CLOUD NATIVE COMPUTING

CLOUD-NATIVE

Unit:

Domain Driven Design

(5) Taktisches Design

Architektur-Pattern

PROF. DR.
NANE KRATZKE

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

INHALTE

Domain Driven Design

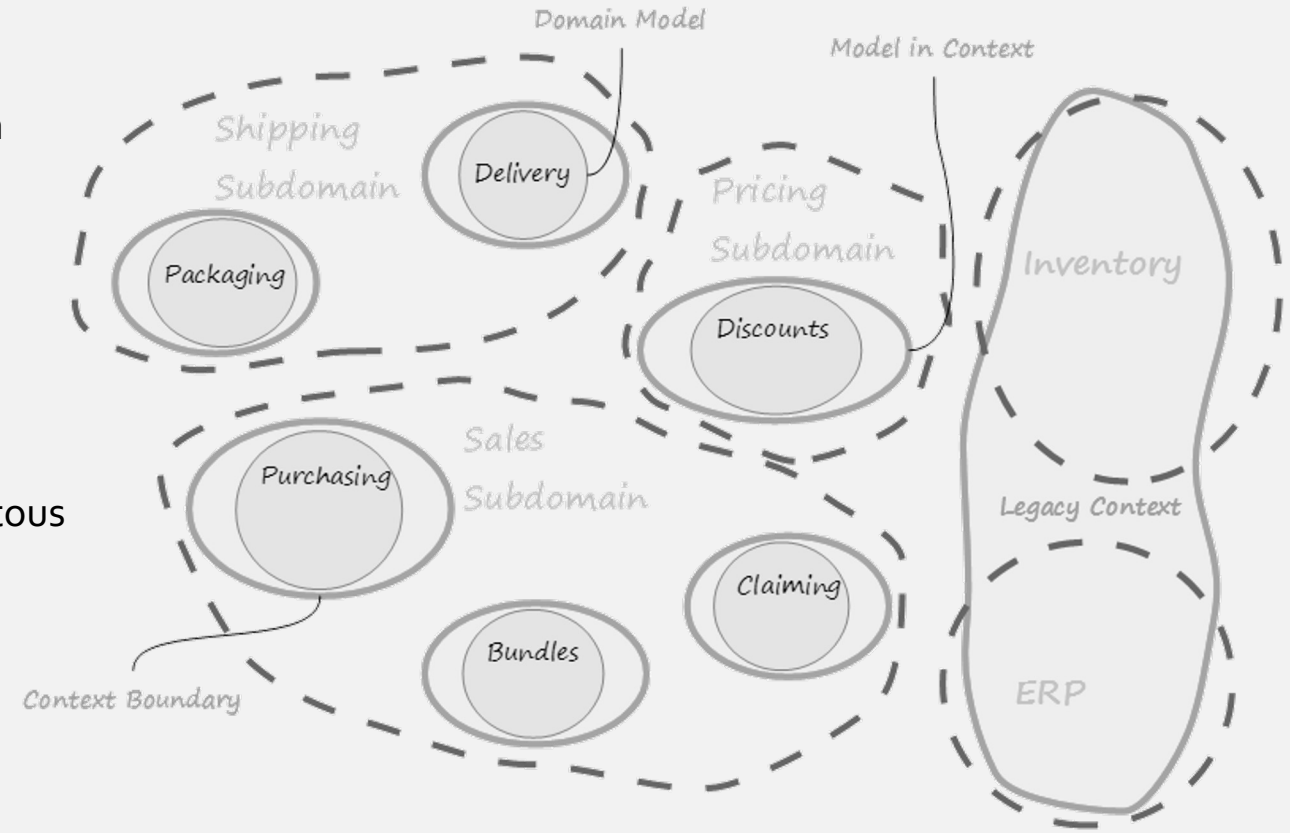
- Was ist das?
- Effektives Software Design
- Strategisches Design
- Taktisches Design

