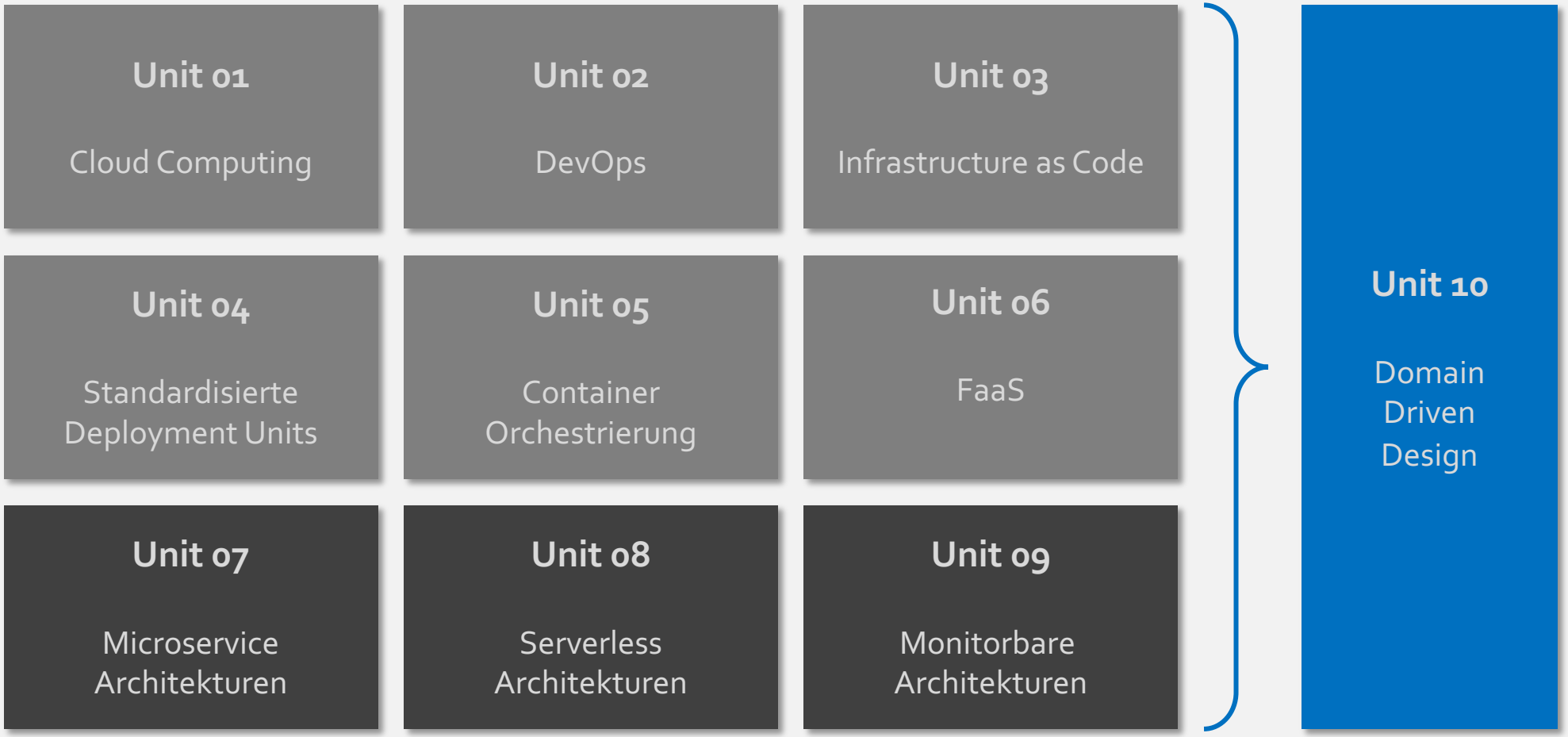
14.3 Taktisches Design

- Oft genutzte Pattern für Geschäftslogik
  (ETL, Active-Record, Domain Model, Event-Sourcing)
- Oft genutzte Architektur-Pattern
  (Layered Architecture, Ports & Adaptor, CQRS)

14.4 Zusammenfassung

INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



DOMAIN DRIVEN DESIGN

Wo sind wir nun?

