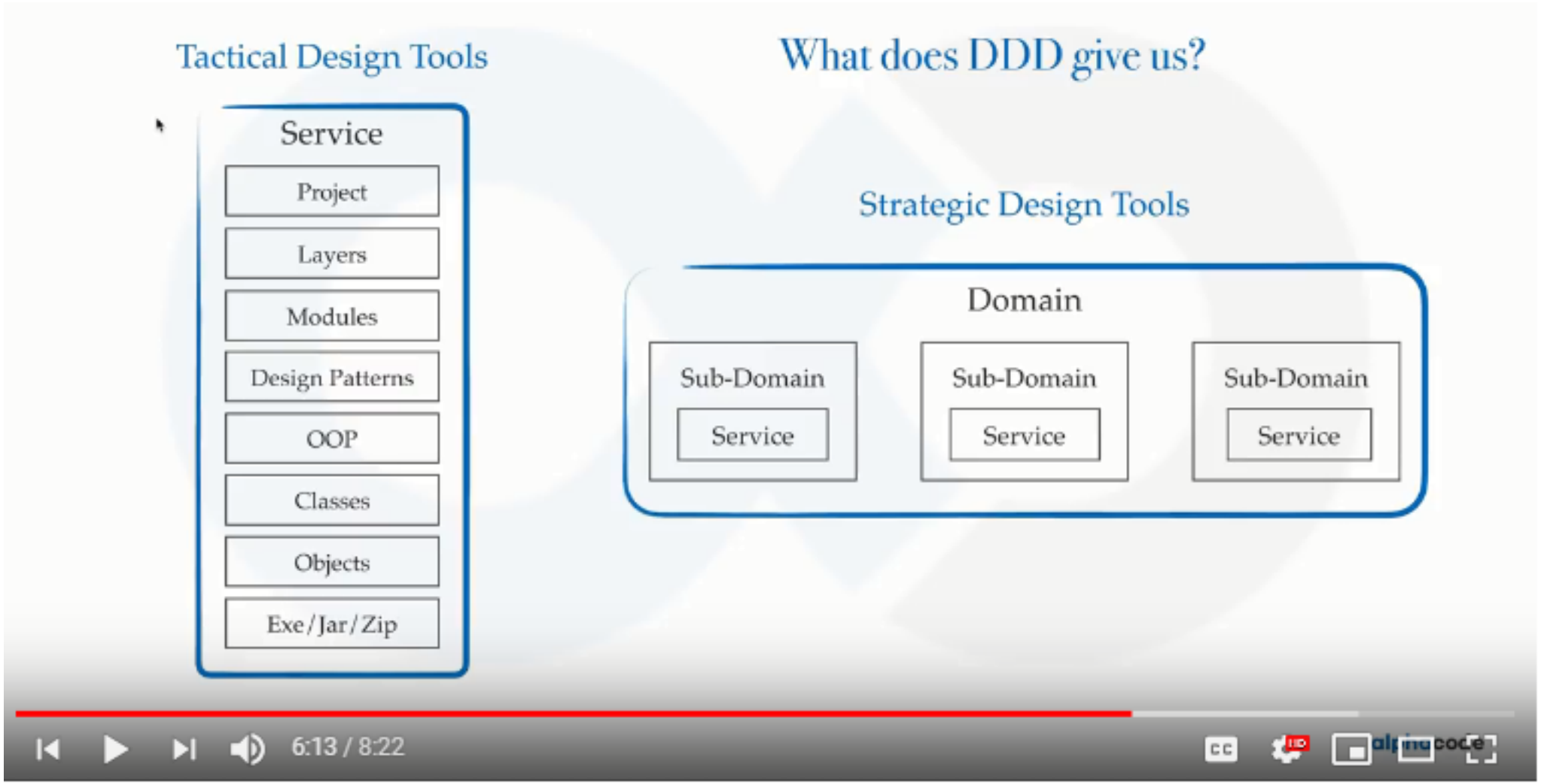


2. What is Domain Driven Design?

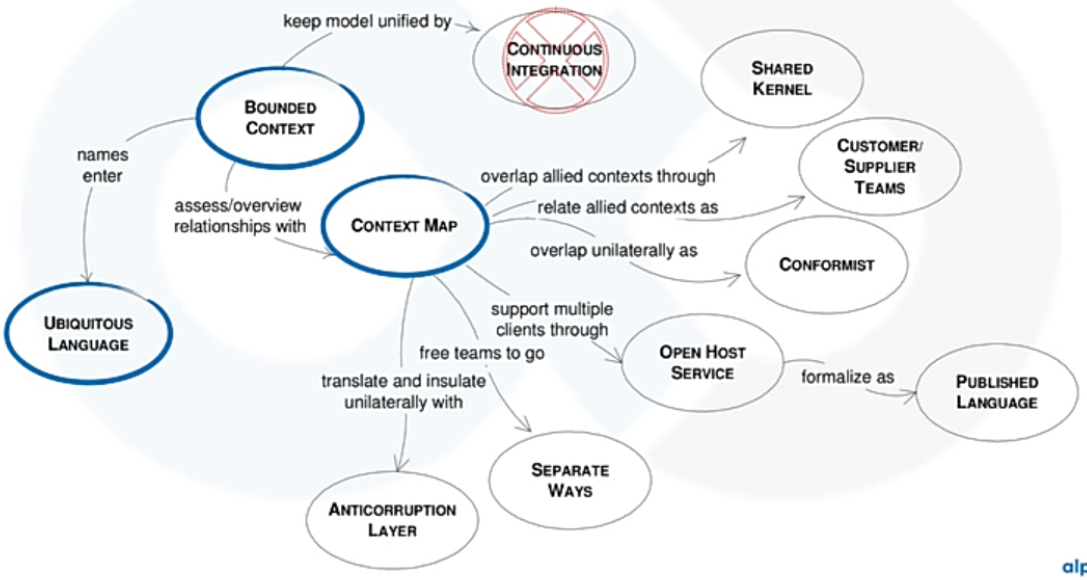
9,056 views

337 117 İndir 720 SHARE SAVE ...