Strategisches Design

- Subdomains
- Bounded Context + Ubiquitous Language
- Context Mapping

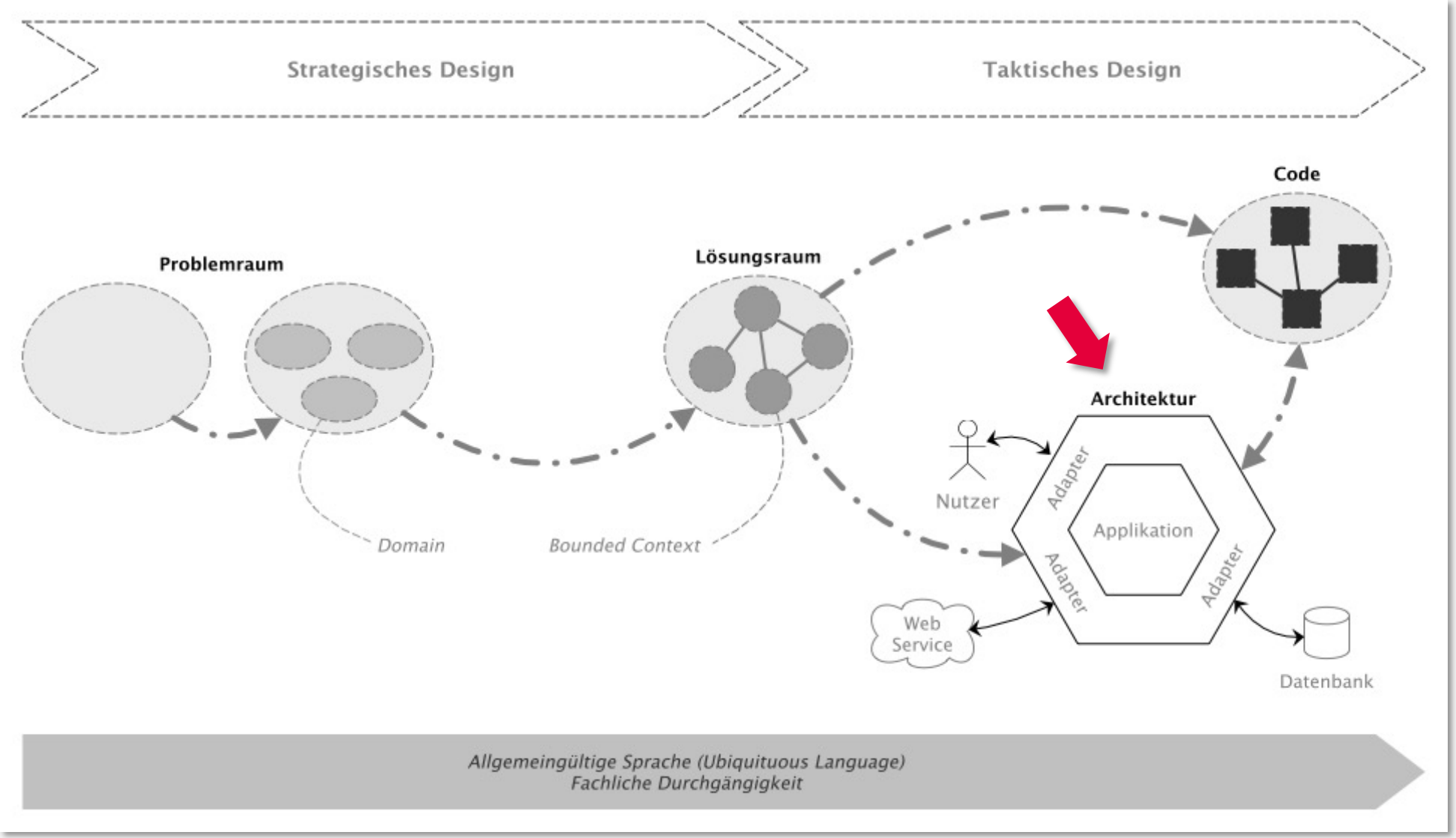
Taktisches Design

- Business Logic Pattern
- Architectural Pattern
- Fallstudie: Entwurf eines Uptime-Service



DOMAIN DRIVEN DESIGN

Fachlichkeit als Treiber der Softwareentwicklung



TACTICAL DESIGN

Architektur-Pattern

CLOUD NATIVE COMPUTING

Welches Architekturmuster verwendet werden soll, ist eine wichtige taktische Designentscheidung. Das richtige Muster unterstützt die Umsetzung der funktionalen und nicht-funktionalen Anforderungen des Systems.