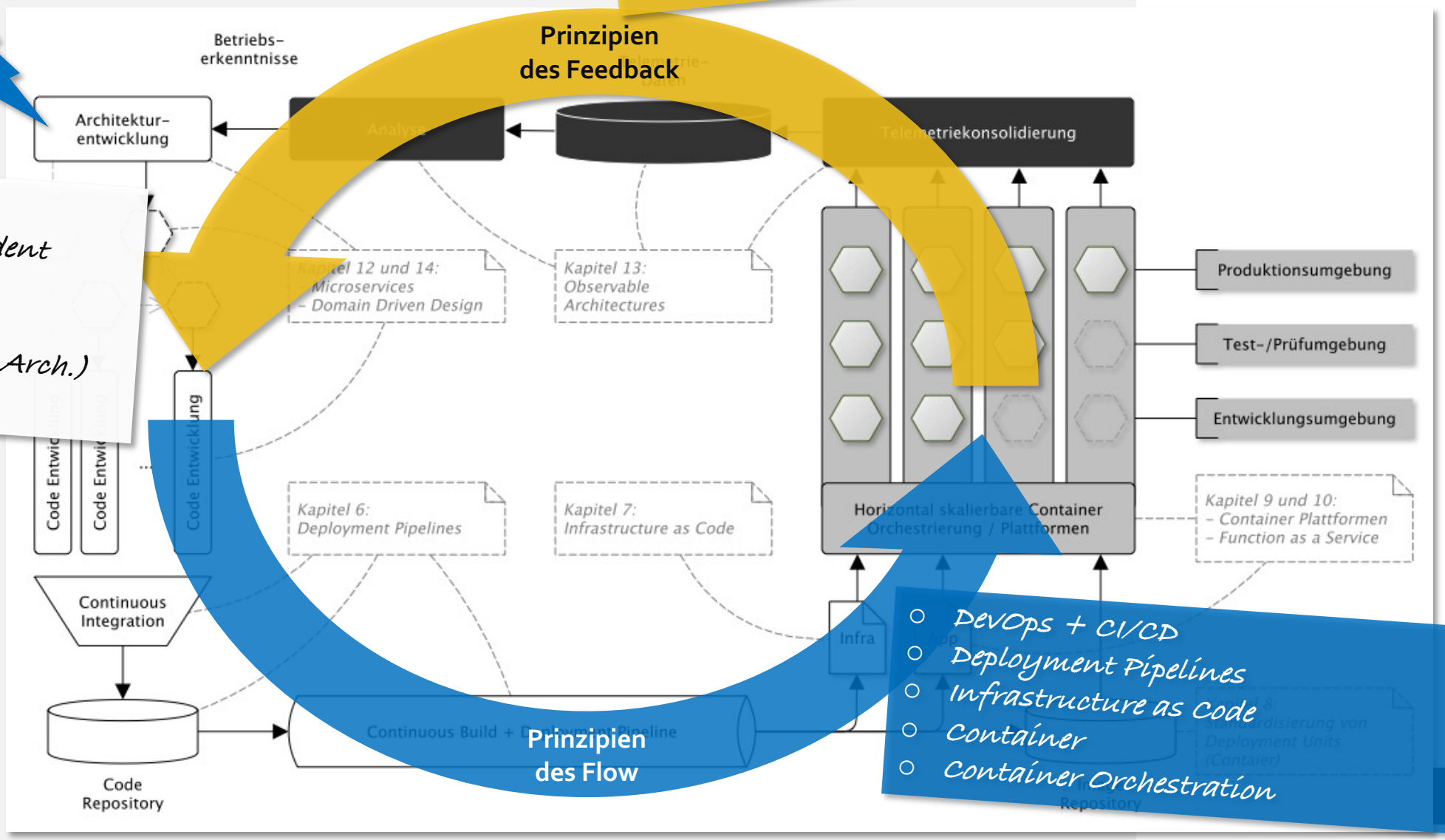
- Monitoring
- Logging
- Distributed Tracing
- Telemetry Data Consolidation

Evolutionäres Design

- Microservices (Independent Deployability)
- Lose Kopplung (REST, GraphQL, Event-Driven Arch.)
- Hohe Kohäsion

Frage:
Wie kommen wir methodisch zu Komponenten mit einer hohen Kohäsion?

