O'REILLY®

Domain-Driven Design for Monoliths

September 2020

Introduction

What are you about to learn?

About You

- New to Domain-Driven Design
- Working on a monolithic codebase
  - Greenfield project
  - Brownfield project
- You care

Learning Objectives

- Analyze a company's business goals and strategy
- Get a business-oriented point of view on software projects
- Align software design with business strategy
- Work effectively with monolithic codebases

About Me

- Over 20 years of industry experience
- Practicing Domain-Driven Design for over 10 years
- Conference speaker, trainer, and blogger
- Author of “What is Domain-Driven Design?”
- Co-author of “DDD: The First 15 Years”

vladikk

O'REILLY®

Domain-Driven Design for Monoliths

September 2020

Prologue: Monolith

What is a Monolith?

Big Ball of Mud

"A Big Ball of Mud is a **haphazardly structured**, **sprawling**, **sloppy**, **duct-tape-and-baling-wire**, **spaghetti-code jungle**. These systems show unmistakable signs of **unregulated growth**, and repeated, **expedient repair**. **Information is shared promiscuously among distant elements of the system**, often to the point where nearly all the important information becomes global or duplicated.” - Joseph Yoder

- A monolith can be a big ball of mud. Just as
- Microservices can be a distributed big ball of mud!
- **A monolith can be modular!**

Business Domain Analysis

Understanding a software project’s business context

What is a Business Domain?

- A company’s overall area of activity
- The service a company is providing to its customers
- A company can operate in multiple business domains
- A company’s business domain can change over time

Business Domain: Example

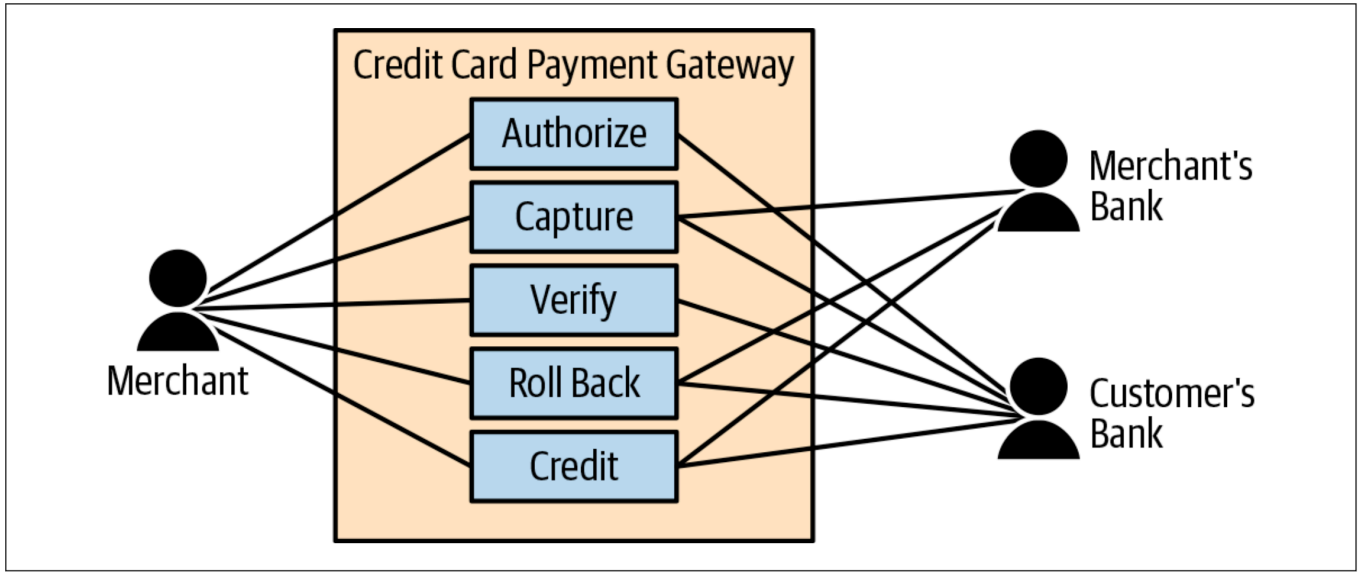
- Internovus
- Marketing services for companies producing products / services
  - Marketing strategy planning
  - Ad creatives – banners, landing pages, etc.
  - Ad campaign publishing and optimization
  - Sales calls to prospective leads

What is a Business Subdomain?

- A fine-grained area of business activity
- Activity required to succeed in the business domain
- Often correlates with departments and organization unit
- Business “building block”

What is a Business Subdomain?

A set of interrelated use-cases



Business Subdomains: Examples

Subdomains in Internovus:

- Creative catalogue
- Campaign management
- Campaign optimization
- Ad serving
- Ad spaces acquisition
- Accounting
- Identity and Access
- Customer relationship management
- Sales commissions accounting
- Telephony

Types of Business Subdomains

- Core
- Supporting
- Generic

Core Subdomain

- The company's competitive advantage
  - What differentiates the company from its competitors
  - What it does differently
  - Business differentiator
- Inventions / Optimizations / Know-How / Intellectual Properties
- How the company is making money

Business Subdomains: Examples

Core Subdomains in Internovus:

- Creative catalogue
- **Campaign management**
- **Campaign optimization**
- Ad serving
- Ad spaces acquisition
- Telephony
- Accounting
- Identity and Access
- **Customer relationship management**
- **Sales commissions accounting**

Generic Subdomain

- Things all companies are doing in the same way
- Provide no competitive advantage
  - Competitors are using the same solution
  - No business differentiation
- No innovation, optimization, or intellectual property

Business Subdomains: Examples

Generic Subdomains in Internovus:

- Creative catalogue
- Campaign management
- Campaign optimization
- Ad serving
- Ad spaces acquisition
- **Accounting**
- **Identity and Access**
- Customer relationship management
- Sales commissions accounting
- **Telephony**

Supporting Subdomain

- Supports implementation of Core Subdomains
- The company is doing differently from its competitors
- Provide no competitive advantage
- No innovation, optimization, or intellectual property
- Not a strategic issue if a competitor copies the solution

Business Subdomains: Examples

Supporting Subdomains in Internovus:

- **Creative catalogue**
- Campaign management
- Campaign optimization
- **Ad serving**
- **Ad spaces acquisition**
- Accounting
- Identity and Access
- Customer relationship management
- Sales commissions accounting
- Telephony

Types of Business Subdomains

- Core
- Supporting
- Generic

Types of Business Subdomains

Which subdomain

- Provides no competitive advantage
- Conducted in the same way as by competitors

Types of Business Subdomains

Which subdomain

- Provides no competitive advantage
- Conducted in the same way as by competitors

Answer: Generic Subdomain

Types of Business Subdomains

Which subdomain

- Provides no competitive advantage
- Differentiates the company from its competitors

Types of Business Subdomains

Which subdomain

- Provides no competitive advantage
- Differentiates the company from its competitors

Answer: Supporting Subdomain

Types of Business Subdomains

Which subdomain

- Provides competitive advantage
- Differentiates the company from its competitors

Types of Business Subdomains

Which subdomain

- Provides competitive advantage
- Differentiates the company from its competitors

Answer: Core Subdomain

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'
Core	Yes	Yes
Supporting	No	Yes
Generic	No	No

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type
Core	Yes	Yes	Interesting
Supporting	No	Yes	Boring
Generic	No	No	Solved

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity
Core	Yes	Yes	Interesting	
Supporting	No	Yes	Boring	
Generic	No	No	Solved	

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity
Core	Yes	Yes	Interesting	High
Supporting	No	Yes	Boring	
Generic	No	No	Solved	

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity
Core	Yes	Yes	Interesting	High
Supporting	No	Yes	Boring	
Generic	No	No	Solved	High

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity
Core	Yes	Yes	Interesting	High
Supporting	No	Yes	Boring	Low
Generic	No	No	Solved	High

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility
Core	Yes	Yes	Interesting	High	
Supporting	No	Yes	Boring	Low	
Generic	No	No	Solved	High	

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility
Core	Yes	Yes	Interesting	High	High
Supporting	No	Yes	Boring	Low	
Generic	No	No	Solved	High	

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility
Core	Yes	Yes	Interesting	High	High
Supporting	No	Yes	Boring	Low	Low
Generic	No	No	Solved	High	

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility
Core	Yes	Yes	Interesting	High	High
Supporting	No	Yes	Boring	Low	Low
Generic	No	No	Solved	High	Low

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility	Uncertainty
Core	Yes	Yes	Interesting	High	High	
Supporting	No	Yes	Boring	Low	Low	
Generic	No	No	Solved	High	Low	

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility	Uncertainty
Core	Yes	Yes	Interesting	High	High	High
Supporting	No	Yes	Boring	Low	Low	
Generic	No	No	Solved	High	Low	

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility	Uncertainty
Core	Yes	Yes	Interesting	High	High	High
Supporting	No	Yes	Boring	Low	Low	Low
Generic	No	No	Solved	High	Low	

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility	Uncertainty
Core	Yes	Yes	Interesting	High	High	High
Supporting	No	Yes	Boring	Low	Low	Low
Generic	No	No	Solved	High	Low	Low

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility	Uncertainty	Solution Strategy
Core	Yes	Yes	Interesting	High	High	High	
Supporting	No	Yes	Boring	Low	Low	Low	
Generic	No	No	Solved	High	Low	Low	

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility	Uncertainty	Solution Strategy
Core	Yes	Yes	Interesting	High	High	High	
Supporting	No	Yes	Boring	Low	Low	Low	
Generic	No	No	Solved	High	Low	Low	Buy / Adopt

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility	Uncertainty	Solution Strategy
Core	Yes	Yes	Interesting	High	High	High	Build inhouse
Supporting	No	Yes	Boring	Low	Low	Low	
Generic	No	No	Solved	High	Low	Low	Buy / Adopt

Types of Business Subdomains

	Provides Competitive Advantage	Different Solution Than Competitors'	Problem Type	Complexity	Volatility	Uncertainty	Solution Strategy
Core	Yes	Yes	Interesting	High	High	High	Build inhouse
Supporting	No	Yes	Boring	Low	Low	Low	Outsource
Generic	No	No	Solved	High	Low	Low	Buy / Adopt

Exercise

Identify core, generic, and supporting subdomains of Gigmaster:

- Gigmaster is a ticket sales and distribution company. Its mobile app analyzes users' music libraries, streaming services' accounts, and social media profiles to identify nearby shows that are relevant to its users.
- Gigmaster's users are conscious about their privacy. Hence, all users' personal information has to be anonimized, or at least encrypted.
- To improve the app's recommendation, a new module was implemented to allow manual entry of past gigs a user has attended, even if the tickets weren't purchased through Gigmaster.

Note: Feel free to make assumptions as long as you are specifying them explicitly

Exercise

Identify core, generic, and supporting subdomains of Gigmaster:

- Gigmaster is a ticket sales and distribution company. Its mobile app analyzes users' music libraries, streaming services' accounts, and social media profiles to identify nearby shows that are relevant to its users.
- Gigmaster's users are conscious about their privacy. Hence, all users' personal information has to be anonimized, or at least encrypted.
- To improve the app's recommendation, a new module was implemented to allow manual entry of past gigs a user has attended, even if the tickets weren't purchased through Gigmaster.

Core:

- Recommendation engine
- Data anonymization

Generic:

- Data encryption
- Identity and access
- Clearing and accounting

Supporting:

- Integration with music streaming services and social networks
- Past gigs tracking module

Key Takeaways

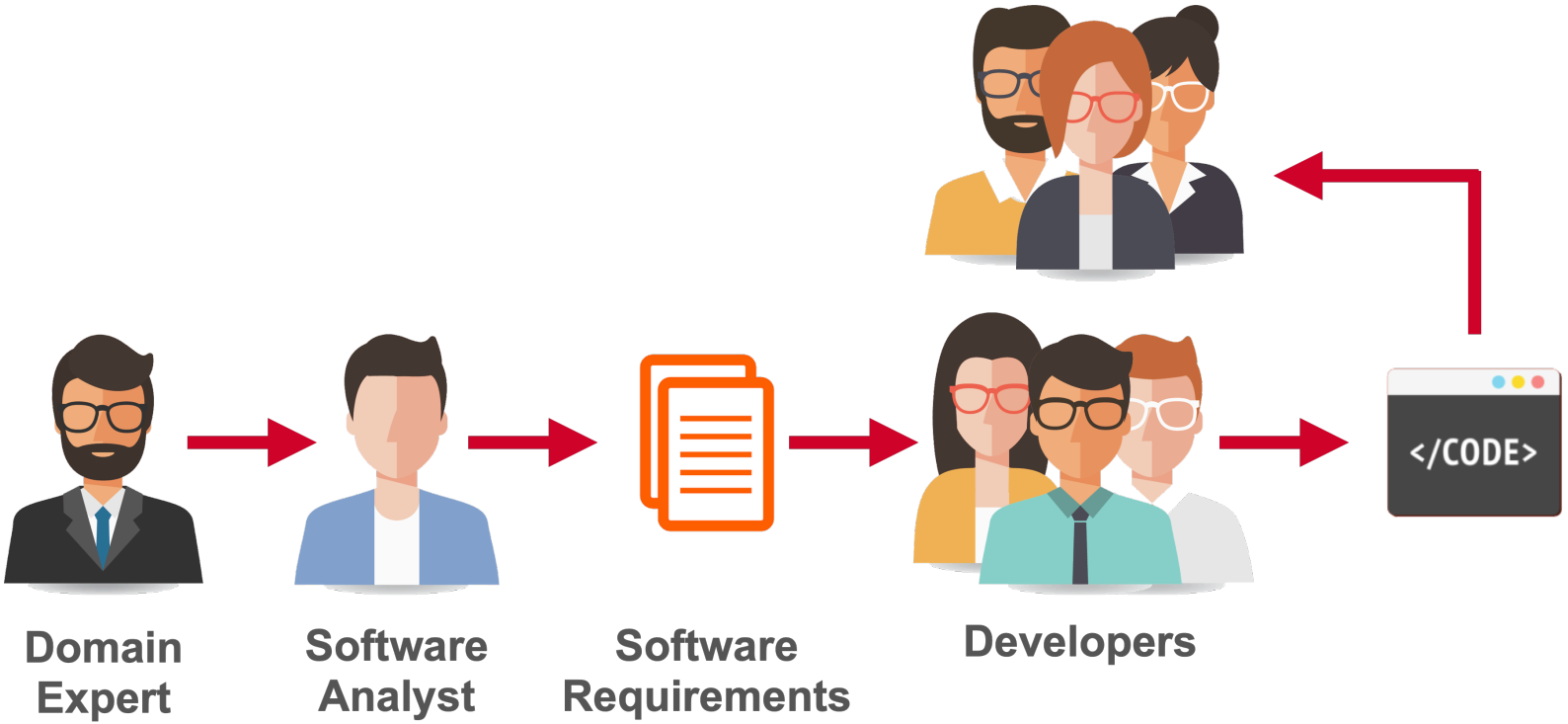
- Monolith ≠ Big Ball of Mud
- Business domain is a company's overall area of activity
- Subdomain – fine-grained area of business activity
- Core Subdomain – a company's competitive advantage
- Supporting Subdomain – an activity that provides no competitive advantage, but required for implementation of a core subdomain
- Generic Subdomain – problem domain with existing proven solutions

Questions?

