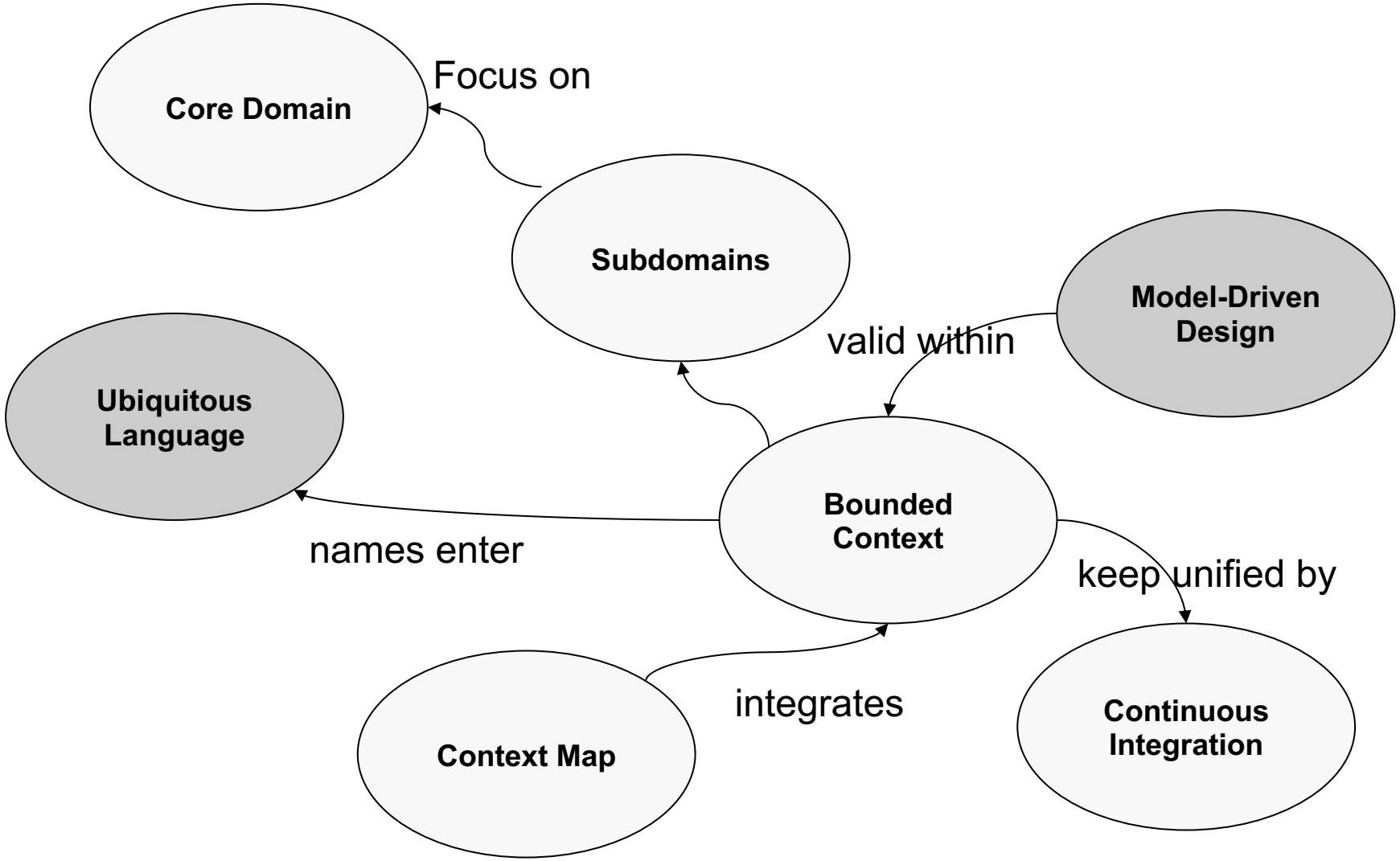
PROFESSIONAL SOFTWARE ENGINEERING

PSE SWE LE 4 und 5 - Domain Driven Design

Vertiefung Strategisches Domain Driven Design

Dominik Neumann

STRATEGISCHE MUSTER IM ÜBERBLICK



„**Tightly relating the code to an underlying model** gives the code meaning and makes the model relevant.“ Eric Evans

STRATEGISCHES DESIGN MIT SUB-DOMAINS

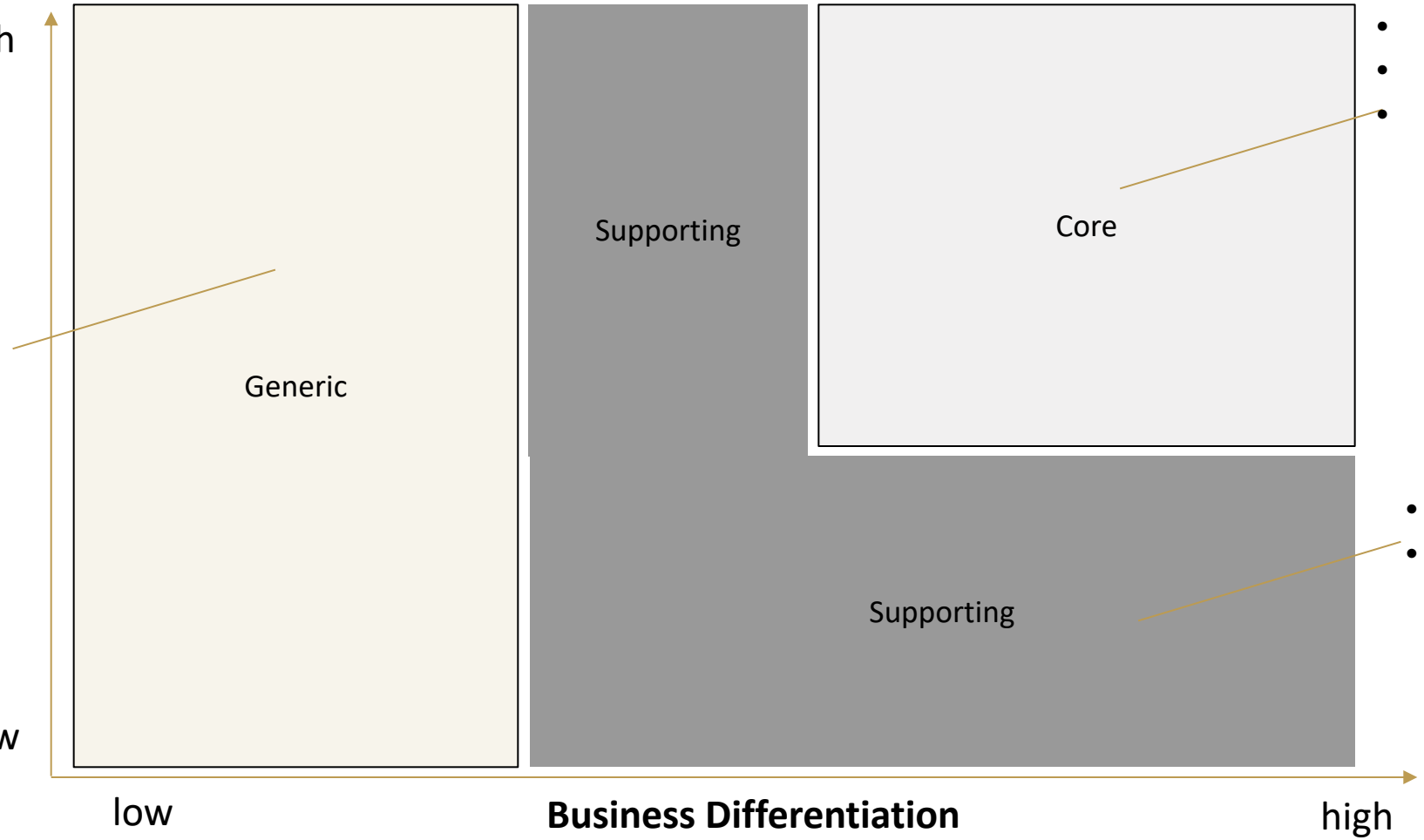
STRATEGISCHES DESIGN MIT SUB-DOMAINS

Wir unterscheiden drei Arten von Subdomains:

- Core Subdomain. (manchmal auch Core Domain genannt.) Das ist unsere Kerndomäne. Sie ist für das Unternehmen am wichtigsten! (oder für das Projekt)
- Supporting Subdomain. Ist nicht die Core Domain. Ist alles, was nicht entweder Core Domain oder Generic Subdomain ist.
- Generic Subdomain. Lösungen für Generic Subdomains kann man in der Regel von der Stange kaufen (COTS – Commercial of the Shelf Lösungen). Dafür gibt es Standardsoftware und eine Individualentwicklung ist nicht wirklich ratsam.

STRATEGISCHES DESIGN MIT SUB-DOMAINS

Knowledge Foundation
@ Reutlingen University



- competitive advantage
- high volatility
- companies continuously innovate and evlove core subdomains

- do not provide any competitive advantage
- all companies are performing in the same way

- do not change often
- do not provide any competitive advantage

Quelle: Vlad Khononov, Learning Domain-Driven Design 2021

STRATEGISCHES DESIGN MIT SUB-DOMAINS

Subdomain type	Competitive advantage	Complexity	Volatility	Sourcing	Problem
Core	Yes	High	High	In-house	Interesting
Generic	No	High	Low	Buy/adopt	Solved
Supporting	No	Low	Low	In-house / outsource	Obvious

Quelle: Vlad Khononov, Learning Domain-Driven Design 2021

STRATEGISCHES DESIGN MIT SUB-DOMAINS

Wie identifizieren wir Subdomain-Grenzen:

- Organisation: Org-Charts → Orgnisations-Design
- Prozesse: Prozess-Diagramme → Geschäftsprozess-Management
- Kohärente Use Cases → Use Cases
- Kohärente Business Capabilities → Business Capability Mapping
- Pivotal Events aus dem EventStorming → Strategisches EventStormimg
- Sprachgrenzen → Bounded Context


Knowledge Foundation
@ Reutlingen University
BOUNDED CONTEXT &
UBIQUITOUS LANGUAGE

Software architecture is the art of drawing line that I call boundaries.

- Robert C. Martin

BOUNDED CONTEXT

Software Architecture:

- Boundaries separate (software) elements from one another and restrict those on one side from knowing those on the other side.

(Clean Architecture, p. 160, Robert Martin)

System Science:

- One way we know we are looking at (perceiving) an object is that we see a boundary that differentiates the object from its environment.

(Principles of System Science, p. 190, Mobus & Calton)

- Unterscheidung ist perfekte Be-Inhaltung: Eine Unterscheidung wird getroffen, indem eine Grenze mit getrennten Seiten so angeordnet wird, dass ein Punkt auf der einen Seite die andere Seite nicht erreichen kann, ohne die Grenze zu kreuzen.

(Laws of Form, p. 1, Spencer Brown)

BOUNDED CONTEXT

- Bounded Context is the delimited applicapility of a particular model.
- Bounded Contexts give team member a clear and shared understanding of what has to be consitent and what can develop independently.

(DDD p.511 Glossary, Eric Evans)

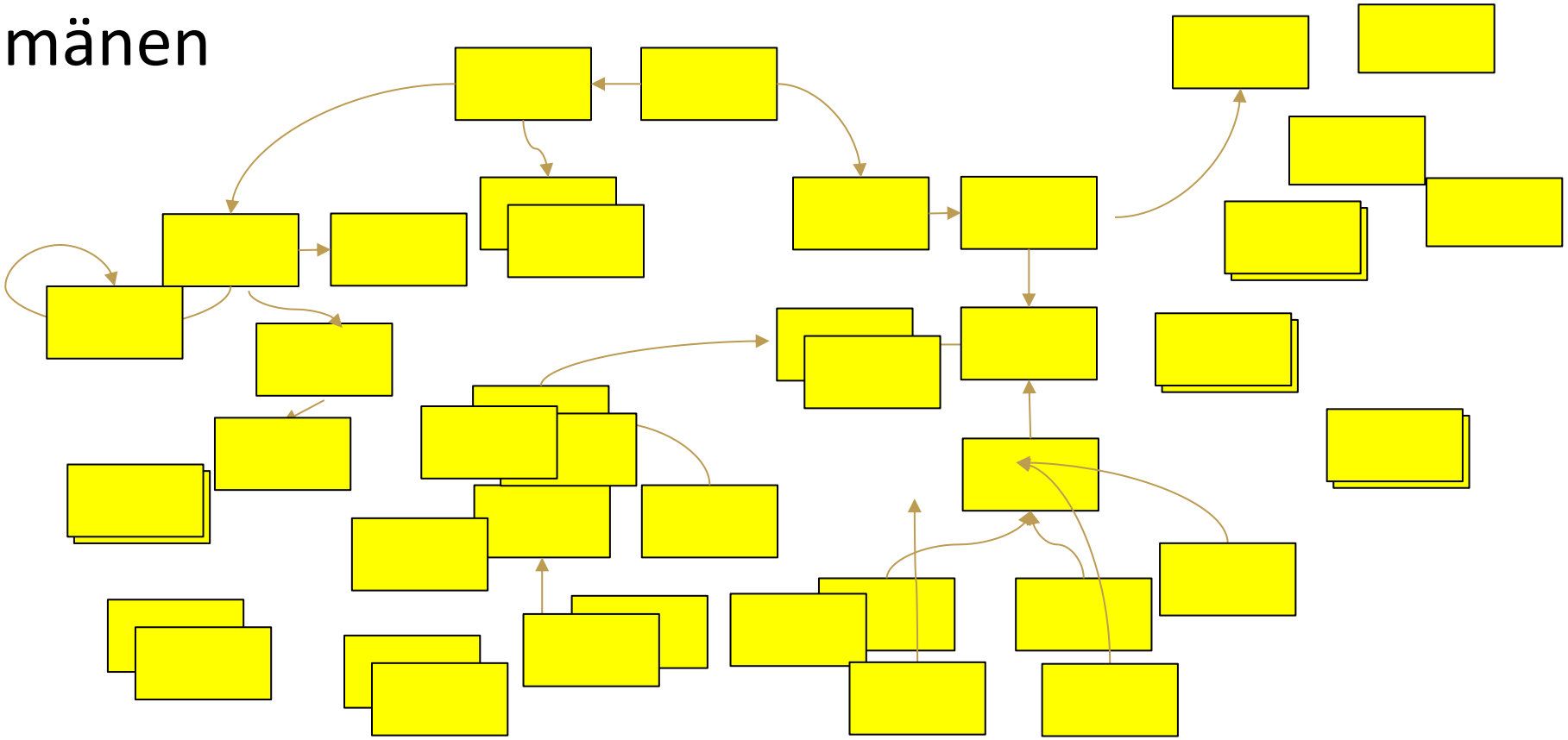
- A Bounded Context is an explicit boundary within which a domain model exists.
- Inside the boundary all terms and phrases of the Ubiquitous Language have specific meaning and the model reflects the Language with exactness.