Wir werden uns hierzu drei häufig anzutreffende Architekturmuster und geeignete Anwendungsfälle hierfür ansehen.

Oft genutzt in:

1. Layered Architecture — *Supporting / ggf. Core*
2. Ports & Adapters — *Core / ggf. Supporting*
3. CQRS (Command-Query-Responsibility-Segregation) — *Core (Stateful)*

OK! Ähh, was für ein Boot?

- *Speedboot? Schlauchboot?*
- *Kanu?*
- *Floß?*
- *...*

TACTICAL DESIGN

Layered Architecture

Präsentiert und definiert die Benutzeroberfläche

Implementiert die Geschäftslogik

Zugriff auf Persistenz-mechanismen (DBen und andere Infrastruktur-komponenten)

Durch die Abhängigkeit zwischen der Geschäftslogik und den Datenzugriffsschichten passt dieses Architekturmuster gut zu einem System, dessen Geschäftslogik mit dem Active-Record-Pattern implementiert wurde.

*Das Pattern ist daher häufig in Bounded Contexts für **Supporting Domains** anzutreffen!*

Bildquelle: Vladik Khononov, *What Is Domain-Driven Design?* O'Reilly Media, 2019

CLOUD NATIVE COMPUTING

Ein Klassiker!
Vielleicht sogar DER Klassiker!

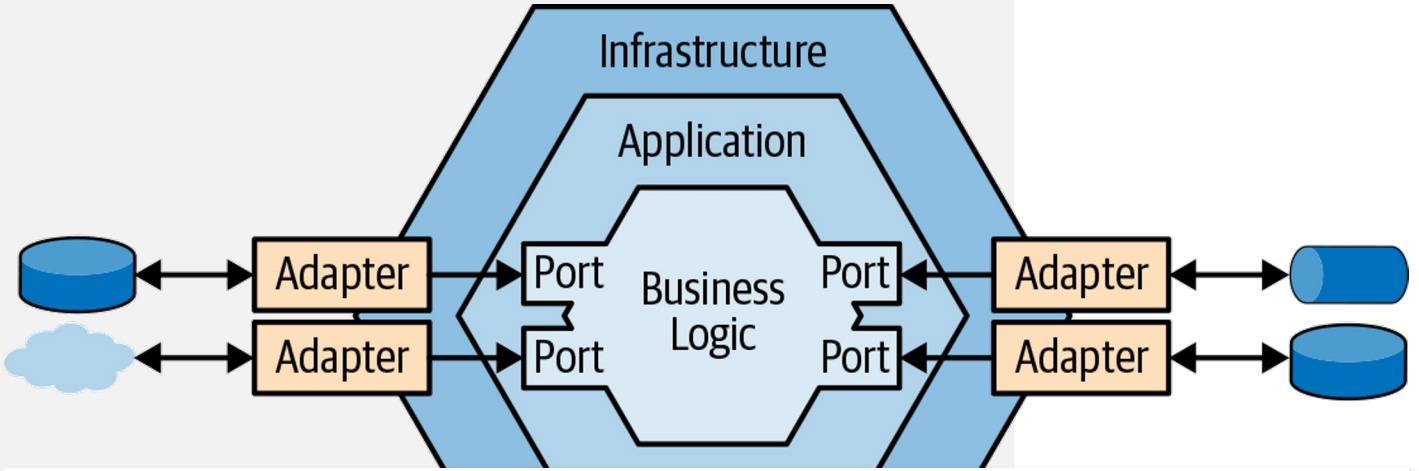
If the only tool you have is a hammer, **you tend to see every problem as a nail**