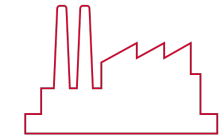
Acquiring and Modeling Business Domain Knowledge

Business Subdomains => Software

- Entities
- Rules
- Invariants
- Processes


Domain Expert
Software Analyst
Software Requirements
Developers
</CODE>

vladikk



DOMAIN KNOWLEDGE

MENTAL MODEL

SOLUTION MODEL

101010010
010101101
110101110

CODE

DISCOVERY

DESIGN

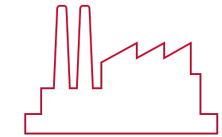
IMPLEMENTATION

vladikk

”It's developers’ **(mis)understanding**, not domain experts’ **knowledge** that gets released in production”

~ Alberto Brandolini

vladikk



DOMAIN KNOWLEDGE

MENTAL MODEL

SOLUTION MODEL

101010010
010101101
110101110

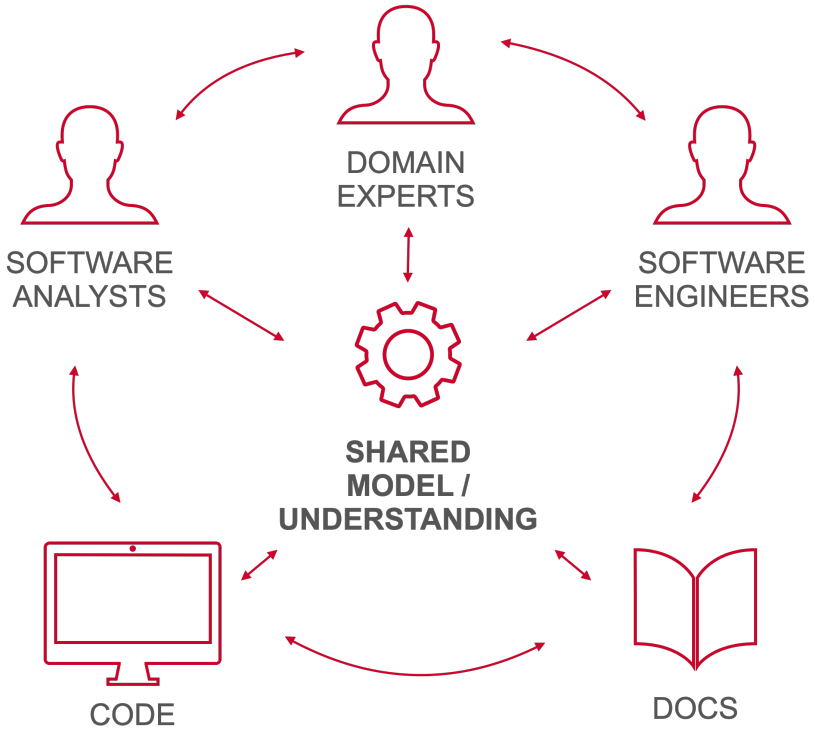
CODE

DISCOVERY

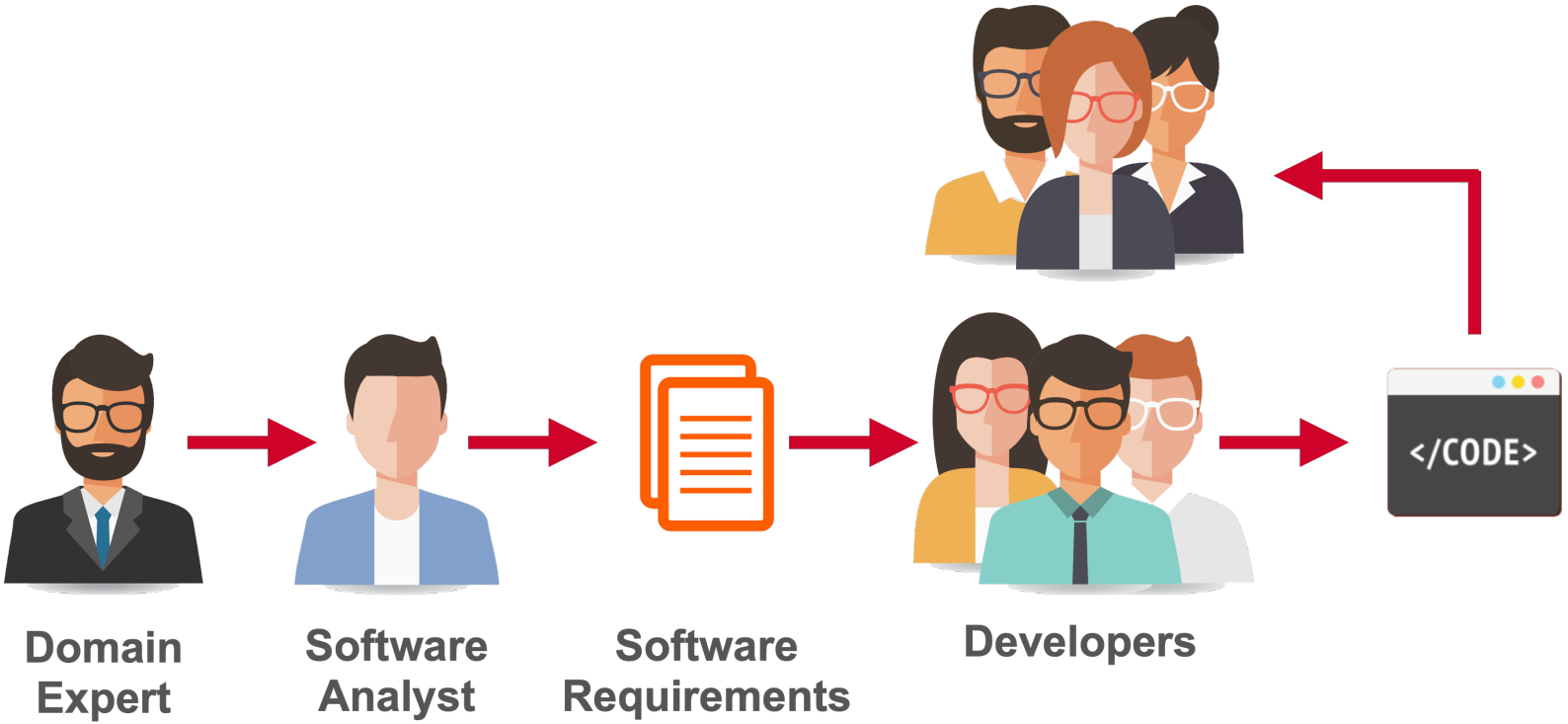
DESIGN

IMPLEMENTATION

vladikk


DOMAIN EXPERTS
SOFTWARE ANALYSTS
SOFTWARE ENGINEERS
SHARED MODEL / UNDERSTANDING
CODE
DOCS

Ubiquitous Language


Domain Expert
Software Analyst
Software Requirements
Developers
</CODE>

vladikk

Ubiquitous Language

A shared language for all project related communication

vladikk

“An advertising campaign is displayed on multiple placements”

“Placement can only be published if it’s associated with a banner”

“Creative cannot be deleted if it is already published”

“A lead is a landing page visitor interested in a product”

“A banner displays a creative”

“A banner displays a creative”

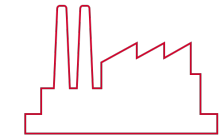
~~*“A banner displays an image”*~~

~~*“A banner displays a creative file”*~~

~~*“A banner displays a content”*~~

Ubiquitous Language

- Each term has one and only one meaning
- Eliminates ambiguity and assumptions
- Language is a model of a business domain
- Should be used in conversations, documentation, tests, code



DOMAIN KNOWLEDGE

MENTAL MODEL

SOLUTION MODEL

101010010
010101101
110101110

CODE

DISCOVERY

DESIGN

IMPLEMENTATION

vladikk

Ubiquitous Language: Exercise

- Come up with 5 business concepts from your domain
- Did you encounter “model translations”: business concepts that were denoted by different names?
- Can you spot business terms that mean different things for different stakeholders in your business domain?

What if…

- What if you cannot meet the ubiquitous language's requirements?
- What if you do have terms that mean different things to different stakeholders?
- What if some domain experts do use different terms to describe the same business concepts?

How can we merge ubiquitous language conflicts?

How can we merge ubiquitous language conflicts? **We don’t!**

Bounded Contexts

Divide-and-conquer business domain knowledge

Bounded Context

- Don’t mix/merge conflicting models
- Follow the domain experts’ mental models
- Split the ubiquitous language
- Define an explicit applicability context for each language / model – it’s bounded context
- Bounded context: boundary of a model

Marketing Bounded Context

Creative

Banner

Lead

Campaign

Payment

Placement

Sales Bounded Context

Payment

Sale

Campaign

Lead

Desk

Deposit

vladikk

Marketing Bounded Context

Creative

Banner

Lead

Campaign

Payment

Placement

Sales Bounded Context

Payment

Sale

Campaign

Lead

Desk

Deposit

Boundary of:

Ubiquitous Language

Physical System

Model

vladikk

Bounded Context

Enables Ubiquitous Language

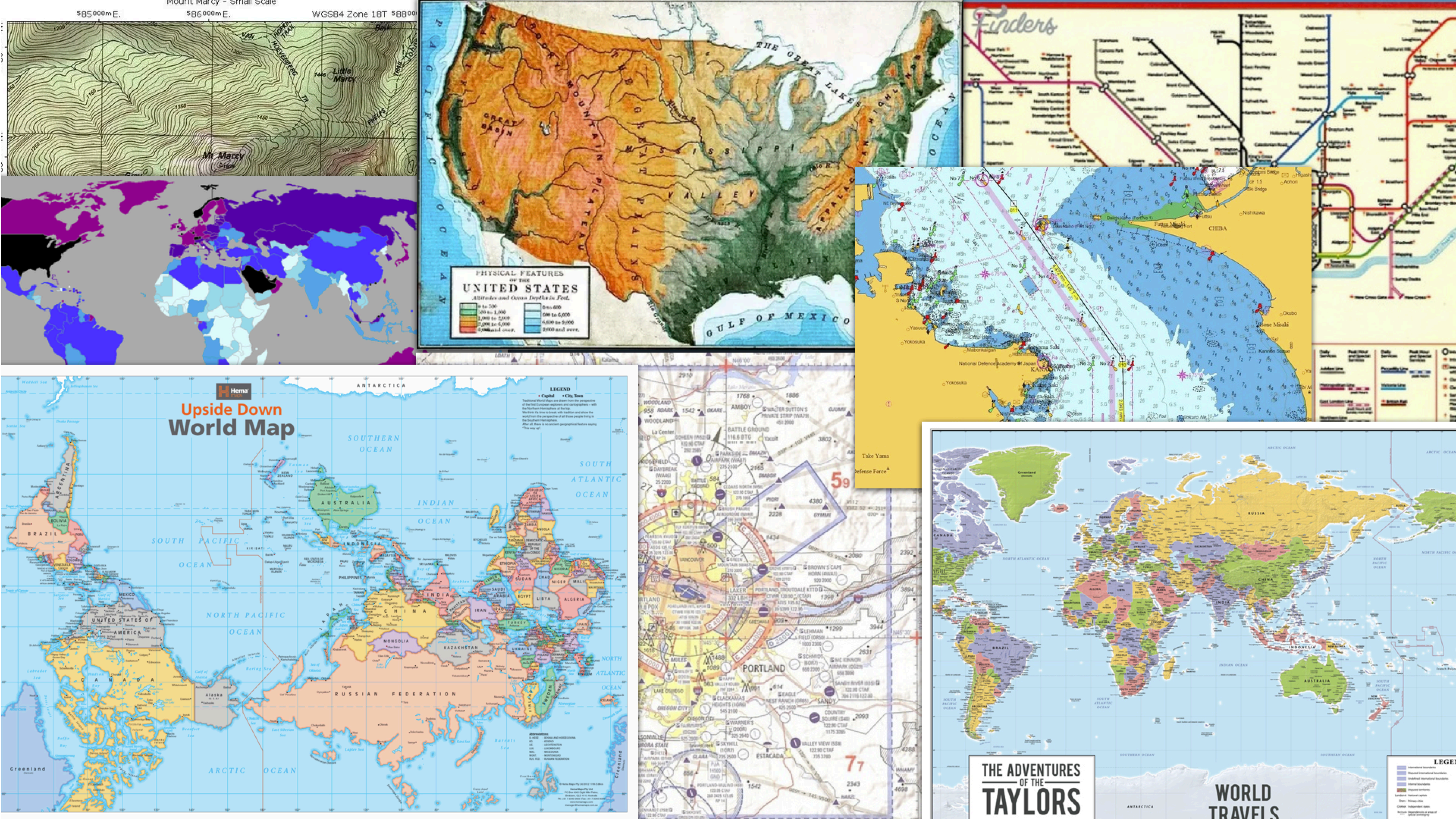
Language is a Model

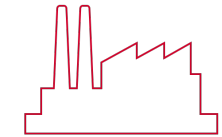
Bounded Context

Enables Modeling

"A model is **not the real world** but merely a human construct to **help us better understand** real world systems."