(Implementing DDD p.62, Vauhgn Vernon)

BOUNDED CONTEXT

Warum ist es gut sich mit Bounded Contexts zu beschäftigen?

Oft starten Domänen Modelle klein.



..und entwickeln sich zu einem ***Big Ball of Mud***, weil Teams nicht wissen, wann sie aufhören sollen.

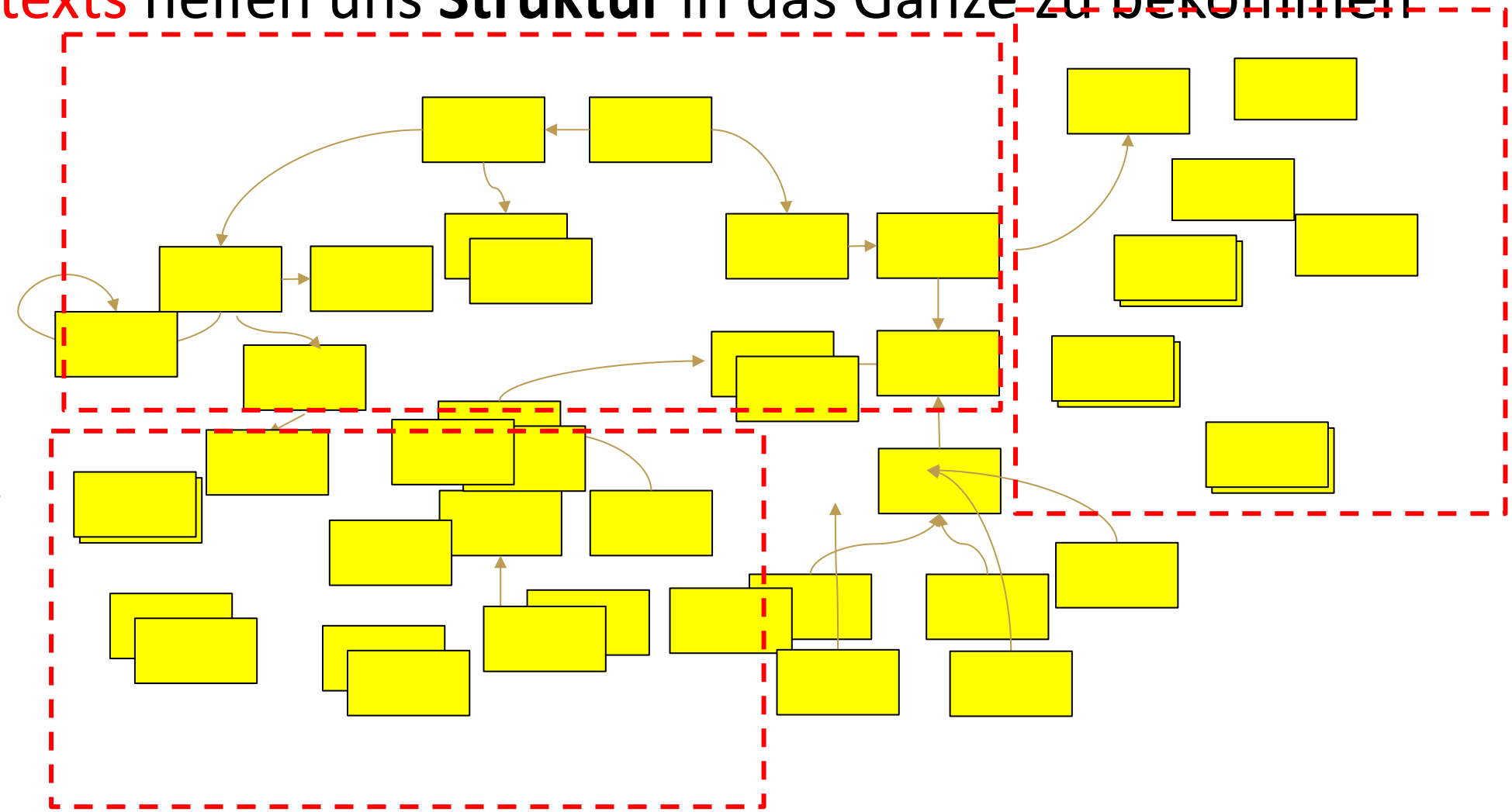
BOUNDED CONTEXT

Bounded Contexts helfen uns **Struktur** in das Ganze zu bekommen

..der erste Schritt um die

„Big Ball of Mud“

Falle zu umgehen.



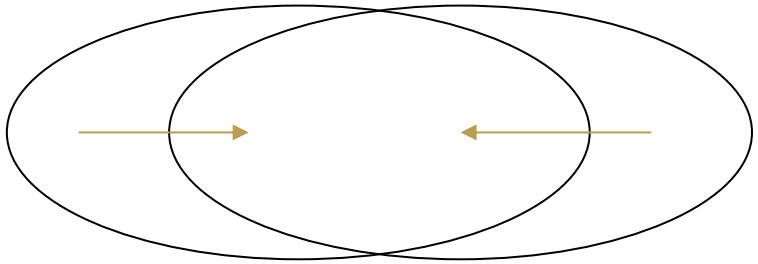
BOUNDED CONTEXT & UBIQUITOUS LANGUAGE

Was ist dabei zu beachten?

- Ein Modell dient immer genau einem Zweck (nicht mehreren Zwecken)
- Das Modell enthält keine impliziten Annahmen.
- Das Modell enthält keinen unnötigen Ballast.
- Es ist die Manifestation des mentalen Modells der Domain Experten.
- Sprache ist nicht eindeutig. Worte können in unterschiedlichen Kontexten unterschiedliche Bedeutung haben.
- Die Ubiquitous Language sollte konsistent sein (eindeutig und vollständig)

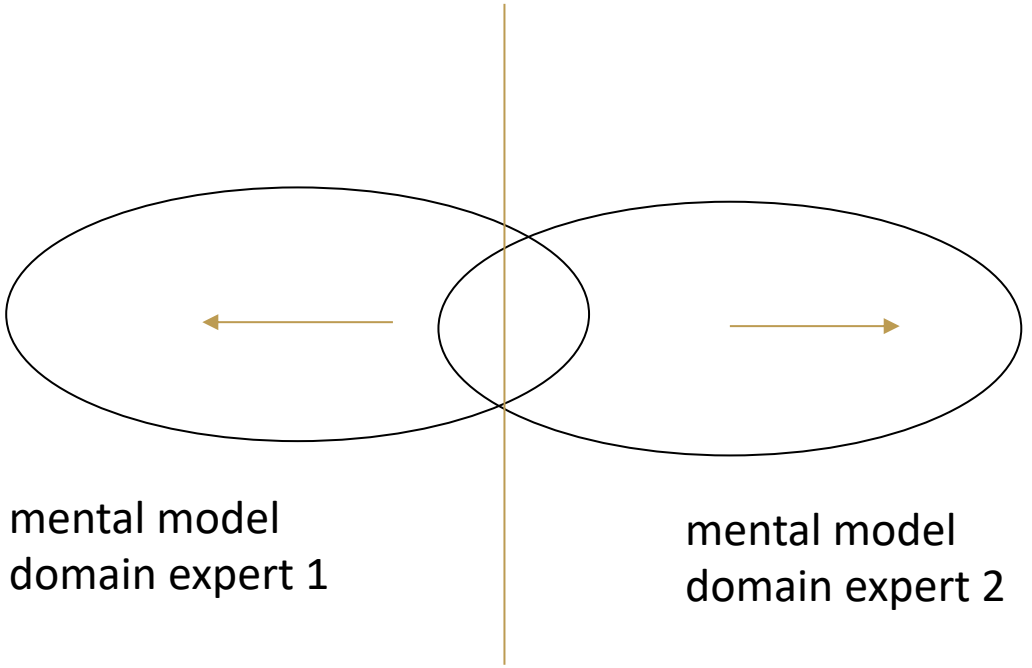
BOUNDED CONTEXT & UBIQUITOUS LANGUAGE

- Was tun, wenn wir die mentalen Modelle der Domain Experten nicht in Einklang bringen können?



mental model
domain expert 1

mental model
domain expert 2



mental model
domain expert 1

mental model
domain expert 2

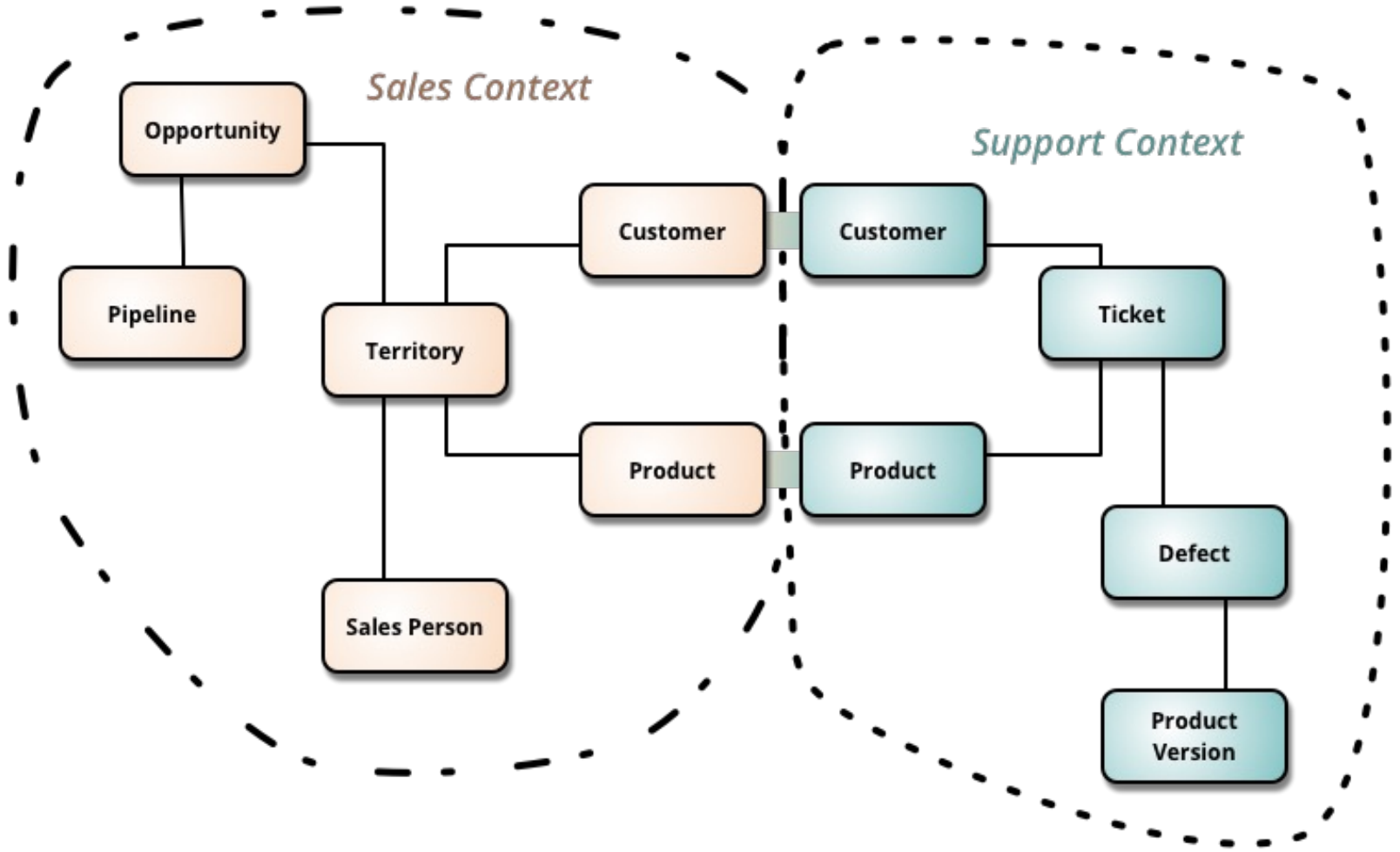
- Wir identifizieren und ziehen eine (neue) Grenze: den Bounded Context

BOUNDED CONTEXT & UBIQUITOUS LANGUAGE

- Beispiele

BOUNDED CONTEXT & UBIQUITOUS LANGUAGE

- Beispiele



Quelle: Martin Fowler, https://martinfowler.com/bliki/BoundedContext.html

BOUNDED CONTEXT & UBIQUITOUS LANGUAGE

- Beispiel Telesales-Company (1/2)

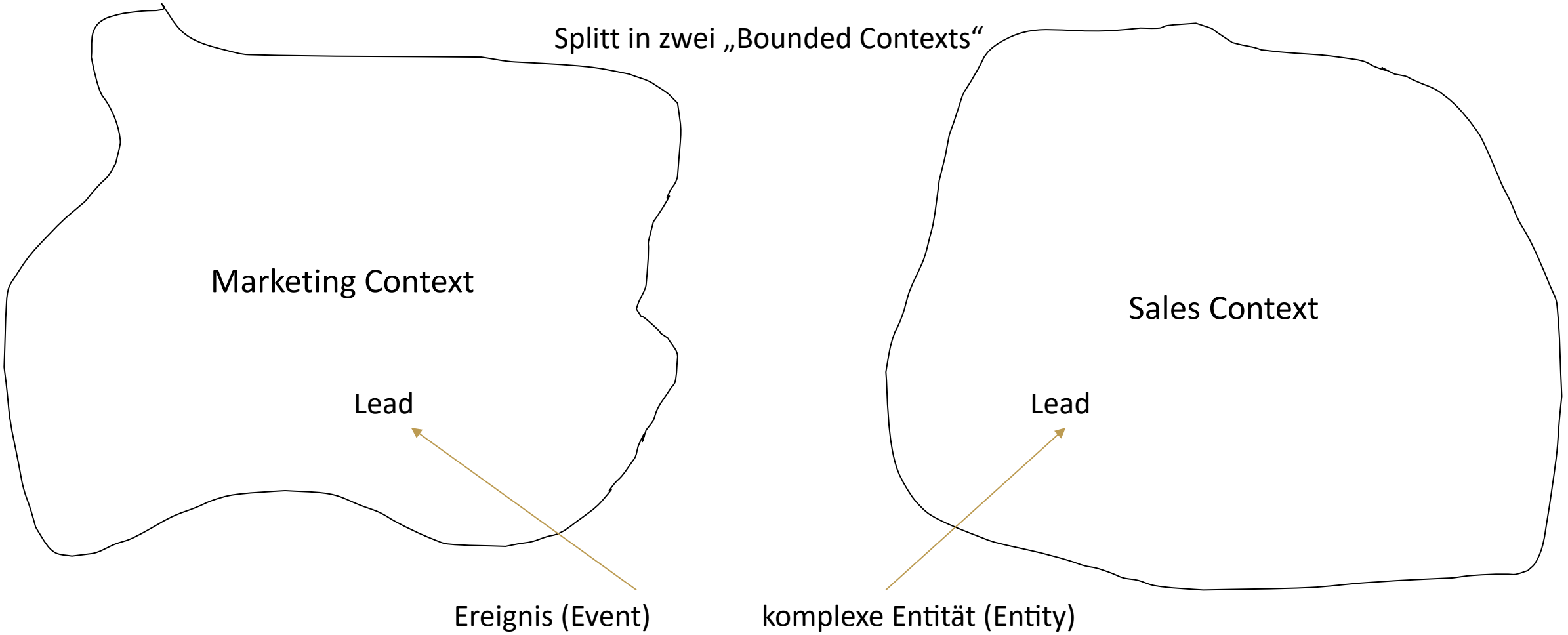
Für das Marketing ist ein Lead die Absicht eines Interessenten ein Produkt zu kaufen.
“Klick auf die Online-Werbung und Eingab der Kontaktinformationen des Interessenten“

Für den Sales ist ein Lead ein Prozess, der mit dem obigen Event startet und mit dem eigentlichen „Kaufen“ endet. Das beinhaltet den kompletten Lebenszyklus des Kaufprozesses.

