Domain-Driven Design Fundamentals

Introducing Domain-Driven Design

Steve Smith

Force Multiplier for
Dev Teams

@ardalis | ardalis.com

Julie Lerman

Software Coach,
DDD Champion

@julielerman | thedatafarm.com

Some Recommended Books on DDD

Domain-Driven Design, Tackling Complexity in the Heart of Software Eric Evans

Applying Domain-Driven Design and Patterns, Jimmy Nillson

Implementing Domain-Driven Design, Vaughn Vernon

Professional Domain-Driven Design Patterns, Scott Millett, Nick Tune

Domain-Driven Design Distilled, Vaughn Vernon

Domain Modeling Made Functional, Scott Wlaschin

Hands-on Domain-Driven Design with .NET Core, Alexey Zimarev

Conferences

Domain-Driven Design Europe (Amsterdam)

KanDDDinsky (Berlin)

Explore DDD Conference (Colorado)

DDD Exchange from Skills Matter (London)

Virtual DDD Meetup (on the web)

Course Introduction

Why should you care about DDD?

What does a DDD solution look like?

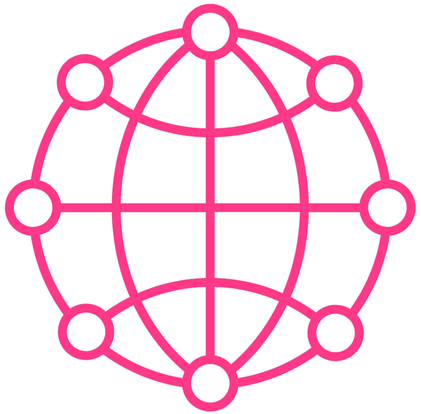
Modeling problems in software

Technical components of DDD

Managing complexity using DDD

Extending our solution as the domain evolves

Goals of This Course



Socrates

“The more you know, the more you realize you know nothing.”

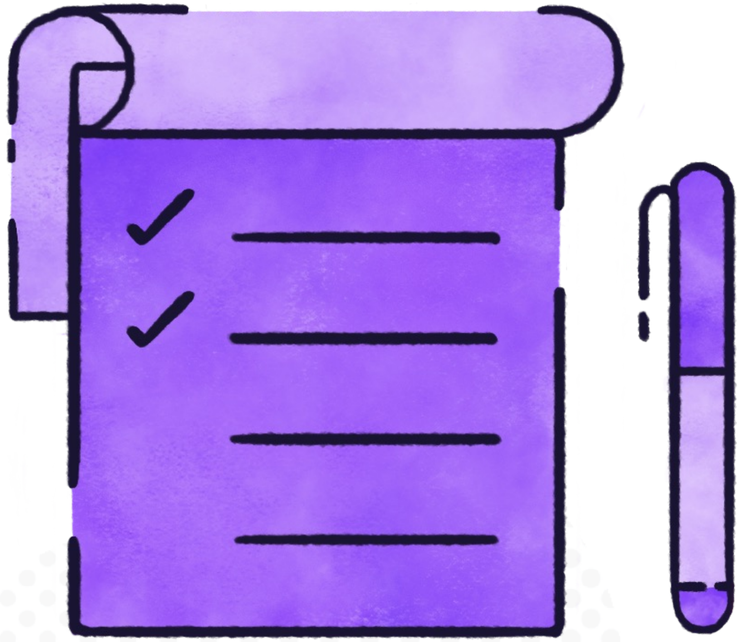
Module Overview

What is Domain-Driven Design?