https://serc.carleton.edu/introgeo/models/WhatIsAModel.html


Mount Marcy - Small Scale
585000m E.
586000m E.
WGS84 Zone 18T
Little Marcy
Mt Marcy
PHYSICAL FEATURES OF THE UNITED STATES
Altitudes and Ocean Depths in Feet.
THE GREAT LAKES
MISSISSIPPI VALLEY
GULF OF MEXICO
PACIFIC OCEAN
CHIBA
Upside Down World Map
SOUTHERN OCEAN
SOUTH ATLANTIC OCEAN
INDIAN OCEAN
SOUTH PACIFIC OCEAN
NORTH PACIFIC OCEAN
ARCTIC OCEAN
AUSTRALIA
CHINA
INDIA
RUSSIAN FEDERATION
UNITED STATES OF AMERICA
CANADA
BRAZIL
ANTARCTICA
PORTLAND
VANCOUVER
THE ADVENTURES OF THE TAYLORS
WORLD TRAVELS



DOMAIN KNOWLEDGE

MENTAL MODEL

SOLUTION MODEL

101010010
010101101
110101110

CODE

DISCOVERY

DESIGN

IMPLEMENTATION

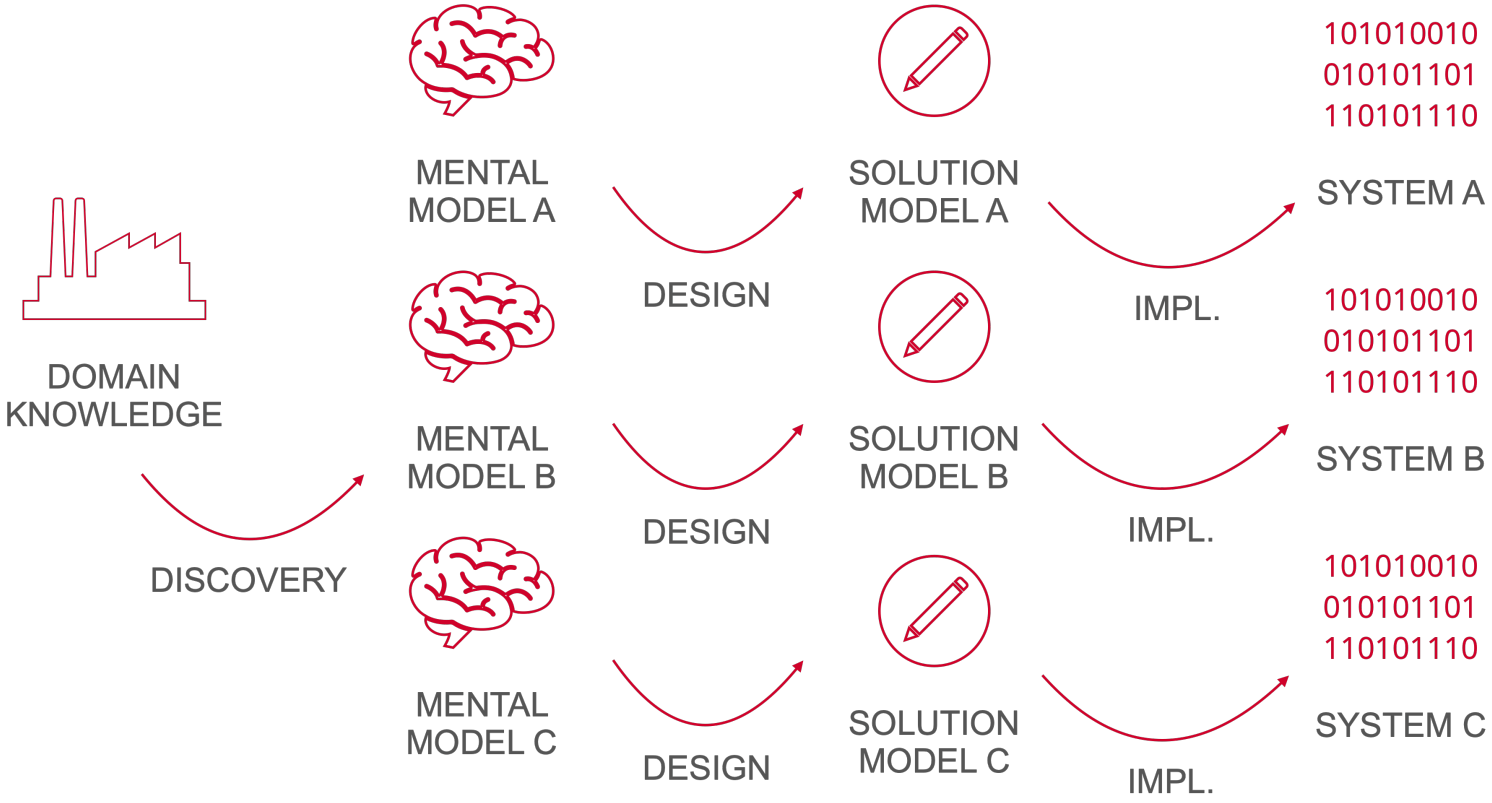
vladikk

DOMAIN
KNOWLEDGE

IMPLEMENTATION

101010010
010101101
110101110

CODE


DOMAIN
KNOWLEDGE
DISCOVERY
MENTAL
MODEL A
MENTAL
MODEL B
MENTAL
MODEL C
DESIGN
DESIGN
DESIGN
SOLUTION
MODEL A
SOLUTION
MODEL B
SOLUTION
MODEL C
IMPL.
IMPL.
IMPL.
101010010
010101101
110101110
SYSTEM A
101010010
010101101
110101110
SYSTEM B
101010010
010101101
110101110
SYSTEM C

vladikk

Discussion

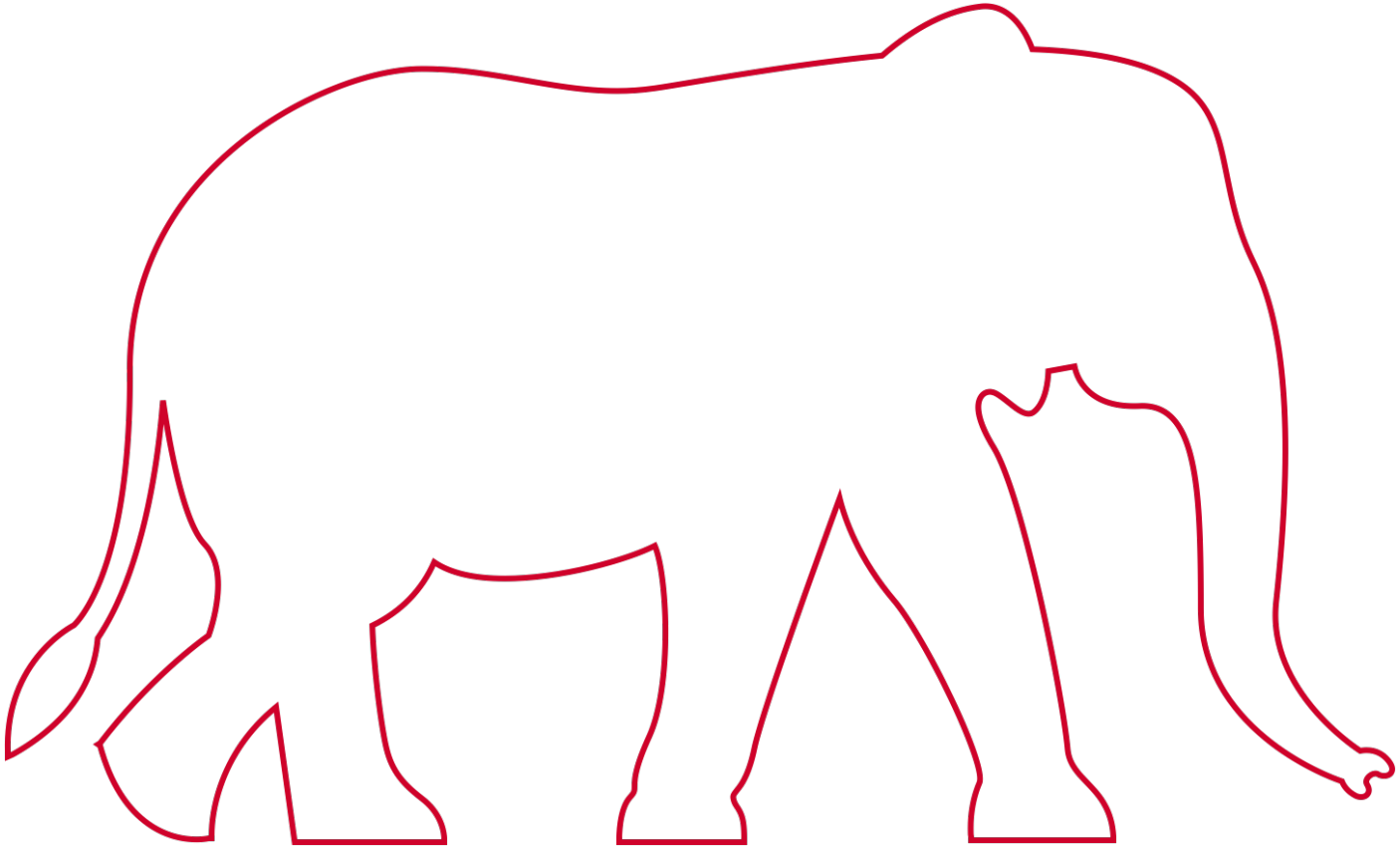
- A subdomain is a boundary
- A bounded context is a boundary as well
- Why do we need both boundaries?

Bounded Contexts vs. Subdomains

- Subdomains belong to the problem space
- Bounded contexts belong to the solution space

Bounded Contexts vs. Subdomains

- Subdomains belong to the problem space
- Bounded contexts belong to the solution space
- **Subdomains are discovered**
- **Bounded contexts are designed**

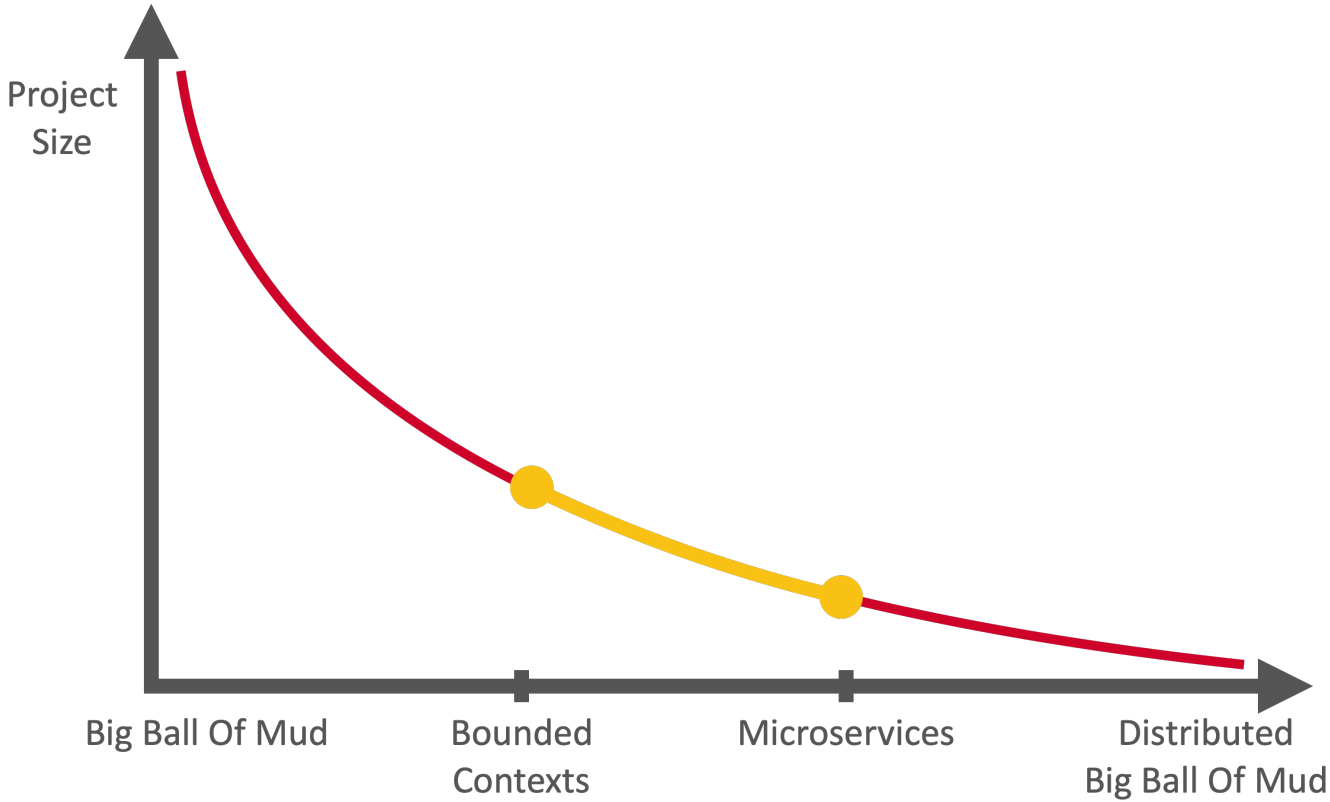


Discussion

- Are bounded contexts microservices?
- Why, or why not?

