DYSFUNCTIONAL DDD

JAREK

Wizard, Anarchitect, Coder

JAREK

Wizard, Anarchitect, Coder

> 25 years of coding


I'VE SEEN COMMITS
YOU PEOPLE WOULDN'T BELIEVE
imgflip.com

DISCLAIMER

DISCLAIMER

- I tried not to attack anyone personally, actually I do like my colleagues doing DDD, and I have learned a lot from some of them

DISCLAIMER

- I tried not to attack anyone personally, actually I do like my colleagues doing DDD, and I have learned a lot from some of them
- not an expert, (some experience wih CQRS/ES not DDD)

WHY DDD?

- People talk about it
- Object oriented world
- Like mixing concepts

My problem with DDD

DDD story resembles me

- UML
- what happened to Agile

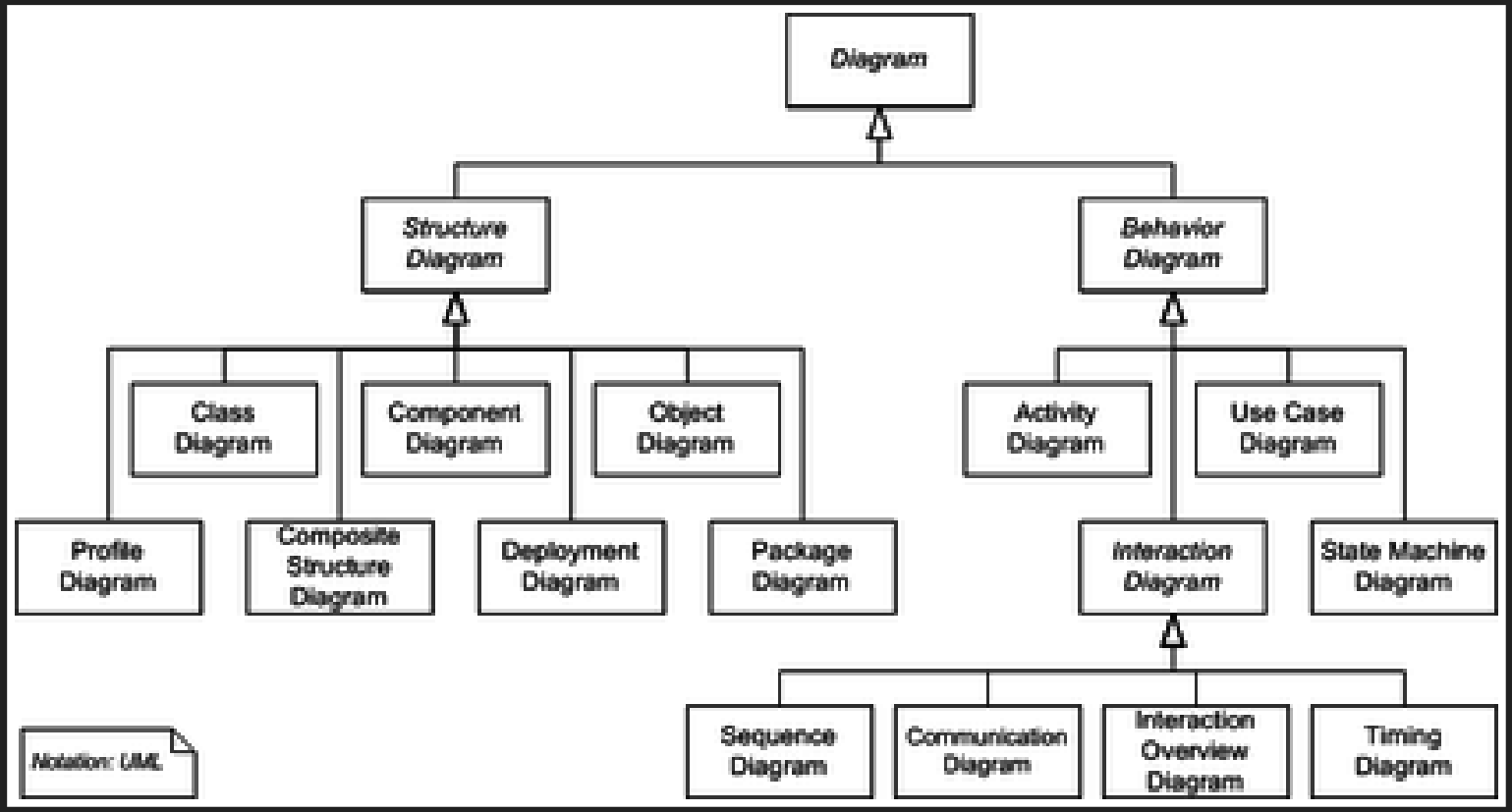


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Activity Diagram
Use Case Diagram
Profile Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Interaction Diagram
State Machine Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram
Notation: UML