Goals of DDD

Benefits and potential drawbacks

View reference application

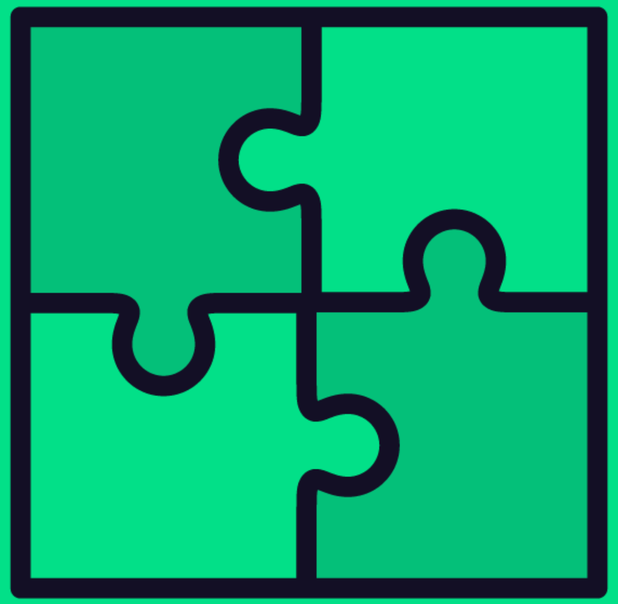


Understanding the Value of Domain-Driven Design

Martin Fowler

“Domain-Driven Design is an approach to software development that centers the development on programming a domain model that has a rich understanding of the processes and rules of a domain.”

Why DDD?



Why DDD?

Value Proposition of DDD

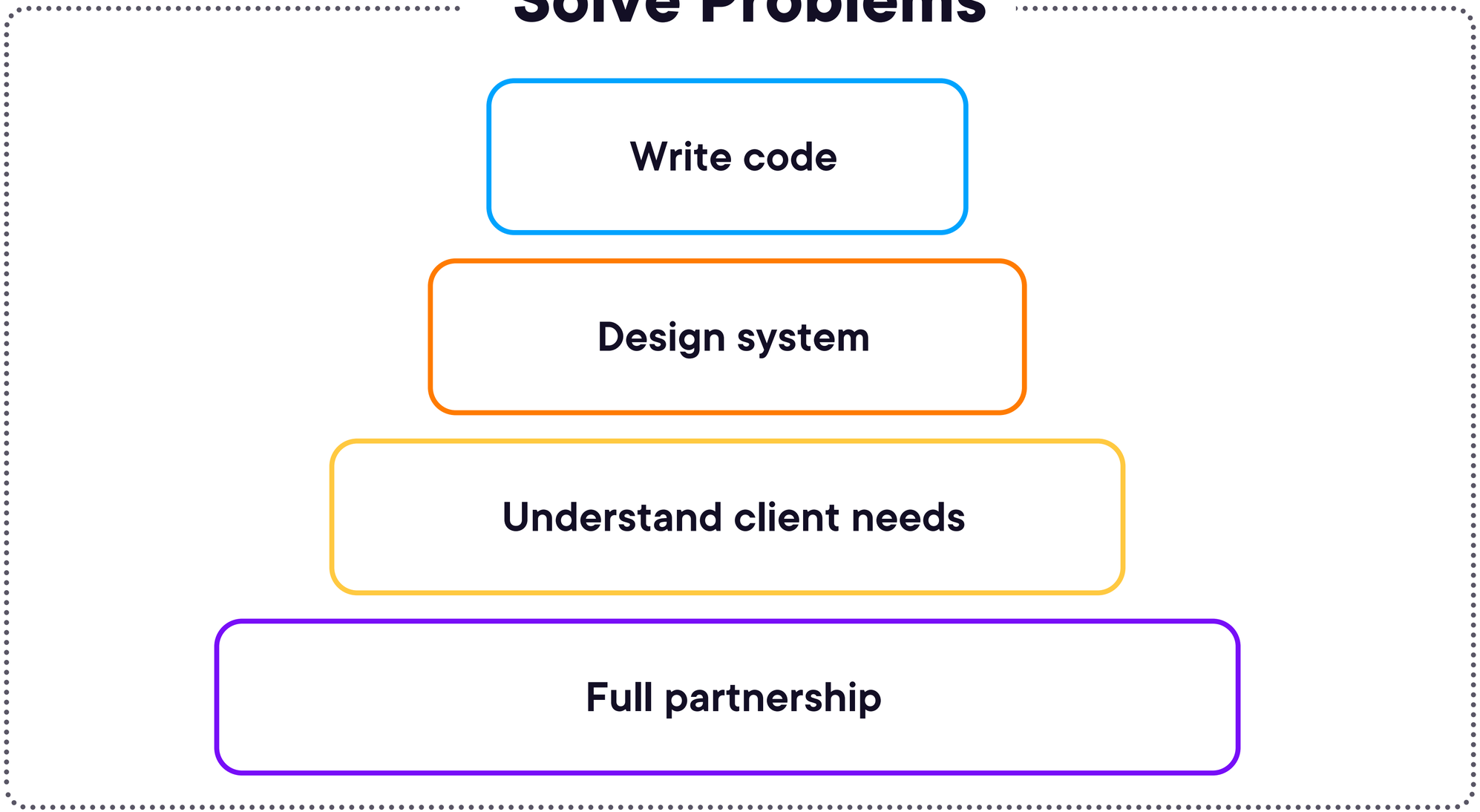
Principles and patterns to
solve difficult problems