Bounded Contexts vs. Microservices

- Bounded context: boundary of the largest valid monolith
- Microservice: the smallest service boundary
- Each microservice is a bounded context
- Not each bounded context is a microservice


Project
Size
Big Ball Of Mud
Bounded
Contexts
Microservices
Distributed
Big Ball Of Mud

Bounded Contexts

Designing modular monoliths

Marketing Bounded Context

Creative Catalogue

Campaign Management

Ad Serving

Campaign Optimization

Ad Spaces Acquisition

Accounting

Sales Bounded Context

Customer Relationship Management

Telephony

Sales Commissions Accounting

Logical Boundaries are Crucial!

Logical Boundaries

- Namespaces, modules, etc.
- Must have for achieving modularity
- Okay to be wrong!
- Easy to refactor
- Easy to step over
- Easy to incur technical debt and turn into a BBoM

Business Logic

Implementation Patterns

Business Logic Implementation Patterns

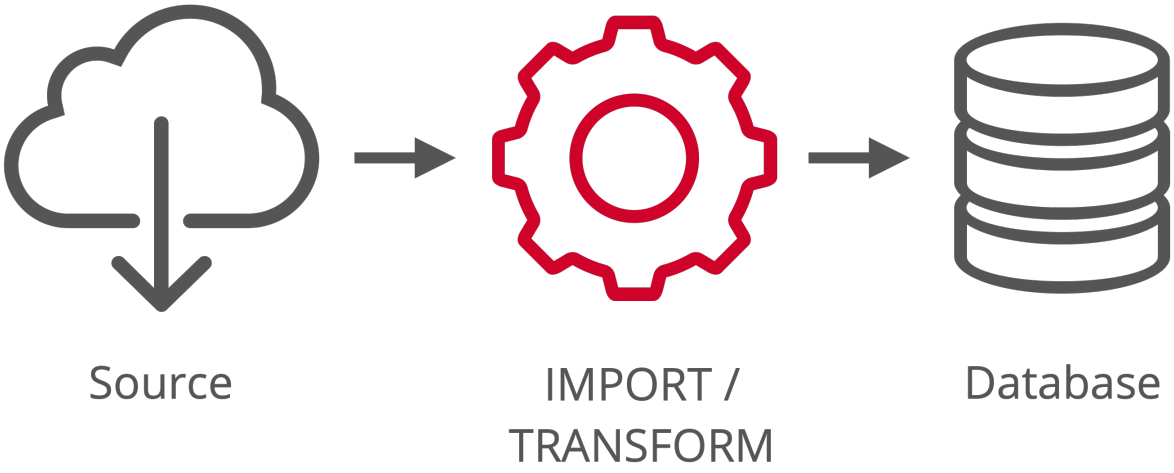
- Transaction Script
- Active Record
- Domain Model
- Event Sourced Domain Model

Transaction Script

Transaction Script

- Simple logic, simple data structures
- Procedural code
- Ensure each operation is transactional - either succeeds or fails
- Data is never corrupted

Transaction Script



Transaction Script

```
DB.StartTransaction();

var job = DB.LoadNextJob();
var json = LoadFile(source);
var xml = ConvertJsonToXml(json);
WriteFile(destionation, xml.ToString();
DB.MarkJobAsCompleted(job);

DB.Commit()
```


Transaction Script

- Simple logic, simple data structures
- Procedural code
- Ensure each operation is transactional - either succeeds or fails
- Data is never corrupted

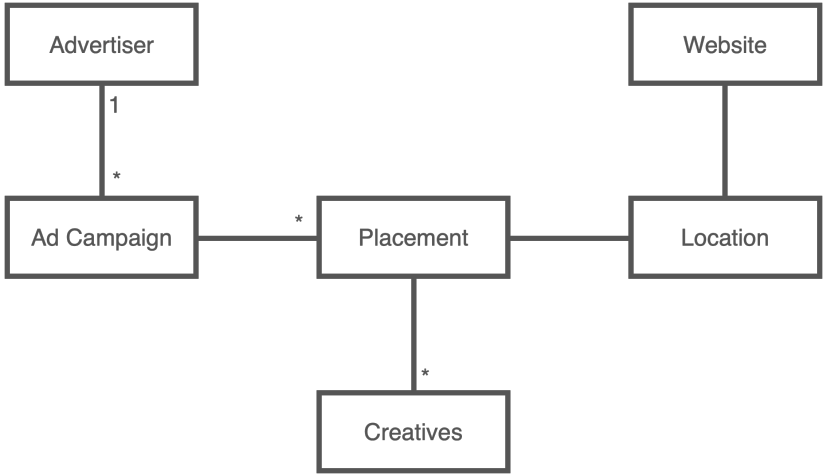
Transaction Script: Examples

- Import events from a client's systems
- Aggregate events from ad serving platform
- Enrich incoming events with information from other systems

Active Record

Active Record

- Simple business logic
- Complex data structures



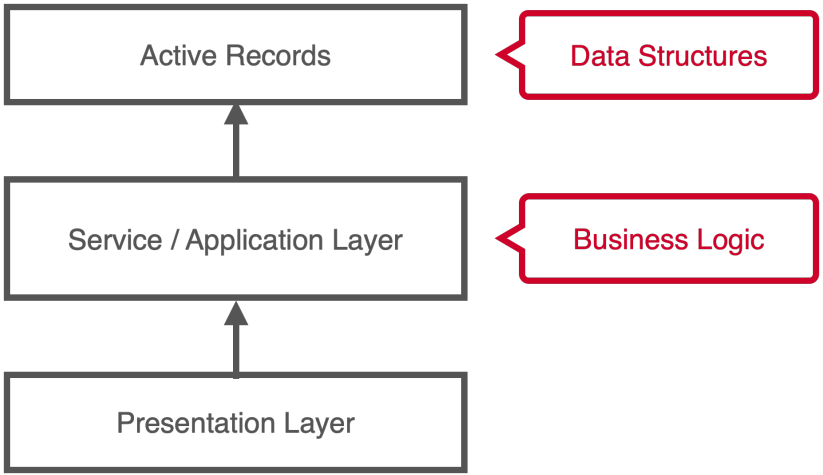
vladikk

Active Record

```
public class User {
    public Guid Id { get; set; }
    public string Name { get; set; }
    public List<Interest> Interests { get; set; }
    public Address Address { get; set; }

    public void Save() { … }
    public void Delete() { … }
    public static User Get(Guid id) { … }
    public static List<User> GetAll() { … }
}
```


Active Record

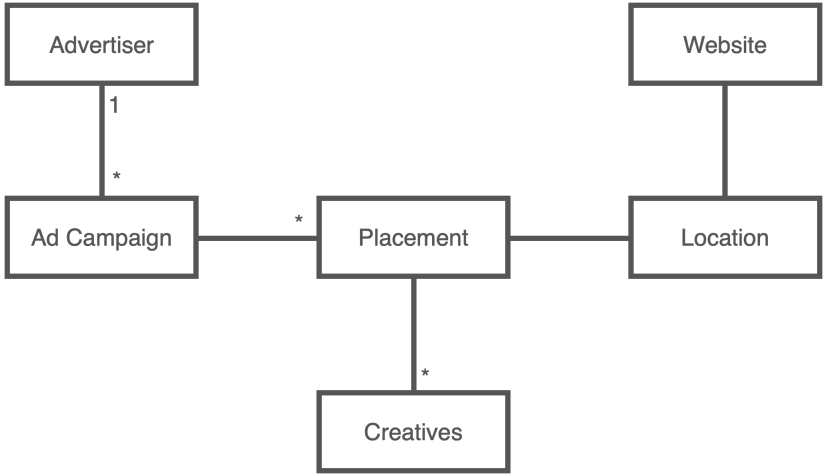


Active Record

```
public class CreateUser {
    public void Execute(userDetails) {
    try {
        DB.StartTransaction();
        var user = new User();
        user.Name = userDetails.Name;
        user.Email = userDetails.Email;
        user.Save();
        DB.Commit();
    } catch {
        DB.Rollback();
        throw;
    }
  }
}
```


Active Record

- Simple business logic
- Complex data structures



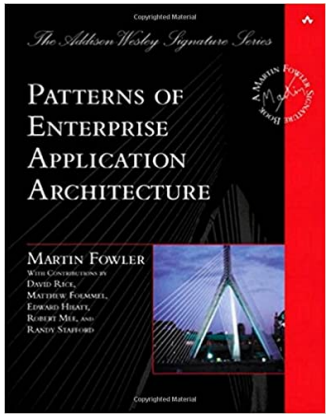
vladikk

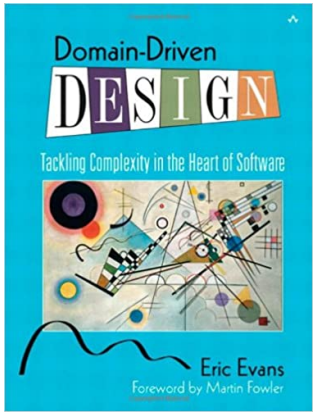
Domain Model

Domain Model

- Complex business logic
- Sophisticated algorithms
- Complicated business rules and invariants

Domain Model





vladikk

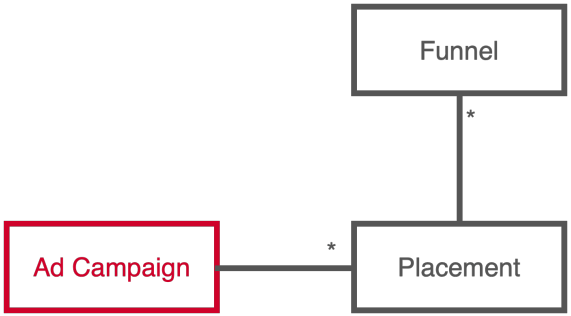
Domain Model Building Blocks

- Value Object
- Entity
- Aggregate
- Domain Event

Domain Model: Aggregate

- A cluster of objects
- Both data and behavior belongs to the aggregate’s boundaries
- Responsible for enforcing business rules and invariants
- No external entity or process can modify an aggregate’s data

Domain Model: Aggregate



Domain Model: Aggregate

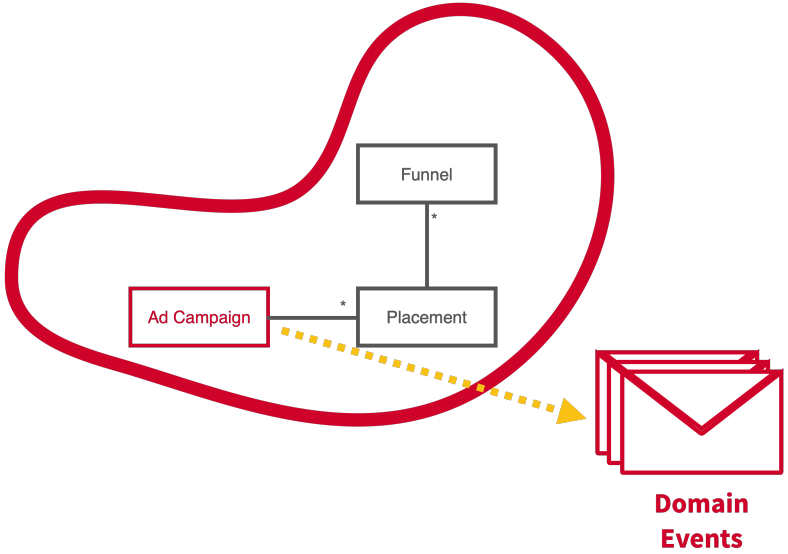
- A cluster of objects
- Both data and behavior belongs to the aggregate's boundaries
- Responsible for enforcing business rules and invariants
- No external entity or process can modify an aggregate's data
- Data inside an aggregate's boundary – strongly consistent
- Data outside – eventually consistent

Aggregate Design Rule:

Always try to find the smallest consistency boundaries

Domain Model: Domain Event

- Record of business significant occurrence
- Part of an aggregate's public interface
- Formulated in the Ubiquitous Language



Event Sourced Domain Model

Event Sourced Domain Model

- Aggregate’s state is represented as a series of events
- Introduces dimension of time
- Enables “time traveling”
- Retroactive debugging
- Flexible state projections
- Deep insight into the production data

State-Based Model

```
{
   “campaignId”: 123,
   “placements”: [
     {
        “placementId”: 100,
        “placementName”: “Wide banner”,
        “funnels”: [
          {
              “funnelId”: 200,
              “bannerId”: 110,
              “landingPageId”: 112
          },
          ….
        ]
        ,….
   ]
}
```

vladikk

Event-Based Model

- CampaignInitialized(“123”, “JCC market”, “01/04/2018”, “yahoo.com”, …)
- PlacementAdded(“100”, "Wide banner”, NotActive, …)
- FunnelAdded(“300”, “Video ads”,…)
- FunnelAdded(“301”, “Text ads”,…)
- FunnelAdded(“302”, “Mixed content”,…)
- PlacementAdded(“101”, "Narrow banner”, NotActive, …)
- FunnelAdded(“400”, “Popup”,…)
- FunnelAdded(“401”, “Popunder”,…)
- FunnelApprovedForPublishing(“400")
- PlacementActivated(“101”)
- CampaignPublished(“123”)
- CampaignExpired(“123”)

Business Logic Implementation Patterns

- Transaction Script
- Active Record
- Domain Model
- Event Sourced Domain Model

Discussion: When to Use Each Pattern?