Antwort:
Z.B. mit **Domain Driven Design**

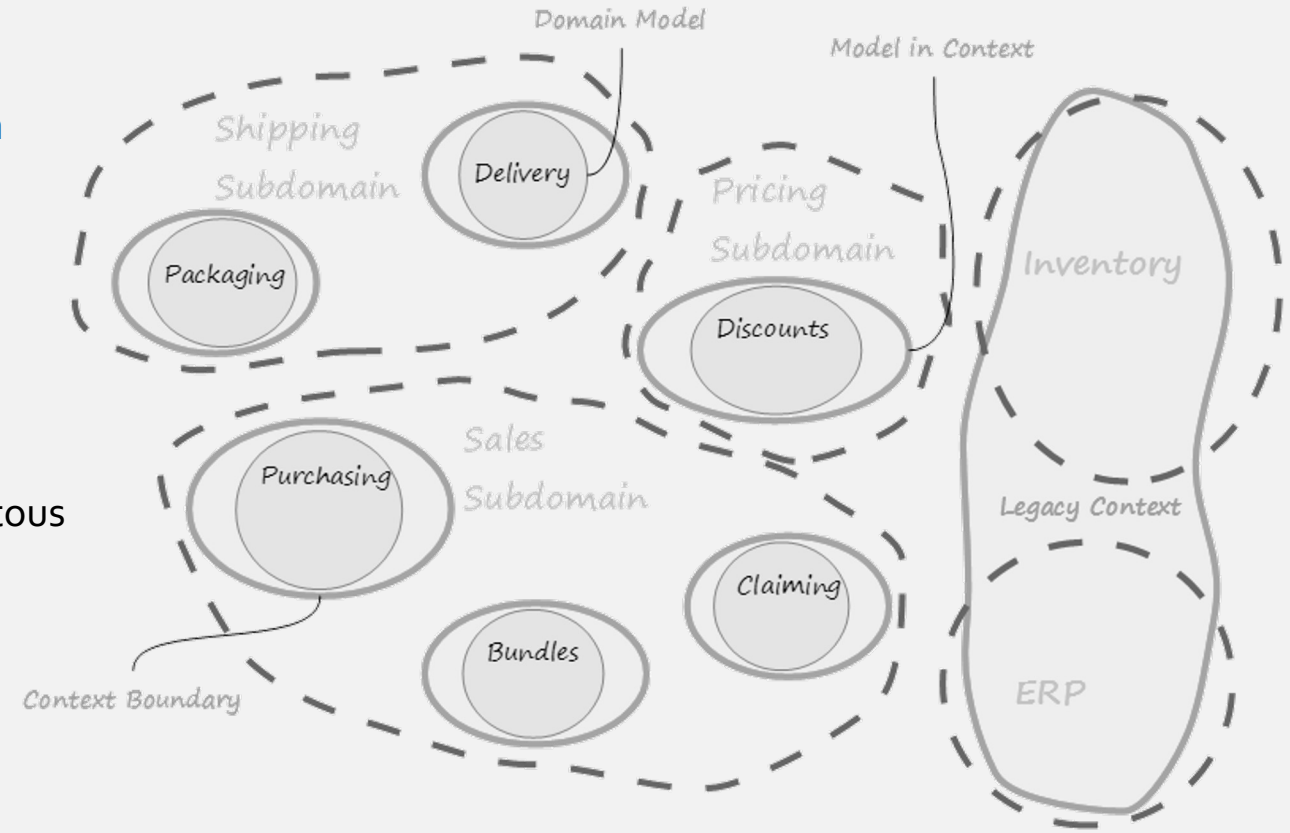


- DevOps + CI/CD
- Deployment Pipelines
- Infrastructure as Code
- Container
- Container Orchestration

INHALTE

Domain Driven Design

- Was ist das?
- Effektives Software Design
- Strategisches Design
- Taktisches Design



Strategisches Design

- Subdomains
- Bounded Context + Ubiquitous Language
- Context Mapping

Taktisches Design

- Business Logic Pattern
- Architectural Pattern

DOMAIN DRIVEN DESIGN

Auf der Suche nach einer Methodik zur Entwicklung Cloud-nativer Architekturen

CLOUD NATIVE COMPUTING

Wir erinnern uns an die Unit 07. Services in Cloud-nativen Systemen sollten u.a. folgenden Prinzipien folgen und folgende Eigenschaften haben:

- **Dekomposition** mittels Services
- **Evolutionäres Design** und Entwicklung des Gesamtsystems
- **Lose Kopplung** von Services (Standardformate, z.B. JSON/XML, Standarddatentypen, Messaging)
- Hohe Kohäsion innerhalb von Services (**Single-Responsibility Prinzip**)
- Services sollten unabhängig von einander austausch- und aktualisierbar sein (**Independent Deployability**)
- Conways Law => Services sollten als langlebende Produkte und nicht Projekte als verstanden wissen (**You build it, you run it**)

Noch mehr als die Programmierung einzelner Services ist die Entwicklung einer tragfähigen Architektur eine „Kunst".