History of success
with complex projects

Aligns with practices
from our own experience

Clear, readable, testable code
that represents the domain

Solve Problems



Gaining a High-level Understanding of DDD

“Tackling Complexity in the Heart of Software”

Interaction with domain experts

Eric Evans

“You really need to cultivate your ability to communicate with business people to free up people's creative modeling.”

Interaction with domain experts

Model a single subdomain at a time

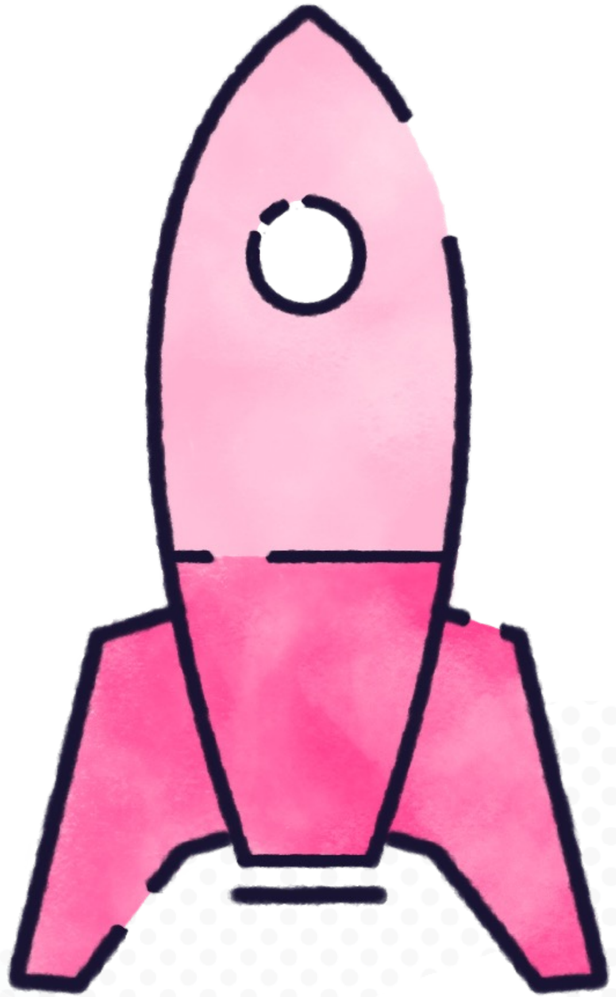
Purchase materials

Engineering

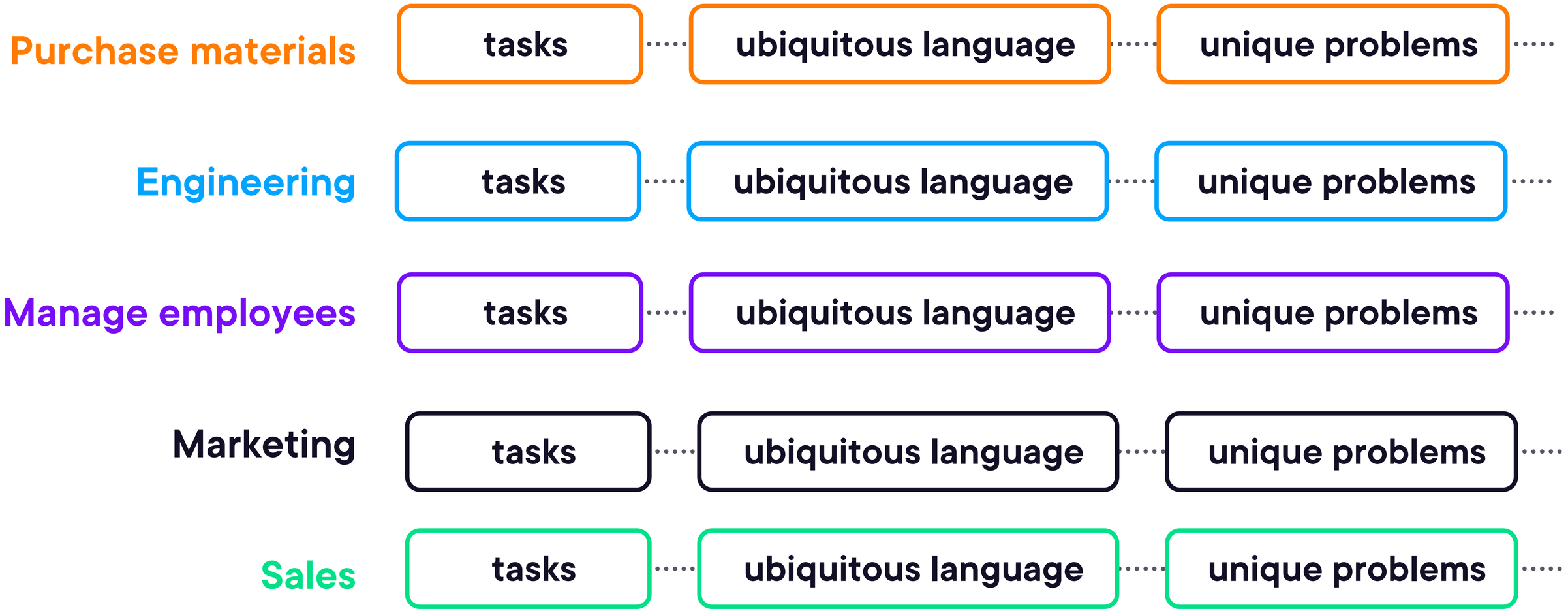
