GOTOpia Chicago Online 2021-04-20

Is DDD Overrated?

Stefan Tilkov
@stilkov

INNOQ

Domain-driven design

Domain-driven design defined

An approach to designing software that emphasizes domain knowledge over technical aspects and supports users within a domain via a model implemented in software

Domain-driven design defined

DDD is an approach to the development of complex software in which we:

1. Focus on the core domain
2. Explore models in a creative collaboration of domain practitioners and software practitioners
3. Speak a ubiquitous language within an explicitly bounded context

Eric Evans, https://www.domainlanguage.com/wp-content/uploads/2016/05/DDD_Reference_2015-03.pdf

Key aspect #1: Ubiquitous Language

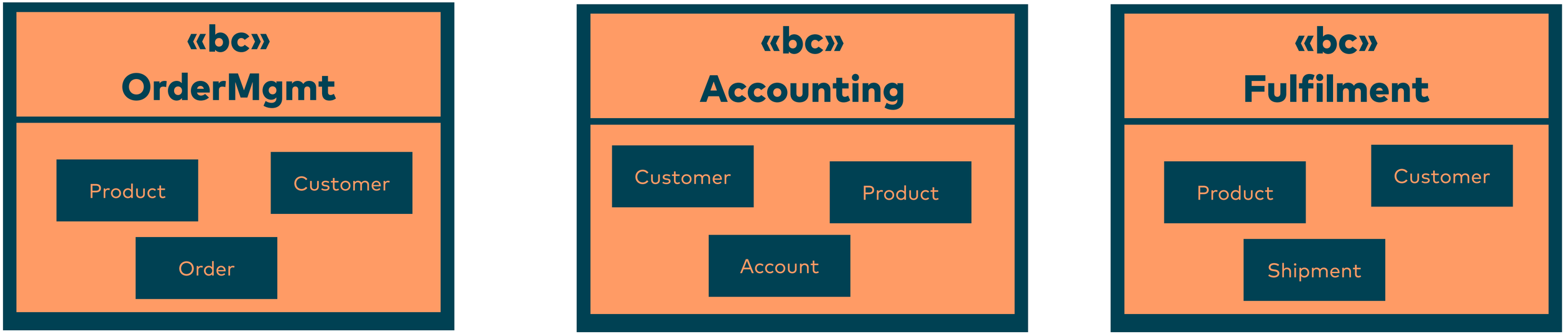
A common language for domain experts and technical team members

Key aspect #2: Tactical patterns

A set of building blocks to structure the implementation of a model according to best practices: Entity, Aggregate, Value Object, Service, Domain Event, Repository, Factory, Module

Key aspect #3: Strategic design

Context maps to visualize bounded contexts and their relationships/connections: Partnership, Conformist, Customer/Supplier, Anticorruption Layer, Open-host Service, Published Language, Shared Kernel


«bc»
OrderMgmt
Product
Customer
Order
«bc»
Accounting
Customer
Product
Account
«bc»
Fulfilment
Product
Customer
Shipment

Conceptual extensibility

Generalization: Models and language

Ubiquitous language exists on multiple levels. On the meta-level, the languages, idioms and patterns used by team members support design collaboration, too

Extensible jargon

Shared language (“jargon”) supports communication among domain team members. It evolves according to reoccurring needs

Extensible, not fixed

Any set of pre-defined, “best practices” patterns is a starting point, not an end in itself

Extending tactical patterns

Entity, Aggregate, Value Object, Service, Domain Event, Repository, Factory, Module,

Filter, Rule, Proxy, Contract, Role, Reference, ...

[insert whatever makes sense to you]

