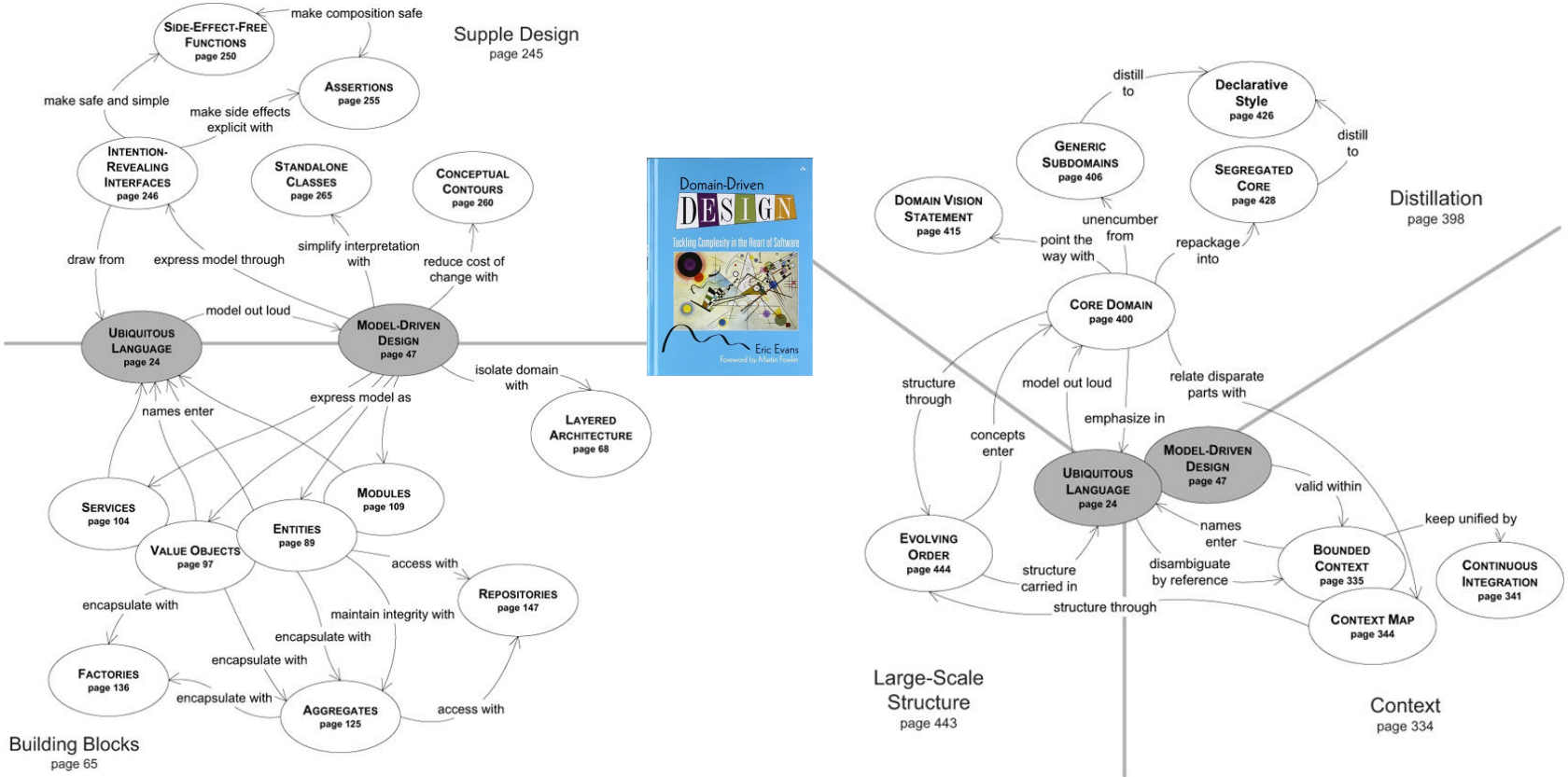
DDD Strategic Design

Introduction to Context Maps

Paulo Clavijo - April 2018

Domain-driven design - 10000 foot view

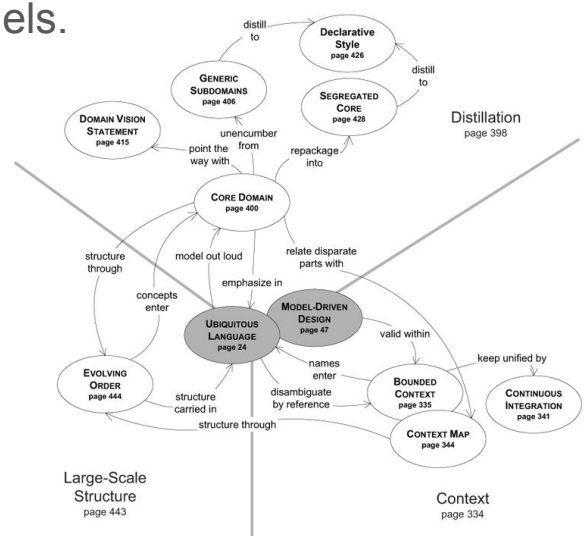


Strategic Design

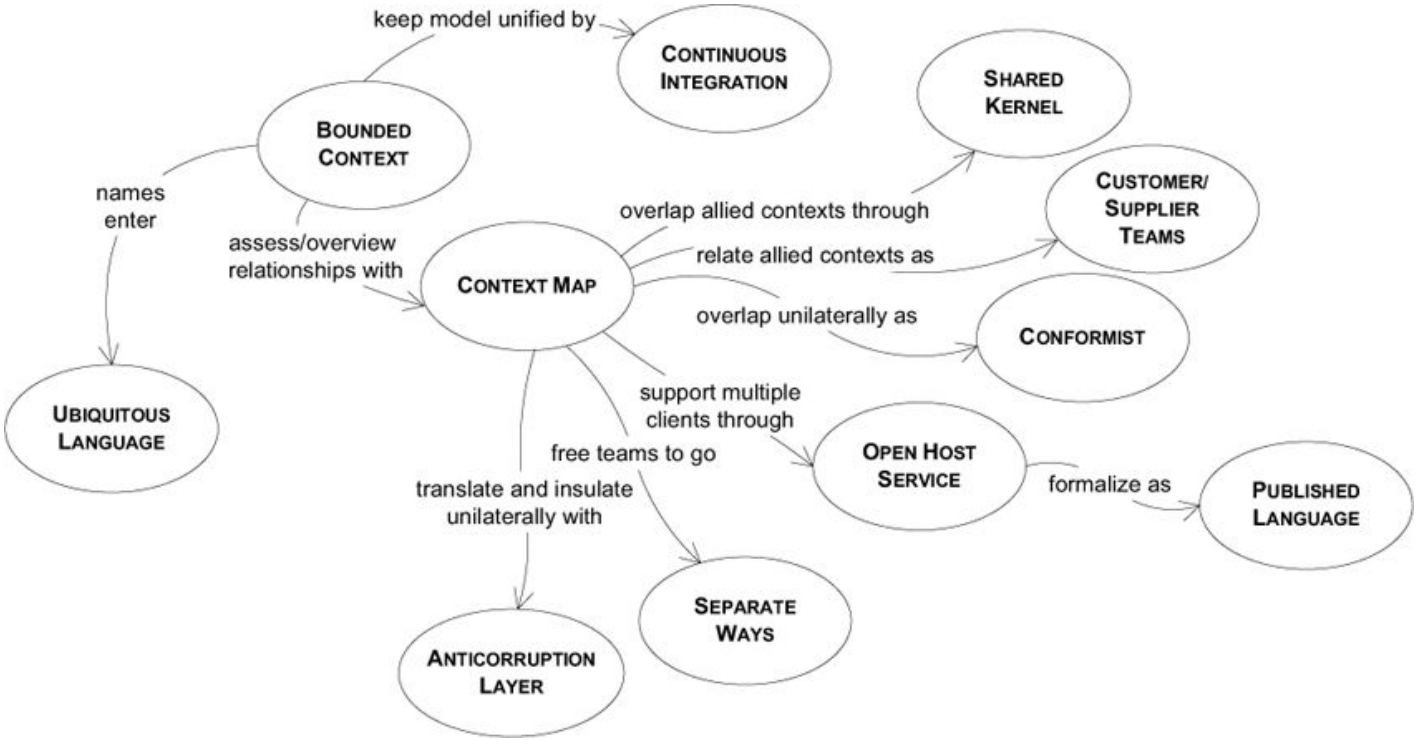
Ideally, it would be preferable to have a single, unified model. While this is a noble goal, in reality it typically fragments into multiple models. It is useful to recognize this fact of life and work with it.

Strategic Design is a set of principles for maintaining model integrity, distillation of the Domain Model and working with multiple models.