Im Zusammenhang mit Microservice Architekturen wird aber immer wieder eine Methodik erwähnt, die (mit einer höheren Wahrscheinlichkeit) Architekturen erzeugt, die gut auf die in Unit 7 genannten Erfordernisse abgestimmt sind.

Insbesondere jüngeren SW-Architekten sei daher die Methodik des Domain Driven Design (DDD) ans Herz gelegt, obwohl DDD weder auf Cloud-native Systeme oder Microservice Architekturen festgelegt ist und auch in anderen Kontexten der Entwicklung tragfähiger SW-Architekturen eingesetzt werden kann.

DOMAIN DRIVEN DESIGN

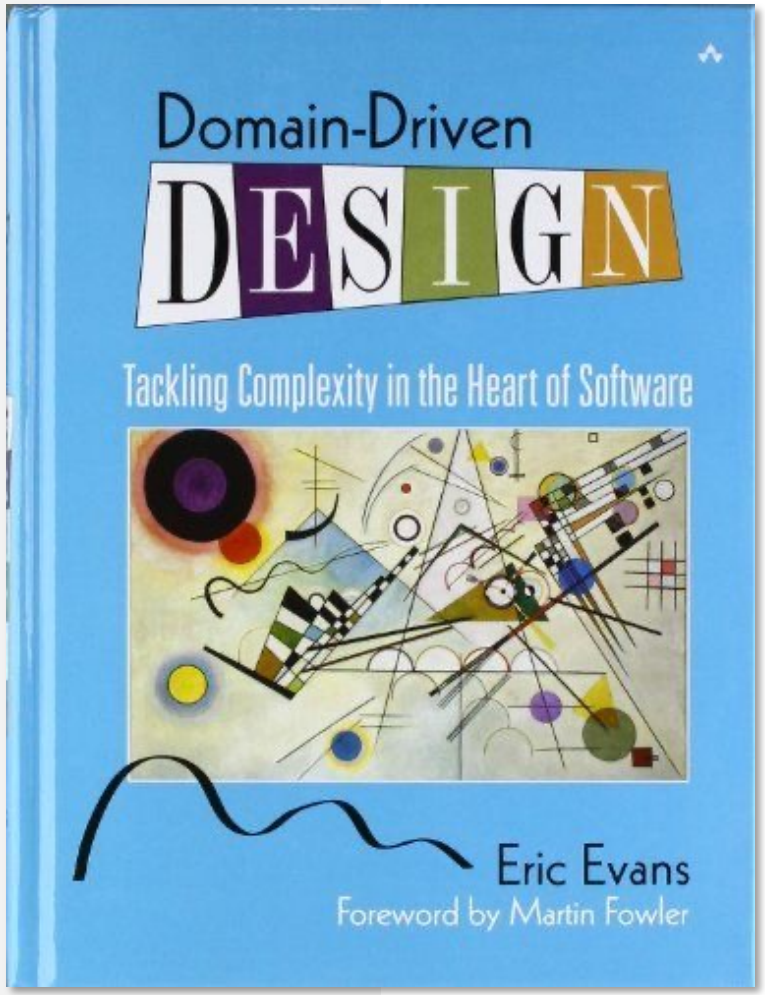
Fachlichkeit auf Basis eines Domänenmodells

Domain-driven Design (DDD) ist eine Methodik zur Modellierung komplexer Software. Der Begriff „Domain-driven Design" wurde 2003 von Eric Evans in seinem gleichnamigen Buch geprägt (also deutlich bevor es den Begriff Cloud-native und entsprechende Systeme überhaupt gab).

Domain-driven Design basiert auf folgenden zwei Annahmen:

- Der Schwerpunkt des Softwaredesigns liegt auf der **Fachlichkeit** und der Fachlogik.
- Der Entwurf komplexer fachlicher Zusammenhänge sollte auf einem Modell der Anwendungsdomäne, dem **Domänenmodell** basieren.

DDD orientiert sich an agiler Softwareentwicklung, ist aber SW-Entwicklungsprozess-agnostisch. Eine iterative Softwareentwicklung und eine enge Zusammenarbeit zwischen Entwicklern und Domänen-Experten ist jedoch erforderlich.