Quelle: Vlad Khononov, p 34 – 37, 2021

BOUNDED CONTEXT & UBIQUITOUS LANGUAGE

- Beispiel Telesales-Company (2/2)



Quelle: Vlad Khononov, p 34 – 37, 2021

BOUNDED CONTEXT & UBIQUITOUS LANGUAGE

Zusammenfassung:

- Ein Modell ist keine Kopie der realen Welt, sondern
  - eine Vereinfachung durch Weglassen,
  - eine Reduktion auf das Wesentliche,
  - sowie eine Abstraktion durch Zusammenfassen

  und es dient einem bestimmten Zweck.

- Ein Modell benötigt deshalb immer eine Grenze, sonst wird es zwangsweise zur Kopie der realen Welt: Sprache, Terminologie, Prinzipien und Geschäftsregeln haben nur innerhalb dieser Grenze ihre Gültigkeit und sind dort konsistent.

- Die Ubiquitous Language (allgemeingültige Sprache) ist damit keine universelle Sprache! Sie ist nur innerhalb ihres Bounded Context gültig.

WIE FINDET MAN DEN RICHTIGEN SCHNITT: BOUNDED CONTEXT

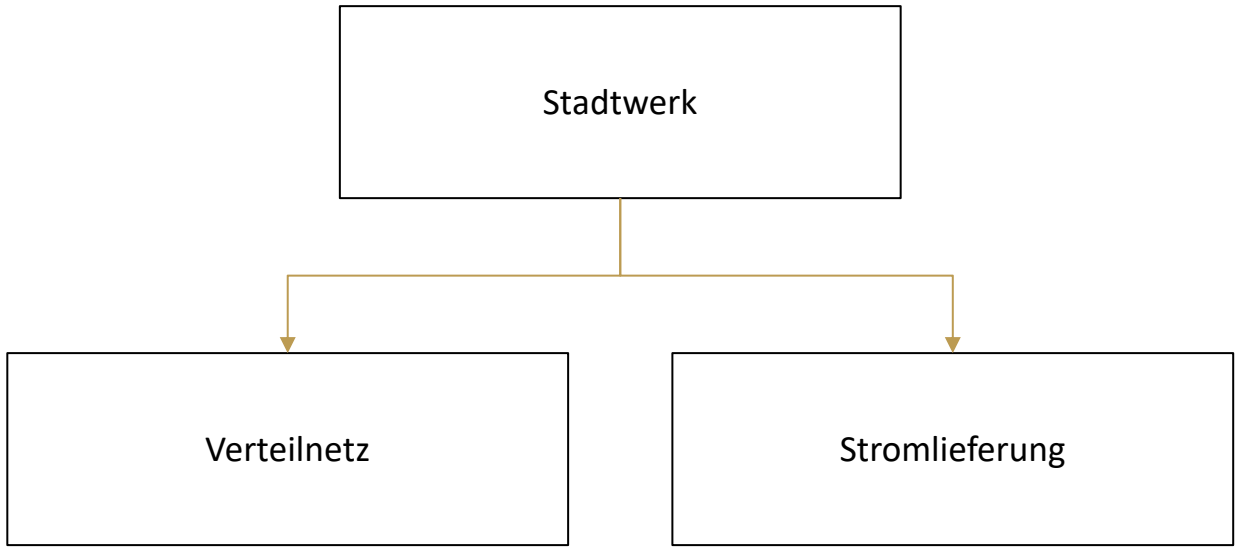
WIE FINDET MAN DIE MODELL-GRENZE?

Ansatzpunkte:

- Aufbau-Organisation mit ihren Abteilungen (Abteilungen haben oft unterschiedliche fachliche Funktionen und Aufgaben)
- Business Capability Model (beschreibt die Fähigkeiten einer Organisation)
- EventStorming als Methode (achte auf Pivotal Events)
- Immer dann, wenn von unterschiedlichen Domain Experten dieselben Dinge unterschiedlich benannt werden. Unterschiedliche Definitionen für die gleichen Begriffe.

WIE FINDET MAN DIE MODELL-GRENZE?

Beispiele: Energiewirtschaft (1/4)



Rolle Netzbetreiber

Rolle Lieferant

Netzstabilität
Physik
Messung

Vertrag mit dem Konsumenten
Lieferung und Abrechnung

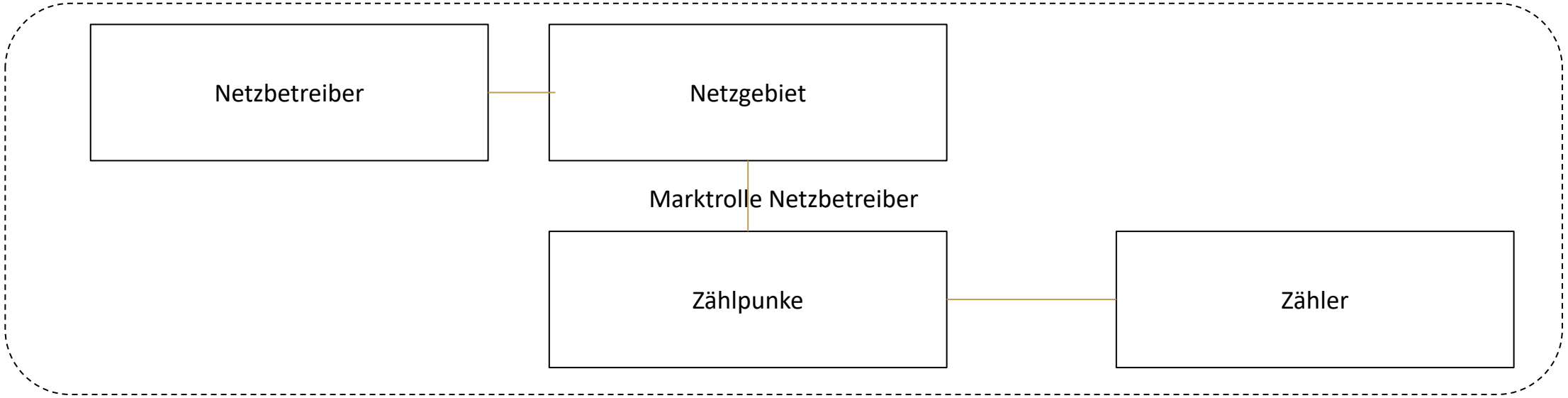
Rolle Messstellenbetreiber

WIE FINDET MAN DIE MODELL-GRENZE?

Beispiel: Energiewirtschaft (2/4)

Wikipedia: **Zählpunkt**

Zählpunkt ist die Bezeichnung in der Energiewirtschaft für den Punkt, an dem Versorgungsleistungen wie z. B. Elektrizität, Erdgas, Fernwärme oder Trinkwasser an Verbraucher geleistet oder von Erzeugern bezogen werden. Dem Zählpunkt wird eine eindeutige Bezeichnung, die **Zählpunktbezeichnung** (ZPB[1]), zugeordnet.
Ein Zählpunkt kann dabei genau einen **Zähler**, z. B. den **Stromzähler** eines Hauses repräsentieren. Es können aber auch mehrere **Messstellen** zu einem **virtuellen Zählpunkt** zusammengefasst werden. Dies kann z. B. ein Unternehmen mit mehreren Übergabestellen sein. Die **Netzbetreiber** verwalten die Lieferbeziehungen zu den verschiedenen Zählpunkten in ihrem Netzgebiet.



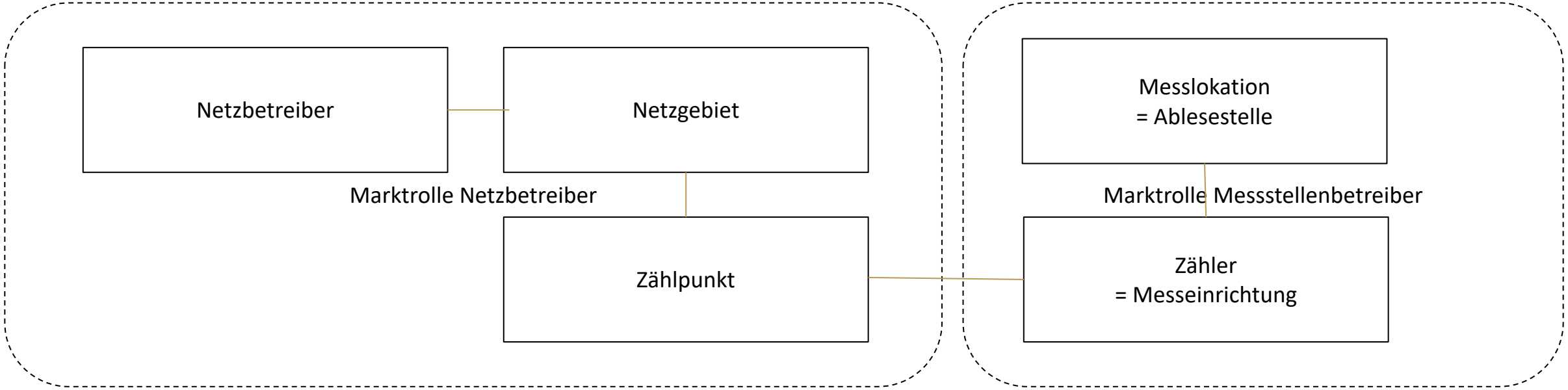
WIE FINDET MAN DIE MODELL-GRENZE?

Knowledge Foundation
@ Reutlingen University

Beispiel: Energiewirtschaft (3/4)

Wikipedia: **Messstellenbetreiber (MSB)**

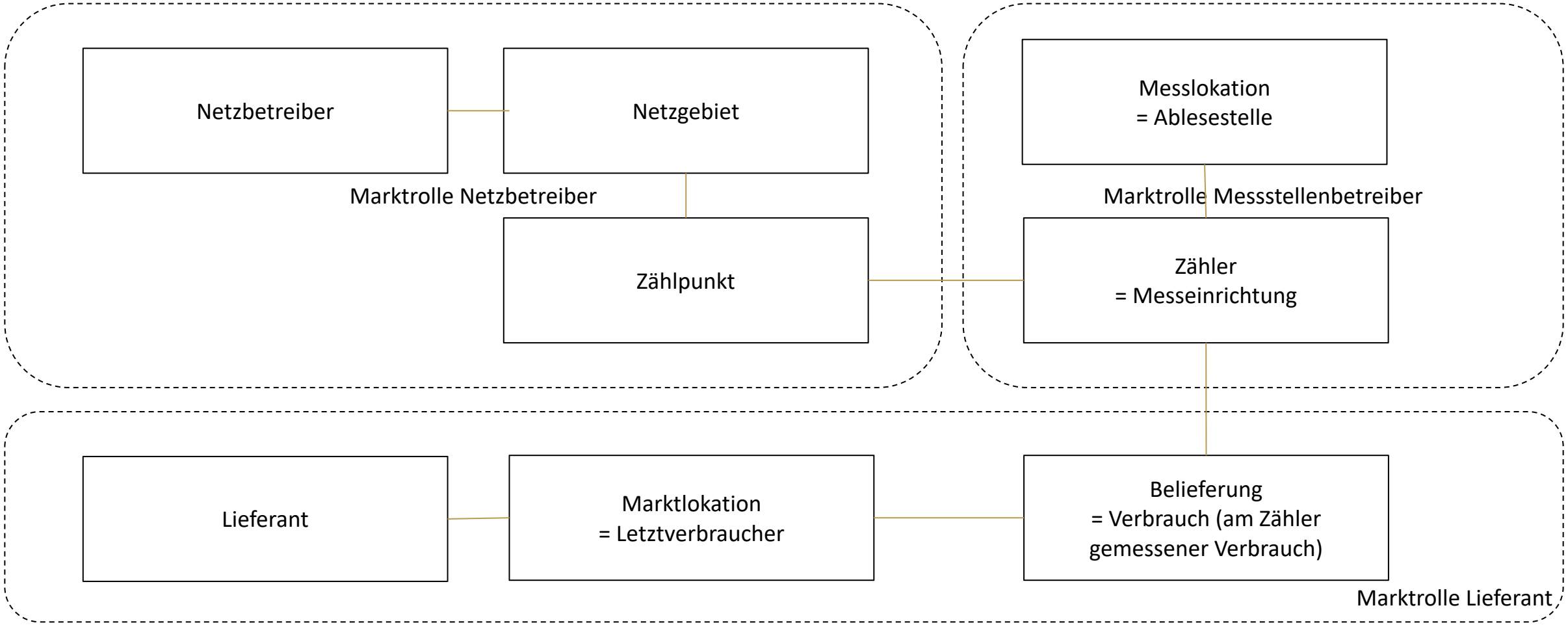
Für die **Messlokation** (also der Ort, an dem Energie gemessen wird) ist es nach § 14 MsbG möglich, dass **Messeinrichtungen** in der Energiewirtschaft (zum Beispiel Stromzähler, Gaszähler) von unabhängigen dritten Messstellenbetreibern eingebaut und betrieben werden können. Dieser Wechsel ist kostenlos.[50] Das politische Ziel ist, einen freien Markt für das Messwesen und damit auch für die Smart Meter Gateway Administration zu schaffen, der im Interesse des Kunden zu sinkenden Messentgelten führen soll. Für den Aufbau und Betrieb der Messeinrichtung erhält der Messstellenbetreiber ein Monatsentgelt.