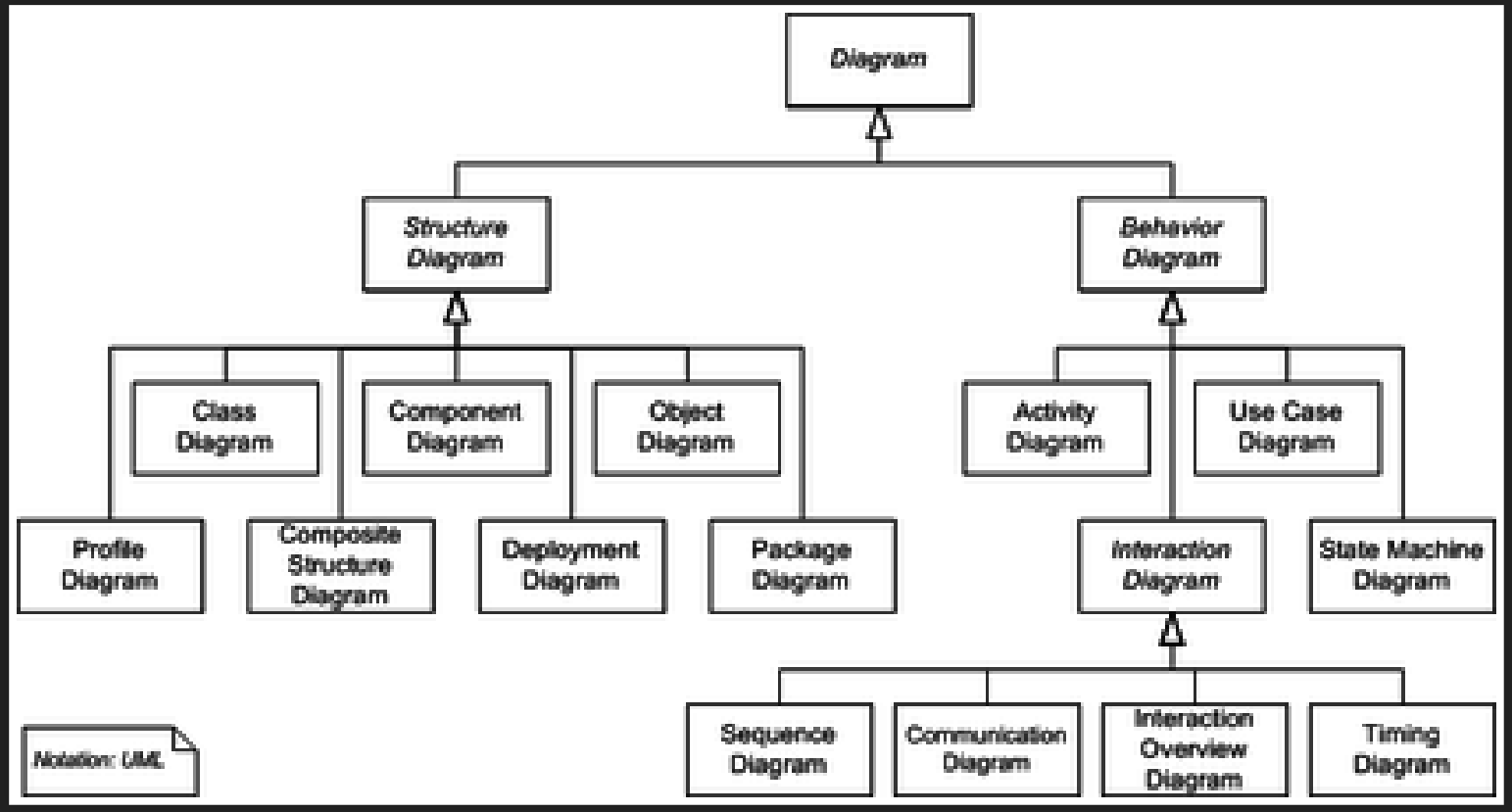
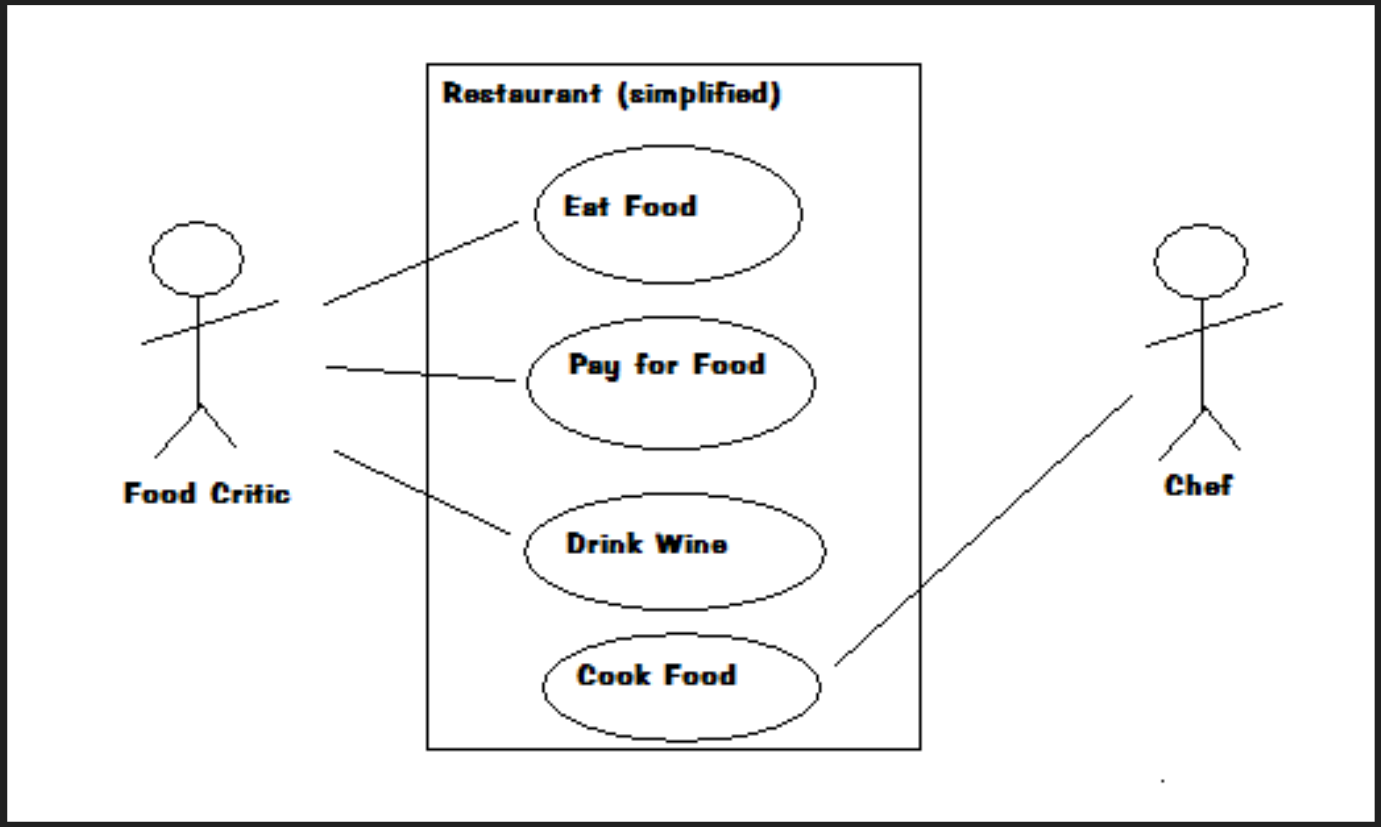
