Workshop "Experiencing Domain Driven Design"

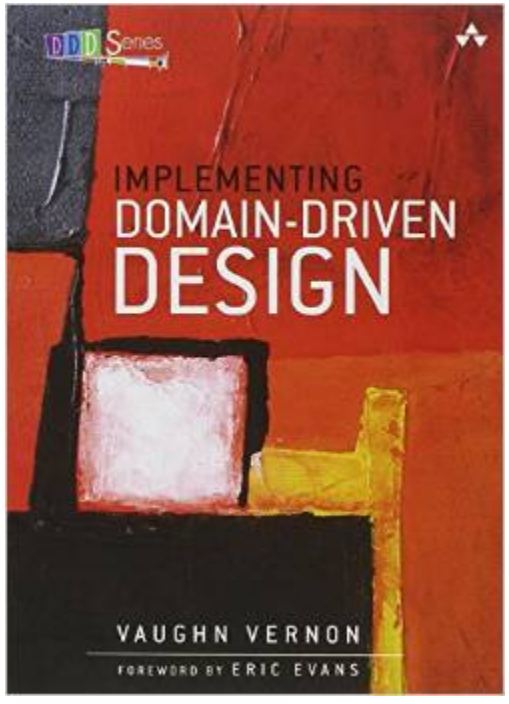
Summary

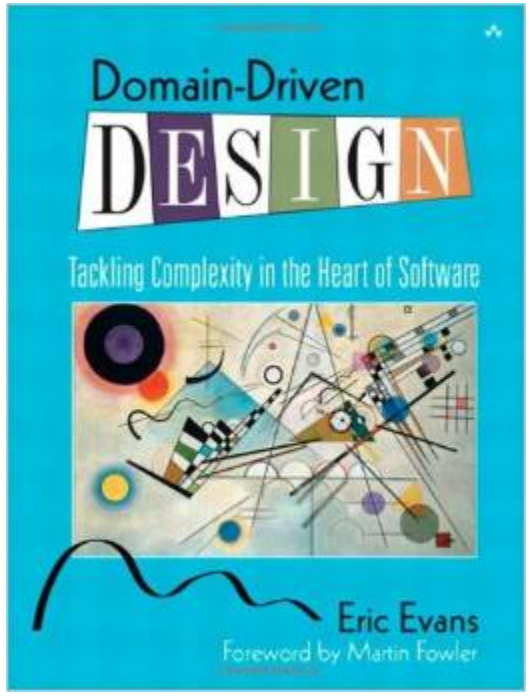
29-sept-2014

Workshop Experiencing Domain Driven Design

- Cursusgever: Mattias Verraes http://verraes.net/#talks
- "Workshop" van drie dagen.
- Programma dag 1: Intro / Systems thinking, Event storming..
- Programma dag 2 (n.a.v. voting): "Heuristics", Boeken, Context mapping.
- Programma dag 3: Event sourcing, CQRS, Event store, Model storming.

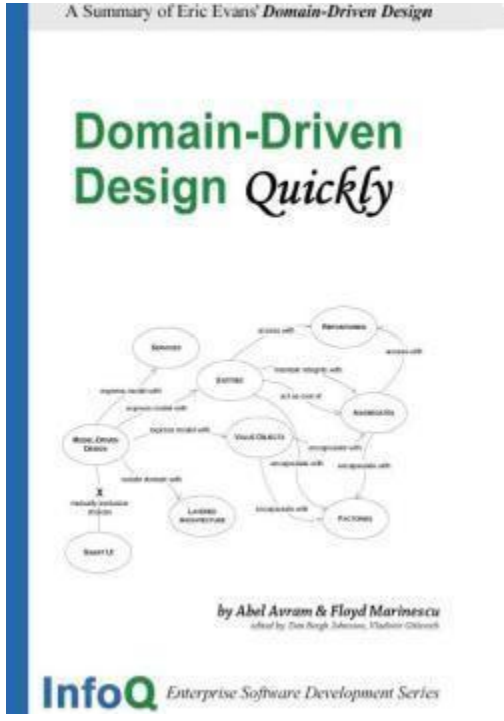
Domain Driven Design





Domain Driven Design Summary

- Summary "Domain Driven Design Quickly"