DOMAIN DRIVEN DESIGN

Effektives Design

- Domain-driven Design (DDD) strebt dabei nicht nach einem perfekten sondern nach einem effektiven Design.
- Effektives Design erfüllt die Anforderungen einer Firma/einer Organisation/einer Einrichtung bis zu einem Grad, der notwendig ist, damit man sich mittels digitalisierter Prozesse von Mitbewerbern/in Benchmarks abheben kann.
- Effektives Design zwingt Organisationen dazu, zu erkennen, worin sie sich hervortun müssen und sich auf diese Bereiche zu fokussieren.
- Und nur in diesen Bereichen wird effektives Design eingesetzt, um ein Domänen-konformes Softwaremodell zu entwickeln und fortzuschreiben.
- Andere nicht Domänen-spezifische Bereiche versucht man durch Standardsoftware oder mittels externer Services abzudecken.

„Die meisten Menschen machen den Fehler zu denken, dass es bei Design nur darum geht wie es aussieht. Die Leute denken, es ist diese Fassade – dass man den Designern einen Kasten übergibt und sagt: "Macht den hübsch!" Das ist nicht unser Verständnis von Design. Es geht nicht nur darum, wie etwas aussieht und sich anfühlt. Design ist wie etwas funktioniert.“

Steve Jobs

DOMAIN DRIVEN DESIGN

Fokus auf die Fachlichkeit

- Software dient zur Umsetzung von Geschäftsanforderungen (nicht umgekehrt)
- Fachlichkeit ist langlebiger als Technologien und bringt somit Stabilität in Softwarearchitekturen
- Fachlichkeit ist der gemeinsame Nenner aller am Entwicklungsprozess Beteiligten
- Domänenwissen begleitet die Softwareentwicklung, d.h. Softwarentwicklung startet und endet mit dem Wissen von Domänen-Experten

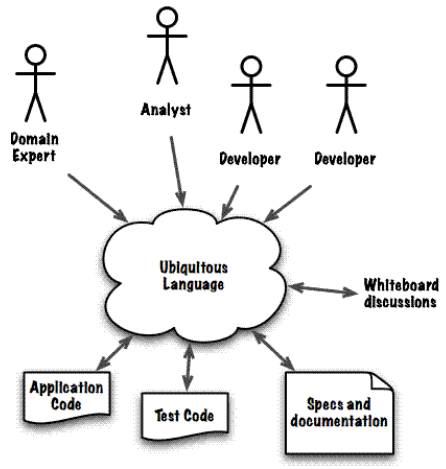
Bei DDD geht es um

- Sprache
- Zusammenarbeit
- Methoden und Verfahren
- Änderung von Denkmustern

DOMAIN DRIVEN DESIGN

Ziele

- **DDD ist eine Philosophie** zur Ausrichtung der SW-Entwicklung an der Fachlichkeit
- DDD beschreibt einen Prozess
- DDD liefert unterstützende Verfahren/Techniken



Fachlichkeit und SW korrespondieren nachvollziehbar

- Durch die Methode
- In welchem SW-Baustein ist diese Fachlichkeit umgesetzt?
- Für welche Fachlichkeit ist dieser Baustein zuständig?