Extending context relationships

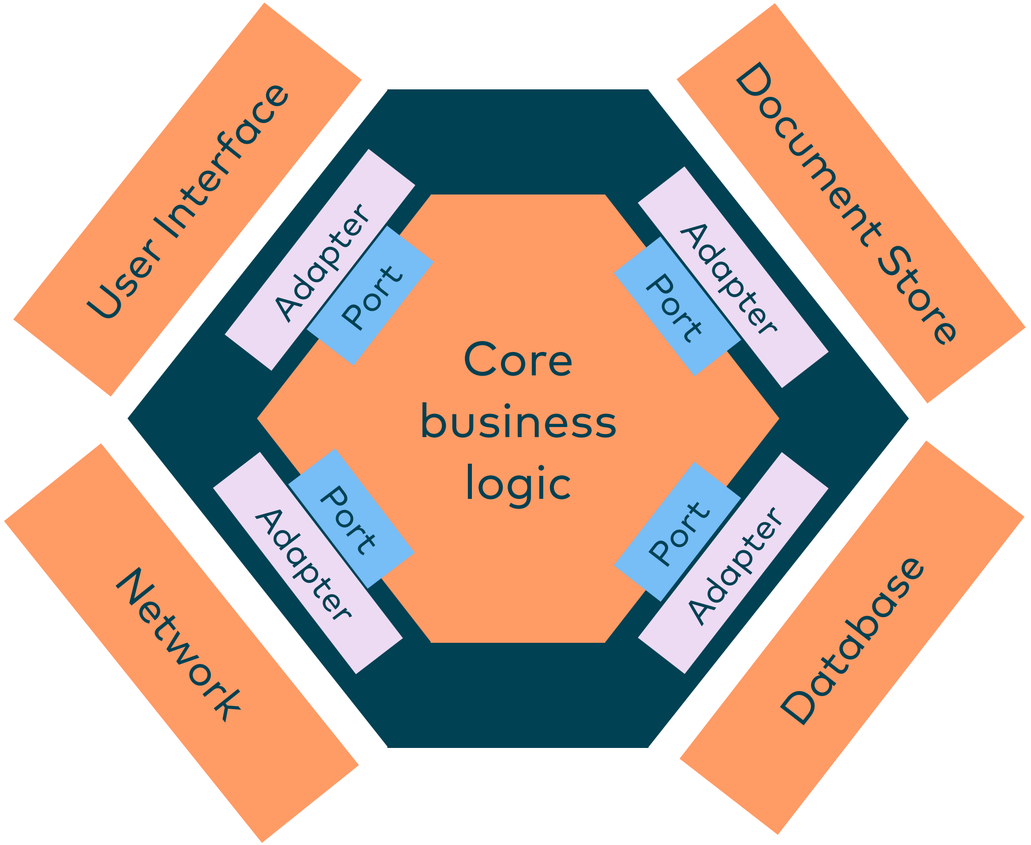
Partnership, Conformist, Customer/Supplier, Anticorruption Layer, Open-host Service, Published Language, Shared Kernel,

Formal Contract, Shared Spec, 3rd Party Standard, ...

[insert whatever makes sense to you]

Should design be domain-driven?

Domain allergy: preferring to explore cool technology to being bothered by learning domain concepts; a disease common among technical developers



Ports and Adapters

Hexagonal Architecture

Clean Architecture

Reality aversion: a failure to recognize that theoretical models tend to break down in practice; a condition often observed among public speakers

Should design be domain-driven?

Yes: Every design should be driven by the domain, not by technology

No: Not every software needs to be built using a technology-neutral OO core

Relational

Focus on using an RDBMS and its abstractions (tables, views, joins, stored procedures ...) for high-performance, data-centric applications

UX/UI/U-driven

Drive design from UI prototypes validated with user experiments, focus on minimalistic, lean implementations to quickly gather feedback, only evolve what works as desired

Algebraic/Denotational

Use mathematical/functional models to generalize/ abstract domain models, apply combinatorial rules to discover new domain logic

Model-driven

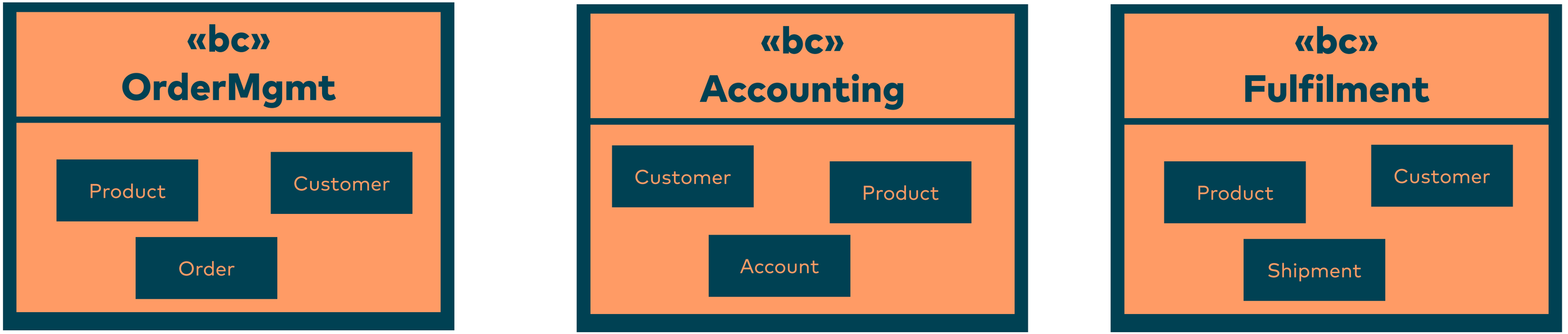
Create technology-independent models outside of programming language environments, use domain-specific languages and model-driven development