- Transaction Script
- Active Record
- Domain Model
- Event Sourced Domain Model

Transaction Script

- Simple supporting subdomains
- Integrating 3-rd party solutions for generic subdomains

Active Record

- Supporting subdomains
- Mostly CRUD operations
- Simple business logic
- At most input validation

Domain Model

- Core subdomains
- Complex business logic

Event-Sourced Domain Model

- Core subdomains
- Complex business logic
- Monetary transactions
- Deep analysis of the system’s data and behavior is required

Marketing Bounded Context

Creative Catalogue

Campaign Management

Ad Serving

Campaign Optimization

Ad Spaces Acquisition

Accounting

Sales Bounded Context

Customer Relationship Management

Telephony

Sales Commissions Accounting

Logical Boundaries are Crucial!

Logical Boundaries

- Make sure each module has its explicit boundary
- Use appropriate design for each module in the monolith
- Make sure the design is chosen consciously, not accidentally
- Leverage the flexible boundaries – it’s okay to be wrong
- Be pragmatic, but not negligent! Yes, it’s hard.

Key Takeaways

- Ubiquitous language is a shared, business domain-oriented language for all project related communication
- Language is a model of the business domain
- A bounded context is a ubiquitous language's/model's applicability context
- A ubiquitous language is consistent in its bounded context
- A bounded context denotes the boundary of the largest valid monolith
- Subdomains are discovered, whereas bounded contexts are designed
- Logical boundaries are crucial for designing a modular monolith
- Business logic implementation patterns: Transaction script, active record, domain model, event sourced domain model

Questions?

Bounded Contexts

Integration Patterns

Bounded Context: Ownership Boundary

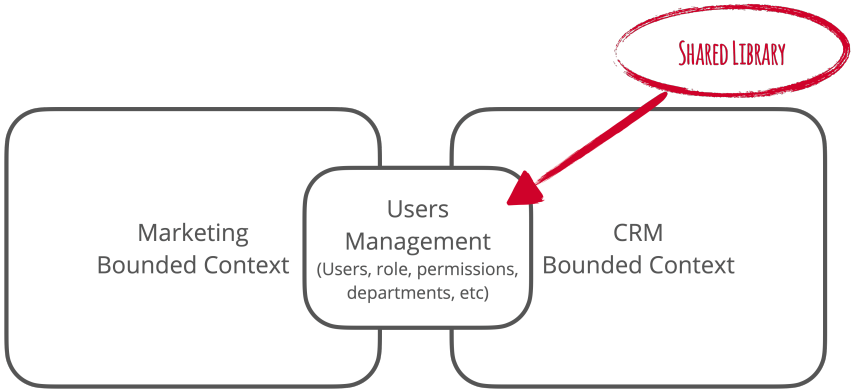
- One team owns a bounded context
- One team can own multiple bounded contexts
- A bounded context **cannot** be developed by multiple teams
- Bounded contexts have to be integrated with each other
- Bounded contexts integrations often define collaboration patterns between teams.

Bounded Context Integration Patterns

- Shared Kernel
- Partnership
- Conformist
- Anti-Corruption Layer
- Open Host Service
- Separate Ways

Shared Kernel

- Shared library that multiple BCs use
- Overlapping code
- Tightly coupled relationship
- The same team owns all the BCs



Partnership

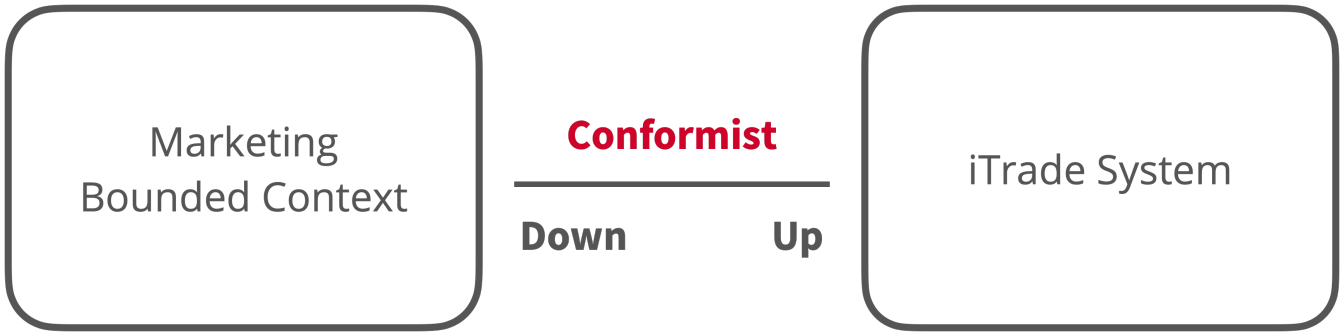
- Ad hoc collaboration
- Shared goals – succeed or fail together
- Collaborating teams, easy to agree and to communicate
- Ideally 1 team owning multiple BCs

Customer – Supplier Integration Patterns

- One bounded context provides a service to other BCs
- Upstream/Downstream – not flow of data!
- Upstream – a service provider
- Downstream – the service's clients
- Not a balanced relationship

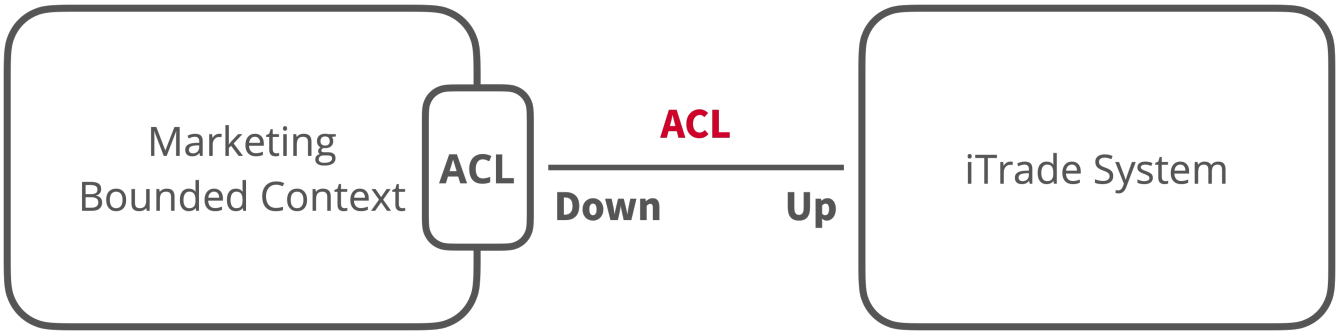
Conformist

- Downstream conforms to the model of upstream
- Client written in the language as defined by upstream



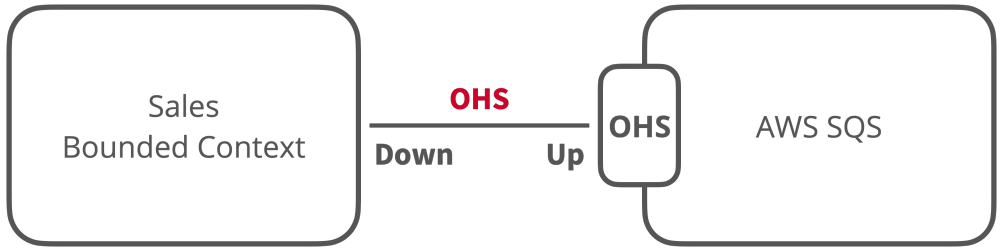
Anti-Corruption Layer

- Protects the language of downstream from the language of upstream
- Part of the client



Open Host Service

- Public API as a part of the upstream service design
- Published model can be evolved independently of the UL of upstream
- Published model and model language evolve independently to protect the service's clients
- Makes the public interface more stable



Separate Ways

- Can’t collaborate well
- Communication is more expensive than duplication

Bounded Context Integration Patterns

- Shared Kernel
- Partnership
- Conformist
- Anti-Corruption Layer
- Open Host Service
- Separate Ways

Discussion: Domain-Driven Integration

Question: Would you use the separate ways pattern for a bounded context implementing a core subdomain?

Discussion: Domain-Driven Integration

Question: Would you use the separate ways pattern for a bounded context implementing a core subdomain?

Answer: No! Duplicate complex and volatile logic is a bad idea.

Discussion: Domain-Driven Integration

Question: Which downstream subdomain is more likely to implement an anti-corruption layer – core or supporting?

Discussion: Domain-Driven Integration

Question: Which downstream subdomain is more likely to implement an anti-corruption layer – core or supporting?

Answer: Core subdomain, to protect its model

Discussion: Domain-Driven Integration

Question: Which upstream subdomain is more likely to implement the open host pattern – core or supporting?

Discussion: Domain-Driven Integration

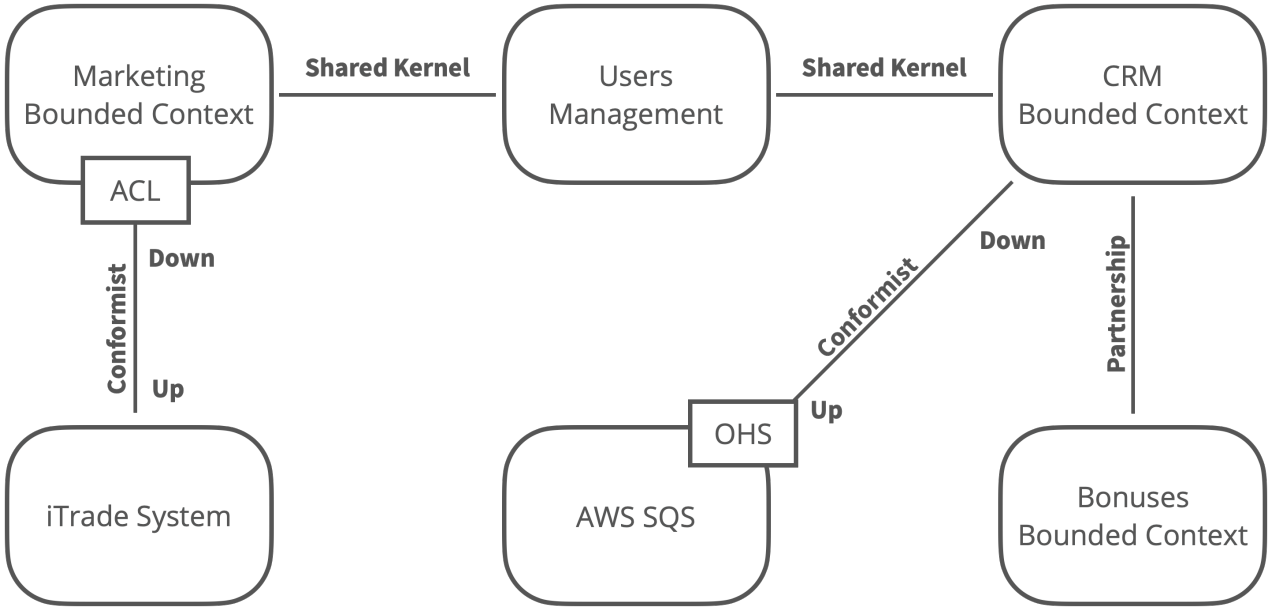
Question: Which upstream subdomain is more likely to implement the open host pattern – core or supporting?

Answer: Core subdomain to protect its clients from changes in its implementation details

Bounded Context Integration Patterns

- Shared Kernel
- Partnership
- Conformist
- Anti-Corruption Layer
- Open Host Service
- Separate Ways

Context Map


Marketing
Bounded Context
ACL
Shared Kernel
Users
Management
Shared Kernel
CRM
Bounded Context
Conformist
Down
Up
iTrade System
Conformist
Down
Up
OHS
AWS SQS
Partnership
Bonuses
Bounded Context

Key Takeaways

- **Shared Kernel**
  Bounded context implemented as a compiled library which is used by multiple other bounded contexts.

- **Partnership**
  Multiple bounded contexts implemented by well-collaborating teams. Models can be shared among the contexts, and teams ready to collaborate to evolve them simultaneously.

- **Customer-Supplier**
  One bounded context acts as a service provider, and other contexts are its consumers.

Key Takeaways

- **Conformist**
  The customer conforms to the supplier's model.

- **Anti-Corruption Layer**
  The customer translates the supplier's model into a more convenient model.

- **Open-Host Service**
  To provide more stable level of service, the supplier decouples its public API model from its internal implementation model.

- **Separate Ways**
  Communication between the teams is challenging, and it's more expensive to cooperate than to duplicate the service logic.