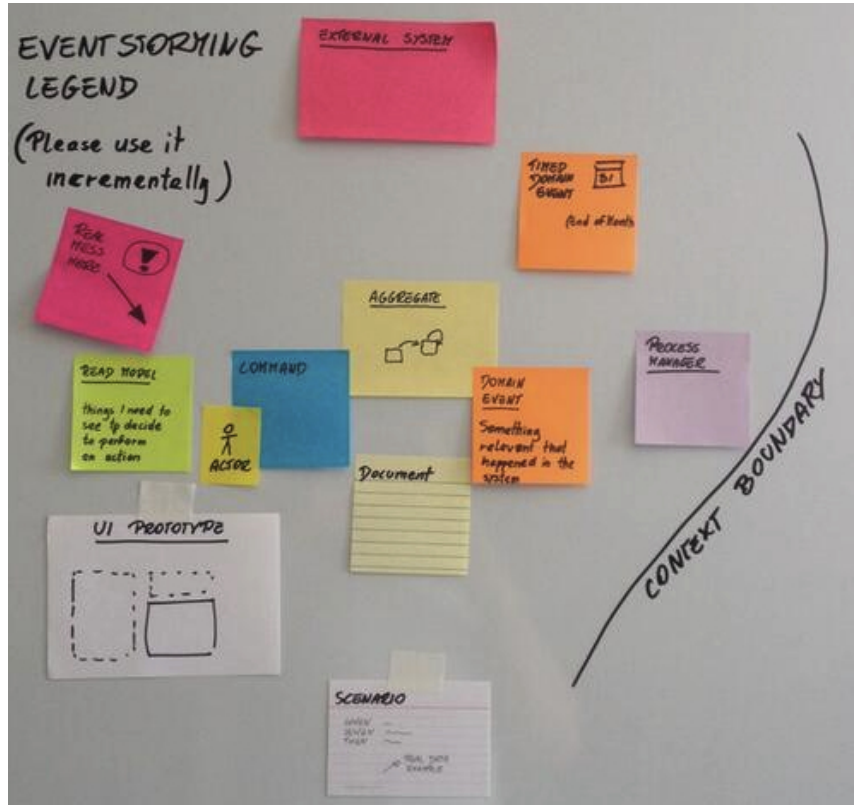


DDD

- Supple design
- Declarative design
- Refactoring
- Bounded contexts
- Distillation
- Large-scale structure
- Modelling whirlpool
- Model storming

Event storming

- Alberto Brandolini (link: here)
- Important events for the business.
- Events ==> Past tense and in the Ubiquitious language.
- No consensus first, better to have alternatives.



1) Find Domain Events

1) Find Domain Events

- Happened in the past (expressed in past form: A car was rented, Customer has subscribed)

1) Find Domain Events

- Happened in the past (expressed in past form: A car was rented, Customer has subscribed)
- Interest the business