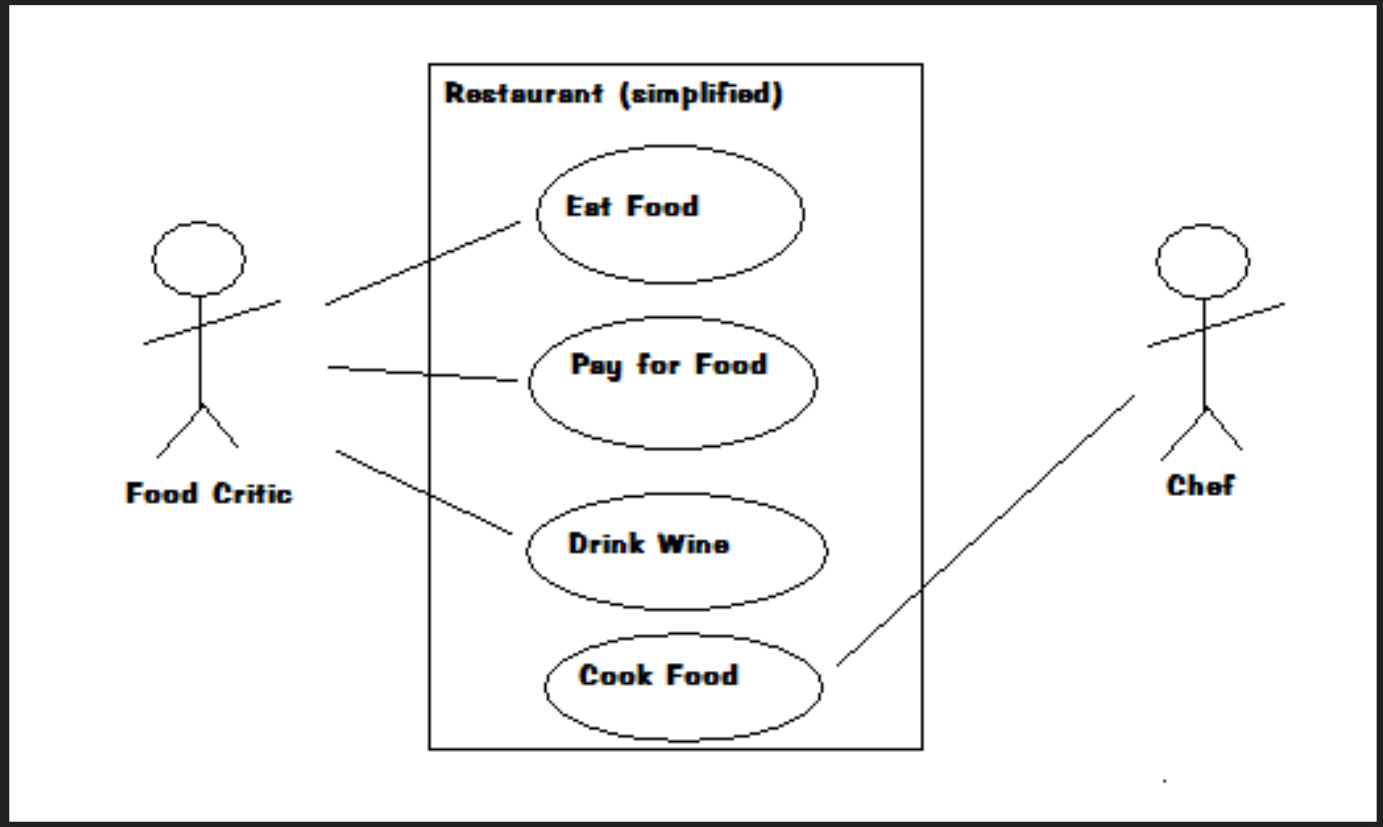