Abraham Maslow

TACTICAL DESIGN

Ports & Adapters (auch Hexagonale Architektur)

- Das Ports & Adapter-Pattern strebt die Zerlegung der Codebasis nach technologischen Gesichtspunkten an
- Ähnlichkeit zum Layered Architecture Pattern
- Unterschiede:
  - **Terminologie:** "Infrastruktur" bezeichnet sowohl Benutzeroberfläche, Datenzugriff und andere infrastrukturelle Belange
  - **Abhängigkeiten:** Geschäftslogik im Zentrum, diese soll nicht direkt von Infrastruktur abhängig sein
  - **Anwendungsschicht:** Fassade für öffentliche Schnittstellen, orchestriert Geschäftslogik



Integration von Infrastrukturkomponenten

Ziel der Ports & Adapter-Architektur ist es, die Geschäftslogik des Systems von den Infrastrukturkomponenten zu entkoppeln.

Anstatt die Infrastrukturkomponenten (z. B. eine konkrete DB) direkt zu referenzieren und aufzurufen, definiert die Geschäftslogikschicht "Ports", die von der Infrastrukturschicht implementiert werden müssen. Die Infrastrukturschicht implementiert "Adapter" der Ports für die Arbeit mit verschiedenen Technologien. Die Anwendungsschicht liefert die Adapter für die Ports der Geschäftslogik durch Dependency Injection.

Die Entkopplung der Geschäftslogik von allen technologischen Belangen macht das Ports & Adapter Pattern vor allem für Geschäftslogiken, die auf dem **Domain Model Pattern** implementiert werden, interessant.

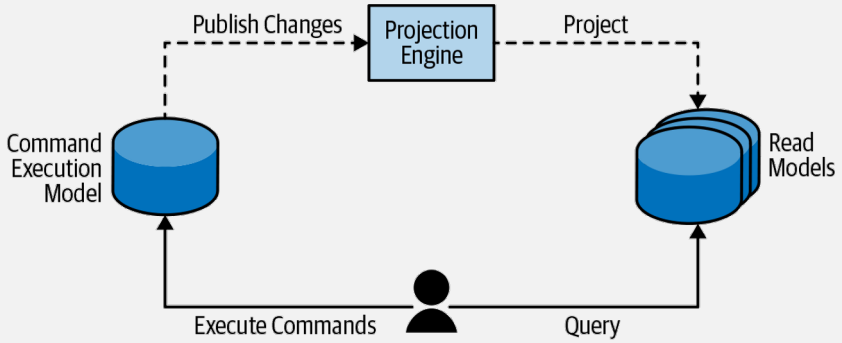
Das Pattern ist daher häufig in Bounded Contexts für **Core Subdomains** anzutreffen!

Bildquelle: Vladik Khononov, *What Is Domain-Driven Design?* O'Reilly Media, 2019
Siehe auch: Alistair Cockburn - https://alistair.cockburn.us/hexagonal-architecture/

TACTICAL DESIGN

CQRS (Command-Query Responsibility Segregation)

CQRS basiert auf einem einzigem Modell zur Ausführung von Operationen, die den Zustand des Systems verändern (Systembefehle) . Gem. dem CQRS-Pattern ist dieses Command Execution Modell ist das einzige Modell, das stark konsistente Daten repräsentiert (Single Source of Truth).



Das System kann beliebig viele Modelle mittels Projektionen erzeugen, die für die Darstellung von Daten für Benutzer oder andere Systeme erforderlich sind. Diese Modelle sind Read-only. Keine der Operationen des Systems kann die Daten der gelesenen Modelle direkt ändern.

Modell-Trennung

In der CQRS-Architektur sind die Zuständigkeiten der Modelle des Systems nach ihrem Schreib-Lese-Charakter getrennt. Ein Command kann nur auf dem stark konsistenten Befehlsausführungsmodell operieren. Eine Query kann keinen der Systemzustände direkt verändern - weder die Lesemodelle noch das Befehlsausführungsmodell.