Manage employees

Marketing

Sales



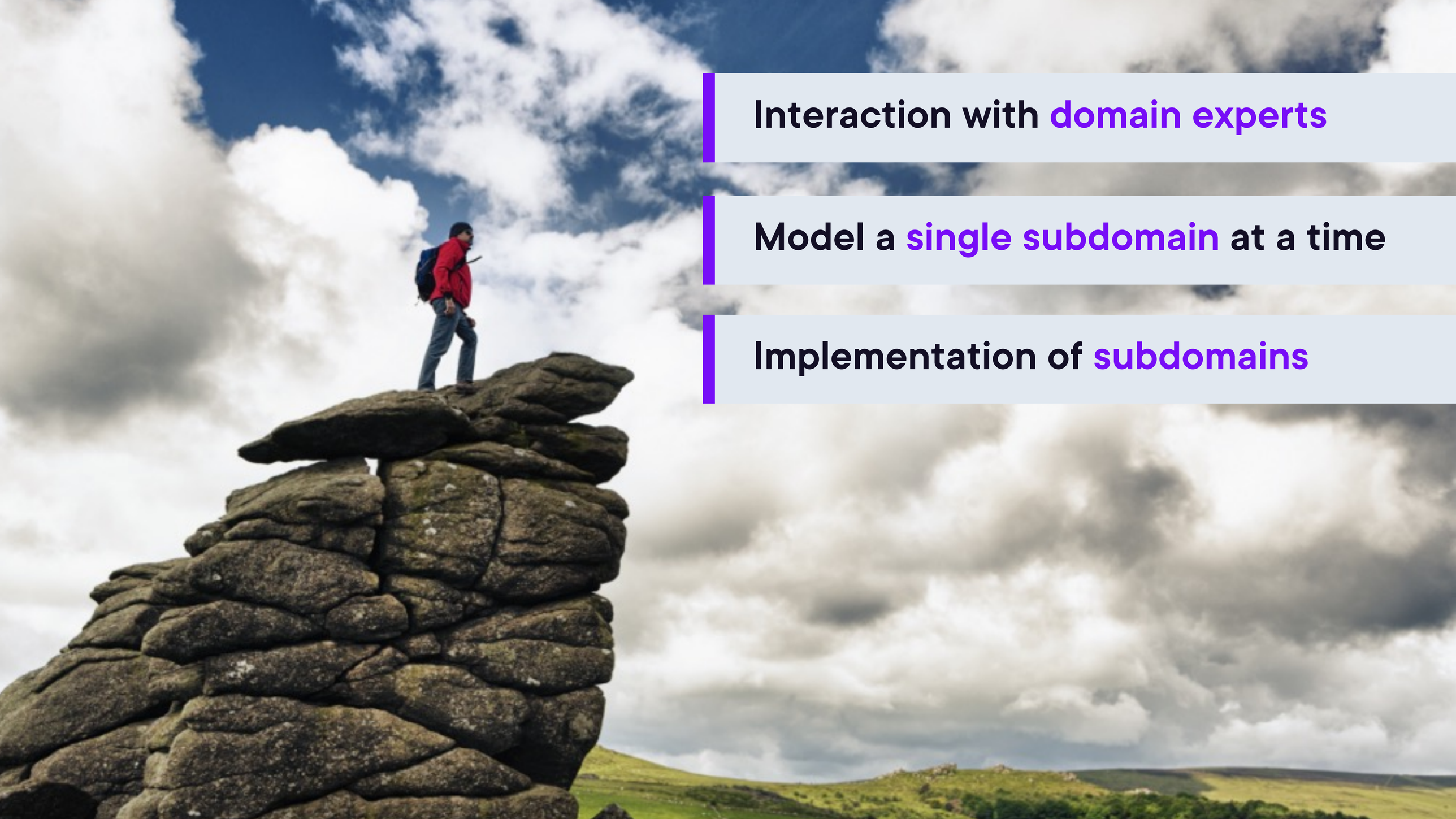
Separate Subdomains for Each Problem



Divide and Conquer

Modeling

How you decipher and design each subdomain



- Interaction with domain experts
- Model a single subdomain at a time
- Implementation of subdomains