Contexts revisited

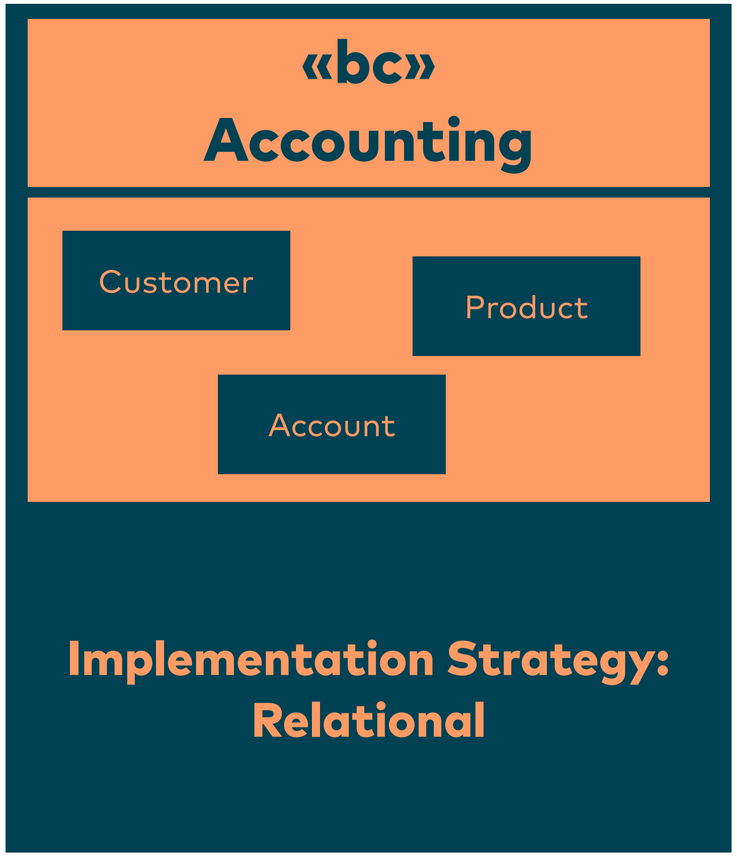
Bounded Context: Definition

A description of a boundary (typically a subsystem or the work of a particular team) within which a particular model is defined and applicable

Eric Evans, https://www.domainlanguage.com/wp-content/uploads/2016/05/DDD_Reference_2015-03.pdf


«bc»
OrderMgmt
Product
Customer
Order
«bc»
Accounting
Customer
Product
Account
«bc»
Fulfilment
Product
Customer
Shipment


«bc»
OrderMgmt
Product
Customer
Order
Implementation Strategy:
UX-driven


«bc»
Accounting
Customer
Product
Account
Implementation Strategy:
Relational


«bc»
Fulfilment
Product
Customer
Shipment
Implementation Strategy:
Tactical DDD

@stilkov

So – is DDD overrated?

Strategic DDD is a great starting point for large systems

... but you may have been doing it with another name

Tactical DDD is one of many great starting points

… but it's best viewed as a micro-level decision

I really appreciate DDD and its community

... but no solution is the only viable one

Summary

1. Don’t look for just one thing

No single approach will workin 100% of all cases, or even 50% – adjust your expectations.

2. Use recipes as starting points

Following rules at the start is fine, but don't be dogmatic

3. Use contexts as decision spheres

Don’t be afraid to use the best tool for the job, even if it’s uncool, old-fashioned or unusual

Thank you! Questions?

Stefan Tilkov
stefan.tilkov@innoq.com
+49 170 471 2625
@stilkov

INNOQ
www.innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin

Ludwigstr. 180E
63067 Offenbach

Kreuzstr. 16
80331 München

Hermannstrasse 13
20095 Hamburg

Erftstr. 15-17
50672 Köln

Königstorgraben 11
90402 Nürnberg