Questions?

Evolving Design Decisions

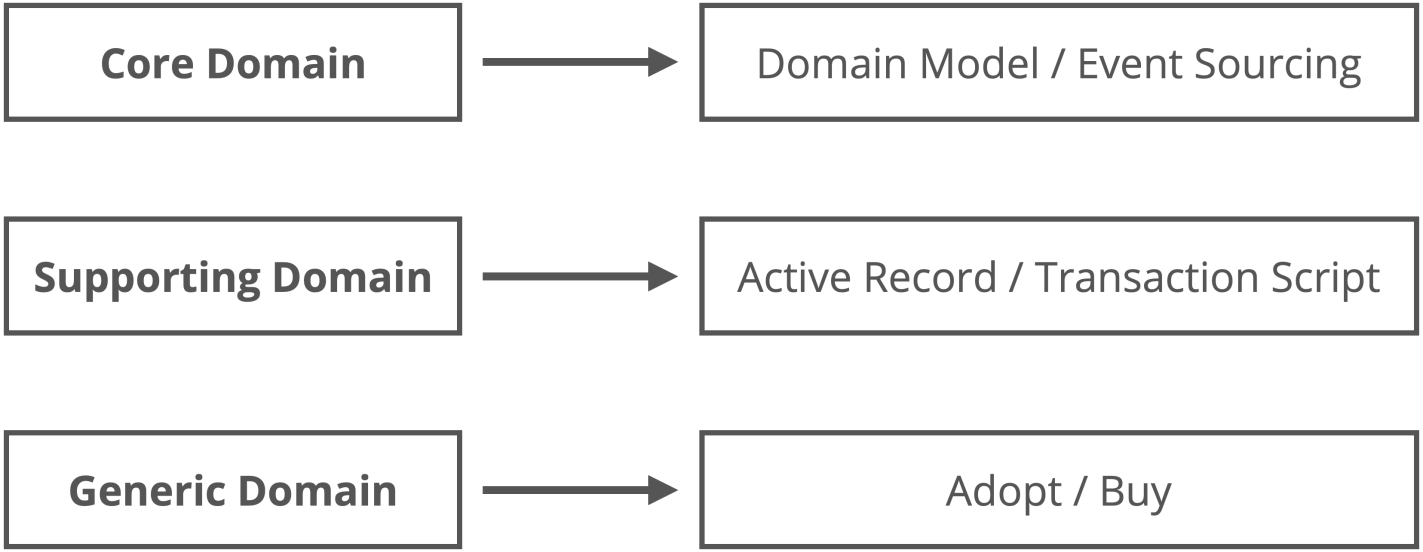
Evolving Design Decisions

- No project starts as a big ball of mud
- Inadequate response to changes in the business domain leads to projects to degrade into big balls of mud
- Requirements will always change
- Core subdomains will change most frequently

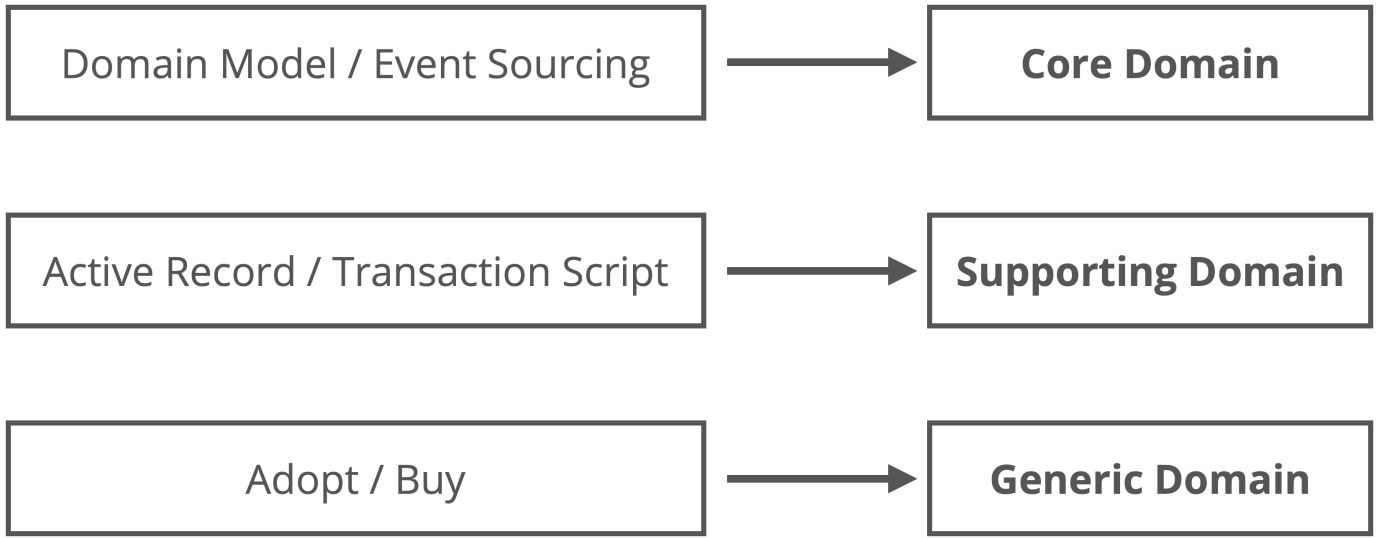
Changes in Subdomains

- Subdomains' types can change
- It's crucial to spot such changes in the business strategy, and adapt the design accordingly
- For example: business logic implementation pattern, bounded contexts integration strategy

Changes in Subdomains



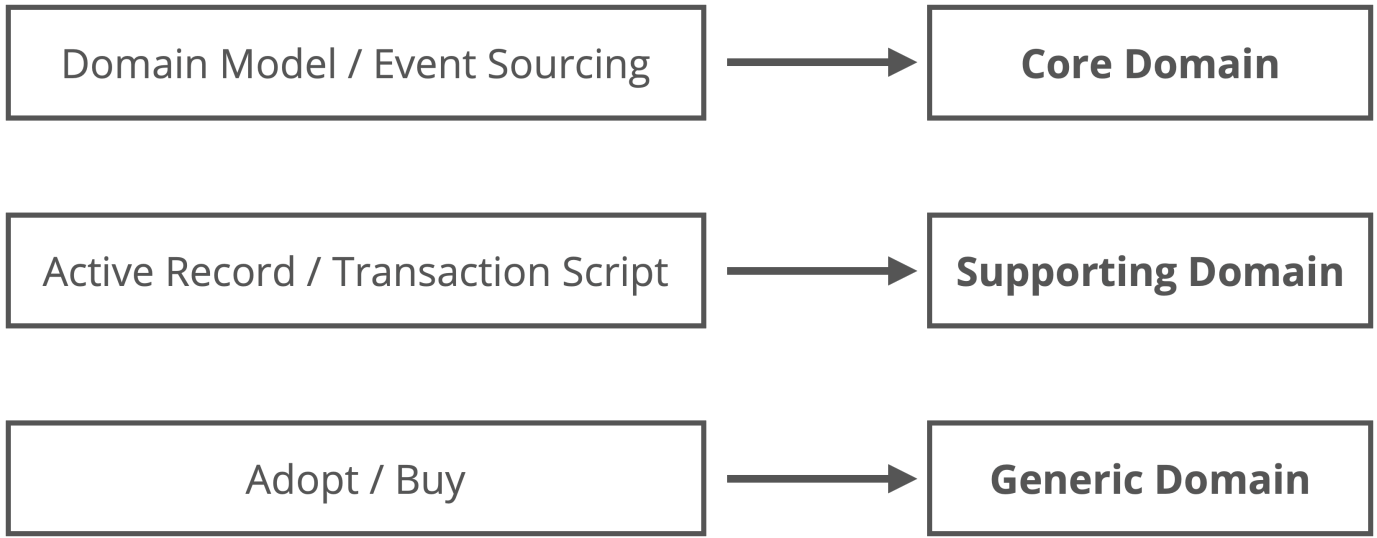
Changes in Subdomains



Changes in Subdomains

- Monetary transactions? Deep data analysis? Audit log? - **Event Sourced Domain Model**
- Complex business logic? **Domain Model**
- Complex data structures? **Active Record**
- Simple logic and data structures? **Transaction Script**

Changes in Subdomains

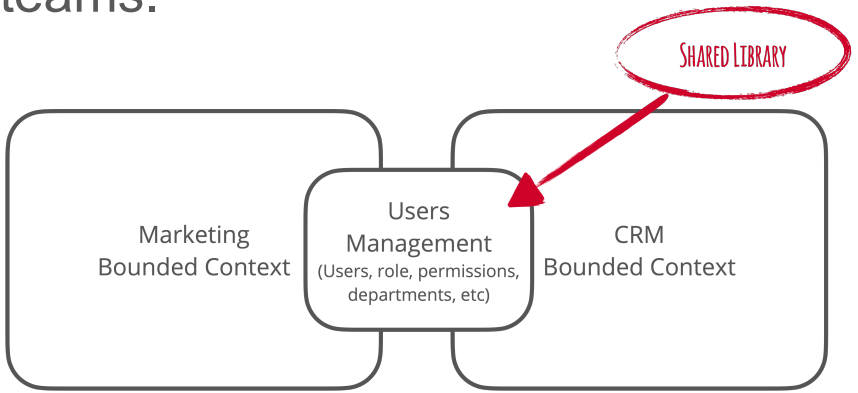


Changes in the R&D Organization

- A company's R&D department size can change overtime
- Locally or globally
- Affects strategic design decisions
- For example: decomposition to bounded contexts, bounded contexts integration strategy

Shared Kernel

- Good fit for a team working on multiple bounded contexts
- The decision has to be re-evaluated if the work on the bounded contexts is distributed between two teams.

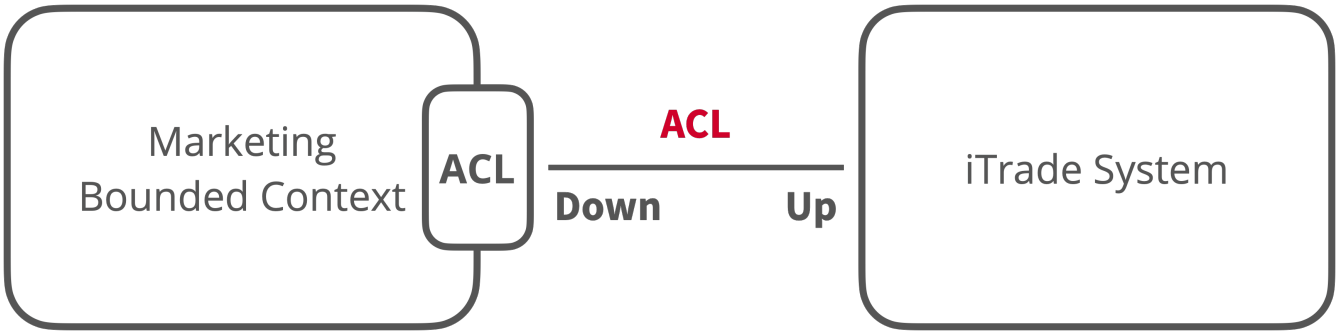


Partnership

- The “partnership” pattern fits well-cooperating teams.
- Preferably co-located, or with good communication between them.
- What if the organization’s growth hinders communication between the teams?

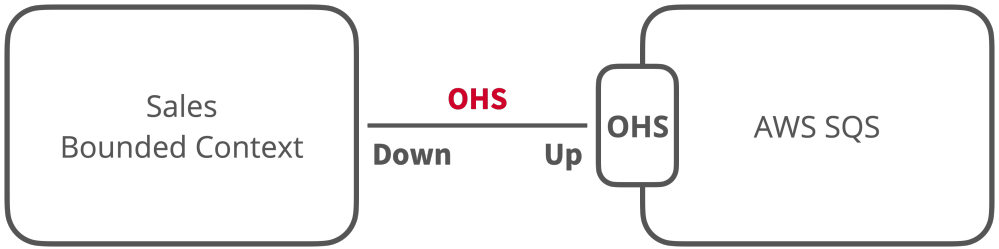
Anti-Corruption Layer

When cooperation issues arise, an Anti-Corruption Layer may protect the downstream team from using a model dictated by the upstream team.



Open Host Service

When organization grows, and suddenly a service is being used by many clients, it can make sense to decouple its public API from its internal implementation model.