- Context
- Distillation
- Large-Scale Structure

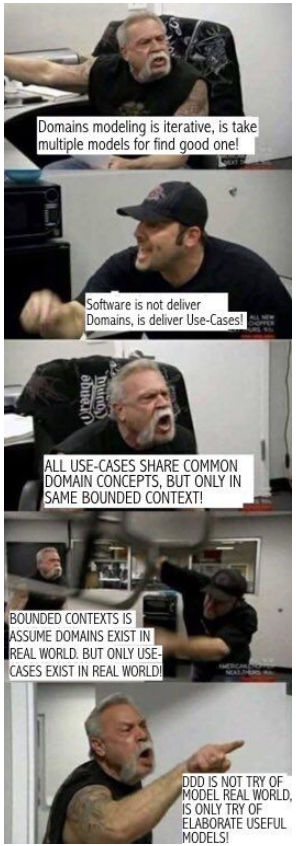


Model Integrity Patterns



DDD Borat
@DDD_Borat

Seguir

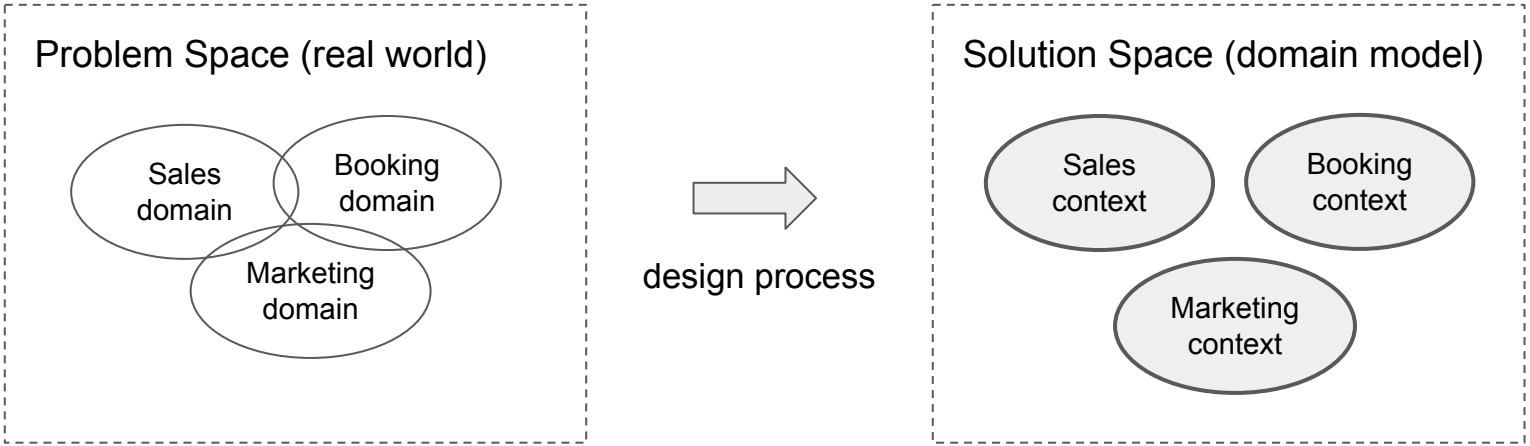


23:58 - 4 abr. 2018

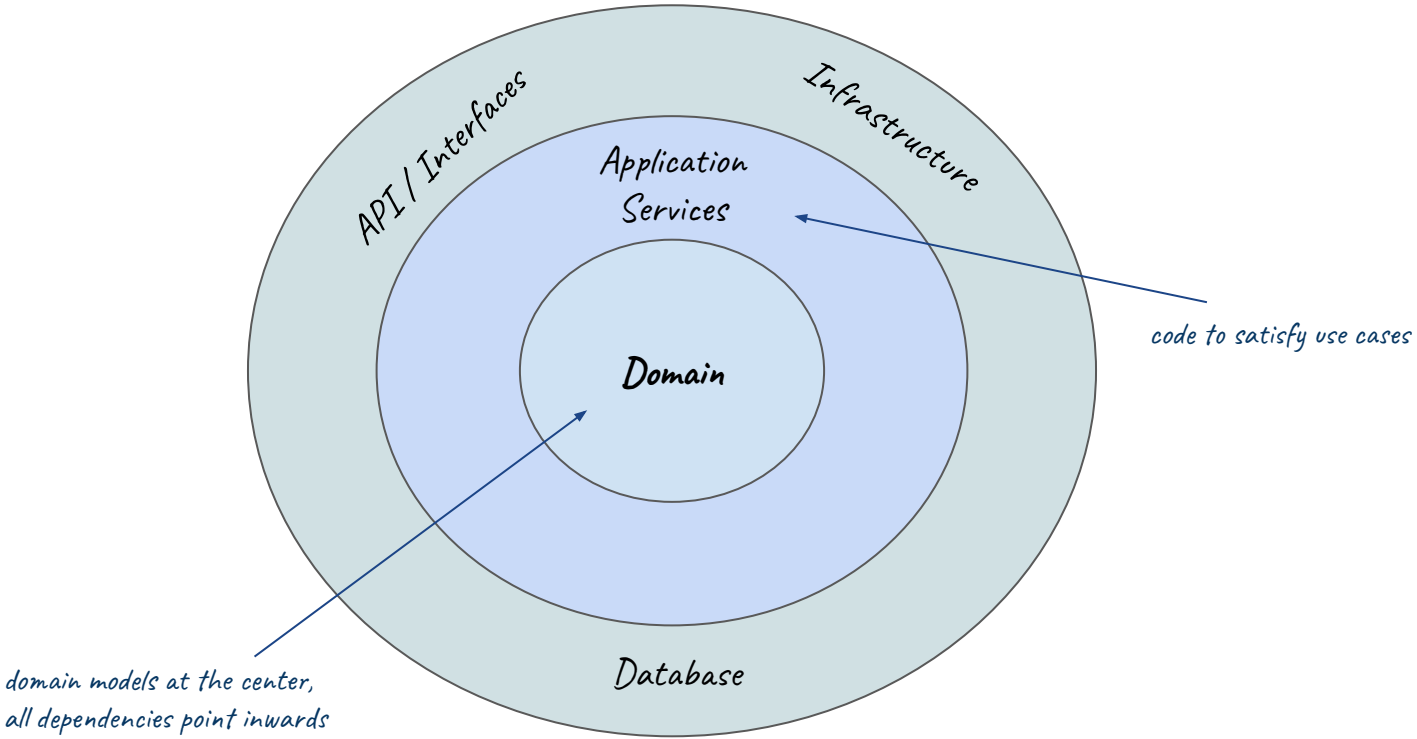
80 Retweets **110** Me gusta

1 80 110

Problem Space vs Solution Space



Architecture

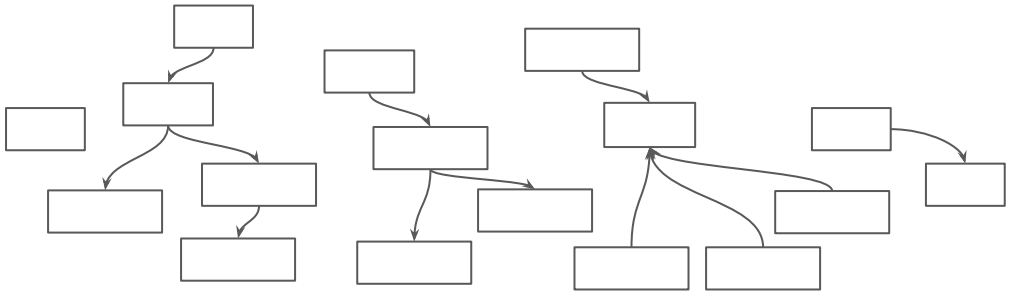


Bounded Context

Multiple models are in play on any large project. Yet when code based on distinct models is combined, software becomes buggy, unreliable, and