Alpha Code

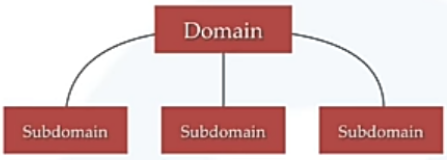
SUBSCRIBE 2.7K

Strategic Design Tools



alphacode

Problem Space



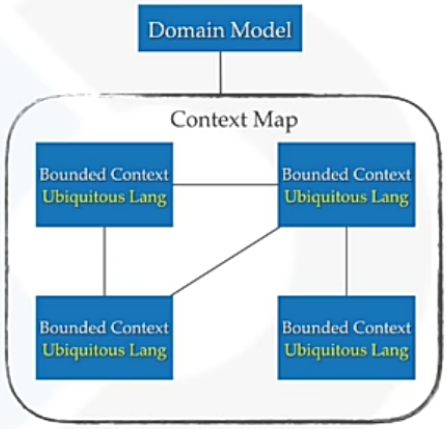
Domain Model - abstracts subdomain

Domain model to Bounded context is like Class to Object

Type of Subdomains

- Core Sub-Domain
- Supporting Sub-Domain
- Generic Subdomains

Solution Space



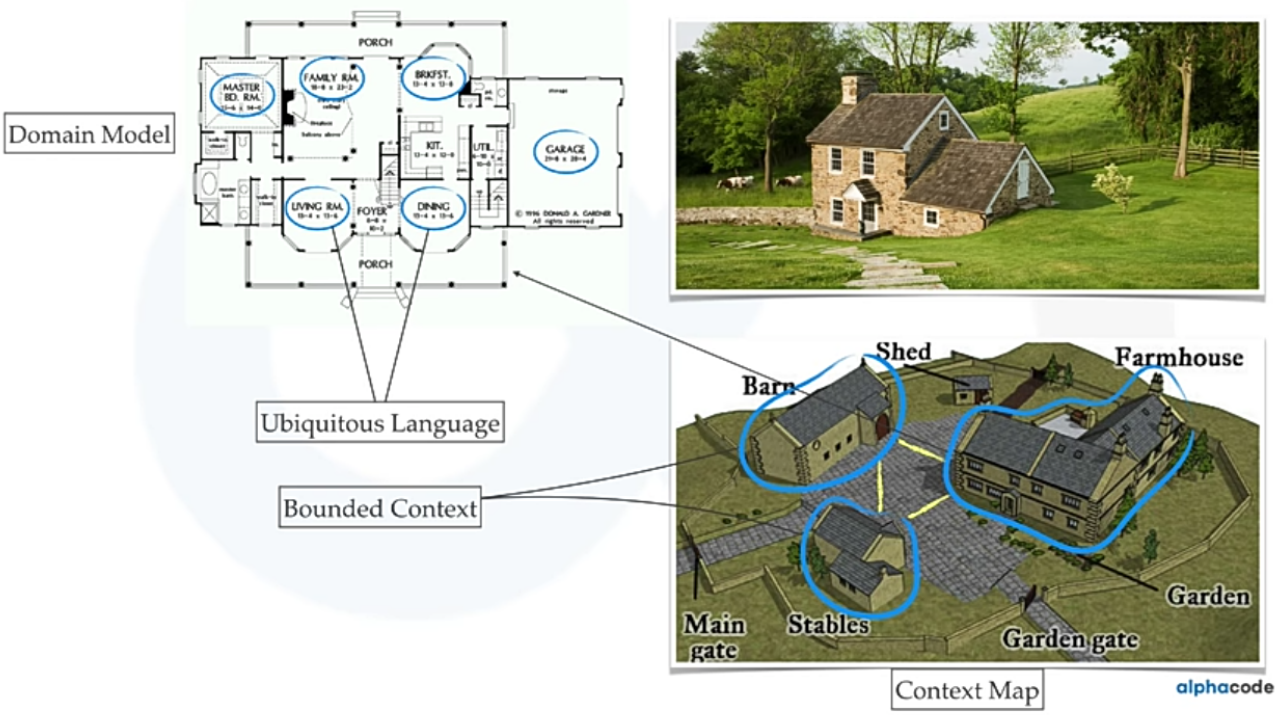
Context Map = Map of relationships between Bounded Contexts

Eg:

Domain: World | Subdomain: Country | Domain model: Country's map | Bounded context: eg Japan

Ubiquitous Language: Japan's administrative nomaclature, which is specific to Japan,
and is only unambigious when mentioned in Japan,