SW-Lösung skaliert mit der Systemgröße

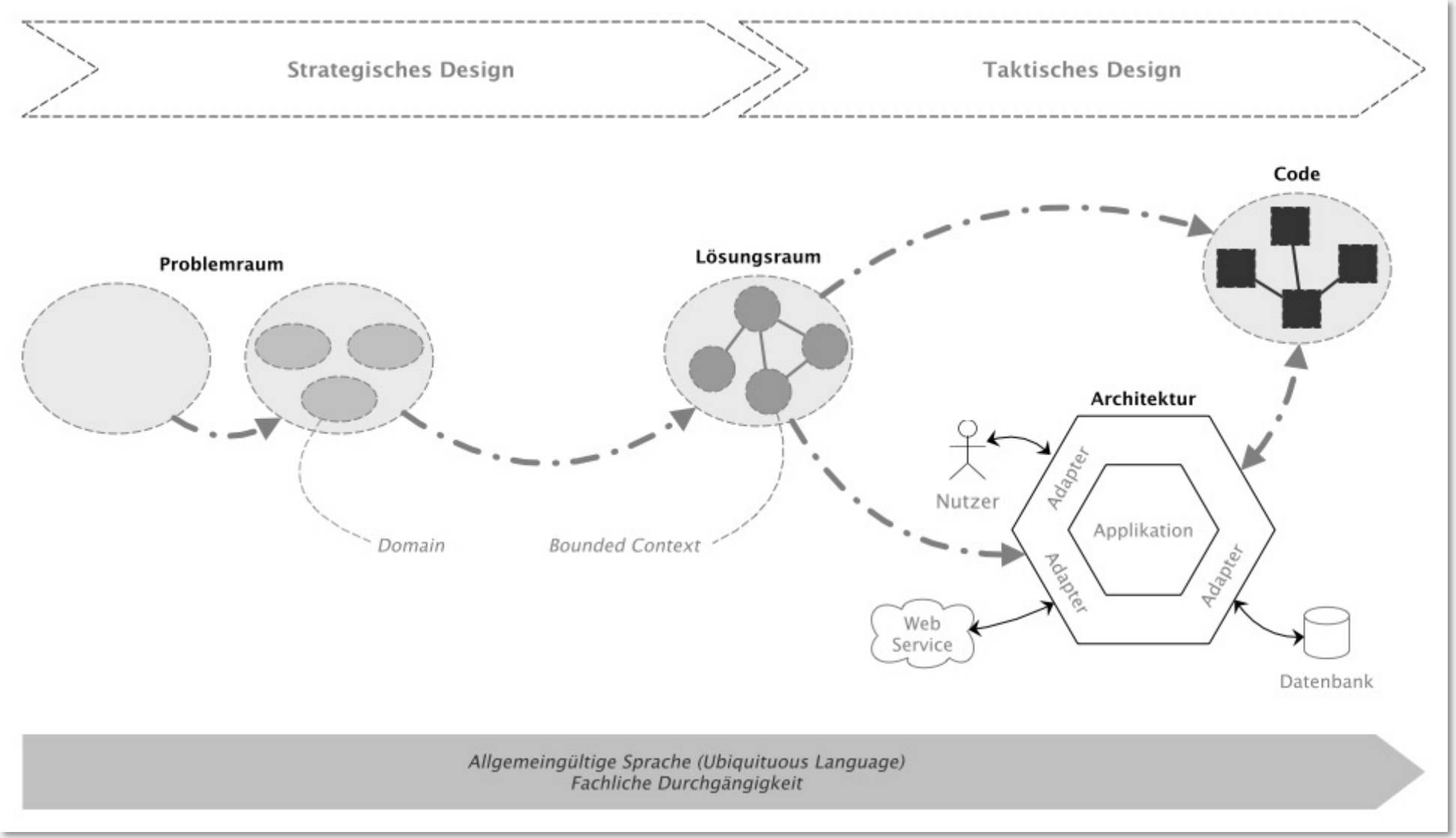
- Integrationskomplexität einer fachlichen Änderung darf nicht mit der Größe eines Systems wachsen
- Wachsende Systeme bleiben wartbar

Zusammenarbeit steht im Mittelpunkt

- Fachexperten und Entwicklung arbeiten Hand in Hand
- Fachexperten und Entwicklung kommunizieren technologiefrei
- Die Zusammenarbeit von Fachexperten und Entwicklung vereint unterschiedliche Fähigkeiten

DOMAIN DRIVEN DESIGN

Fachlichkeit als Treiber der Softwareentwicklung



DOMAIN DRIVEN DESIGN

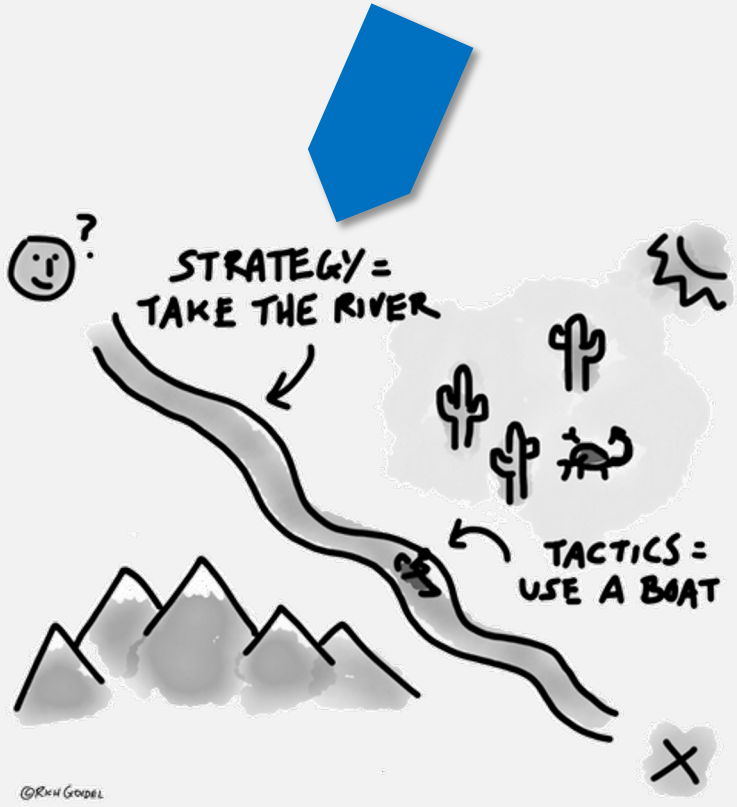
Wie findet man methodisch raus was wichtig ist? => Strategisches Design