difficult to understand. Communication among team members becomes confusing. It is often unclear in what context a model should not be applied.

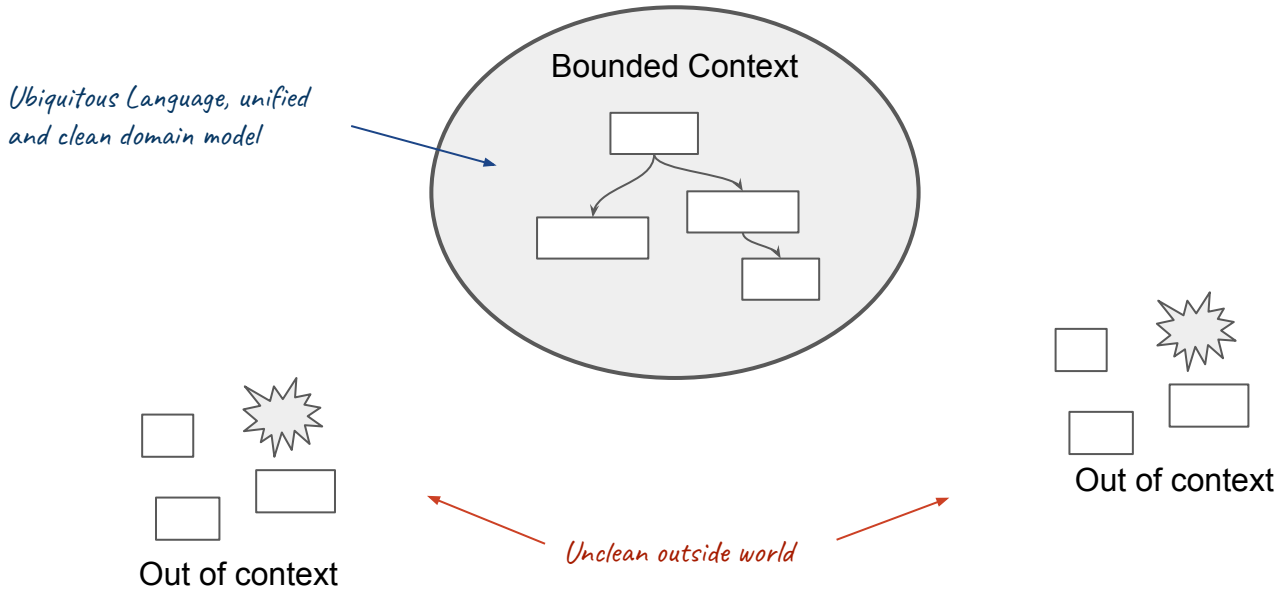
Therefore: Explicitly define the context within which a model applies. Explicitly set boundaries in terms of team organization, usage within specific parts of the application, and physical manifestations such as code bases and database schemas. Keep the model strictly consistent within these bounds, but don't be distracted or confused by issues outside.