Evolving Service Providers (Upstream)

Shared Kernel  Partnership  Open Host Service

Evolving Service Consumers (Downstream)

Partnership → Conformist → Anti-Corruption Layer

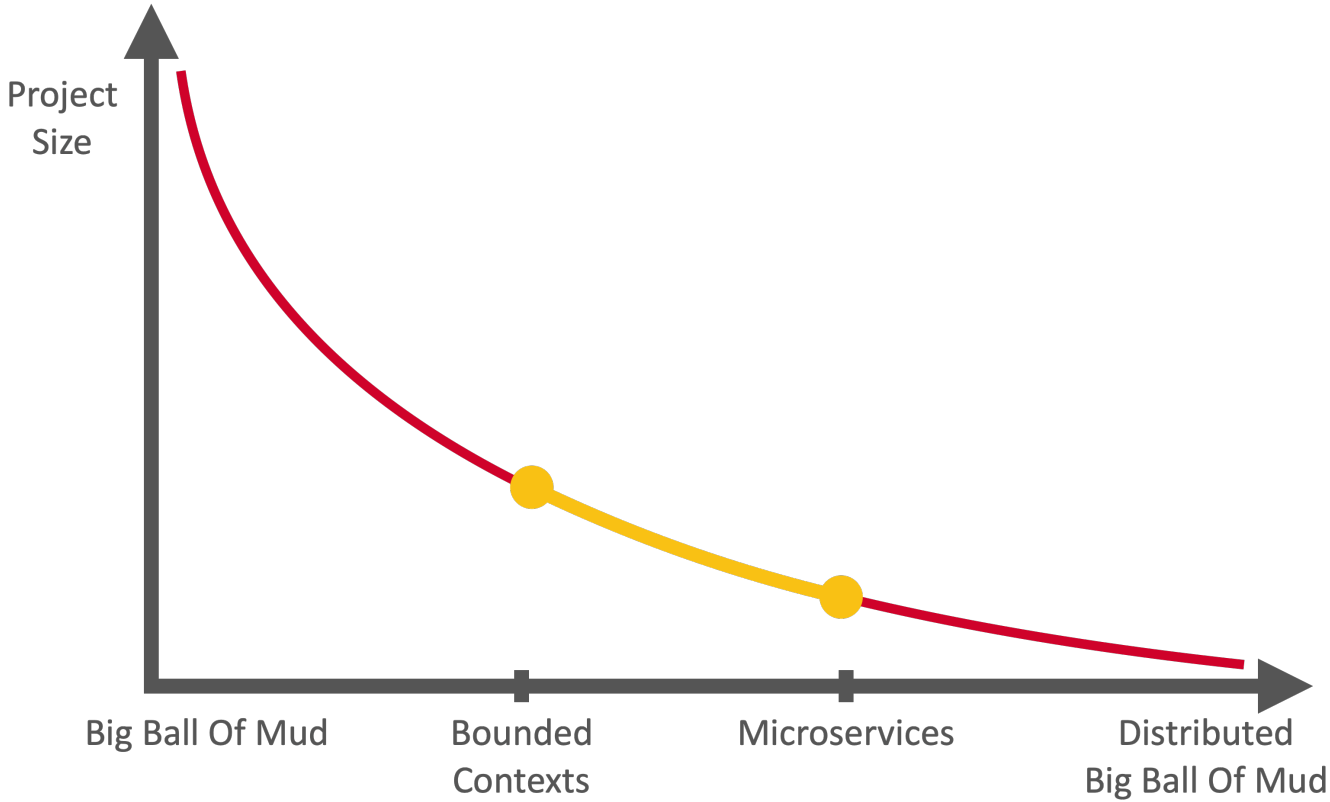
Changes in Domain Knowledge

- With time you gain more business domain knowledge
- Especially true for core subdomains
- New insights can and should affect design decisions
- For example: business logic implementation pattern, business logic model, transactional boundaries, etc.

THE LESS YOU KNOW ABOUT THE DOMAIN

THE WIDER THE PHYSICAL BOUNDARIES

START WITH BIGGER PHYSICAL BOUNDARIES
DECOMPOSE LATER, AS YOU GAIN KNOWLEDGE


Project
Size
Big Ball Of Mud
Bounded
Contexts
Microservices
Distributed
Big Ball Of Mud

Changing Decomposition to Bounded Contexts

Gradually refactoring logical boundaries into physical:

- Tighten up the logical boundaries
- Ensure no code/process steps over them (permissions, static code analysis)
- Eliminate context coupling
- Expose the extracted module as a service

Key Takeaways

- Inadequate response to changes degrades projects into BBoM
- Types of business subdomains change – make sure your design is adapted when such change happens
- Changes in the R&D department affect bounded contexts and interactions between them
- New domain knowledge insights should be leveraged to improve the design
- When needed change bounded contexts decomposition gradually

Questions?

Event Storming

What is Event Storming?

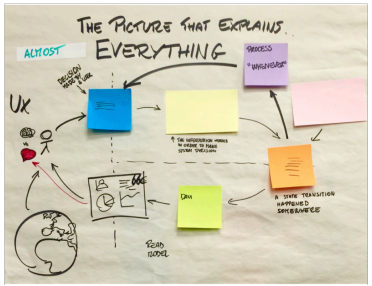
- Easy to learn, low-tech collaboration technique
- Brings together software engineers and domain experts for shared learning experience
- Allows to quickly understand what happens in your business domain
- Allows to quickly start cultivating and refining a Ubiquitous Language

Who Can Participate in Event Storming?

- Software engineers
- Product owners
- Domain experts
- UX Designers
- QA engineers
- DBAs
- … and all other stakeholders related to the project

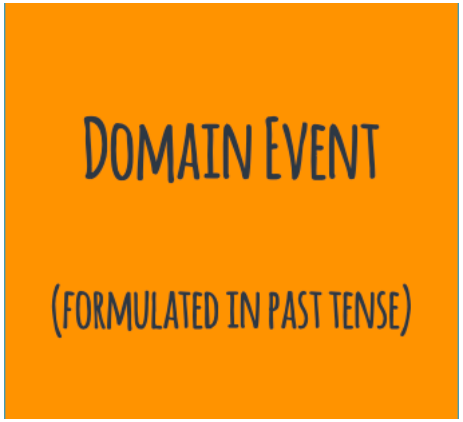
What Do You Need for Event Storming?

- Modeling surface:
  - Big wall + roll of paper
  - Big whiteboard
  - Online tool (Miro)
- Lots of markers
- Lots of colored stickies
- Visible Legend

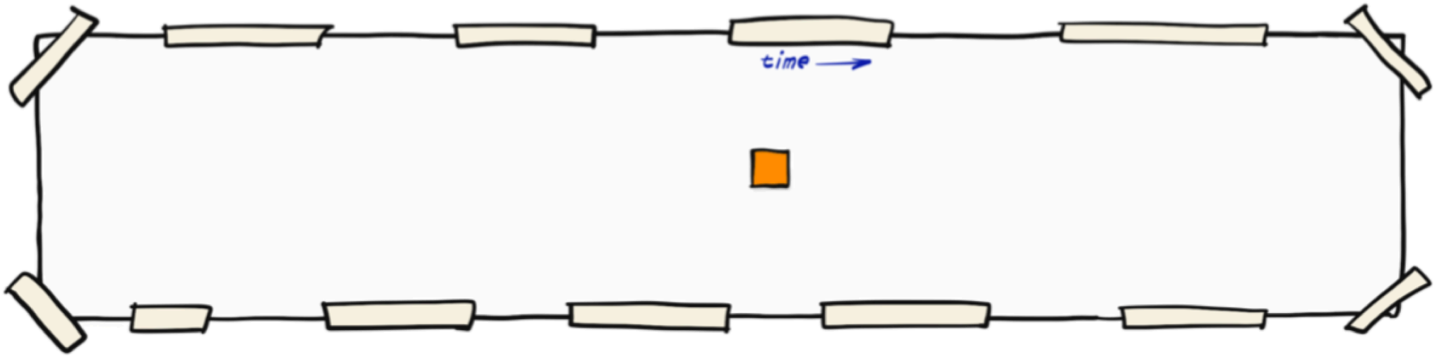


Step 1: Unstructured Exploration

- Brainstorming exercise
- Generating information to be used later
- Everybody should write and stick whatever they believe is a domain event
- Not caring about order, correctness, or redundancy
- Facilitator should not be the first to put a sticky



Step 1: Unstructured Exploration



Step 2: Organizing Events Into Timelines

- Refining the “events only” model
- Moving them around to form timelines
- Eliminating redundant domain events
- Throwing away incorrect domain events
- Adding missing events

Step 3: Augmenting with Pain Points

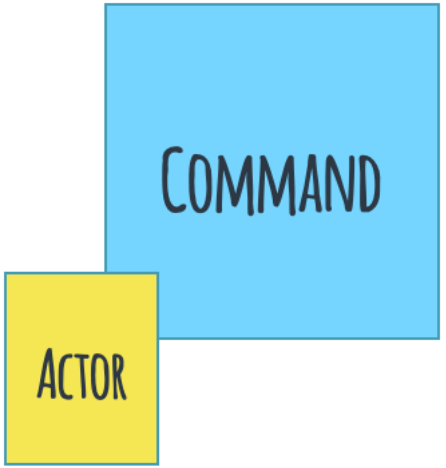
- Marking pain points
- Bottlenecks
- Manual processes
- Inefficiencies
- Unknown behavior

Step 4: Augmenting with Commands

- What action triggered an event?
- Formulated in the imperative tense

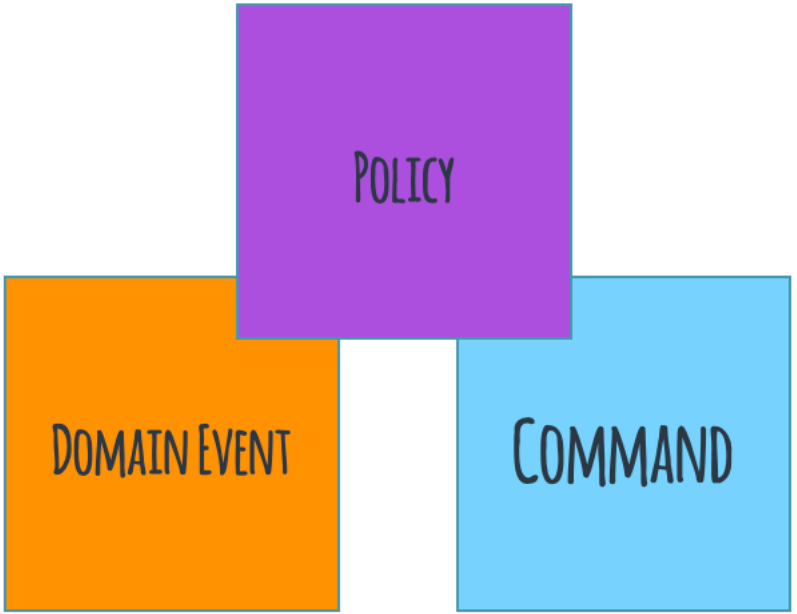
Step 5: Augmenting with Actors

- Who executes those commands?
- A user persona within the problem domain



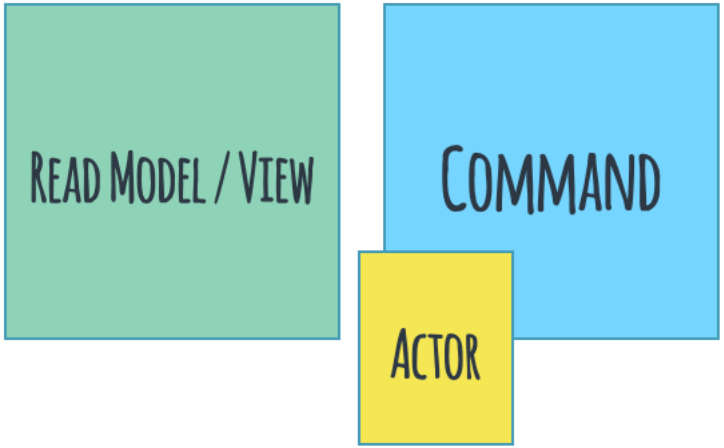
Step 6: Augmenting with Policies

- Automation that executes a command whenever some domain event is observed
- Connects domain events to commands through policy
- Alternative to actor



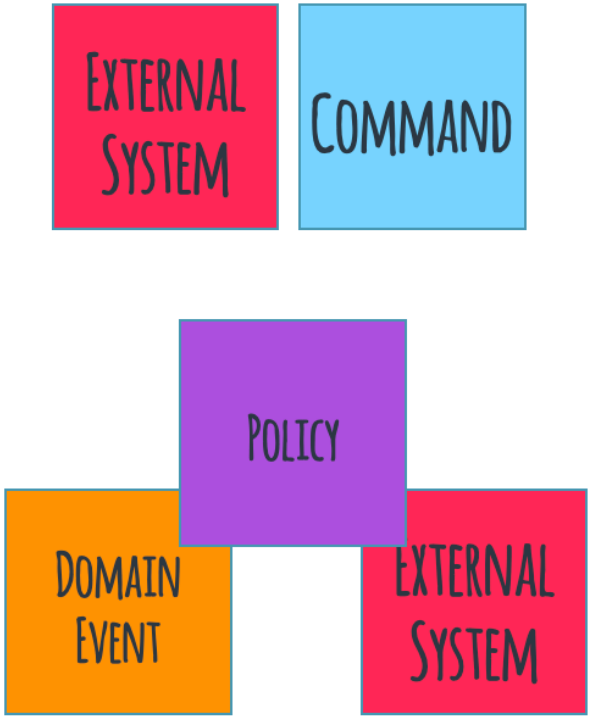
Step 7: Augmenting with Read Models

- View of data captured within the domain that the actor uses to make a decision to execute a command
- Decision supporting information
- The system's major screens



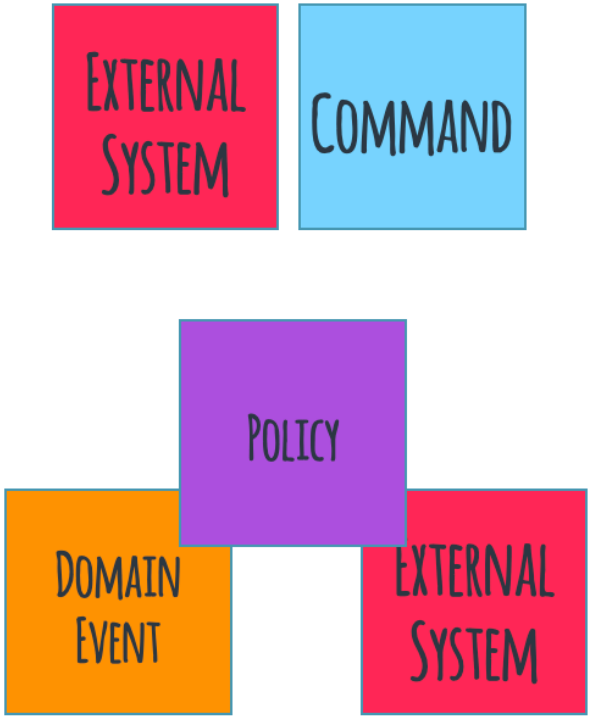
Step 8: Adding External Systems

- Systems outside of the modeled domain that we are collaborating with
- Not the systems we are currently designing
- Can be used as input and/or output