Strategisches Design

- Was ist wichtig für die Zielerreichung (das Geschäft)?
- Wie teilt man Arbeit anhand von Prioritäten auf?
- Wo fasst man Dinge zusammen?

Vorgehensweisen

- Domain Matter Experts mittels einer gemeinsamen Sprache einbeziehen (Ubiquitous Language)
- Komplexe Domänenmodelle aufteilen (mittels Bounded Contexts und Subdomains)
- Subdomains mittels Context Mapping und definierter Beziehungen integrieren



Daraus werden letztlich Services definiert und gem. Conways Law Teams gebildet

Nicht erschrecken: DDD hat Ähnlichkeiten mit militärischen Vorgehensweisen.

Strategisches Design schneidet Services anhand von Fachlichkeit.

DOMAIN DRIVEN DESIGN

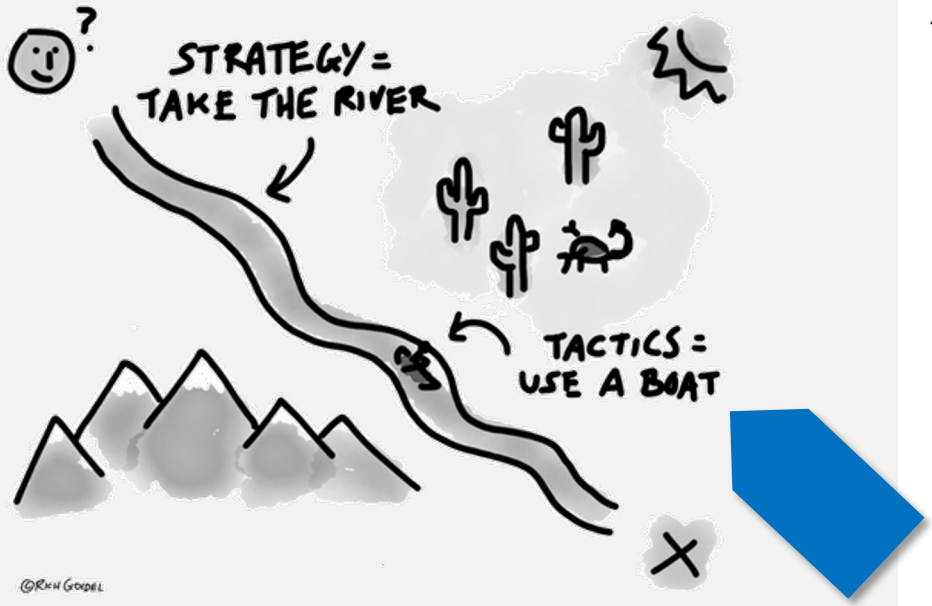
Wie bricht man methodisch herunter was wichtig ist? => Taktisches Design

Taktisches Design

- „Der dünne Pinselstrich, um die feinen Details des Domänenmodells zu zeichnen" (Vaughn Vernon)
- Ausarbeitung des Domänenmodells in wertschöpfenden Bereichen

Vorgehensweisen

- Arbeiten im Inneren eines Bounded Contexts
- Entities und Value Objekts in Aggregaten zusammenfassen
- Service Interaktionen über die Grenzen von Bounded Contexts mittels Domain Events modellieren und realisieren



Daraus werden letztlich Schnittstellen zwischen Services definiert

Nicht erschrecken: DDD hat Ähnlichkeiten mit militärischen Vorgehensweisen.

Taktisches Design arbeitet einzelne Services (Bounded Contexts) im Detail aus.