Domain-Driven Design, Eric Evans



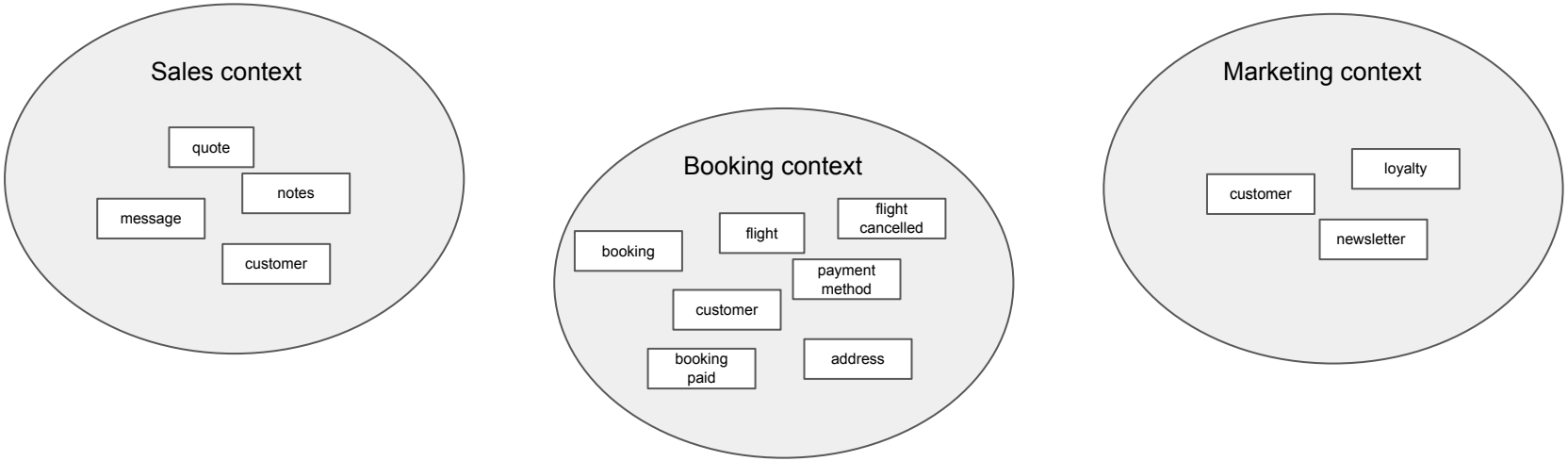
Bounded Context

Bounded Context gives team members a clear and shared understanding of what has to be consistent and what can develop independently



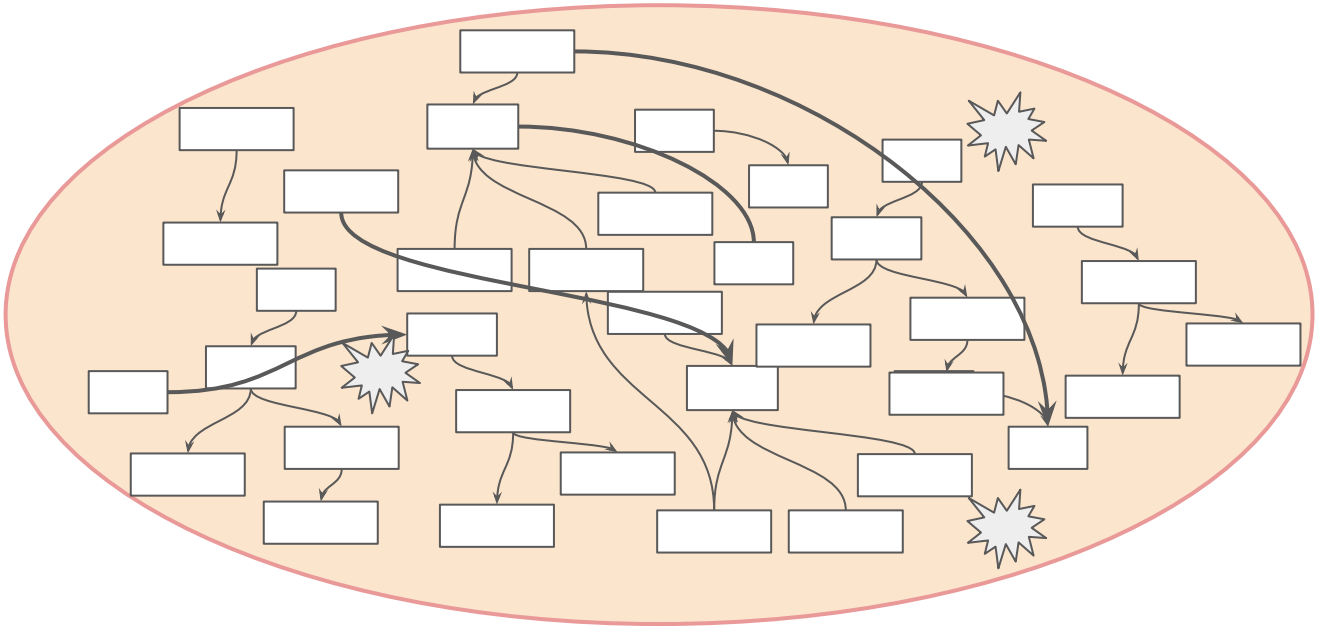
Bounded Context

Large or complex models should be divided into bounded contexts where a model can be explicit by being understood in a specific context.