Step 9: Augmenting with Aggregates

- Systems outside of the modeled domain that we are collaborating with
- Not the systems we are currently designing
- Can be used as input and/or output



vladikk

Event Storming Doesn’t Work When

- Not enough people are participating in the workshop
- Not a diverse group of participants
- Modeling supporting subdomains - nothing to model

Getting Started with DDD

Greenfield Projects

- Do event storming to dive into the business domain and start cultivating a ubiquitous language
- Identify conflicting models
- Define bounded contexts
- Make business domain-driven design decisions

Brownfield Projects

- Assess what has changed: business domain, organization, knowledge
- Do event storming to dive into the business domain, recover lost business knowledge, and start cultivating a ubiquitous language
- Identify existing bounded contexts and build a context map

Brownfield Projects

- Look for inefficient models and bad context boundaries
- Apply gradual refactoring to boundaries
- Identify subdomains and business logic implementation patterns
- Evolve design decisions where needed

Summary

Business Domain Analysis

- Monolith ≠ Big Ball of Mud
- Business domain is a company's overall area of activity
- Subdomain – fine-grained area of business activity
- Core Subdomain – a company's competitive advantage
- Supporting Subdomain – an activity that provides no competitive advantage, but required for implementation of a core subdomain
- Generic Subdomain – problem domain with existing proven solutions

Acquiring and Modeling Business Domain Knowledge

- Ubiquitous language is a shared, business domain-oriented language for all project related communication
- Language is a model of the business domain
- A bounded context is a ubiquitous language's/model's applicability context
- A ubiquitous language is consistent in its bounded context
- A bounded context denotes the boundary of the largest valid monolith
- Subdomains are discovered, whereas bounded contexts are designed
- Logical boundaries are crucial for designing a modular monolith
- Business logic implementation patterns: Transaction script, active record, domain model, event sourced domain model

Bounded Contexts Integration Patterns

- **Shared Kernel**
  Bounded context implemented as a compiled library which is used by multiple other bounded contexts.

- **Partnership**
  Multiple bounded contexts implemented by well-collaborating teams. Models can be shared among the contexts, and teams ready to collaborate to evolve them simultaneously.

- **Customer-Supplier**
  One bounded context acts as a service provider, and other contexts are its consumers.

Bounded Contexts Integration Patterns

- **Conformist**
  The customer conforms to the supplier's model.

- **Anti-Corruption Layer**
  The customer translates the supplier's model into a more convenient model.

- **Open-Host Service**
  To provide more stable level of service, the supplier decouples its public API model from its internal implementation model.

- **Separate Ways**
  Communication between the teams is challenging, and it's more expensive to cooperate than to duplicate the service logic.

Evolving Design Decisions

- Inadequate response to changes degrades projects into BBoM
- Types of business subdomains change – make sure your design is adapted when such change happens
- Changes in the R&D department affect bounded contexts and interactions between them
- New domain knowledge insights should be leveraged to improve the design
- When needed change bounded contexts decomposition gradually

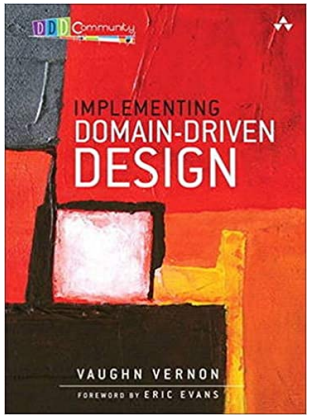
Greenfield Projects

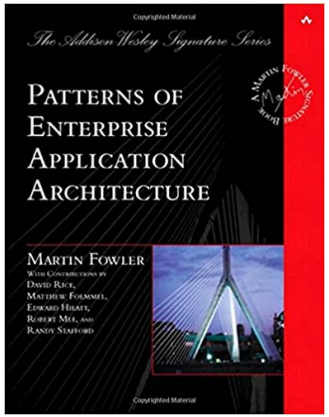
- Do event storming to dive into the business domain and start cultivating a ubiquitous language
- Identify conflicting models
- Define bounded contexts
- Make business domain-driven design decisions

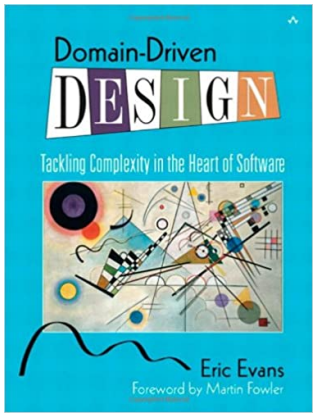
Brownfield Projects

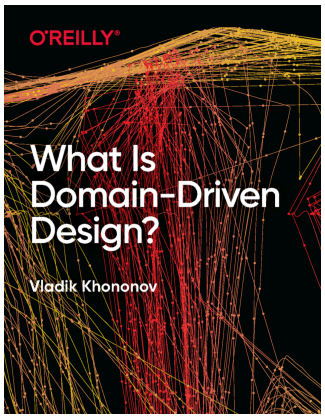
- Assess what has changed: business domain, organization, knowledge
- Do event storming to dive into the business domain, recover lost business knowledge, and start cultivating a ubiquitous language
- Identify existing bounded contexts and build a context map

Further Learning





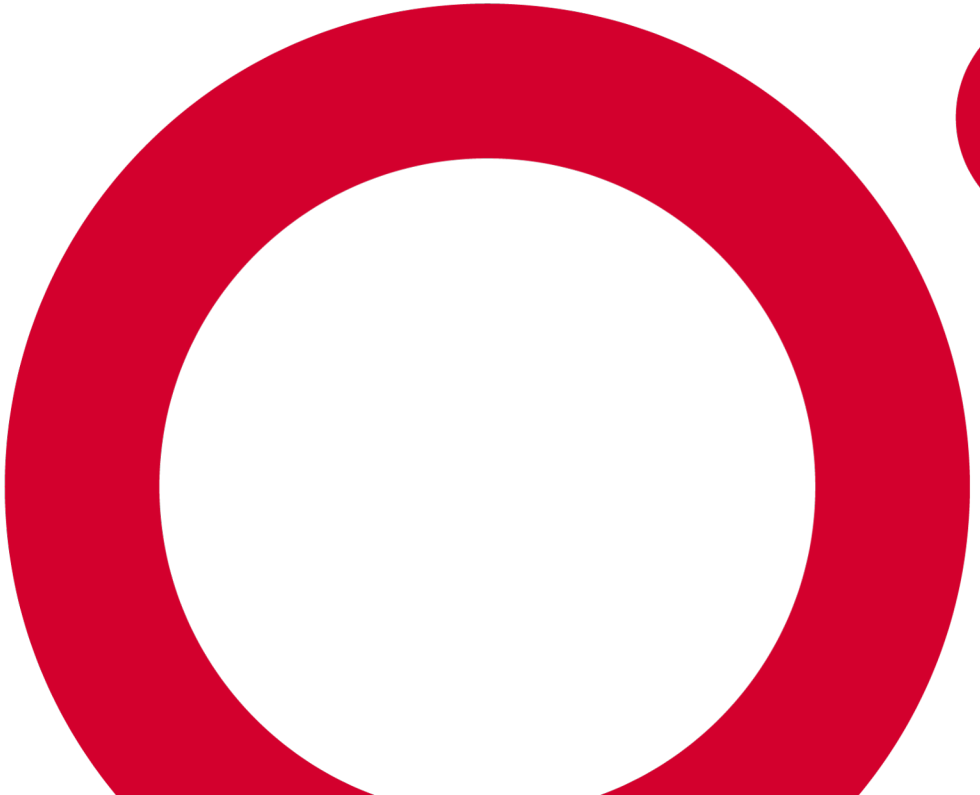




vladikk

O'REILLY®

Thank you



vladikk

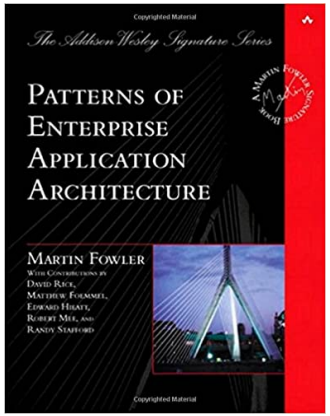
P.S.

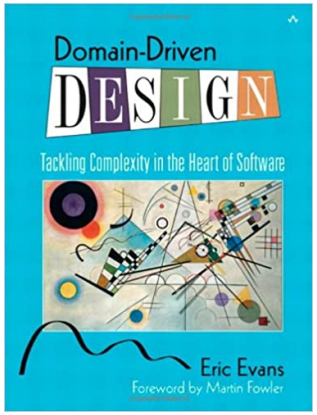
Domain Model

Domain Model

- Complex business logic
- Sophisticated algorithms
- Complicated business rules and invariants

Domain Model





vladikk

Domain Model Building Blocks

- Value Object
- Entity
- Aggregate
- Domain Event

Domain Model: Value Object

- Object identified by its value
- Examples: Color, status, address
- Implemented as immutable object

Domain Model: Value Object

Id	Value
1	Red
2	Yellow
3	Green

Domain Model: Value Object

```
public struct Color {
    public readonly int Red;
    public readonly int Green;
    public readonly int Blue;

    public Color MixWith(Color other) { … }
}
```


Domain Model: Value Object

- Object identified by its value
- Examples: Color, status, address
- Implemented as immutable object

Domain Model: Entity

- Opposite of Value Object
- Cannot be identified by value alone
- Sate changes, but identity remains the same
- Example: Person

Domain Model: Entity

```
public class Person {
    public Guid Id { get; }
    public String FirstName { get; }
    public String LastName { get; }

    …
}
```

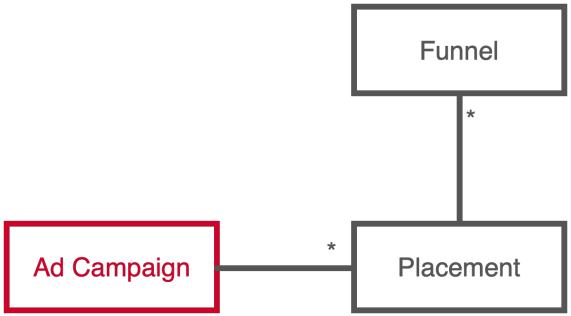
Entities vs. Value Objects

- Value Object: Identified by its value
- Entity: Identified by explicit Id

Domain Model: Aggregate

- A cluster of objects
- Both data and behavior belongs to the aggregate’s boundaries
- Responsible for enforcing business rules and invariants
- No external entity or process can modify an aggregate’s data

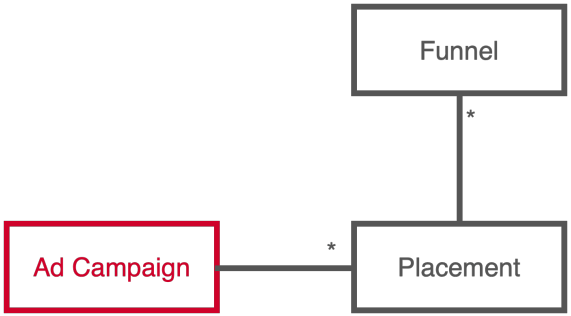
Domain Model: Aggregate



Domain Model: Aggregate

- A cluster of objects
- Both data and behavior belongs to the aggregate's boundaries
- Responsible for enforcing business rules and invariants
- No external entity or process can modify an aggregate's data
- Data inside an aggregate's boundary – strongly consistent
- Data outside – eventually consistent

Domain Model: Aggregate



Domain Model: Aggregate

- A cluster of objects
- Both data and behavior belongs to the aggregate's boundaries
- Responsible for enforcing business rules and invariants
- No external entity or process can modify an aggregate's data
- Data inside an aggregate's boundary – strongly consistent
- Data outside – eventually consistent

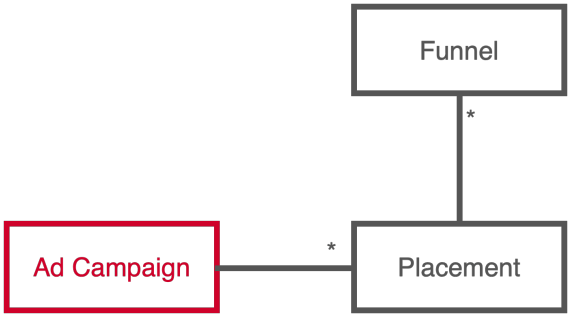
Aggregate Design Rule:

Always try to find the smallest consistency boundaries

Domain Model: Aggregate Root

- Aggregate’s root object
- Acts as a public interface to the aggregates data and methods
- Can contain other aggregates

Domain Model: Aggregate



Domain Model: Domain Event

- Record of business significant occurrence
- Part of an aggregate's public interface
- Formulated in the Ubiquitous Language