DDD

DDD is an approach to software development for complex needs by connecting the implementation to an evolving model. Domain-driven design is predicated on the following goals:

- placing the project's primary focus on the core domain and domain logic
- basing complex designs on a model of the domain
- initiating a creative collaboration between technical and domain experts to iteratively refine a conceptual model that addresses particular domain problems

Taken from "Domain Driven Design" Eric Evans

DDD Concepts

Context

The setting in which a word or statement appears that determines its meaning. DDD deals with large models by dividing them into different Bounded Contexts and being explicit about their interrelationships.

Domain

A sphere of knowledge (ontology), influence, or activity. The subject area to which the user applies a program is the domain of the software

Model

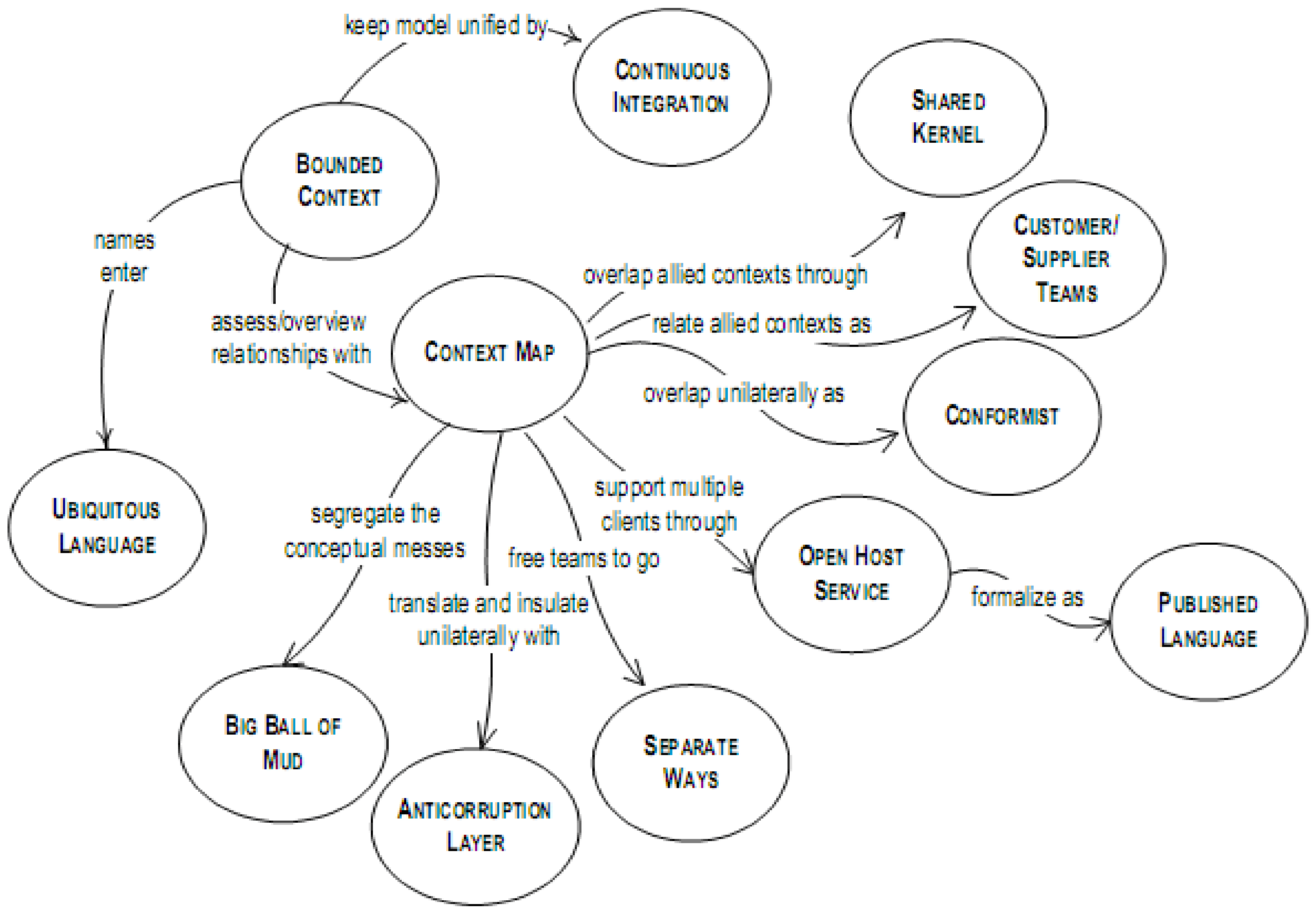
A system of abstractions that describes selected aspects of a domain and can be used to solve problems related to that domain

Ubiquitous Language

A language structured around the domain model and used by all team members to connect all the activities of the team with the software.

https://en.wikipedia.org/wiki/Domain-driven_design
https://martinfowler.com/bliki/BoundedContext.html

Maintaining Model Integrity



DDD components

Entity

An object that is not defined by its attributes, but rather by a thread of continuity and its identity.