eg: Prefecture in Japan is 1+ provinces. In other countries province means 1+ prefectures

alphacode



alphacode

Context Relationship Types

Published Language: The interacting BCs agree on a common a language (for example a bunch of XML schemas over an enterprise service bus) by which they can interact with each other

Open Host Service: BC specifies a protocol (for e.g a RESTful web service) by which any other BC can use its services

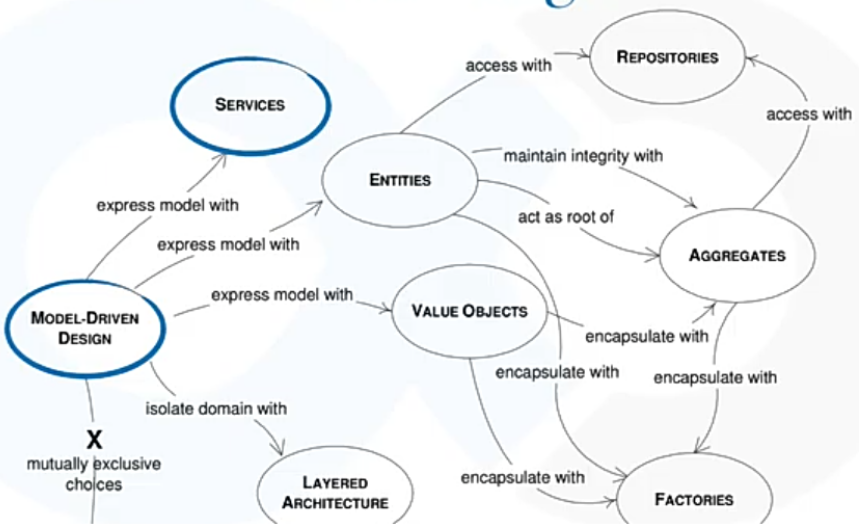
Shared Kernel: Two BCs use a common kernel of code (for example a library) as a common lingua-franca, but otherwise do their other stuff in their own specific way