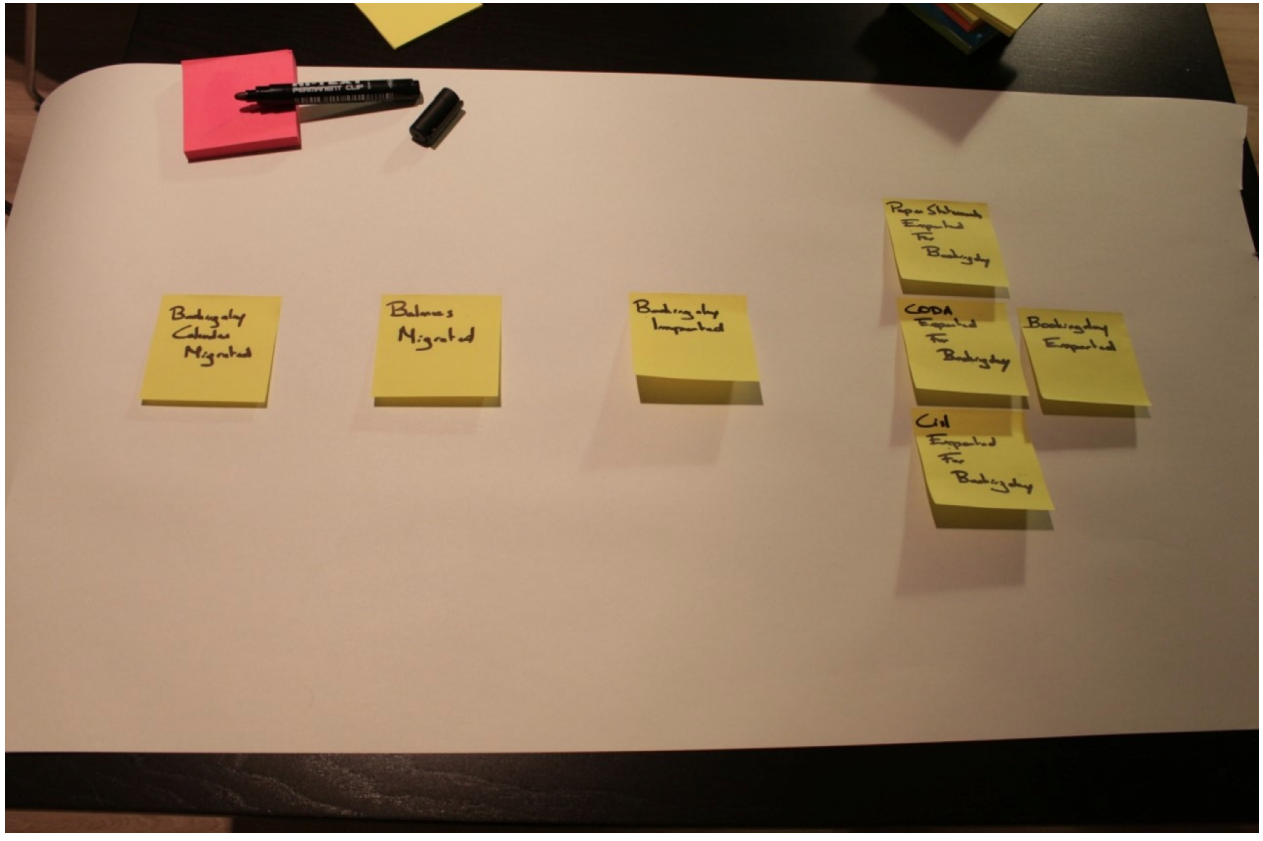
1) Find Domain Events

- Happened in the past (expressed in past form: A car was rented, Customer has subscribed)
- Interest the business
- Expressed in Ubiquitous language

1) Find Domain Events

- Happened in the past (expressed in past form: A car was rented, Customer has subscribed)
- Interest the business
- Expressed in Ubiquitous language

NO technical events like Network is down, Transaction has failed...


Balances
Migrated
Budgetary
Imported
COOA
Budgetary
Budgetary
Imported

2) Find Commands

- Expressed in imperative phrase (Rent a car, Subscribe..)

2) Find Commands

- Expressed in imperative phrase (Rent a car, Subscribe..)
- They express intent/cause for the business