Bounded Context

Big Ball of Mud



Bounded Context

There is no direct one to one correlation between

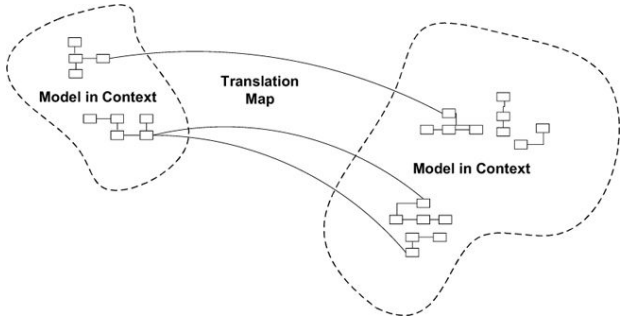
- Domain - Context
- Context - Module
- Context - Micro-service

Context Map

An individual bounded context leaves some problems in the absence of a global view. The context of other models may still be vague and in flux.

People on other teams won't be very aware of the context bounds and will unknowingly make changes that blur the edges or complicate the interconnections. When connections must be made between different contexts, they tend to bleed into each other.

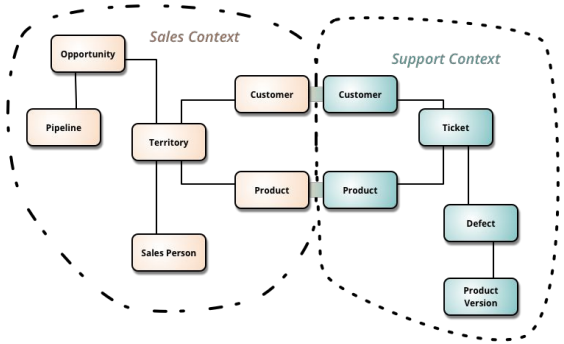
Therefore: Identify each model in play on the project and define its bounded context. This includes the implicit models of non-object-oriented subsystems. Name each bounded context, and make the names part of the ubiquitous language. Describe the points of contact between the models, outlining explicit translation for any communication and highlighting any sharing. Map the existing terrain.



Domain-Driven Design, Eric Evans

Context Map

A model is valid in a specific context. Different context might need different version of the model, due to specific purposes.



Define Bounded Contexts according to the way things are now. This is crucial. To be effective, the Context Map must reflect the true practice of the teams, not the ideal organization

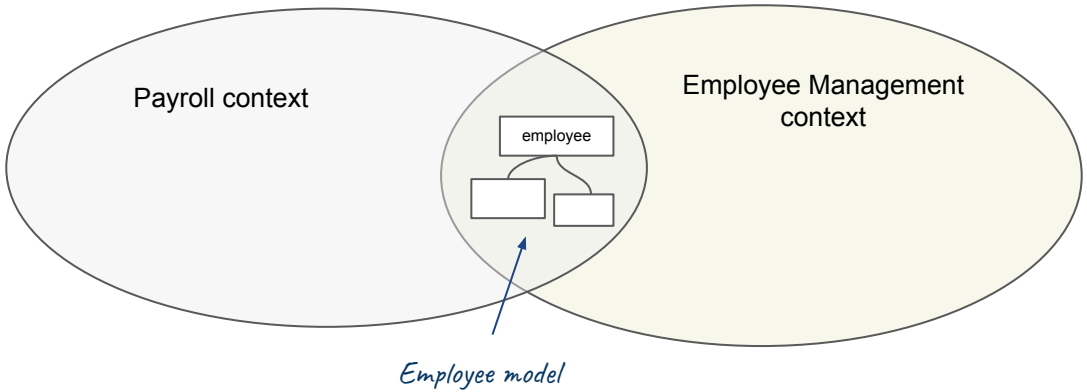
Types of Bounded Context Relationships

- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open Host Service
- Published Language

Shared Kernel

Two teams share a subset of the domain model including code and maybe the database.

Teams involved must collaborate, any change in a kernel model must be done only in consultation with the other team.

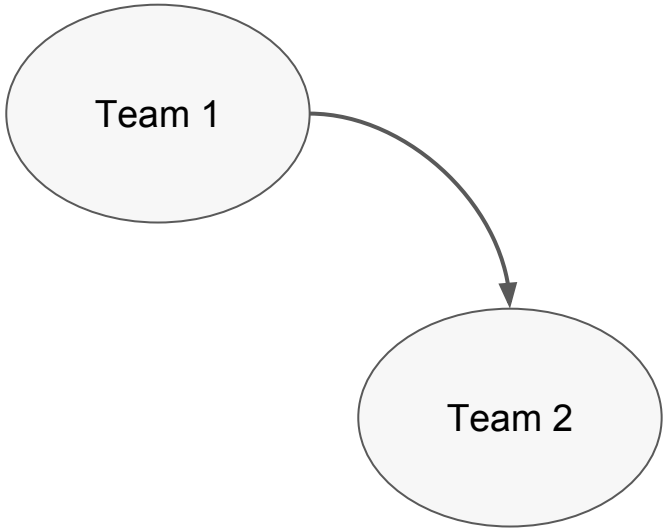


Customer / Supplier

A Customer/Supplier or Consumer-Driven Contract relationship is where the downstream context defines the contract that they want the upstream context to provide.