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Activity Diagram
Use Case Diagram
Profile Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Interaction Diagram
State Machine Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram
Notation: UML

cool

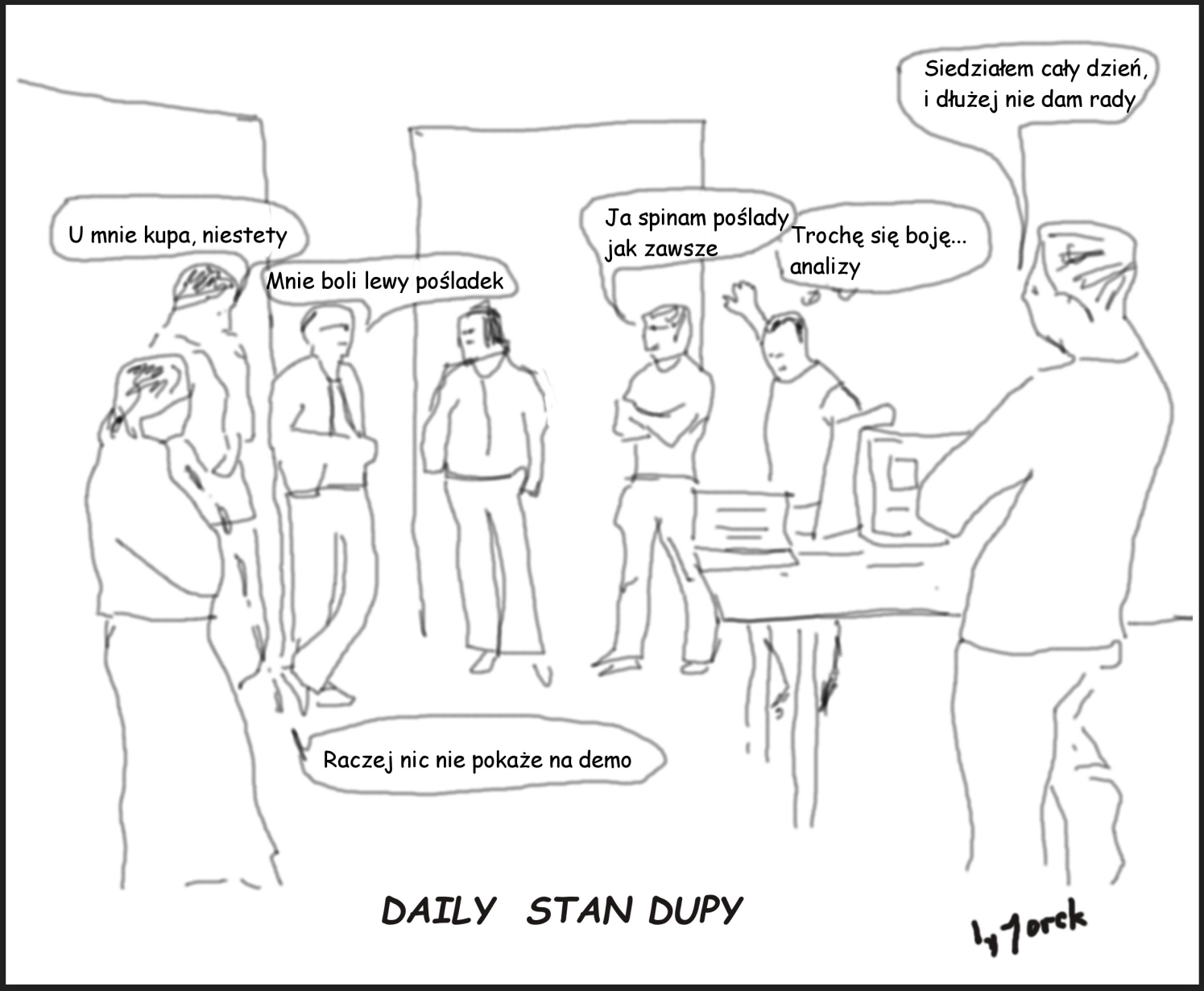

Restaurant (simplified)
Eat Food
Pay for Food
Drink Wine
Cook Food
Food Critic
Chef


Restaurant (simplified)
Eat Food
Pay for Food
Drink Wine
Cook Food
Food Critic
Chef

What??

AGILE COOL

- Tests, TDD
- CI,
- Code over comments
- contact with user


Siedziałem cały dzień,
i dłużej nie dam rady
U mnie kupa, niestety
Mnie boli lewy pośladek
Ja spinam poślady
jak zawsze
Trochę się boję...
analizy
Raczej nic nie pokaże na demo
DAILY STAN DUPY
Lytorck

AGILE REALITY

- boring meetings
- agile coaches,
- certificates
- agile tools
- books
- conferences
- trainings
- velocity :-)

There is nothing wrong about money.

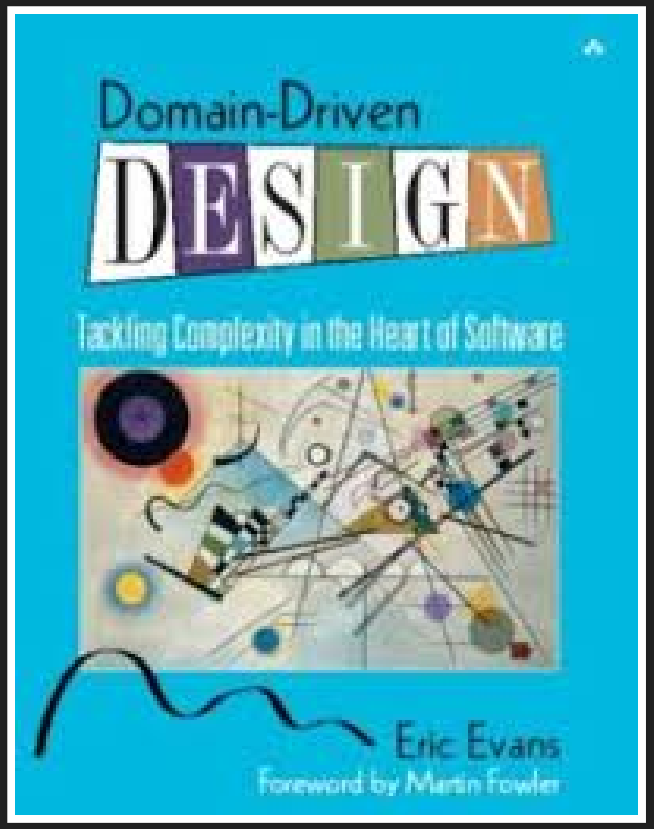
I work for money!

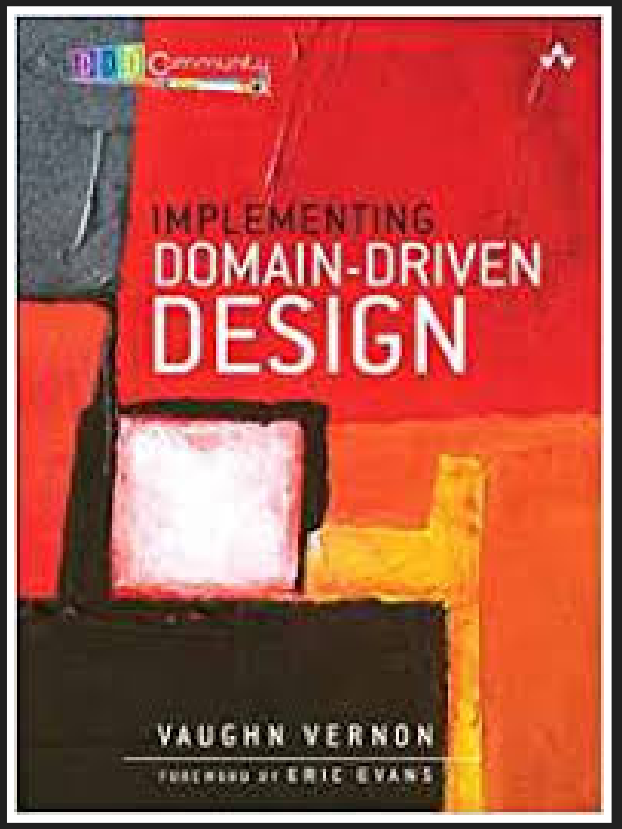
But who likes marketing b...it?!


BUSINESS
imgflip.com

For years I was scared of DDD marketing

BOOKS


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler


IMPLEMENTING
DOMAIN-DRIVEN
DESIGN
VAUGHN VERNON

Make logic of the system visible in code

DDD GOOD PARTS

- community
- constant improvement
- lots of patterns / ideas
- interesting stories