2) Find Commands

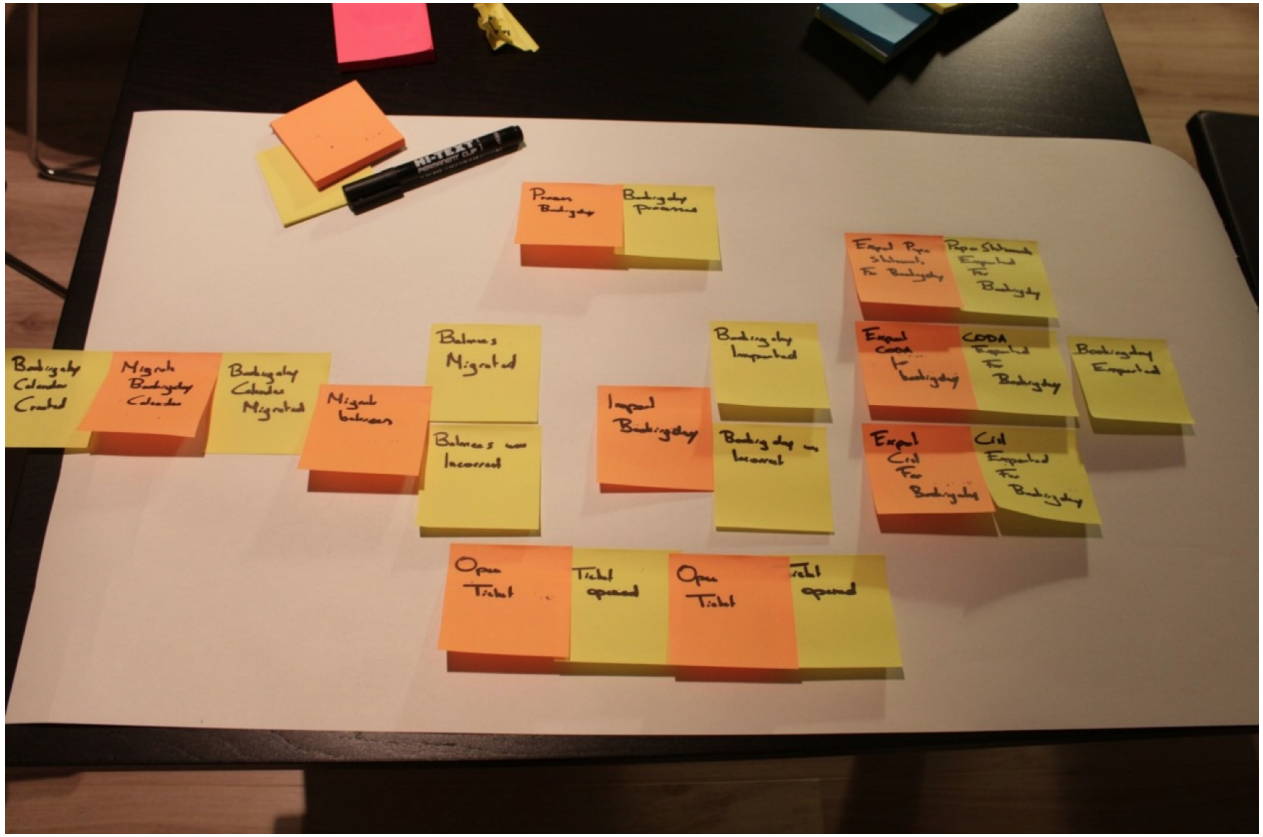
- Expressed in imperative phrase (Rent a car, Subscribe..)
- They express intent/cause for the business
- They can fail

2) Find Commands

- Expressed in imperative phrase (Rent a car, Subscribe..)
- They express intent/cause for the business
- They can fail
- Multiple outcomes (including exceptions)

2) Find Commands

- Expressed in imperative phrase (Rent a car, Subscribe..)
- They express intent/cause for the business
- They can fail
- Multiple outcomes (including exceptions)
- Possible sources: human, time, process.


Budgeting Calendar Created
Migrate Budgeting Calendar
Budgeting Calendar Migrated
Migrate balances
Balances Migrated
Balances are Incorrect
Import Budgeting
Budgeting Imported
Budgeting is Incorrect
Open Ticket
Ticket opened
Open Ticket
Ticket opened