Separation of Concerns

plays an important role in implementing subdomains

Exploring the Benefits and Potential Drawbacks of DDD

Benefits of Domain-Driven Design

Flexible

Customer's vision/perspective of the problem

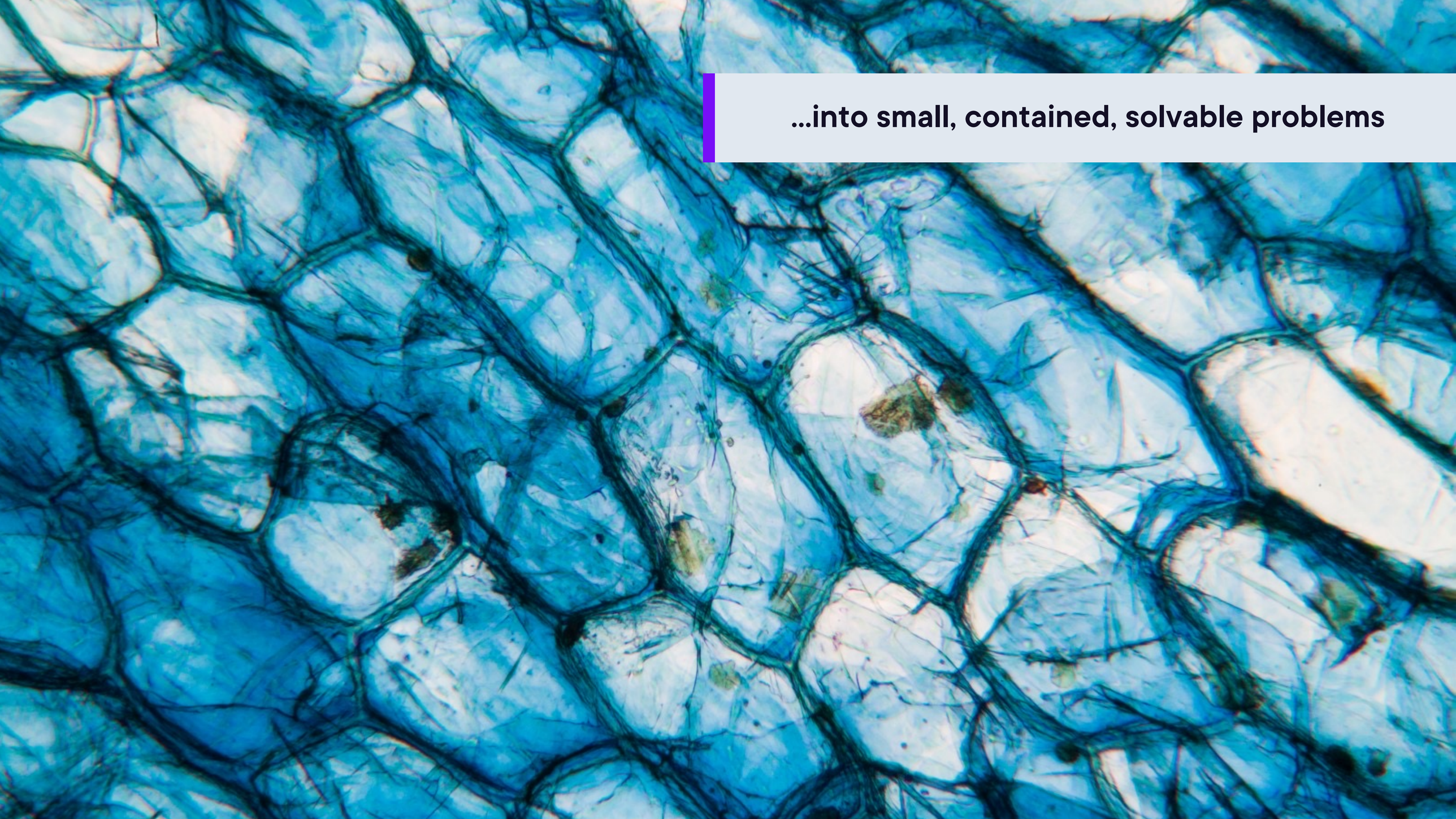
Path through a very complex problem

Well-organized and easily tested code

Business logic lives in one place

Many great patterns to leverage

Transform big, messy problems...



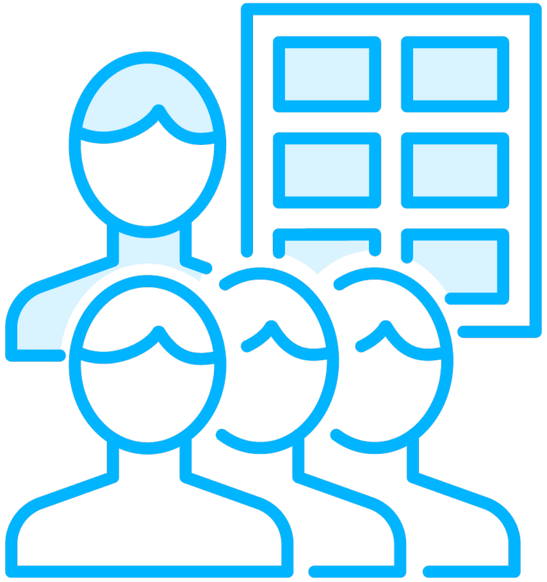
...into small, contained, solvable problems

DDD aims to
tackle business complexity,
not technical complexity.