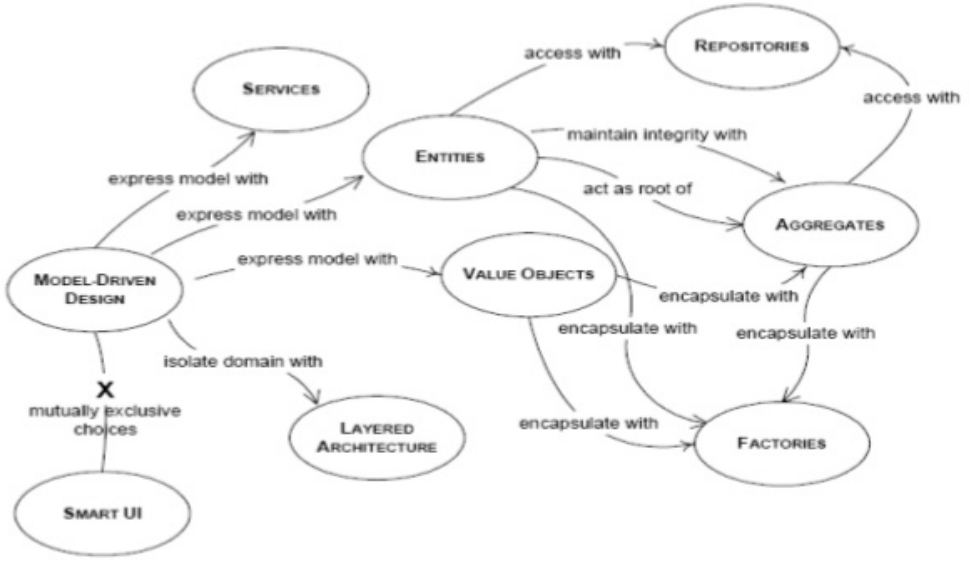
DDD bad(?) parts

- marketing
- example projects
- over-engineering
- partly toxic community
- hard to see the point

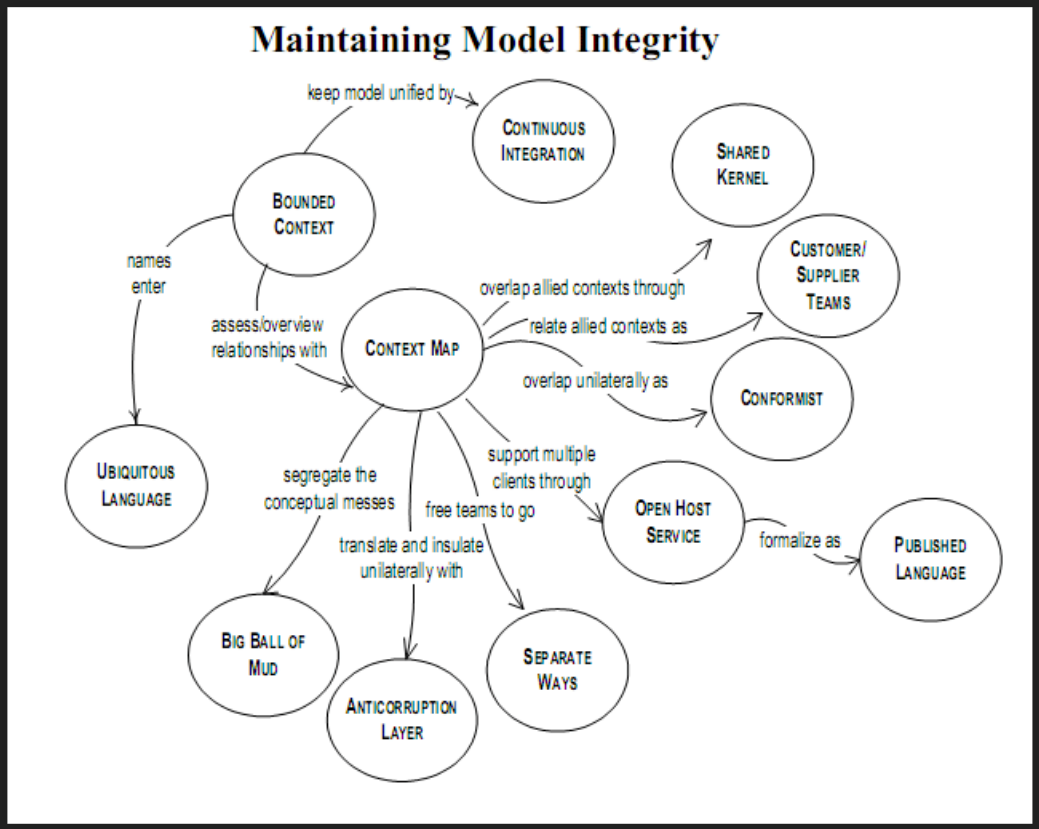
If a tool, methodology, framework makes simple case **complex**

it will probably not make a real life, complex project **simple**

Building blocks of DDD



Maintaining Model Integrity



BUT

DDD

HARD WAY

Skipping some essential parts.

Like `bounded context` etc.