3) Discover Bounded Contexts

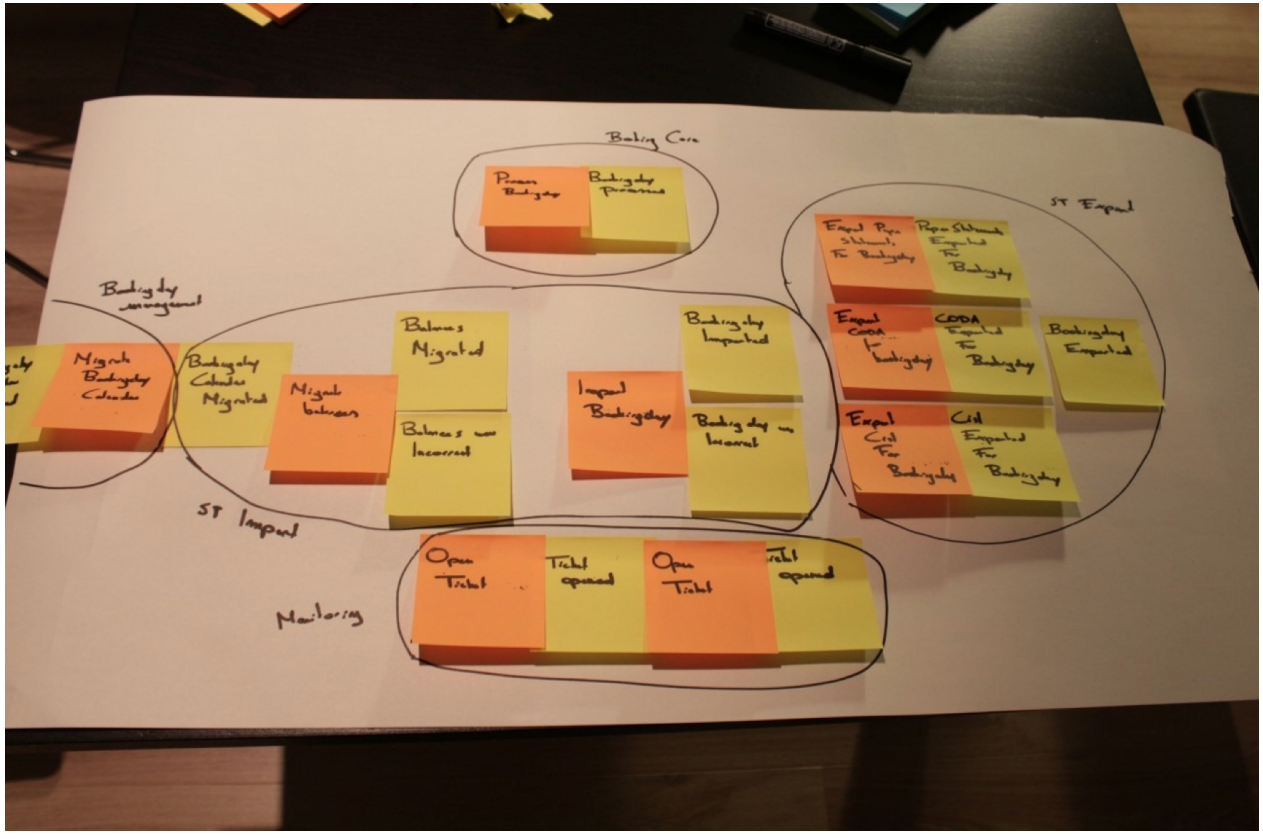
- Explicit boundary between domain models.

3) Discover Bounded Contexts

- Explicit boundary between domain models.
- Each context has it's unique ubiquitious language

3) Discover Bounded Contexts

- Explicit boundary between domain models.
- Each context has it's unique ubiquitious language
- Example: Fleet Management, Scheduling, ..


Booking Core
Person Booking
Booking Processed
ST Export
Export Pax Submit. Fr Booking
Pax Submit Exported Fr Booking
Export CODA Fr booking
CODA Exported Fr Booking
Booking Exported
Export Ctrl Fr Booking
Ctrl Exported Fr Booking
Booking management
Migrate Booking Calendar
Booking Calendar Migrated
Migrate balances
Balances Migrated
Balances are incorrect
ST Import
Import Booking Day
Booking Day Imported
Booking day is incorrect
Monitoring
Open Ticket
Ticket opened
Open Ticket
Ticket opened

4) Find Business rules / decisions

- Put them on the map to make the different scenarios explicit

4) Find Business rules / decisions

- Put them on the map to make the different scenarios explicit
- Example: An already cancelled reservation cannot be cancelled anymore

5) Group events that influence decision

6) Discover & experiment with aggregate boundaries

- Aggregate is a group of objects that work together and are treated as a unit.

6) Discover & experiment with aggregate boundaries

- Aggregate is a group of objects that work together and are treated as a unit.
- Provide a specific functionality.

6) Discover & experiment with aggregate boundaries

- Aggregate is a group of objects that work together and are treated as a unit.
- Provide a specific functionality.
- *Transactional*

6) Discover & experiment with aggregate boundaries

- Aggregate is a group of objects that work together and are treated as a unit.
- Provide a specific functionality.
- *Transactional*
- *Cohesive*

Heuristics

Heuristics are different approaches we want to confront with while building our models so that we can push our mind to influence our design in different directions.