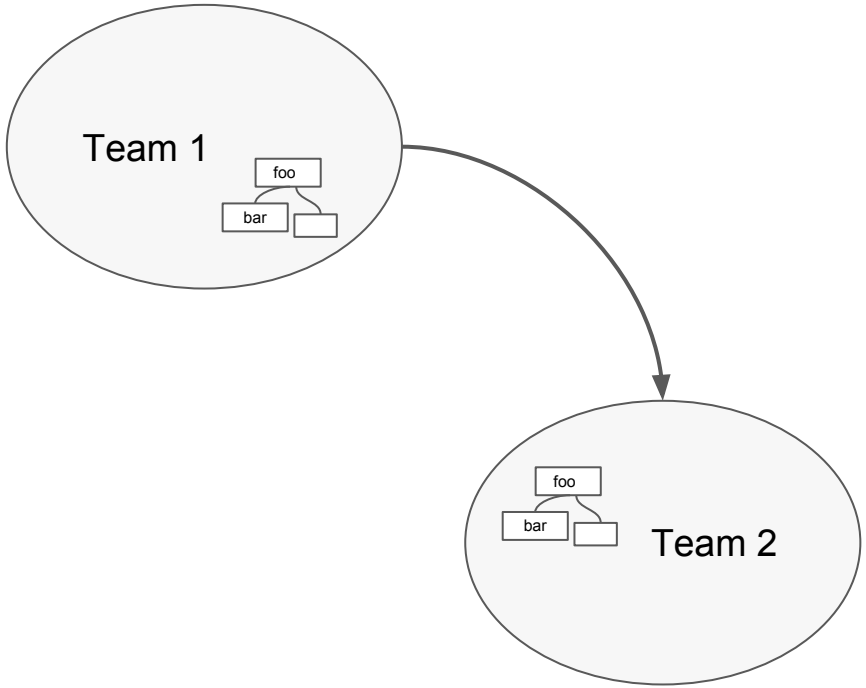
The two domains can still evolve independently, as long as the upstream context fulfills its obligations under the contract.

The downstream team, considered as the customer, sometimes has veto rights.



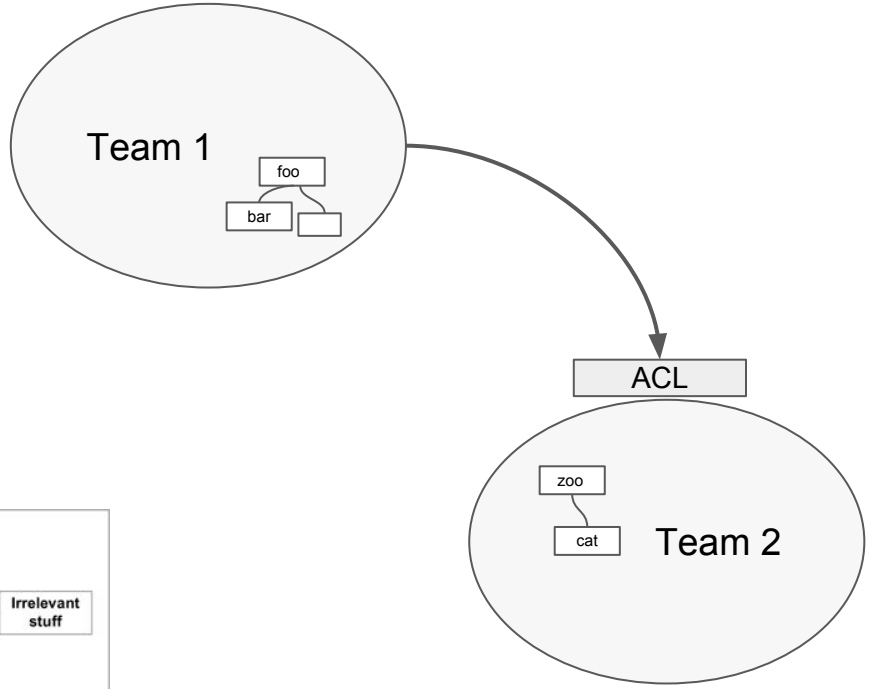
Conformist

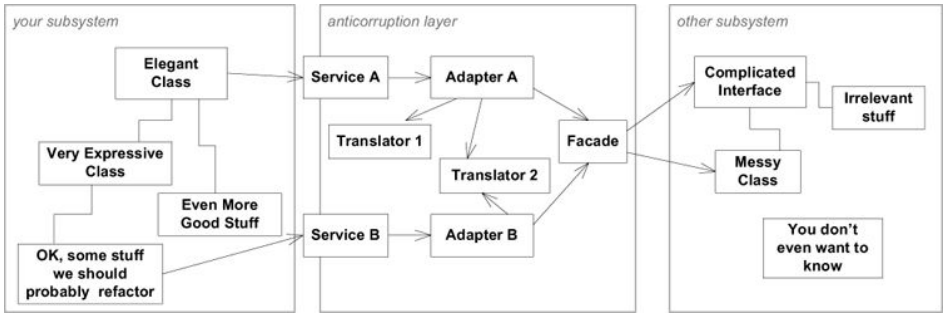
The downstream team conforms to the model of the upstream team. There is no translation of models and not vetoing. If the upstream model is a mess, it propagates to the downstream model.



Anticorruption Layer

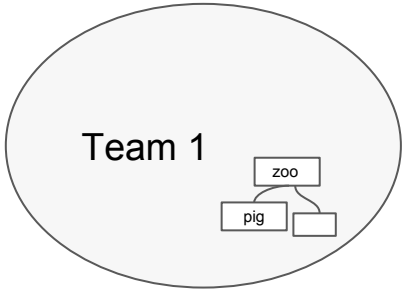
A layer that isolates a client's model from another's system model by translation.