WIE FINDET MAN DIE MODELL-GRENZE?

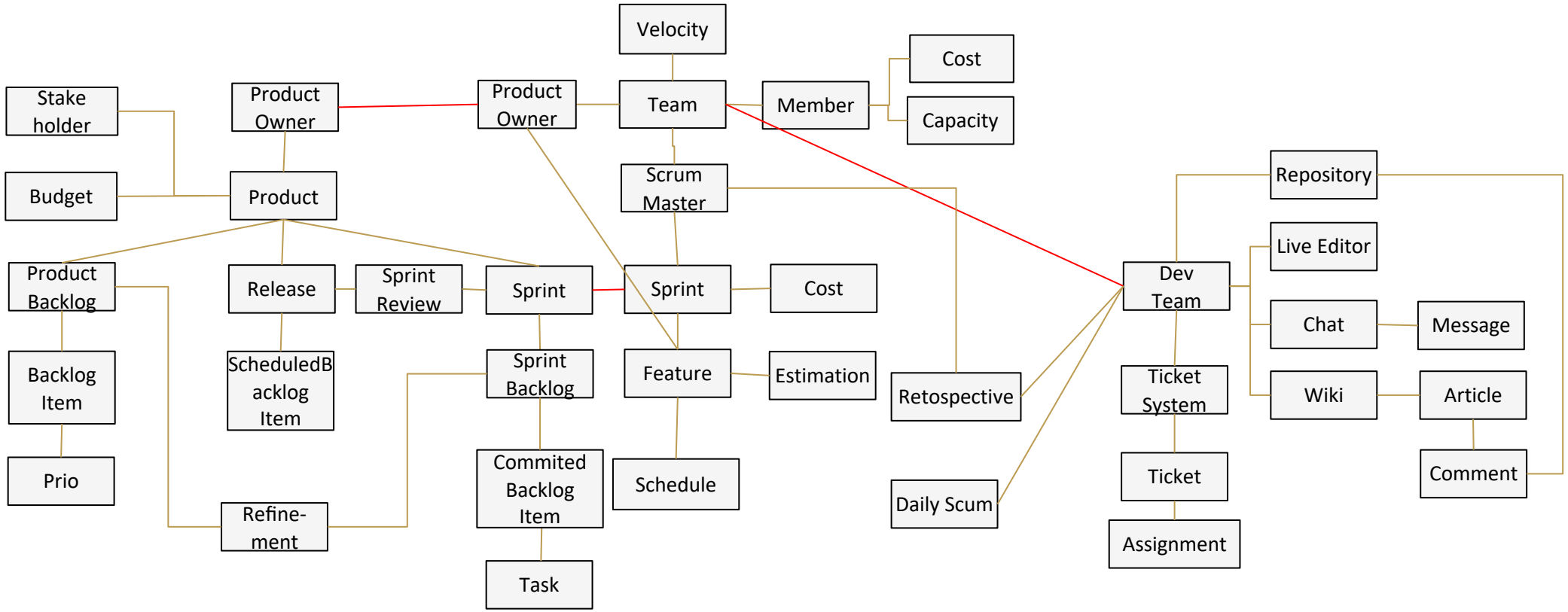
Beispiel: Energiewirtschaft (4/4)

BDEW: Der **Lieferant** ist verantwortlich für die **Belieferung** von **Marktlokationen**, die Energie verbrauchen, und die **Abnahme** von Energie von **Marktlokationen**, die Energie erzeugen.



WIE FINDET MAN DIE MODELL-GRENZE?

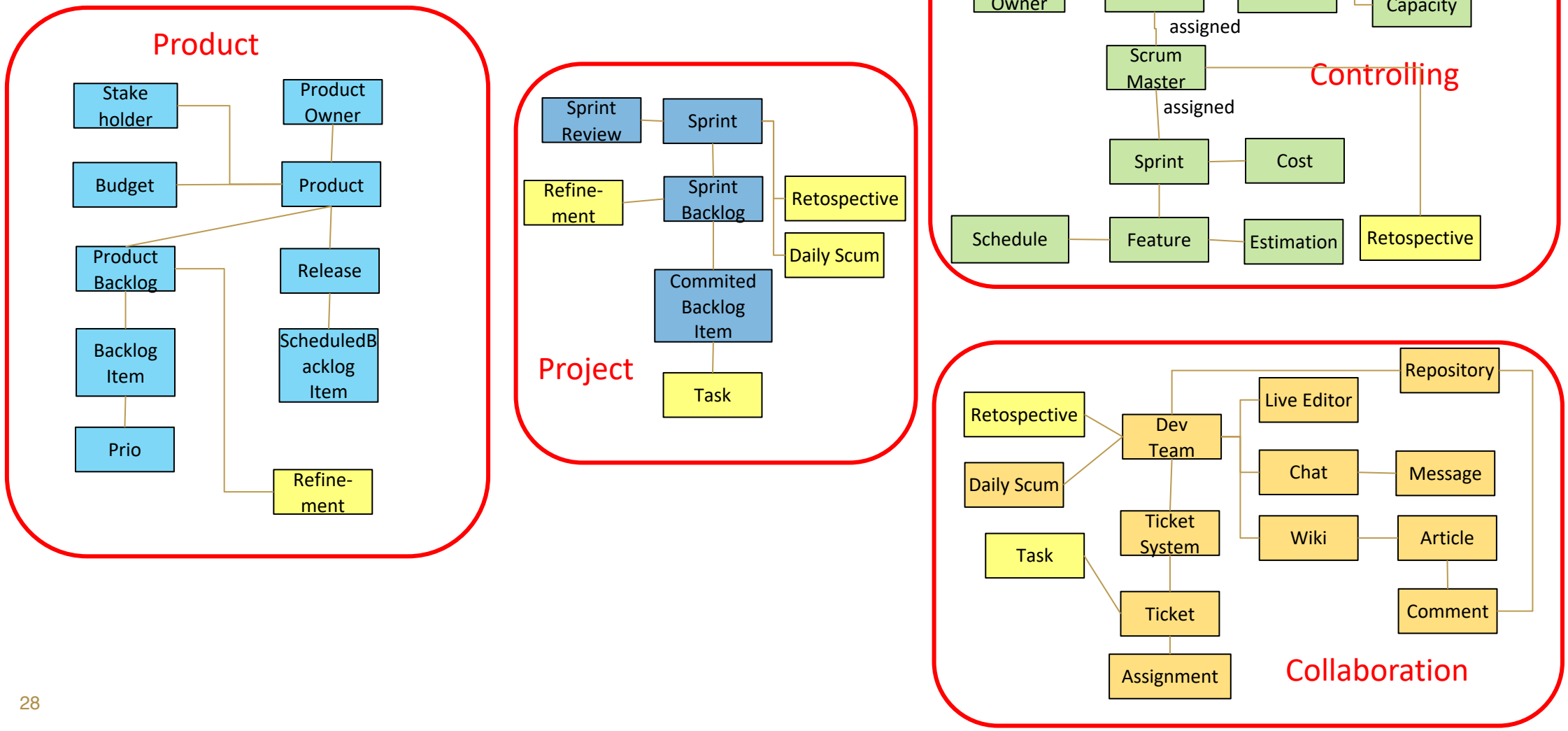
Beispiel: Projekt-Management (1/1)



WIE FINDET MAN DIE MODELL-GRENZE?

Beispiel: Projekt-Management (1/2)

Split in mehere Kontexte



STRATEGISCHES DESIGN MIT CONTEXT MAPPING

CONTEXT MAPPING

Ownership: Jeder Bounded Context hat einen Owner

Ownership-Regeln:

- 1 Team ist verantwortlich für einen Bounded Context
- 1 Team kann für mehrere Bounded Contexte verantwortlich sein

In jedem Fall vermeiden, dass mehrere Teams für einen Bounded Context verantwortlich sind!

CONTEXT MAPPING

Daraus ergeben sich jetzt zwei Fragestellungen:

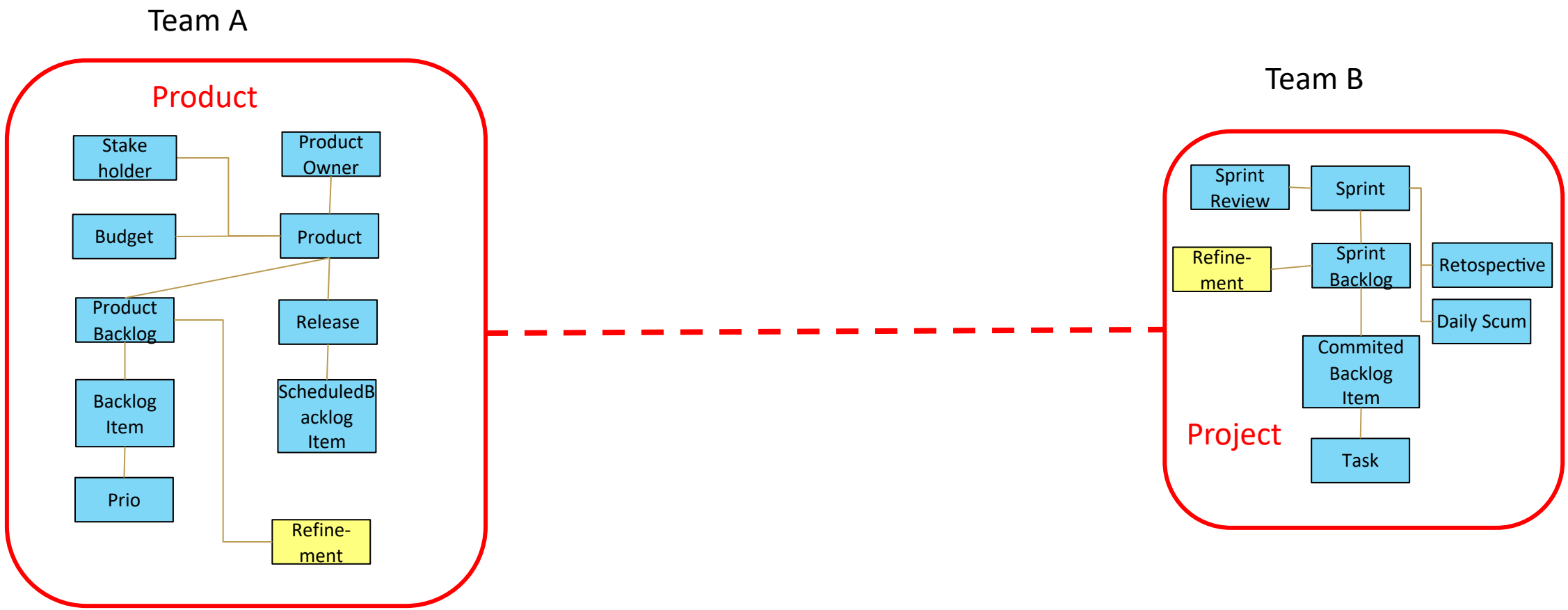
- Separation: Wie können wir als Team unser Domänenmodell vor Verschmutzung und Einflüssen schützen?

- Integration: Wie können wir als Team mit anderen Teams zusammenarbeiten?

→ Lösung: Context Mapping

CONTEXT MAPPING

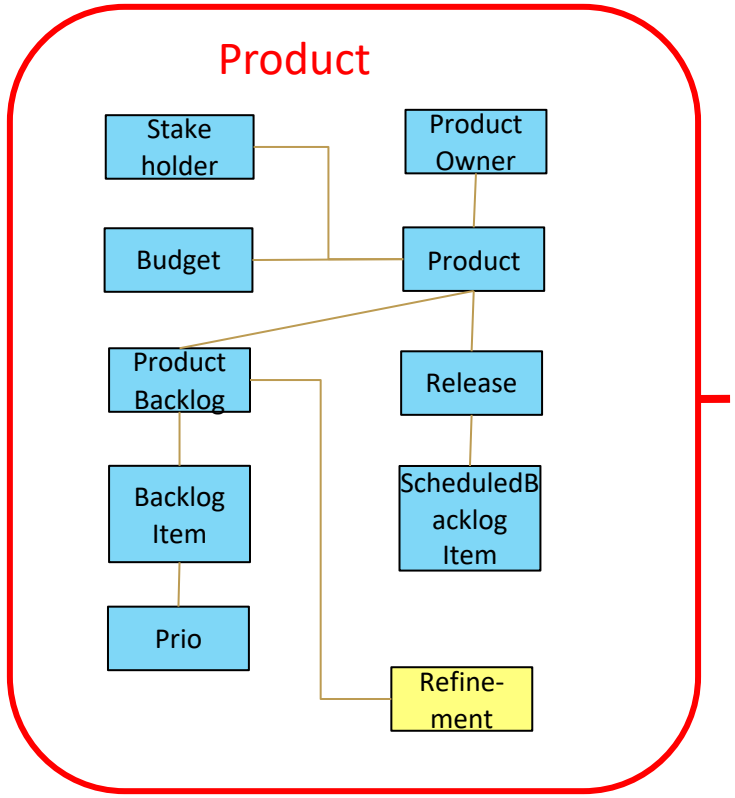
Context Mapping befasst sich mit den drei folgenden Fragestellungen:



CONTEXT MAPPING

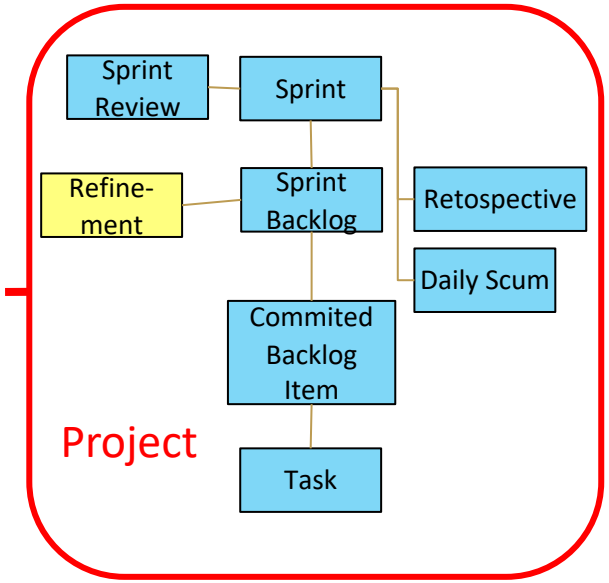
Context Mapping befasst sich mit den drei folgenden Fragestellungen:

Team A



1. Wie beeinflusst die Sprache beide Modelle?

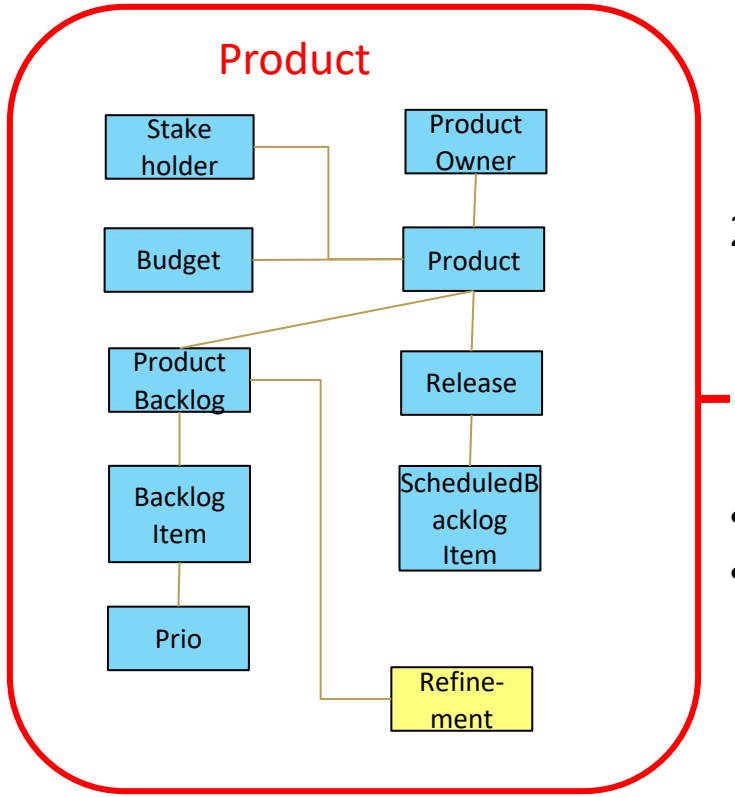
Team B



CONTEXT MAPPING

Context Mapping befasst sich mit den drei folgenden Fragestellungen:

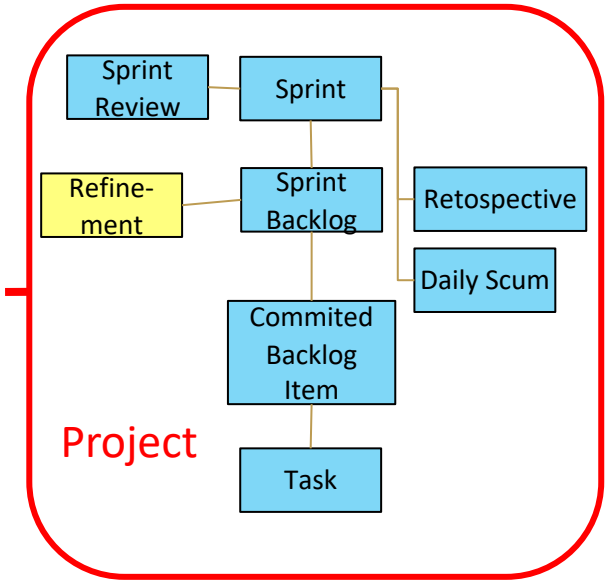
Team A



2. Wie interagieren die Teams miteinander?

- Sprechen die Teams dieselbe Sprache?
- oder benötigen wir einen Übersetzen?

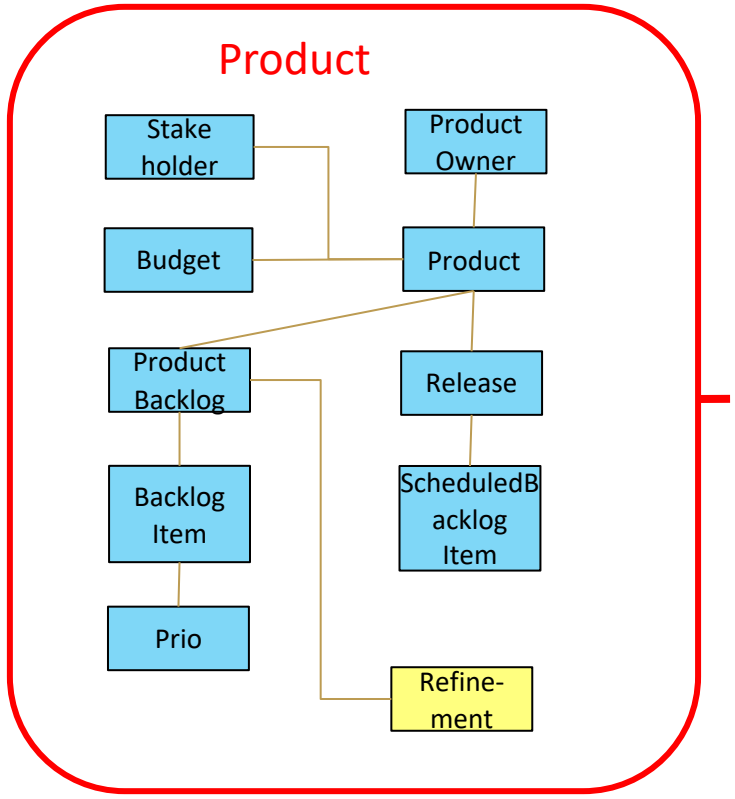
Team B



CONTEXT MAPPING

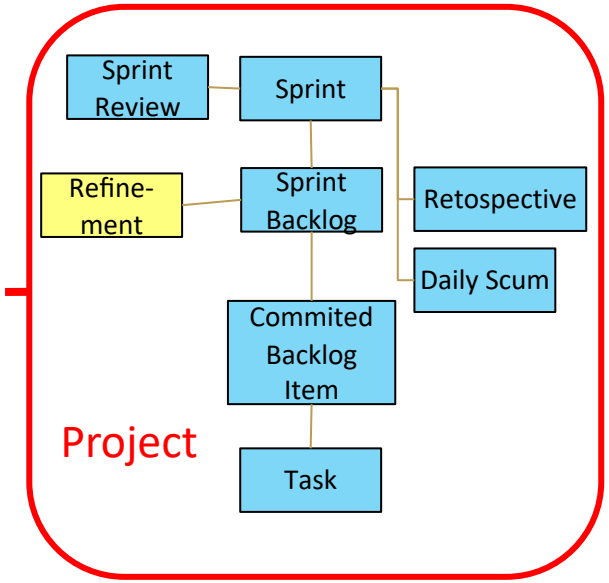
Context Mapping befasst sich mit den drei folgenden Fragestellungen:

Team A



3. Wie interagieren die beiden Systeme technisch miteinander? Wie stark müssen die Systeme voneinander entkoppelt werden?

Team B



Entkoppelung

- der Implementierung (Daten, Funktionen)
- zur Laufzeit (müssen A und B gleichzeitig verfügbar sein?, Performanz vs. Transaktionalität?)
- des Deployment: Gemeinsames Release notwendig?

WAS GENAU BEDEUTET KOPPELUNG?

WELCHE FORMEN DER KOPPELUNG GIBT ES?

KOPPELUNG

1.) Implementation Coupling: (technischen Kopplung)

Herausforderung:

System A ist von der Art der Implementierung von System B abhängig. Ändert sich B, dann muss auch A angepasst werden.

Beispiele sind:

- Nutzung einer gemeinsamen Datenbank. Dies kann durch Kapseln mittels eines API aufgelöst werden (A nutzt das API von B und nur B hat Zugriff auf die Datenbank)
- Nutzung einer gemeinsamen Integrations-Technologie für das API: RMI (remote method invocation), SOAP, ORB (object request broker), ESP (Enterprise Service Bus)

KOPPELUNG

2.) Temporal Coupling: (Kopplung zur Laufzeit)

<u>Herausforderung:</u>

Service A benutzt Service B. Damit A nun B nutzen kann, muss B verfügbar sein. Ist der Aufruf synchron, dann haben wir eine zeitliche Koppelung.

- Aufgelöst werden könnte das z.B. mittels Cache oder asynchroner Kommunikation.
- State (Zustand) ist auch eine zeitliche Koppelung. Deshalb sollen Microservices stateless implementiert werden. In der Vergangenheit wurde oft z.B. der Warenkorb eines Shops in einer Statefull Session Bean implementiert. (Performance und Skalierbarkeit standen da im Vordergrund der Design Entscheidung)
-

KOPPELUNG

3.) Deployment Coupling:

Herausforderung:

a) Technisch - Wenn die ganze Anwendung bestehend aus mehreren logischen Modulen nur im Ganzen deployed werden kann, z.B. weil sie in einem JAR File steckt, dann sprechen wir von Deployment Coupling.

b) Fachlich - Wenn zwei Anwendungen nur gemeinsam released werden können, dann sind beide voneinander abhängig. Und es wartet in der Regel immer eine Anwendung auf die Fertigstellung der anderen.

Das lässt sich z.B. mit einer Architektur aus Self-Contained Systems (oder Microservices) auflösen.

KOPPELUNG

4.) Organisationale und Domain Koppelung:

Herausforderung:

Die Gesamtfunktionalität (Problem Domain, Kontext) lässt sich nicht so in zerlegen, dass daraus zwei voneinander gut abgrenzbare Bounded Contexte entstehen.

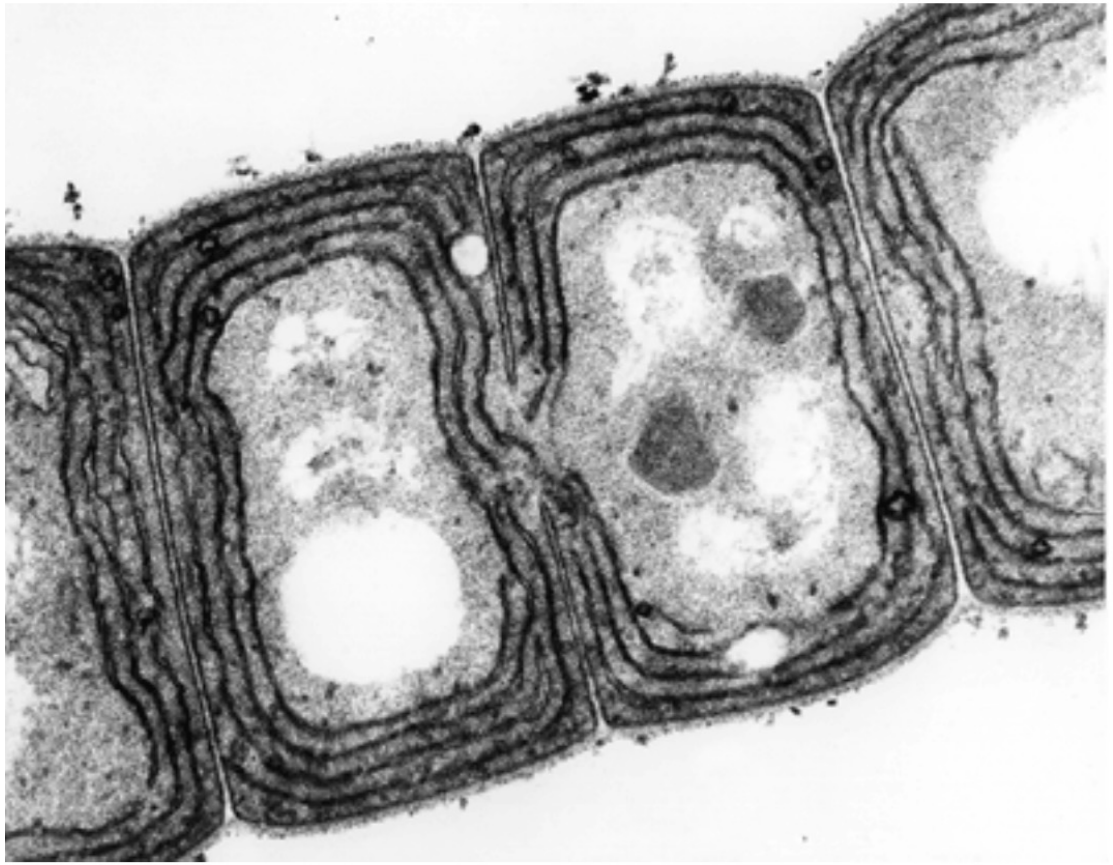
- Gemeinsame (doppelt implementierte) Funktionalität über Bounded Contexte hinweg
- Ändert sich die Funktionalität in System B, dann muss System A angepast werden
- Ändert sich die Organisation, also die Zuständigkeiten. Dann müssen die Systeme angepasst werden

WIE KANN CONTEXT MAPPING HELFEN?

CONTEXT MAPPING

- Eine Zelle existiert, weil ihre Membran definiert, was innen und was außen ist.
- Zell-Membran als Bounded Context.