Polyglotte Persistenz

In Microservice-Architekturen ist eh häufig ein polyglotte Persistenz existent. D.h. es werden mehrere Datenbanken für unterschiedliche Anforderungen an den Datenzugriff verwendet werden. Ein einzelnes System könnte zum Beispiel eine Dokumentendatenbank (document store) als operative Datenbank, einen Column Store für Analysen/Berichte und eine Suchmaschine für die Implementierung von Suchfunktionen verwenden.

Use Cases

CQRS eignet sich insbesondere für ereignisgesteuerte Domänenmodelle.

Reines Event-Sourcing lässt es nicht zu, Datensätze basierend auf den Zuständen der Aggregate abzufragen, aber CQRS kann dies ermöglichen, indem die Zustände in abfragbare Datenbanken projiziert werden.

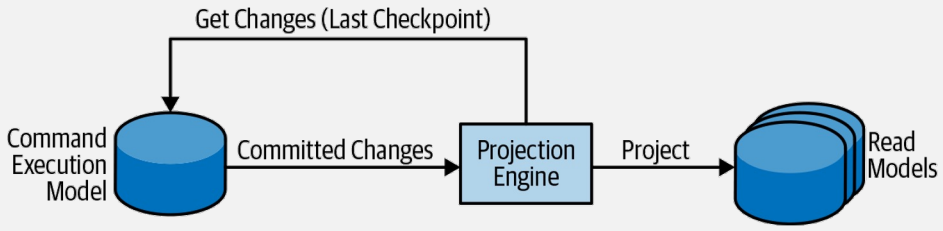
*CQRS ist daher häufig in Bounded Contexts mit **Event Sourcing** von **Core Subdomains** anzutreffen!*

Siehe auch: Greg Young, CQRS Documents - https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf

TACTICAL DESIGN

CQRS (Projections)

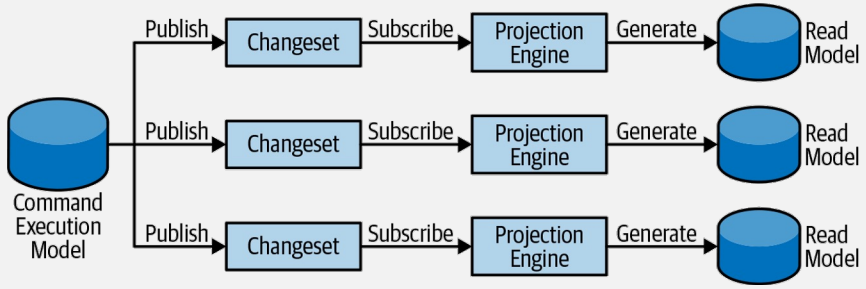
SYNCHRONE PROJEKTIONEN



Verfahren:

- Die Projektions-Engine fragt das das Command Execution Model nach neuen Datensätzen ab
- Die Projektions-Engine verwendet die aktualisierten Daten, um die Read Modells des Systems zu aktualisieren
- Die Projektions-Engine speichert den Checkpoint des zuletzt verarbeiteten Datensatzes
- Dieser Wert wird bei der nächsten Iteration verwendet, um Datensätze abzurufen, die nach dem letzten verarbeiteten Datensatz hinzugefügt oder geändert wurden.

ASYNCHRONE PROJEKTIONEN



Verfahren:

- Das Command Execution Model veröffentlicht alle bestätigten Änderungen mittels eines Publish/Subscribe Messaging Bus.
- Die Projektions-Engines des Systems können die veröffentlichten Nachrichten abonnieren und sie zur Aktualisierung der Projektionen der Read Models verwenden.

Tipp:
Es ist ratsam eine synchrone Projektion als Basis zu nutzen und nur bei Bedarf eine zusätzliche asynchrone Projektion zu ergänzen.