DDD is mostly not about `technology`

FINDING COMMON LANGUAGE

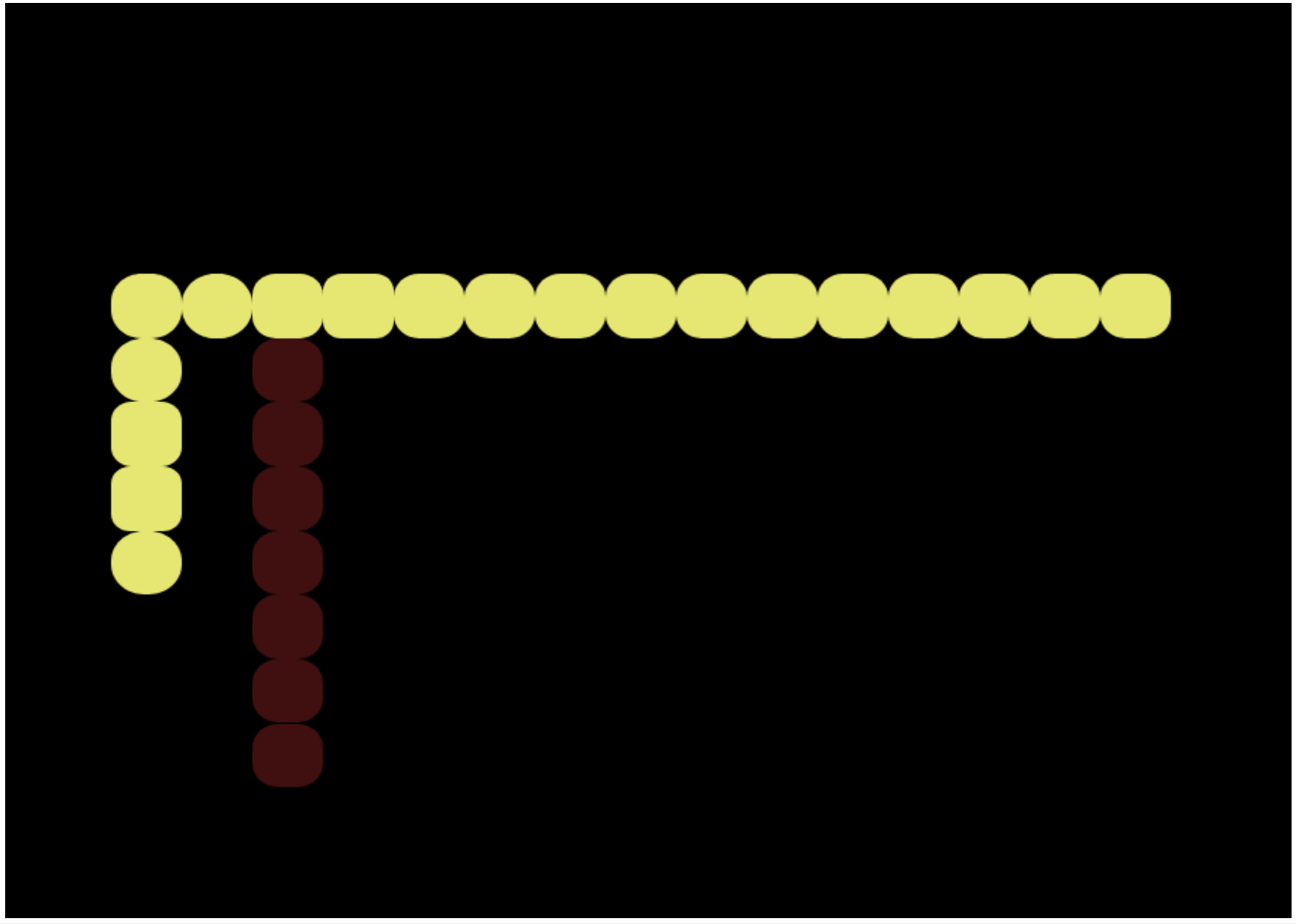
Ubiquitous Language

Some concepts are useful, but have nothing to do with FP

I just love to code more

And you wil not pay me areal money (for a fancy car) :-(

LETS PLAY SNAKE (MULTIPLAYER)



COMMANDS

```
data SnakeCommand
   = SetDirection { wantedDirection :: SnakeDirection }
   | MakeStep
   | Begin {
             initName :: String
           , initCell :: SnakeCell }
```

COMMANDS

- user or subsytem `wants` to do something
- commands may be associated with validation

EVENTS

```
data SnakeEvent
   = DirectionChanged { newDirection :: SnakeDirection }
   | StepMade
   | Killed
   | Born { bornName :: String
          , bornCell :: SnakeCell }
   deriving (Eq, Show, Generic)
```

EVENT

- it has happened
- no validation (it really happened)
- a single command is associated in 0..n events

Fact

You will not find anything about this in Eric Evans book (blue))

Even though nowadays DDD community seems to be all around those concepts

EVENT SOURCING

What if just stored only events?

```
[
  {
    "event": {
      "tag": "Born",
      "bornName": "aa",
      "bornCell": {
        "cellY": 18,
        "cellX": 14
      }
    },
    "snakeId": "4d09ac06-0375-4cb0-ad08-c70d14968677"
  },
  {
    "event": {
      "tag": "DirectionChanged",
```

COMMAND SOURCING?

Also possible... but in fact harder - validation is a problem

VALUE OBJECT

- immutable...
- represents value (from `real` life)
- has no identity
- properties define equality

```haskell
type SnekeId = String

type NickName  = Text

data SnakeDirection
   = SnakeUp
   | SnakeRight
   | SnakeDown
   | SnakeLeft
```

ENTITY

- has identity
- in OOP may be mutable
- ID defines equality
- entity may contain `value objects`
- entity may contain `entities`

```haskell
data SnakeCell = SnakeCell
   { cellX :: Int
   , cellY :: Int
   }

data SnakeState
   = Alive { direction :: SnakeDirection
           , cells     :: [SnakeCell]
           , maxLength :: Int }
   | Dead
   | Init

data Snake = Snake
   { name  :: String
```

Easy?

- `SnakeCell(x,y)` is value object or entity?

To think

I have never introduced `type SnakeEntity = (SnakeId, Snake)`

Only have SnakeState which does not (physically) contain Id

To think

I have never introduced `type SnakeEntity = (SnakeId, Snake)`

Only have SnakeState which does not (physically) contain Id

an Entity without id??

Some DDD concepts may not be explicitly existing in code

`(SnakeId, Snake)`

It gets worse

AGGREGATE

Cluster of objects (entities, value objects, +)

Aggregate remains consistent

Keeps invariants

Transactions should not cross aggregates

AGGREGATE ROOT

- Selected entity from Aggregate (root)
- outside world communicates with it (sends commands)
- outside world only keeps reference to this `root object`
- command handler
- event handler

```
class Aggregate s where
    data Error s :: *
    data Command s :: *
    data Event s :: *

    execute :: s -> Command s -> Either (Error s) (Event s)
    apply :: s -> Event s -> s
    seed :: s
```

Typeclass source:

https://gist.github.com/Fristi/7327904

COMMAND HANDLER

```
executeCommand :: Snake -> SnakeCommand -> [SnakeEvent]
```

BETTER COMMAND HANDLER

```
executeCommand :: Snake -> SnakeCommand
      -> Either MyError [SnakeEvent]
```

EVENT HANDLER

```
applyEvent::Snake-> SnakeEvent -> Snake
```

SUMMARY

- define commands
- define events
- select root Entity
- define commands handler
- define events handler

```
executeCommand :: SnakeData -> SnakeCommand -> [SnakeEvent]
executeCommand SnakeData {state = Alive {}} MakeStep = [StepMa
executeCommand _ MakeStep = []
executeCommand SnakeData {state = Alive {direction = od}} SetD
   | opposite = []
   | otherwise = [DirectionChanged {newDirection = nd}]
  where
    opposite = V.dirIs0 $ V.dirPlus newVec currentVec
    newVec = V.dirVector nd
    currentVec = V.dirVector od
executeCommand anySnake SetDirection {} = []
executeCommand SnakeData {state = Alive {}} Die = [Killed]
executeCommand _ Die = []
executeCommand SnakeData {state = Init} Begin {initName = d, i
executeCommand _ Begin {} = []
```