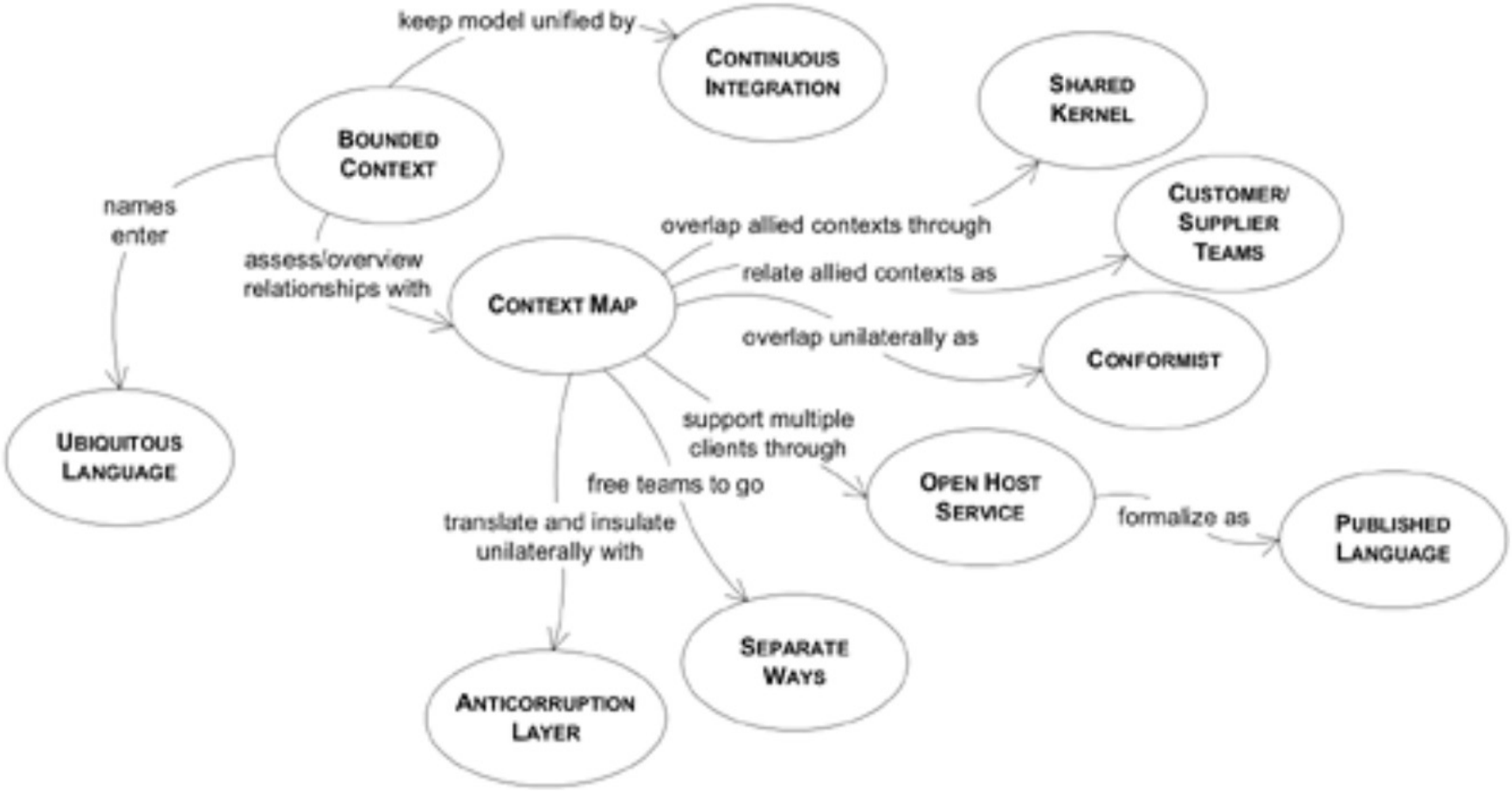
- Sie definiert aver auch was hinein und hinaus darf. Sie legt fest, wie Zellen Miteinander kommunizieren.
- Zell-Membran ermöglicht Context Mapping.



Quelle: Evans p. 334 Kapitel 14

CONTEXT MAPPING

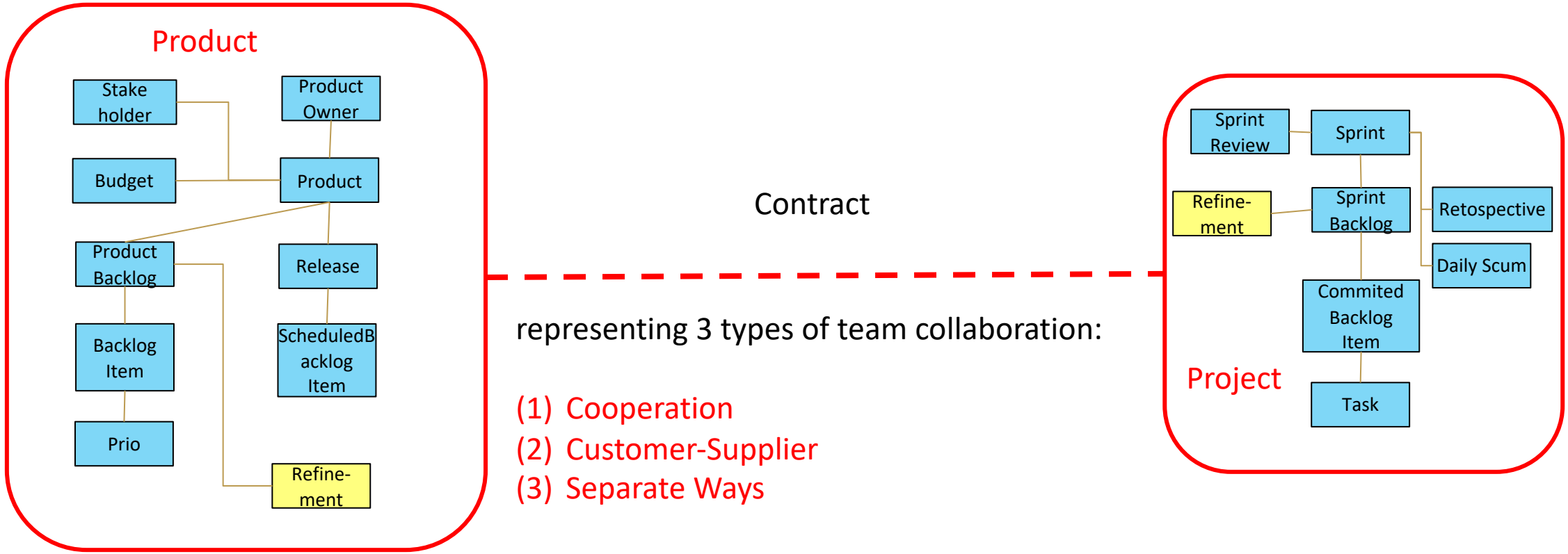
Es gibt verschiedene Arten von Context Mapping und entsprechend geeignete Architektur-Muster



Quelle: Evans p. 334 Kapitel 14

CONTEXT MAPPING

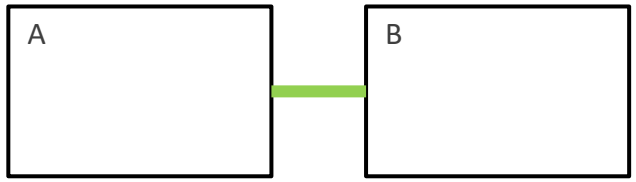
Touchpoints between Bounded Contexts are called Contracts



CONTEXT MAPPING

(1) Cooperation

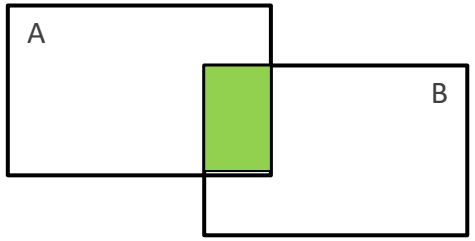
(Im Folgenden bezeichnen A und B Teams. Boxen sind die Bounded Contexte für die die jeweiligen Teams verantwortlich sind.)



Partnership: Team A & B haben

- sind gemeinsam für ein System verantwortlich
- gemeinsame (oder von einander abhängige Ziele): OKR
- sind eng miteinander verbunden. (Scrum of Scrum)
- nutzen Continuous Integration. (Release Train)

Partnership sollte nur solange bestehen, wie sie einen Vorteil bietet. Sie ist ungeeignet für geographisch verteilte Teams sowie Team mit starken voneinander abweichenden Kulturen.



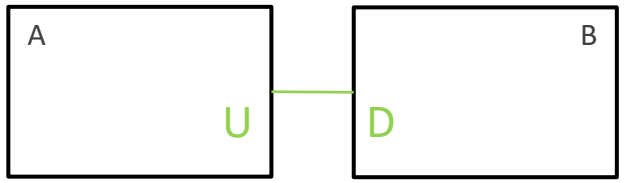
Shared Kernel: Team A & B teilen sich im Schnittpunkt ihrer Bounded Contexte ein kleines aber gemeinsames Modell.

Es ist denkbar, dass nur eines der beiden Teams die Code, Build und Test Verantwortung für das gemeinsame Modell übernimmt. (Gemeinsmes Repo oder Linked Library)

CONTEXT MAPPING

(2) Customer-Supplier (1/2)

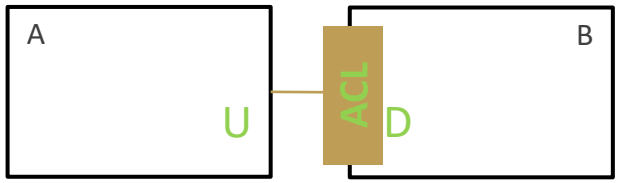
(Im Folgenden bezeichnen A und B Teams. Boxen sind die Bounded Contexte für die die jeweiligen Teams verantwortlich sind.)



Customer-Supplier: Team A bietet Funktionalität (Service) über ein API an. Team B ist Kunde oder Konsument des Service.

Da der Supplier vorgeschaltet (upstream) ist wird er in der Literatur mit U bezeichnet. Der Customer mit D (downstream).

API-First: The Supplier (U) has to design the API for the Customer (D)



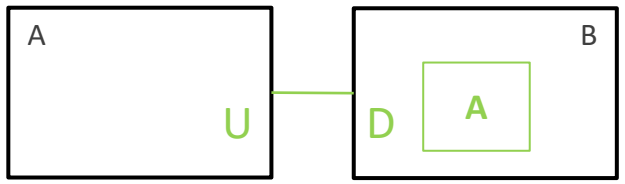
Anticorruption Layer: A bietet API an, B ist der Kunde von A.

Damit das Modell von B nicht “verschmutzt” wird, implementiert B eine Fasade vor der eigentlichen Schnittstelle. Diese wird Anti-Corruption Layer genannt.
Das Pattern wird häufig eingesetzt, wenn auf Legacy-Anwendungen zugegriffen werden muss.

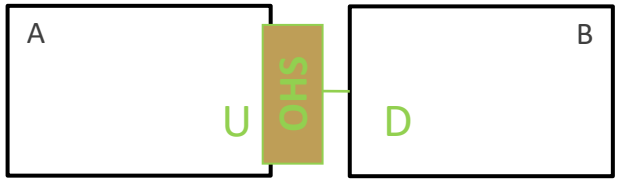
CONTEXT MAPPING

(2) Customer-Supplier (2/2)

(Im Folgenden bezeichnen A und B Teams. Boxen sind die Bounded Contexte für die die jeweiligen Teams verantwortlich sind.)



Conformist: ist ein weiterer Spezialfall von Customer-Supplier. Team A bietet Funktionalität (Service) über ein API an. Team B ist Kunde oder Konsument des Service. B past sich aber A soweit an, dass B die Modell-Sprache von A übernimmt.