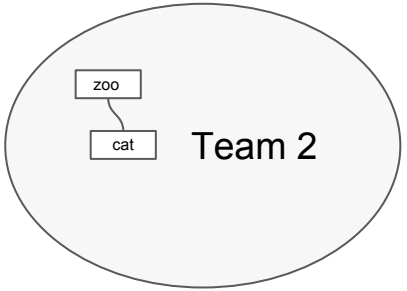




Separate Ways

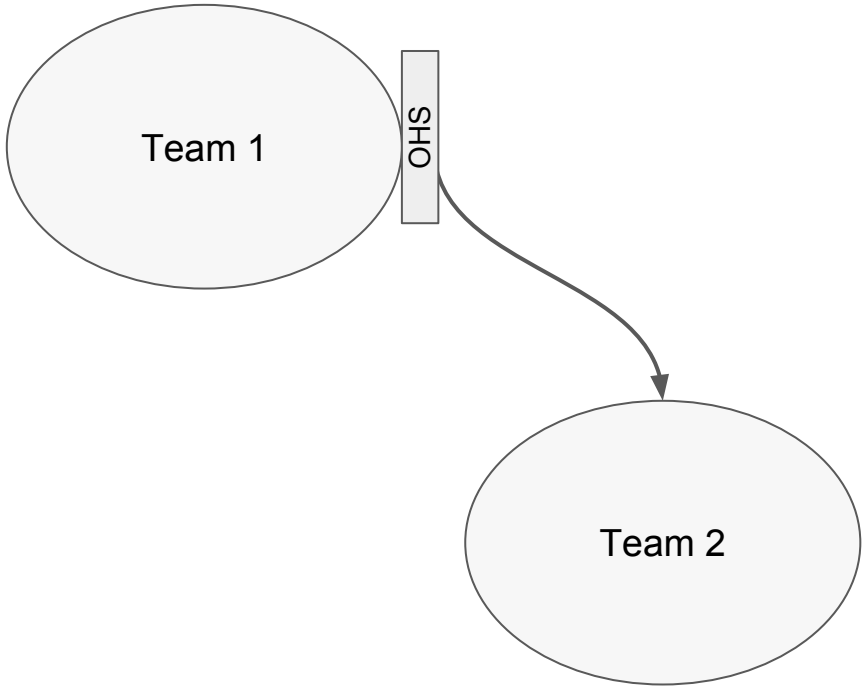
There is no connection between the bounded contexts of a system. This allows teams to find their own solutions in their domain.





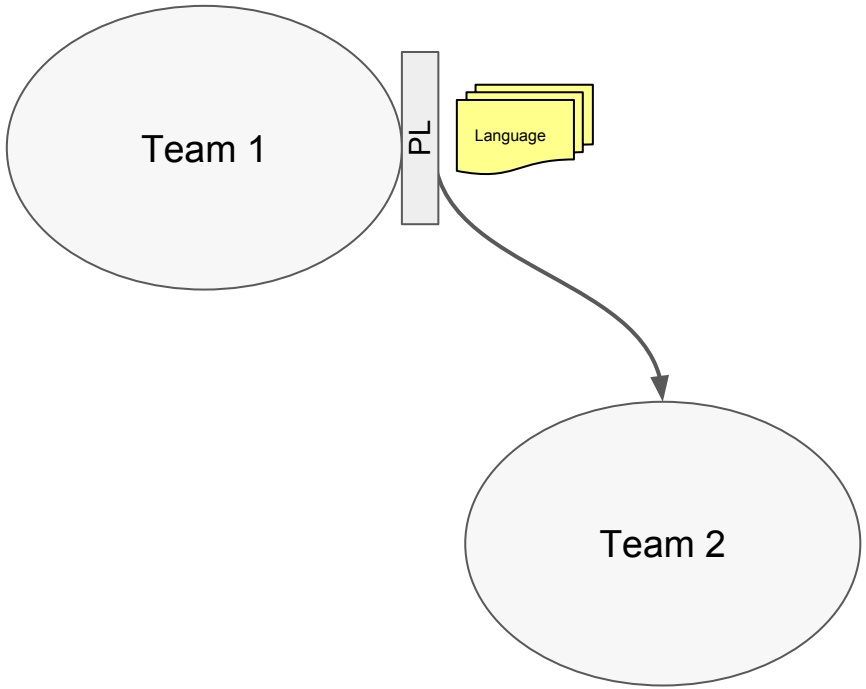
Open Host Service

Each Bounded Context offers a defined set of services that expose functionality for other systems. Any downstream system can then implement their own integration. This is especially useful for integration requirements with many other systems.

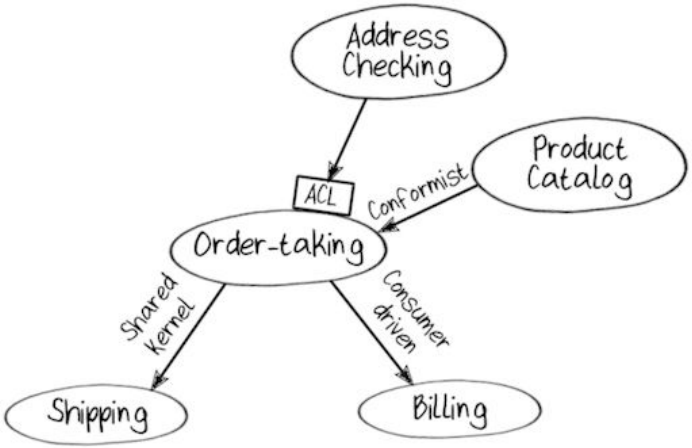


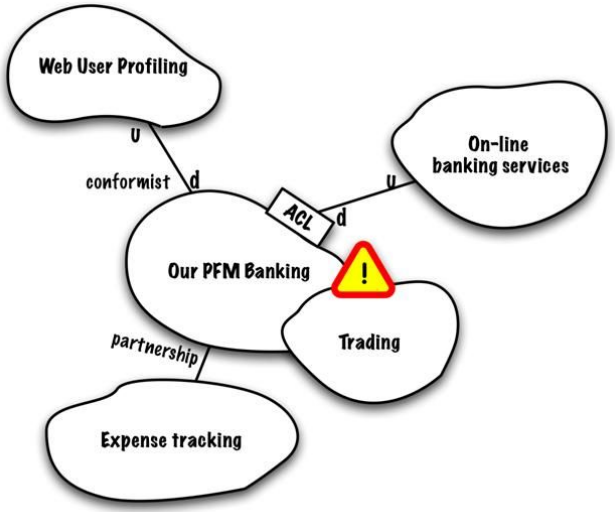
Published Language

Similar to OHS. However it goes as far as to model a Domain as a common language between bounded contexts.