Customer/Supplier: One BC uses the services of another and is a stakeholder (customer) of that other BC. As such it can influence the services provided by that BC

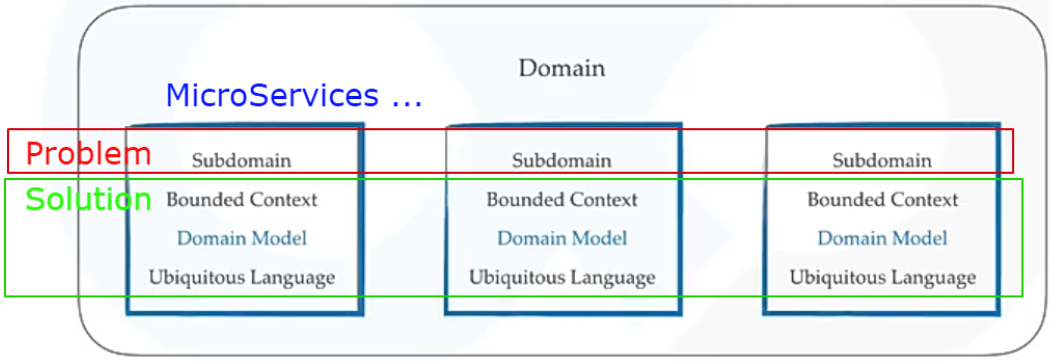
Conformist: One BC uses the services of another but is not a stakeholder to that other BC. As such it uses "as-is" (conforms to) the protocols or APIs provided by that BC

Anti-Corruption Layer: One BC uses the services of another and is not a stakeholder, but aims to minimize impact from changes in the BC it depends on by introducing a set of adapters.

Tactical Design



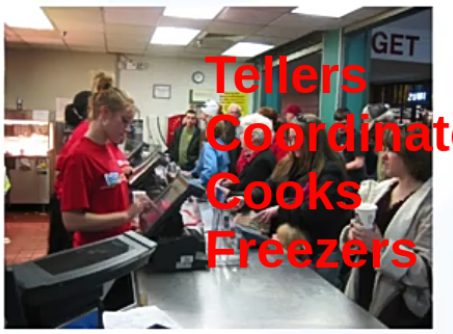
Model Driven Design



alphacode

Layered Architecture

McDonalds analogy



Request Handlers	Presentation, UI, View
Controllers	Application, Service
Business	Business logic, Domain
Persistence	Data access, logging, networking, and other services which are required to support a particular business layer

alphacode

Value Objects

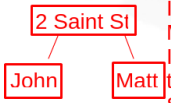
- One of the best things about good design
- Eg. What is a String?
- It's a general purpose value object designed to handle complexities of char arrays
- Value objects reduces complexity and forces ubiquitous language

Float for Money

String for Email

Features of a Value Object?

- Don't care about uniqueness
- Always immutable
- Rich domain logic
- Auto-validating
- Strong equality
- Thread safe



If address (2 Saint St) is value object.
Matt & John both stay there.
If John move houses,
then the value object (2 Saint St)
SHOULDNT change.
If otherwise,
then Matt's address would also change,
which is wrong

Entities

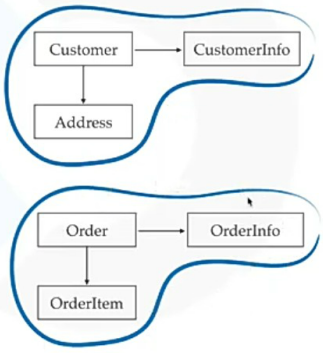
- Can be uniquely identified using an ID
- Consists of value objects
- Generally persisted as a row in DB
- Typically mutable
- Generally implements some business logic