Heuristics

Heuristics are different approaches we want to confront with while building our models so that we can push our mind to influence our design in different directions. All those heuristics lead us to try different models and throw them away at each time until we are happy with our model.

Heuristics 1/5

- Stable (static data) vs Volatile (changing data);

Heuristics 1/5

- Stable (static data) vs Volatile (changing data);
- Type (hierarchy of objects) vs Properties (of an object);

Heuristics 1/5

- Stable (static data) vs Volatile (changing data);
- Type (hierarchy of objects) vs Properties (of an object);
- Actor vs Roles;

Heuristics 1/5

- Stable (static data) vs Volatile (changing data);
- Type (hierarchy of objects) vs Properties (of an object);
- Actor vs Roles;
- Change together vs change separately;

Heuristics 1/5

- Stable (static data) vs Volatile (changing data);
- Type (hierarchy of objects) vs Properties (of an object);
- Actor vs Roles;
- Change together vs change separately;
- Different lifecycle dependencies;

Heuristics 1/5

- Stable (static data) vs Volatile (changing data);
- Type (hierarchy of objects) vs Properties (of an object);
- Actor vs Roles;
- Change together vs change separately;
- Different lifecycle dependencies;
- Immutable vs Mutable;

Heuristics 1/5

- Stable (static data) vs Volatile (changing data);
- Type (hierarchy of objects) vs Properties (of an object);
- Actor vs Roles;
- Change together vs change separately;
- Different lifecycle dependencies;
- Immutable vs Mutable;
- Immutability within a context only;

Heuristics 1/5

- Stable (static data) vs Volatile (changing data);
- Type (hierarchy of objects) vs Properties (of an object);
- Actor vs Roles;
- Change together vs change separately;
- Different lifecycle dependencies;
- Immutable vs Mutable;
- Immutability within a context only;
- Retroactive Business rules;

Heuristics 2/5

- Cohesive vs Decoupled;

Heuristics 2/5

- Cohesive vs Decoupled;
- Afferent coupling (things that are coupled to me) vs Efferent coupling (things that I couple to); Noun < Sentences < Phrases;

Heuristics 2/5

- Cohesive vs Decoupled;
- Afferent coupling (things that are coupled to me) vs Efferent coupling (things that I couple to); Noun < Sentences < Phrases;
- Follow the money;

Heuristics 2/5

- Cohesive vs Decoupled;
- Afferent coupling (things that are coupled to me) vs Efferent coupling (things that I couple to); Noun < Sentences < Phrases;
- Follow the money;
- Use personas;

Heuristics 3/5

3 archetypes we find in almost every software :

Heuristics 3/5

3 archetypes we find in almost every software :

- Collaborative construction;

Heuristics 3/5

3 archetypes we find in almost every software :

- Collaborative construction;
- Execution;

Heuristics 3/5

3 archetypes we find in almost every software :

- Collaborative construction;
- Execution;
- Business intelligence *tracking* monitoring;

Heuristics 4/5

- Consistency

Heuristics 4/5

- Consistency
- Same rules but different semantics;

Heuristics 4/5

- Consistency
- Same rules but different semantics;
- Warnings vs errors;

Heuristics 4/5

- Consistency
- Same rules but different semantics;
- Warnings vs errors;
- Synchronous (ordered) vs Asynchronous (unordered) [on a business point of view, not a technical one]; Happy path vs divergent path;

Heuristics 4/5

- Consistency
- Same rules but different semantics;
- Warnings vs errors;
- Synchronous (ordered) vs Asynchronous (unordered) [on a business point of view, not a technical one]; Happy path vs divergent path;
- Responsibility Layers (see Evans);

Heuristics 4/5

- Consistency
- Same rules but different semantics;
- Warnings vs errors;
- Synchronous (ordered) vs Asynchronous (unordered) [on a business point of view, not a technical one]; Happy path vs divergent path;
- Responsibility Layers (see Evans);
- Atomicity / Transactionality;