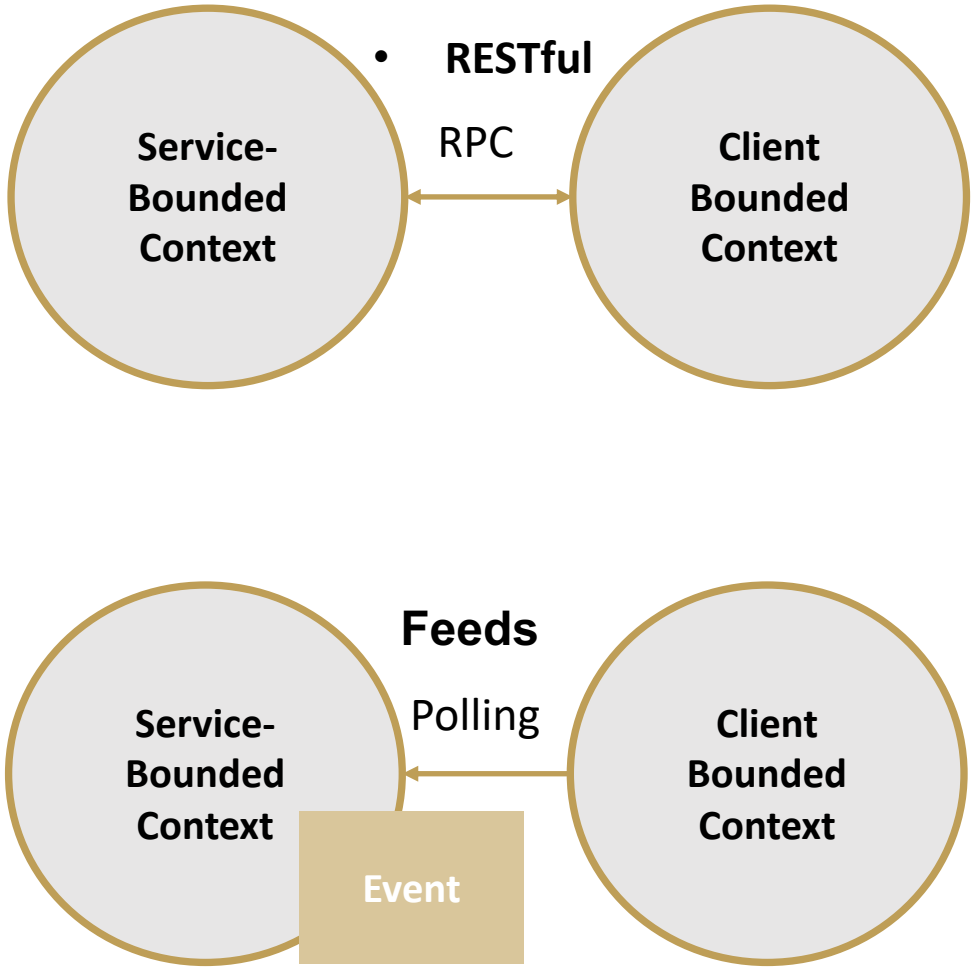


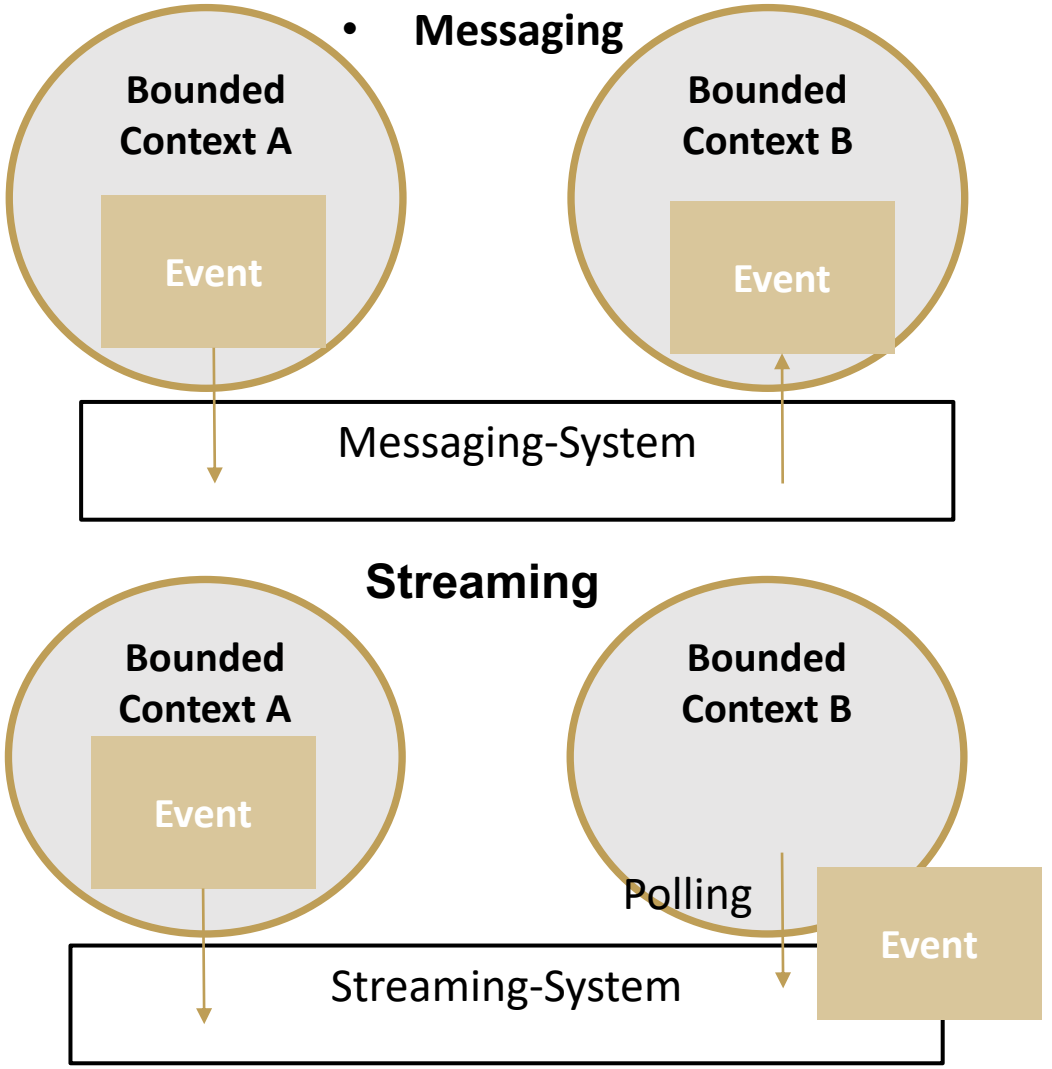
Open Host Serrvice: A bietet einen Service an, der public ist und Über eine offene Schnittstelle konsumiert werden kann.

Da A voraussichtlich viele Kunden hat, haben die Kunden in der Regel keine Mitsprache bei der Definition des API.

CONTEXT MAPPING

Technisch gibt es mehre Möglichkeiten zu integrieren:





DISKUSSION

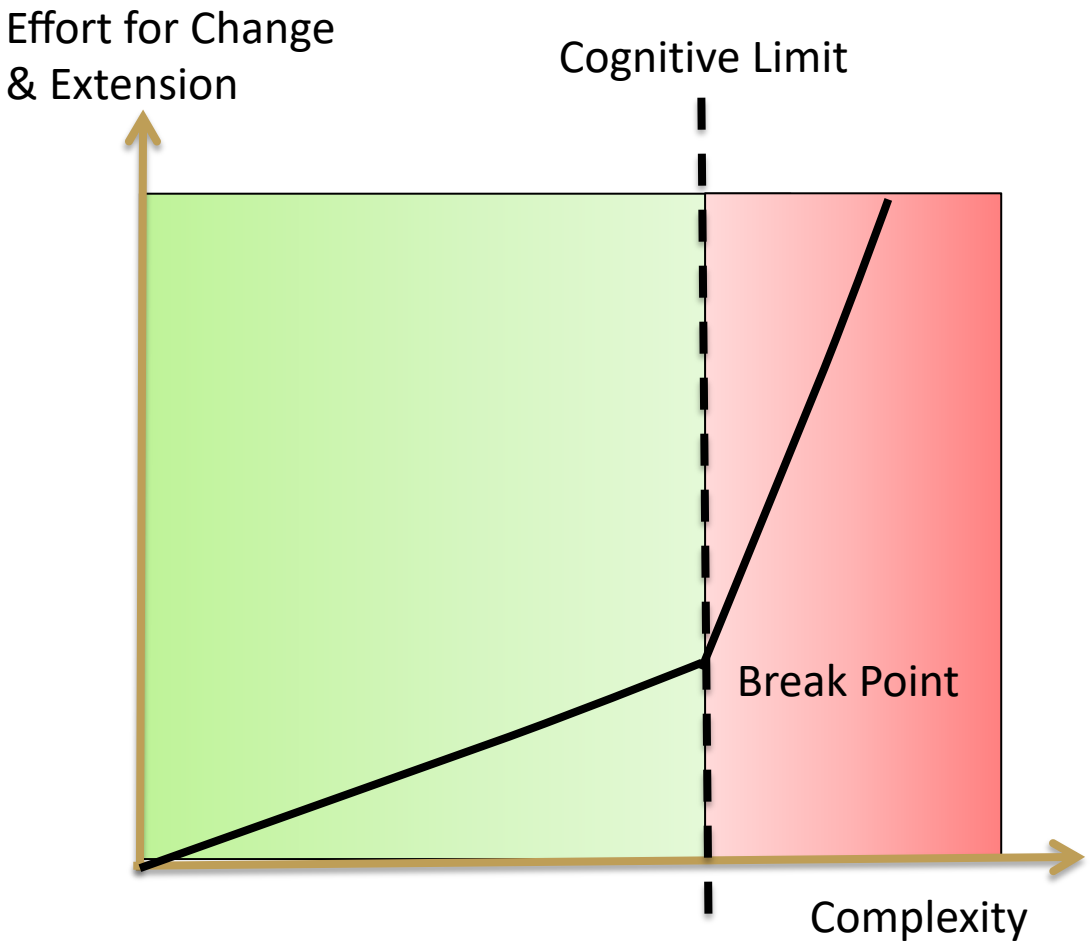
- Koppelung und Integration

- Was sind Treiber für starke Koppelung?
- Wann ist Koppelung problematisch?
- Wie kann man sie verweiden?

ORGANISATION: TEAM TOPOLOGIEN

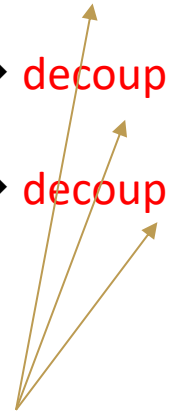
PROBLEM

Complexity kills progress in Software Evolution:



What can we do, if cognitive limit is reached?

- Refactor & Remodularize → decouple implementation
- Split Team → decouple deployment
- Team API → decouple domain



How ?

CONWAY‘S LAW

Das Gesetz von Conway ist eine nach dem US-amerikanischen Informatiker Melvin Edward Conway benannte Beobachtung, dass die Arbeitsergebnisse durch die Kommunikationsstrukturen der sie umsetzenden Organisationen vorbestimmt sind.

Es wurde von Conway 1968 folgendermaßen formuliert:

“Organizations which design systems […] are constrained to produce designs which are copies of the communication structures of these organizations.”

„Organisationen, die Systeme entwerfen, […] sind gezwungen, Entwürfe zu erstellen, die die Kommunikationsstrukturen dieser Organisationen abbilden.“

– Melvin E. Conway

CONWAY‘S LAW

Das Gesetz von Conway basiert auf der Überlegung, dass für die Definition der Schnittstellen zwischen getrennten Softwaremodulen zwischenmenschliche Kommunikation notwendig ist.

Daher haben die Kommunikationsstrukturen der Organisationen einen großen Einfluss auf die Strukturen dieser Schnittstellen.

Eine Studie der Harvard Business School kam zu dem Schluss, dass es starke Hinweise für die Korrektheit des Gesetzes von Conway gibt. Bei allen von ihnen untersuchten 12 Produkten aus 5 unterschiedlichen Anwendungsgebieten (Finanzmanagement, Textverarbeitung, Tabellenkalkulation, Betriebssystem, Datenbanksystem) korrelierte die Kopplung der sie entwickelnden Organisationen mit der Modularität der Produkte

SOLUTION

Team Topologies

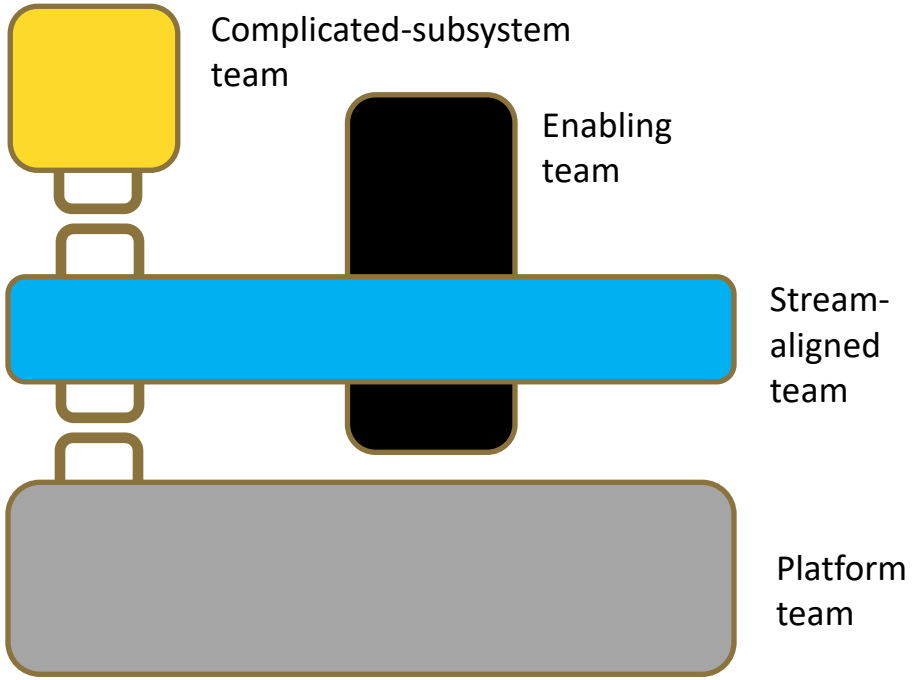


Figure: *The Four Fundamental Team Topologies* from *Team Topologies*, Matthew Skeleton and Manuel Pais

TEAM TOPOLOGIES

Stream-aligned teams:

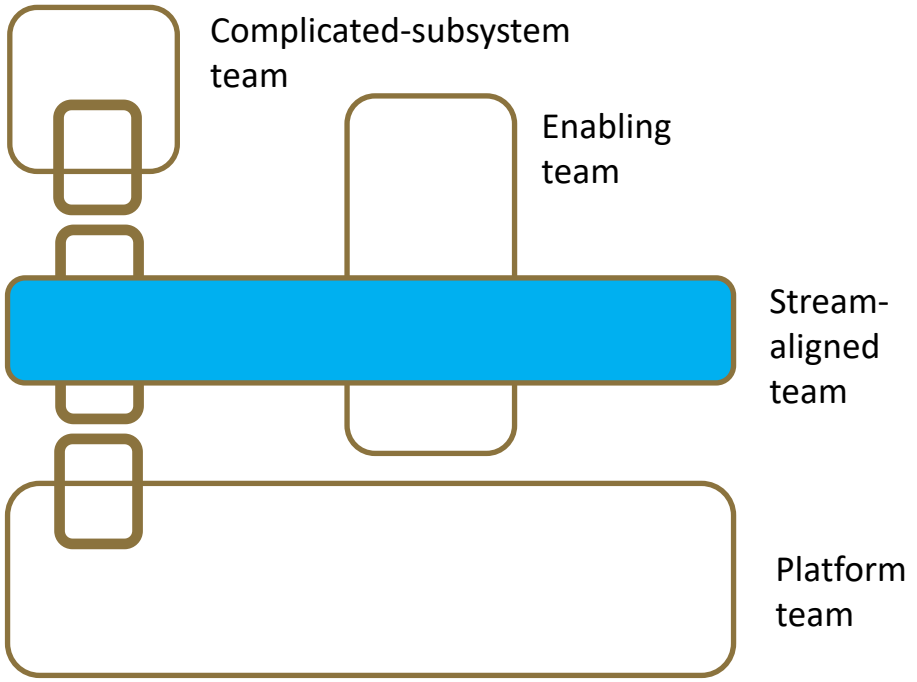


Figure: *The Four Fundamental Team Topologies* from *Team Topologies*, Matthew Skeleton and Manuel Pais

A stream is the continuous flow of work aligned to a business domain or organizational/business capability.

A stream-aligned team is a team aligned to a single, valuable stream of work.

This might be a
- single product or service,
- single set of features,
- single user journey or single user persona

The team is empowered to build and deliver customer or user value as quickly, safely, and independently as possible, without requiring hand-offs to other teams to perform parts of the work.

The purpose of the other fundamental team topologies is to reduce the burdon on the stream-aligned teams.

There maight be different coexisting streams in an organization: specific customer streams, business area streams, geography streams, product streams, user-persona streams or compliance streams.

Whichever kind of stream of changes a stream-aligned team is aligned to, that team is funded in a long-term sustainable manner as part of a portfolio or program of work.