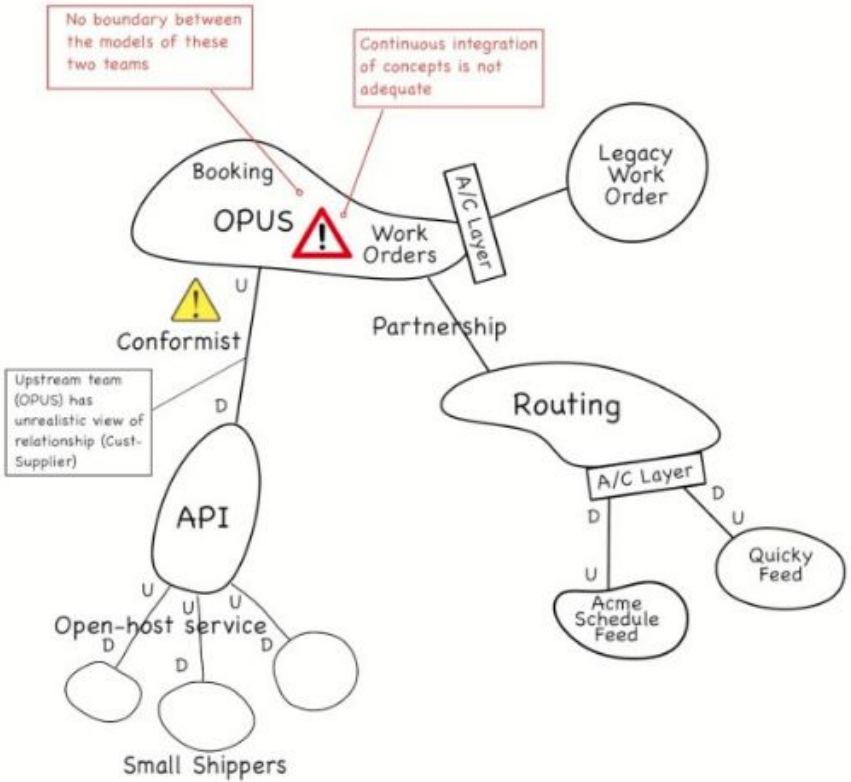


Context Map - Examples



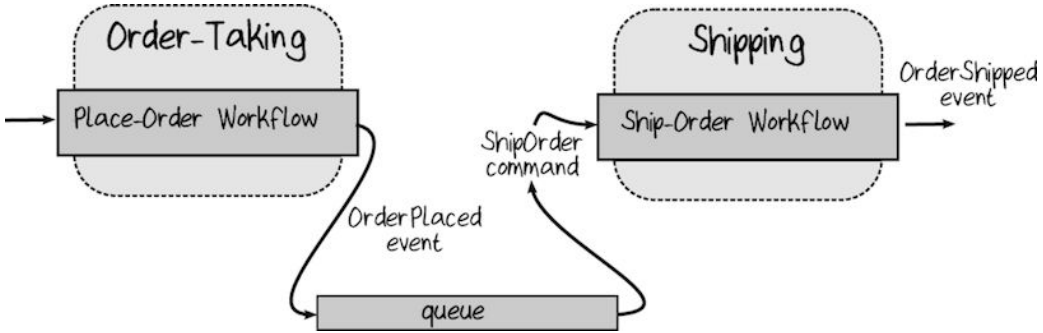


Context Map - Examples

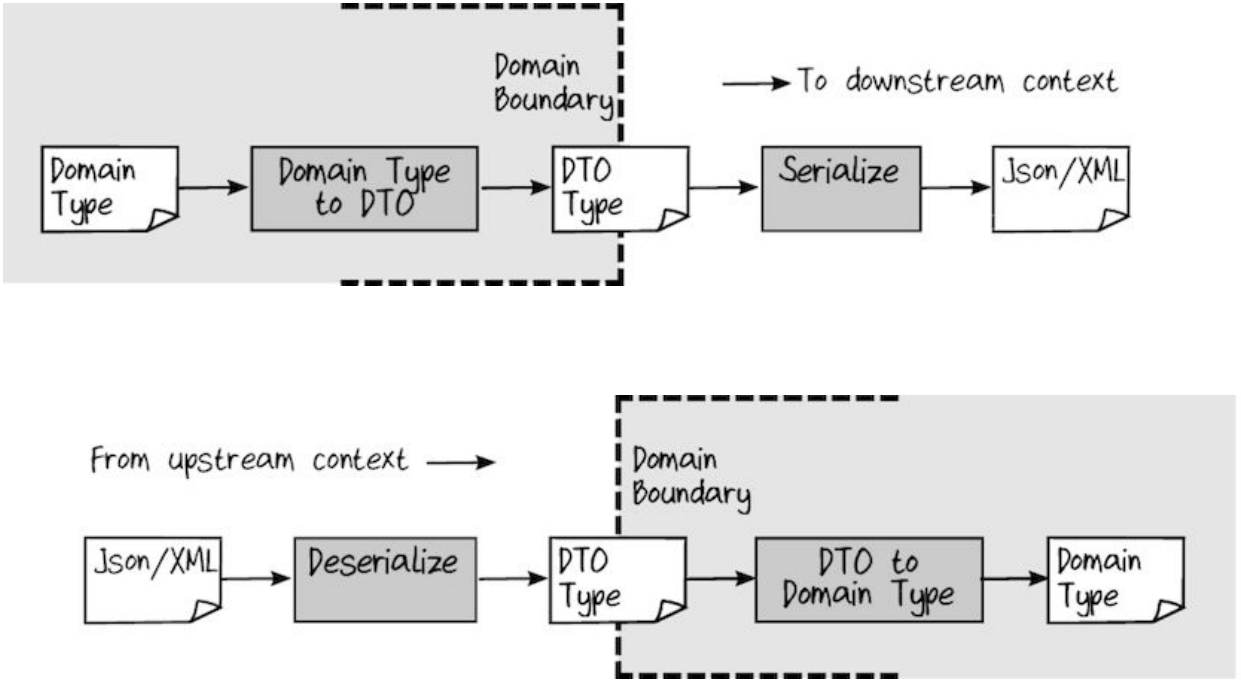


Communication between Bounded Contexts

- Interfaces beetween modules (monolith)
- RPC with SOAP
- REST
- Using Events



Transfering data between Bounded Contexts



References

- Domain-Driven Design. Eric Evans (2003).
- Domain-Driven Design Distilled. Vaughn Vernon (2016).
- Implementing Domain Driven Design. Vaughn Vernon (2013).
- The Anatomy Of Domain-Driven Design. Scott Millett and Samuel Knight.
- Domain Modeling Made Functional. Scott Wlashchin (2018).