If address matters, then it should have ID and be an Entity.
EG: Electricity billing company charging to a particular address.
If address dont matter, then it can be a Value Object.
EG: Post office deliver mails to John and Matt, who stays at the same address.
The important thing here is an unique person ID, not an unique address ID.

alphacode

Aggregates

- When the object graph becomes big it becomes difficult to maintain
- An aggregate is collection of entities and values which comes under a single transaction boundary
- An aggregate controls the change
- An aggregate always has a root entity
- The root entity governs the lifetime of other entities in the aggregate
- An aggregate is always consistent
- Domain events are generated to ensure eventual consistency



alphacode

Factories & Repositories

Entities & Values

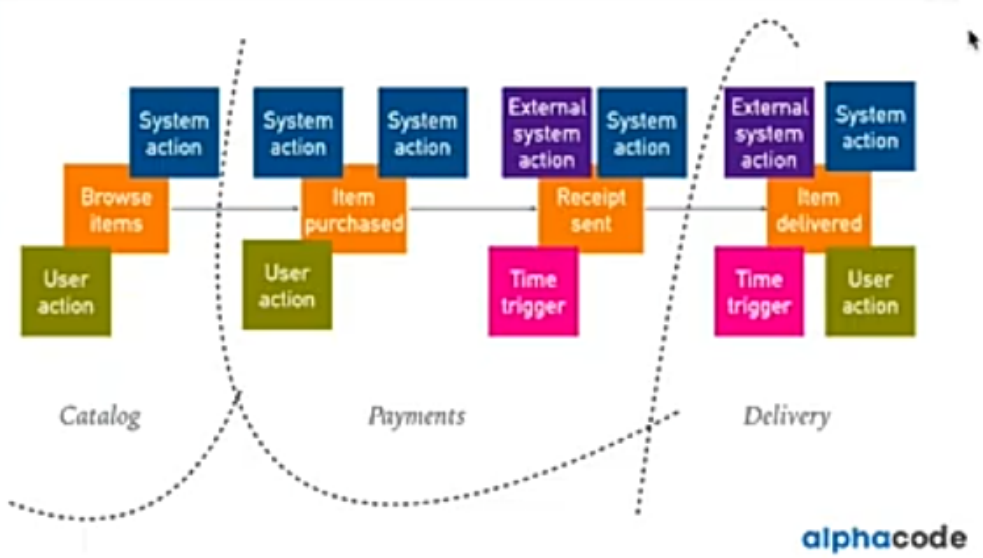
Aggregate

- Factories helps you create new aggregates
- Repository helps you get persisted aggregates
- Repository ≠ DAO

alphacode

How does event storming work?

- First bring the right people in
- Then take sticky notes and colour code them to events, commands, policies, processes, errors, roles and aggregates etc
- Take a white board and start writing interesting events in sequence
- Then start adding commands, aggregates, policies etc
- After the first pass, start identifying bounded contexts



alphacode

Implementing DDD in code

```
public class ShippingController extends Controller{
  private CartService cartService;

  public void process(ShippingForm form) throws Exception, SQLException{
    CartBean cartBean = cartService.getCart(form.getCartId());
    BigDecimal cost = null;

    if (cartBean.getTotal() <= 100){
      cost = new BigDecimal(4.99);
      if (form.getOption() == 1) {
        for (Item i : cartBean.getItems()) {
          if (i.getCat() == 'B') {
            cost.add(new BigDecimal(2.99));
          }
          if (i.getCat() == 'U') {
            cost.add(new BigDecimal(i.getWeight() / 1000)
                .multiply(new BigDecimal(2.99)));
          }
        }
      }
      if (form.getOption() == 2) {
        for (Item i : cartBean.getItems()) {
          if (i.getCat() == 'B') {
            cost.add(new BigDecimal(4.99));
          }
          if (i.getCat() == 'U') {
            cost.add(new BigDecimal(i.getWeight() / 1000)
                .multiply(new BigDecimal(2.99)));
          }
        }
      }
    }
    form.setCost(cost.setScale(0, ROUND_HALF_EVEN));
  }
```

- Magic Numbers
- Duplicate code
- Primitive Obsession
- Mixed Concerns
- Fuzzy Terminology