Example: Most airlines distinguish each seat uniquely on every flight. Each seat is an entity in this context. Southwest Airlines, EasyJet and Ryanair do not distinguish between every seat; all seats are the same. In this context, a seat is actually a value object.

Value object

An object that contains attributes but has no conceptual identity. They should be treated as immutable.

Example: When people exchange business cards, they generally do not distinguish between each unique card; they are only concerned about the information printed on the card. In this context, business cards are value objects.

https://en.wikipedia.org/wiki/Domain-driven_design

DDD components

Aggregate

A collection of objects that are bound together by a root entity, otherwise known as an aggregate root. The aggregate root guarantees the consistency of changes being made within the aggregate by forbidding external objects from holding references to its members.

Example: When you drive a car, you do not have to worry about moving the wheels forward, making the engine combust with spark and fuel, etc.; you are simply driving the car. In this context, the car is an aggregate of several other objects and serves as the aggregate root to all of the other systems.

Domain Event

A domain object that defines an event (something that happens). A domain event is an event that domain experts care about.

Service

When an operation does not conceptually belong to any object. Following the natural contours of the problem, you can implement these operations in services. See also Service (systems architecture).

https://en.wikipedia.org/wiki/Domain-driven_design

Practice

Part #1. Strategic Design

Define the problem space by dividing the System Domain into a set of subdomains: Core Domain, Supporting/Generic Subdomains.

Define the solution space by identifying the Bounded Contexts. The Context Map should be built to show the relationships between the Bounded Contexts.

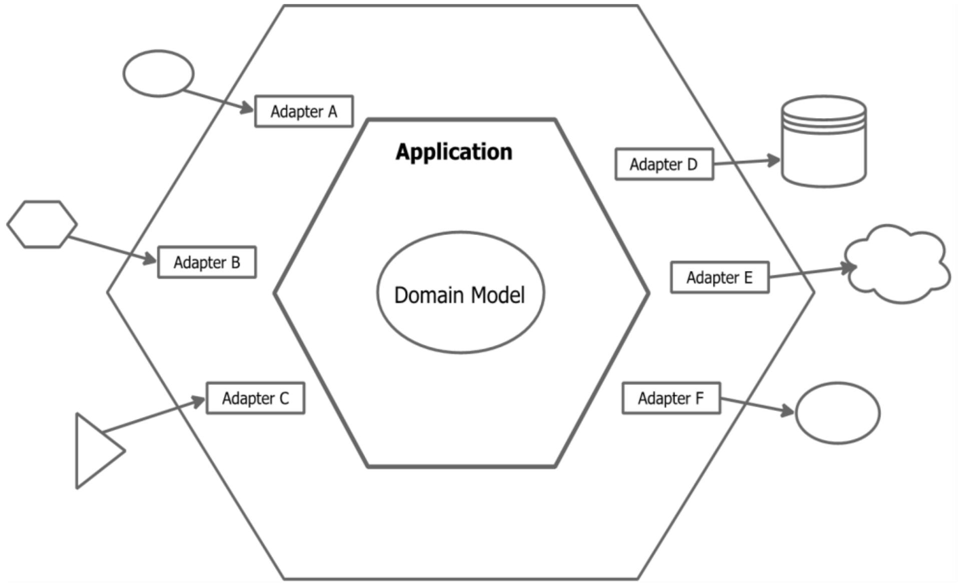
Define the Ubiquitous Language Vocabulary used in each Bounded Context.

Practice

Part #2 Architectural View

Define the overall system architecture. Show how Contexts are going to be integrated. Apply Event Driven Architecture.

For each Bounded Context specify what type of Architecture will be used.


Adapter A
Application
Adapter D
Adapter B
Domain Model
Adapter E
Adapter C
Adapter F