Real Command handler

```
applyEventX snake@(SnakeData {state = alive@Alive {}}) Directi
      makeRes $ snake { state = alive{direction = nd} }
applyEventX snake@(SnakeData {state = Alive {}}) Killed = make
applyEventX SnakeData {state = Init} Born {bornName = nm, born
            newSnake = SnakeData { name = nm , state = initial
               newCells = [cell], removedCells = []
      }
   where initialState = Alive { direction = SnakeUp, cells = [
applyEventX snake@(SnakeData {state = alive@Alive {maxLength =
         makeRes snake { state = alive { maxLength = n+3} }
applyEventX _ _ = error "todo"
```

Real event handler

Modelling with events, commands is not needed in DDD

It was not even considered in an original DDD book

behaviour first seem to be quite efficient

Event storming

Alternative to command handler / Aggregate

```
data SnakeCommand
   = SetDirection { wantedDirection :: SnakeDirection }
   | MakeStep
   | Begin {
             initName :: String
           , initCell :: SnakeCell }
   | Die
```

Commands

```
data SnakeCommand next
   = SetDirection { wantedDirection :: SnakeDirection }
   | MakeStep next
   | Begin {
             initName :: String
           , initCell :: SnakeCell } next
   | Die
```

Have You seen that before?

Free monad DSL

Seems to be more usable in sequencing

In typical REST we have one http call -> one command. Sequencing is not that needed.

HOW TO FIND AGGREGATES?

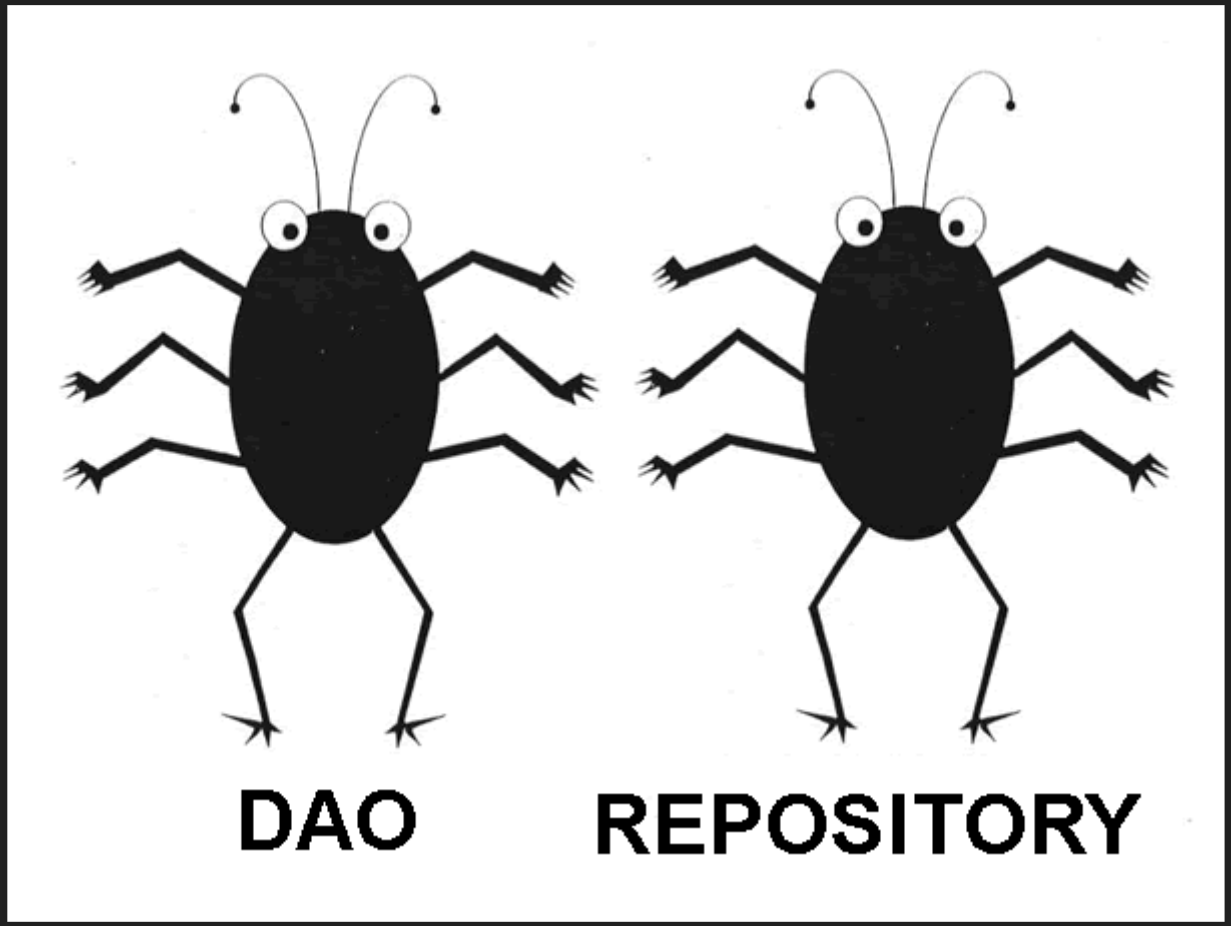
Whole system as an aggregate?

(One Big Aggregate)

REPOSITORY

Remember DAO?

Magic...


DAO
REPOSITORY

- loadEntity::Id->IO Entity
- saveEntity::Id->Entity->IO ()
- etc...

Fact Lots of magic Java frameworks trace state of objects and automatically persist changes to database

This means that a sensible repository `save` method may look like:

```
void save(MyObject t) {

}
```

```
void save(MyObject t) {

}
```

Yep, this works. There are lot of such projects.

But what in case of event sourcing?

```
data SnakeAggregate = SnakeAggregate {
      state :: SnakeState,
      uncommittedEvents :: [SnakeEvent]
}
```

Common pattern in DDD style event sourcing is to save those `uncommited events`

I find it unnatural

I started to send commands to a *Repository*

```
applyCommand :: SnakesRepo-> SnakeId -> Snake.SnakeCommand
   -> IO SnakesRepo
```

```
applyCommand :: SnakesRepo-> SnakeId -> Snake.SnakeCommand
   -> IO SnakesRepo
```

This is so unDDD

Makes for more sense than repeating a code with save events

In my aggregates I do not have those uncommited events (is this a domain?)