Heuristics 4/5

- Consistency
- Same rules but different semantics;
- Warnings vs errors;
- Synchronous (ordered) vs Asynchronous (unordered) [on a business point of view, not a technical one]; Happy path vs divergent path;
- Responsibility Layers (see Evans);
- Atomicity / Transactionality;
- Risks;

Heuristics 5/5

- Strictness (do we restrict user) vs Allow the user to abuse the system and monitor the abuse;

Heuristics 5/5

- Strictness (do we restrict user) vs Allow the user to abuse the system and monitor the abuse;
- Start from the end;

Heuristics 5/5

- Strictness (do we restrict user) vs Allow the user to abuse the system and monitor the abuse;
- Start from the end;
- Let the human do it;

Brownfield DDD

Start by drawing the Current Model and the Desired Model. Make a small step toward the Desired Model. Throw away both drawing models regularly while in the process of leading the current model to the desired model, so that desired model also evolved with the maturity of our understanding.

Brownfield DDD

Have a wall were people write identified Technical Debt. Each time someone stumble upon the same debt, put a dot in front of it. This helps us identify the most critical debt.

Context Mapping

How do our different contexts communicate?

Context Mapping

How do our different contexts communicate?

Relationships fall into repeatable patterns..

Context Mapping 1/2

- Shared kernel

Context Mapping 1/2

- Shared kernel
- Customer / supplier

Context Mapping 1/2

- Shared kernel
- Customer / supplier
- Published language ("language from a long running domain")

Context Mapping 1/2

- Shared kernel
- Customer / supplier
- Published language ("language from a long running domain")
- Partnership

Context Mapping 1/2

- Shared kernel
- Customer / supplier
- Published language ("language from a long running domain")
- Partnership
- Separate ways (rebuild the stuff you need)

Context Mapping 2/2

- Anticorruption layer (translate from upstream to downstream model)

Context Mapping 2/2

- Anticorruption layer (translate from upstream to downstream model)
- Conformist (mimic upstream model)

Context Mapping 2/2

- Anticorruption layer (translate from upstream to downstream model)
- Conformist (mimic upstream model)
- Open Host Service (public 3rd party api we cannot influence (google maps)).

Context Mapping 2/2

- Anticorruption layer (translate from upstream to downstream model)
- Conformist (mimic upstream model)
- Open Host Service (public 3rd party api we cannot influence (google maps)).
- Big Ball of Mud

Context Mapping 2/2

- Anticorruption layer (translate from upstream to downstream model)
- Conformist (mimic upstream model)
- Open Host Service (public 3rd party api we cannot influence (google maps)).
- Big Ball of Mud
- Put yourself on the map

CQRS

CQRS challenge the assumption that reading and writing are supposed to share the same abstractions & models & databases & applications.

CQRS

CQRS challenge the assumption that reading and writing are supposed to share the same abstractions & models & databases & applications.

So in short : "Split Write Model and Read Models"

CQRS

CQRS challenge the assumption that reading and writing are supposed to share the same abstractions & models & databases & applications.

So in short : "Split Write Model and Read Models"

The presentation : https://speakerdeck.com/mathiasverraes/fighting-bottlenecks-with-cqrs

Event Sourcing

Using object's *history* to reconstitute its *state*. The history is expressed as a series of Domain events.

Event Sourcing

Using object's *history* to reconstitute its *state*. The history is expressed as a series of Domain events.

The aggregate records events and protect invariant but does not expose the state. Aggregate is the write model. State is exposed by projector (one of the read models).

Event Sourcing

Using object's *history* to reconstitute its *state*. The history is expressed as a series of Domain events.

The aggregate records events and protect invariant but does not expose the state. Aggregate is the write model. State is exposed by projector (one of the read models).

The presentation and some code at

https://speakerdeck.com/mathiasverraes/practical-event-sourcing

Event Sourcing

Do not put invariant in the apply method because you couldn’t restore the aggregate anymore if invariant change. It is then possible to have aggregate that do not comply with new invariants and it is a business decision to know what to do with these.

Event Sourcing

Projectors should be easy to write. Don’t use an ORM here since they will slow the process down.

Testing the write model is like:

- Given { previous events }
- When { command }
- Then { expected new events or expected exception }

Testing projector / read model is like:

- Given { previous events }
- Then { expected states }


THE
END