DOMAIN DRIVEN DESIGN

DDD is overrated

Ein Wort der „Warnung“

„There is a life beyond DDD. Not every good design needs to be Domain-Driven (though I can accept it should always be driven by the domain, just not necessarily in the DDD sense). ***You can design good systems even if you’re not a DDD expert.“***

Stefan Tilkov
Blog Post

Stefan Tilkov
Geschäftsführer und Principal Consultant bei INNOQ; Strategische Beratung im Umfeld von Software-Architekturen

Siehe auch: Stefan Tilkov, **GOTO 2019:** Good Enough Architectures, https://www.youtube.com/watch?v=PzEox3szeRc

AUSBLICK

Domain Driven Design

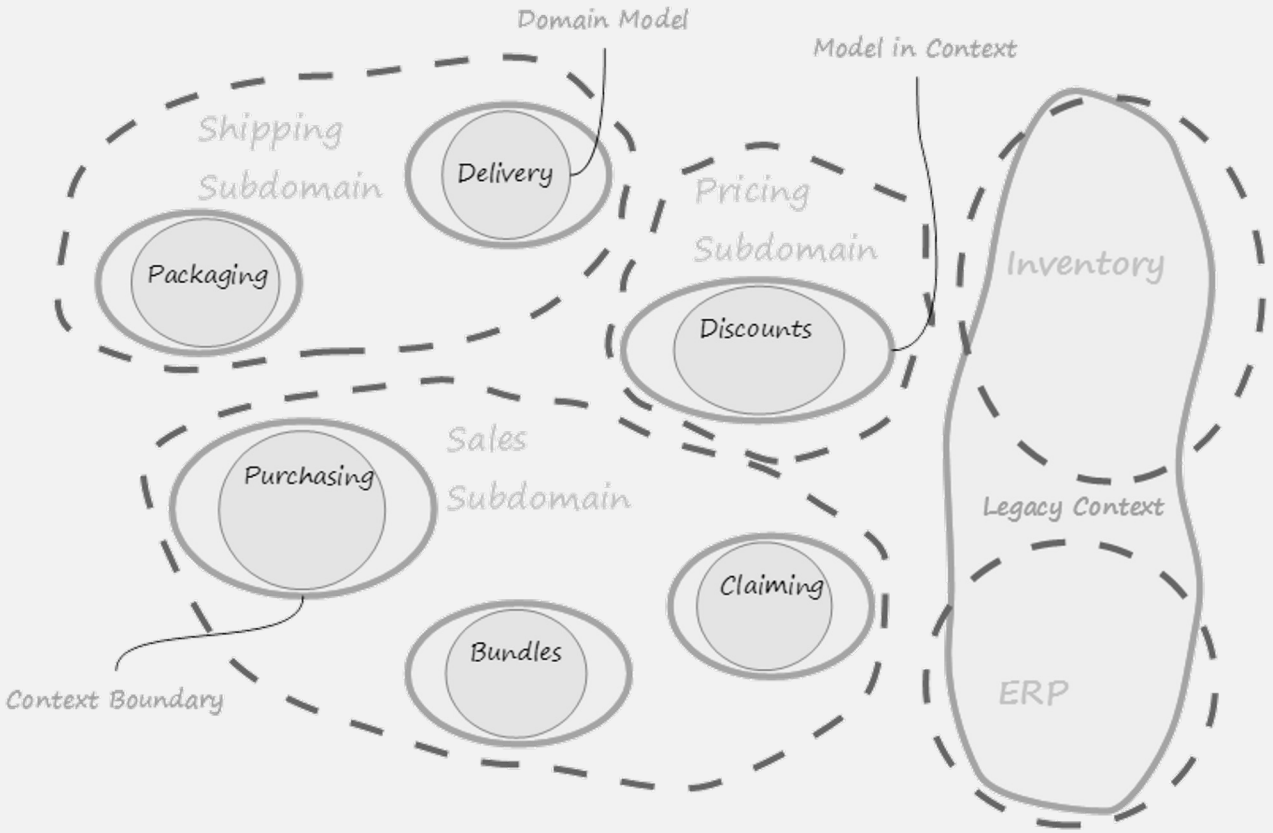
- Was ist das?
- Effektives Software Design
- Strategisches Design
- Taktisches Design

Strategisches Design

- Subdomains
- Ubiquitous Language
- Bounded Context
- Context Mapping

Taktisches Design

- Business Logic Pattern
- Architectural Pattern



KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de