CQS

Command Query separation

If you ask (Query) do not change the state

If You change state (Command) do not expect result

A Stack

```
void stack.push( T  t);
T  stack.pop();
```

A CQS Stack

```
void stack.push( T  t);
T stack.top();
void stack.pop();
```

Simple?

what if called on empty stack?

```
void stack.pop();  // boom
```

OO world consensus:

commands may return exceptions, some status, etc.

Is an `error` not a result?

In FP world

```
push::Stack a->a->Stack a
top::Stack a->a
pop::Stack a->Stack a
```

In FP world each operation gives a result

IO ()

```
whatAPop::Stack a->(Stack a, a)
```

Is it really bad?

Actually I do not see much sense in classical CQS

- nice to have separated queries
- Error/Exception **is** an result
- It only makes API easier to use ... in some mediocre languages
- async?

```
push::Stack a -> MonadAsync (Stack a)
```

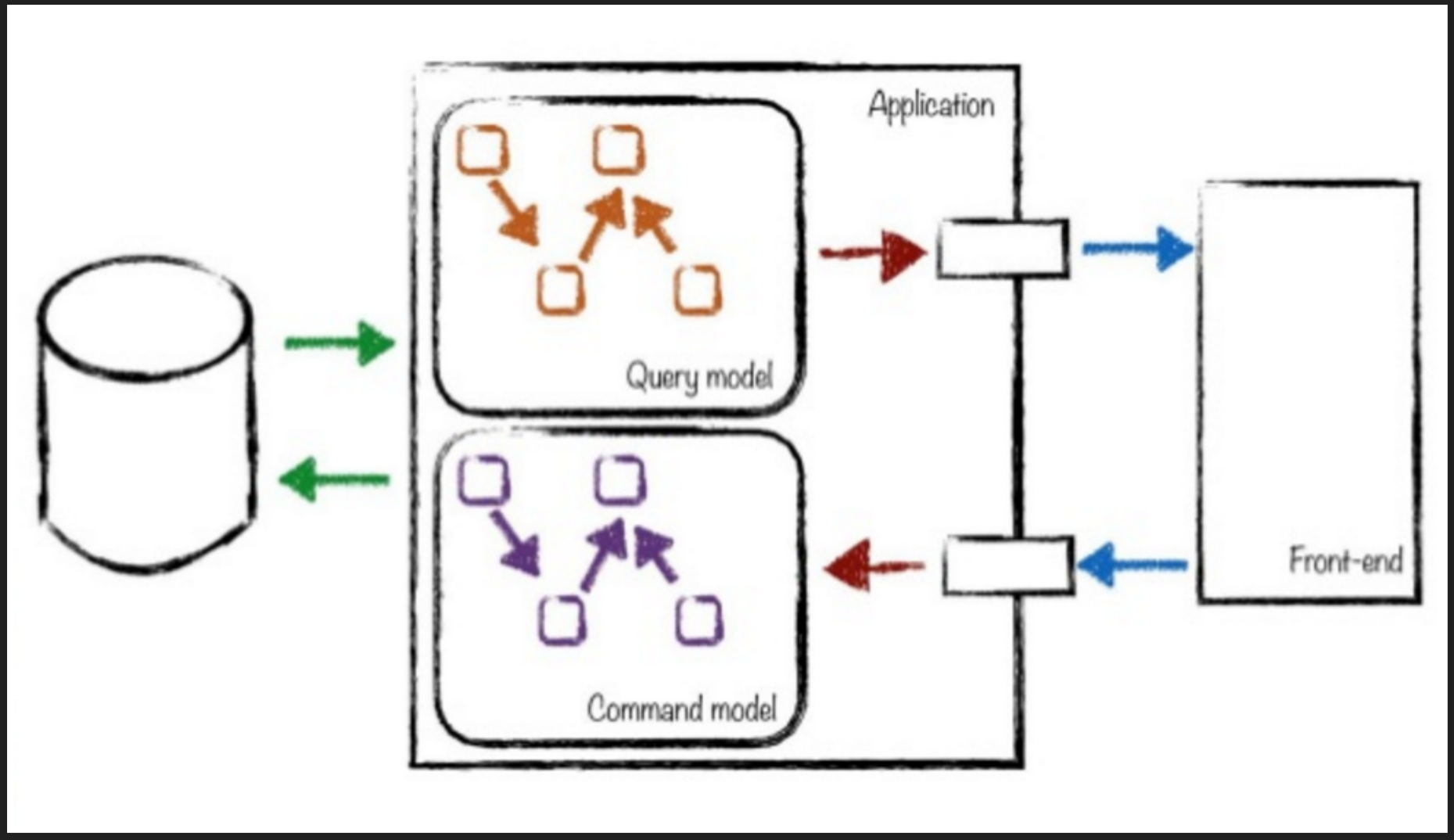
CQRS

Command Query Response Segregation

CQRS ~ CQS on a higher level

Write/Command model - Aggregates

Read/Query model - Projections


Application
Query model
Command model
Front-end

Some CQRS principles

- Critical: Events application cannot use any external data (projection)
- Softer: Commands should not use projection to produce Events
- You can always recreate Aggregates using events
- You can always recreate multiple projections using events

Fact

In `less impure` languages it is easy to make a mistake

```
LocalDate.now()
someRandom.nextInt()
```

Fact

In typical Event souring operations like `findMeIdsOfAllAggregates` are performed using projections

```haskell
data PlaneState = PlaneState
   { allSnakes :: Repo.SnakesMap
   , allCells  :: CellsMap
   , changes :: Changes
   } deriving (Show, Generic)

-- projection
applyEvent :: PlaneState -> Repo.SnakeQualifiedEvent -> (Plane
```

My game field projection

Used by browser

Projections are great:

- potential performance
- make UI code simpler

I use them to detect collisions

If I find collision on a gameField I send `Die command` to snake

I read few times this is wrong...

But I do not see better solution (other than One Big Aggregate)

In CQRS some operations are quite hard

```
nextId::Sequence->Int
```

CODE

https://github.com/jarekratajski/dysfunctional_ddd

dsnake - haskell rest server, (yesod)

Work in progress

Please, do not use it as a sensible DDD example resource (yet)

RESOURCES ON DDD/CQRS

Bottega presentations

"The" books

http://CQRS.nu (FAQ) page

https://github.com/ddd-by-examples/event-source-cqrs-sample by Kuba Nabrdalik

https://github.com/vlingo vlingo platform

Q?