TEAM TOPOLOGIES

Enabling teams:

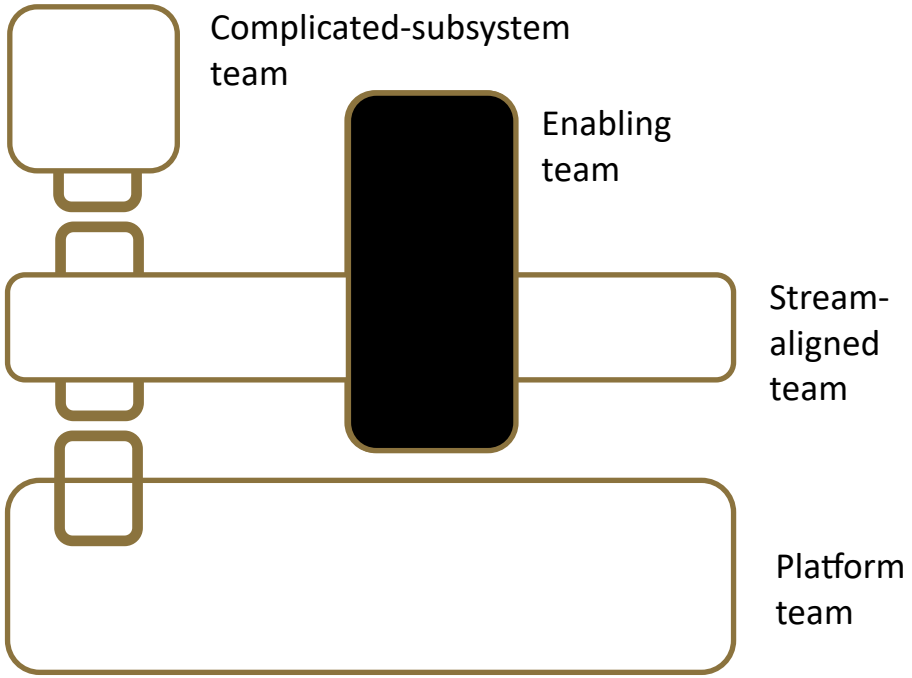


Figure: *The Four Fundamental Team Topologies* from *Team Topologies*, Matthew Skeleton and Manuel Pais

Stream-aligned teams are under constant pressure to deliver and respond to changes quickly. In reality they do not have the capassity to continuously improve their capabilites in order to stay ahead.

An enabling team is composed of specialists in a giving technical(or product) domain and help bridge the above mentioned capability gap.

Enabling teams have a strongly colaborative nature. They thrive to understand the problems and shortcomings of stream-aligned teams in order to provide effective guidance.

Often they are called „Technical Consulting Teams“.

TEAM TOPOLOGIES

Complicated Subsystem teams:

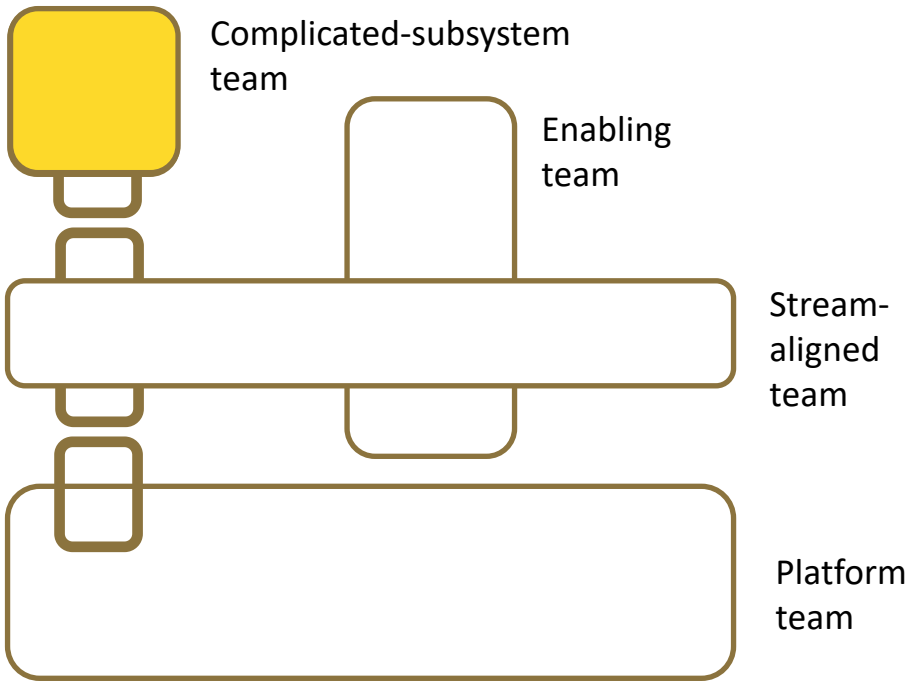


Figure: *The Four Fundamental Team Topologies* from *Team Topologies*, Matthew Skeleton and Manuel Pais

A complicated-subsystem team is responsible for building and maintaining a part of the system that depends heavily on specialist knowledge. The goal of this team is to reduce the cognitive load of stream-aligned teams, working on systems that include or use the complicated subsystem.

TEAM TOPOLOGIES

Platform teams:

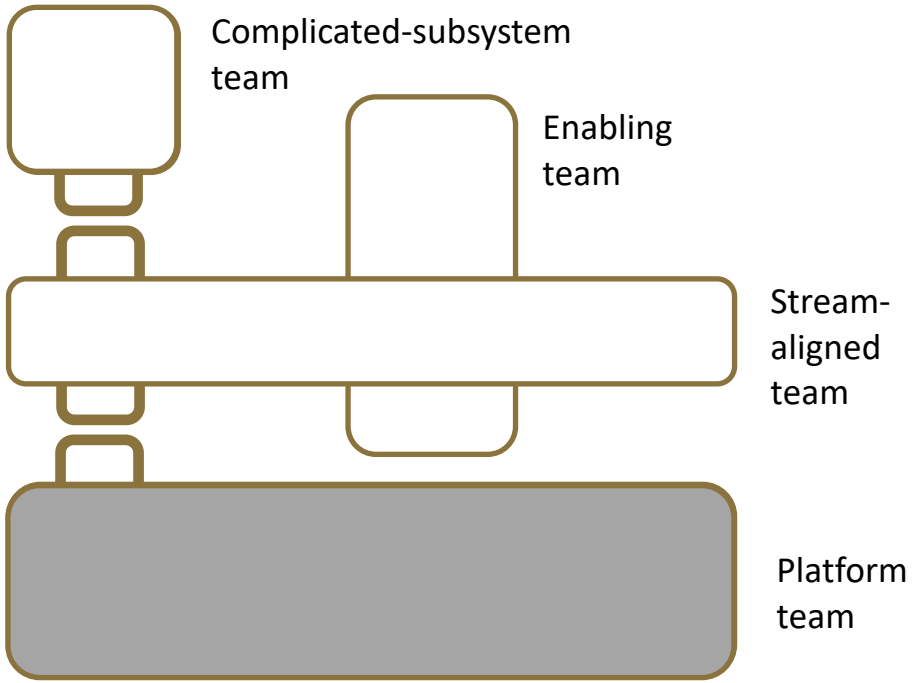


Figure: *The Four Fundamental Team Topologies* from *Team Topologies*, Matthew Skeleton and Manuel Pais

The purpose of a platform team is to enable stream-aligned teams to deliver work with substantial autonomy. Stream-aligned teams maintain full ownership of building, running, and fixing their applications in production.

The platform team provides internal services to reduce the cognitive load that would be required from stream-aligned teams to develop these underlying services.

Ease-of use is fundamental to platform adaption. Platform teams treat the servives they offer as products.

A platform team uses strong collabolration with stream-aliugned teams to understand their needs.

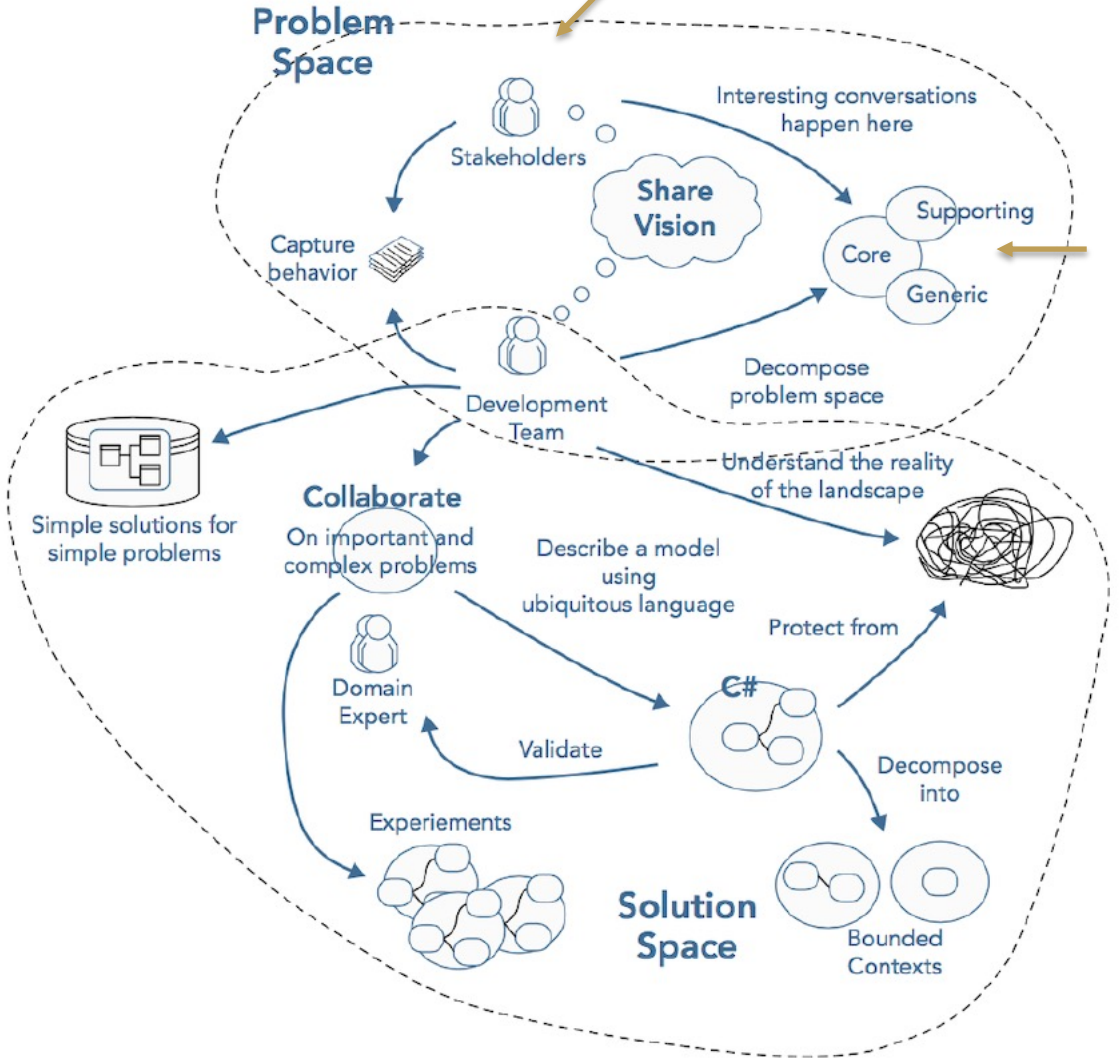
ZUSAMMENFASSUNG

Wir haben gelernt,

- wie man Bounded Context verwenden kann um eine Problemdomäne in Teilbereich zu strukturieren. BC sind immer aktive Design-Entscheidungen. (Teile und Herrsche - Prinzip)
- dass das Business eine natürliche Gliederung vorgibt, die sich am Geschäftsmodell orientiert. Die Business Domain lässt sich in Subdomains zerlegen. Diese beschreiben, welche davon einen Wettbewerbsvorteil stiften und welche generisch sind.
- wie man Teams nach diesen Schnitten (Kontexte) organisieren kann und wie man mit Context Mapping für Ordnung sorgen kann.

ZUSAMMENFASSUNG

Apply Knowledge Crunching



Apply Strategic Design

Apply Tactical Design

Quelle: DDD – The first 15 Years. p.6

ZUSAMMENFASSUNG

DDD in a Nutshell:

- Distill a large problem domain into smaller sub domains.
- Identify the core sub domains to reveal what is Important. The core domains are those of greater value to the business which require more focus, effort and time.
- Collaborate with experts to discover an analysis model that will provide solutions to solve problems or reveal opportunities particularly in the core domain.
- Split the model (if necessary) into smaller models where there is ambiguity in language or the model is too large for a single team. Enclose the model within a boundary to protect the models integrity. When working with multiple models it's important that they are understood in context.
- Keep a context map to understand the relationships, social and technical, of all models in play.

In the next lesson we learn:

- How to use the same ubiquitous language to bind the analysis model to the code model.
- How to use tactical patterns to separate technical code from domain code to prevent accidental complexity.

OPEN QUESTIONS

WIE FINDET MAN DIE MODELL-GRENZE?

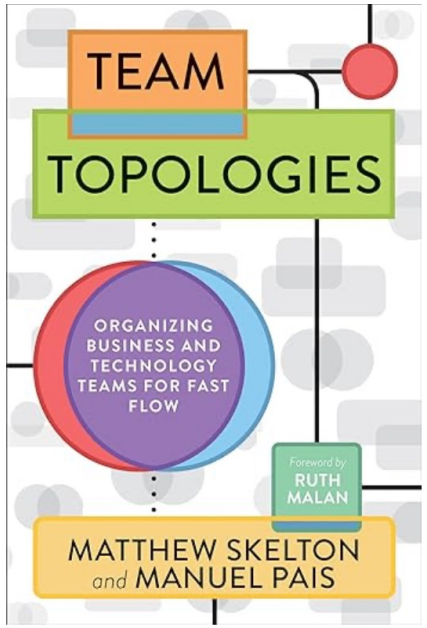
Typische Fragen:

- How big or small should a bounded context be designed?
- What is the difference between a bounded context and a subdomain?
- Should we align subdomain with bounded context?
- What if subdomain and bounded context is overlapping?
- Shoud we align team organization with bounded context?

FURTHER READING

MORE LITERATURE

Team Topologies



2019 – Team Topologies: Organizing Business and Technology Teams for Fast Flow

Matthew Skelton

DDD

Essays from the DDD Community (LeanPub)