Bildquelle: Vladik Khononov, *What Is Domain-Driven Design?* O'Reilly Media, 2019

AUSBLICK

Domain Driven Design

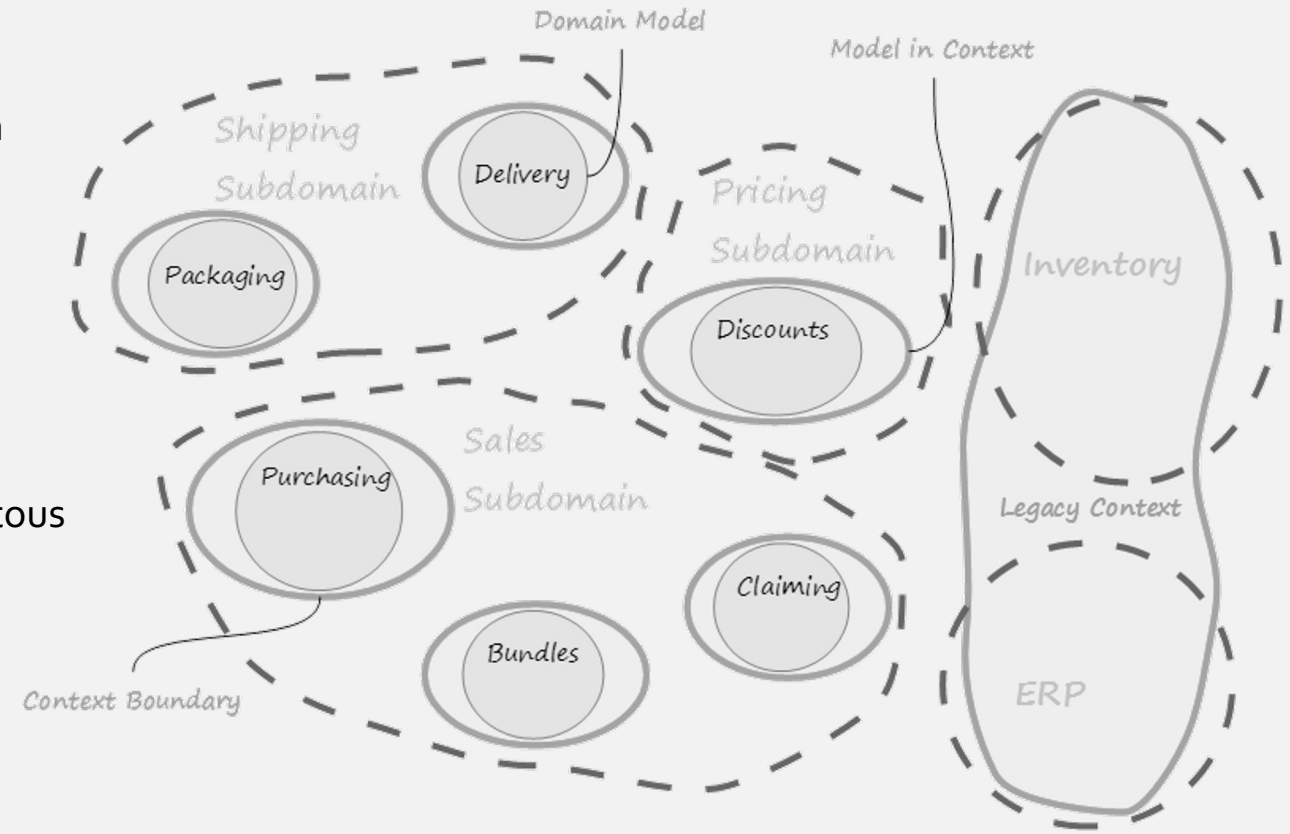
- Was ist das?
- Effektives Software Design
- Strategisches Design
- Taktisches Design

Strategisches Design

- Subdomains
- Bounded Context + Ubiquitous Language
- Context Mapping

Taktisches Design

- Business Logic Pattern
- Architectural Pattern
- Fallstudie: Entwurf eines Uptime-Service



KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de