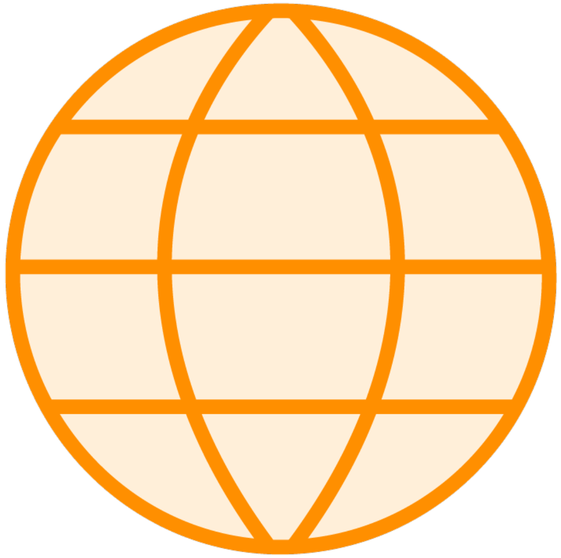
Eric Evans, Domain-Driven Design

“While Domain-Driven Design provides many technical benefits, such as maintainability, it should be applied only to complex domains where the model and the linguistic processes provide clear benefits in the communication of complex information, and in the formulation of a common understanding of the domain.”

Be Prepared for Time and Effort

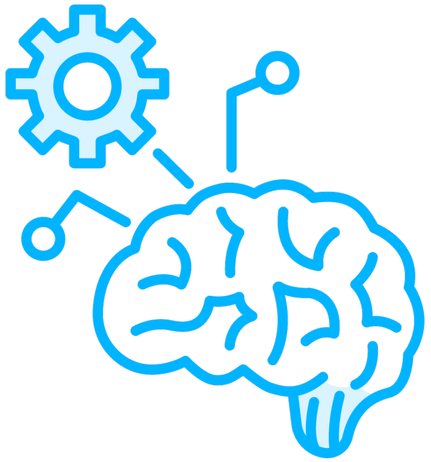


Discuss and model the problem with domain experts

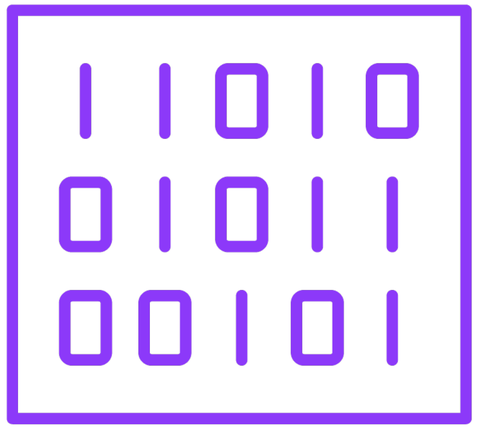


Isolate domain logic from other parts of application

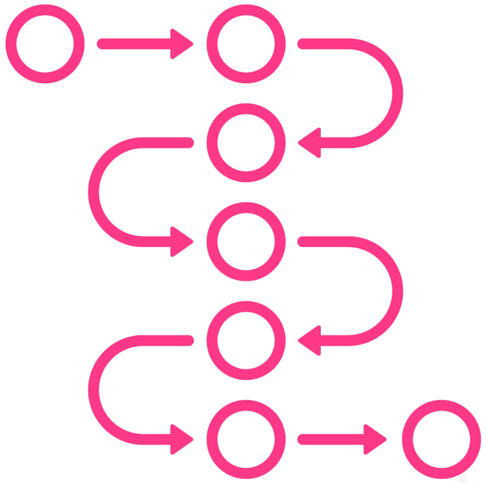
Be Prepared for a Learning Curve



New principles



New patterns



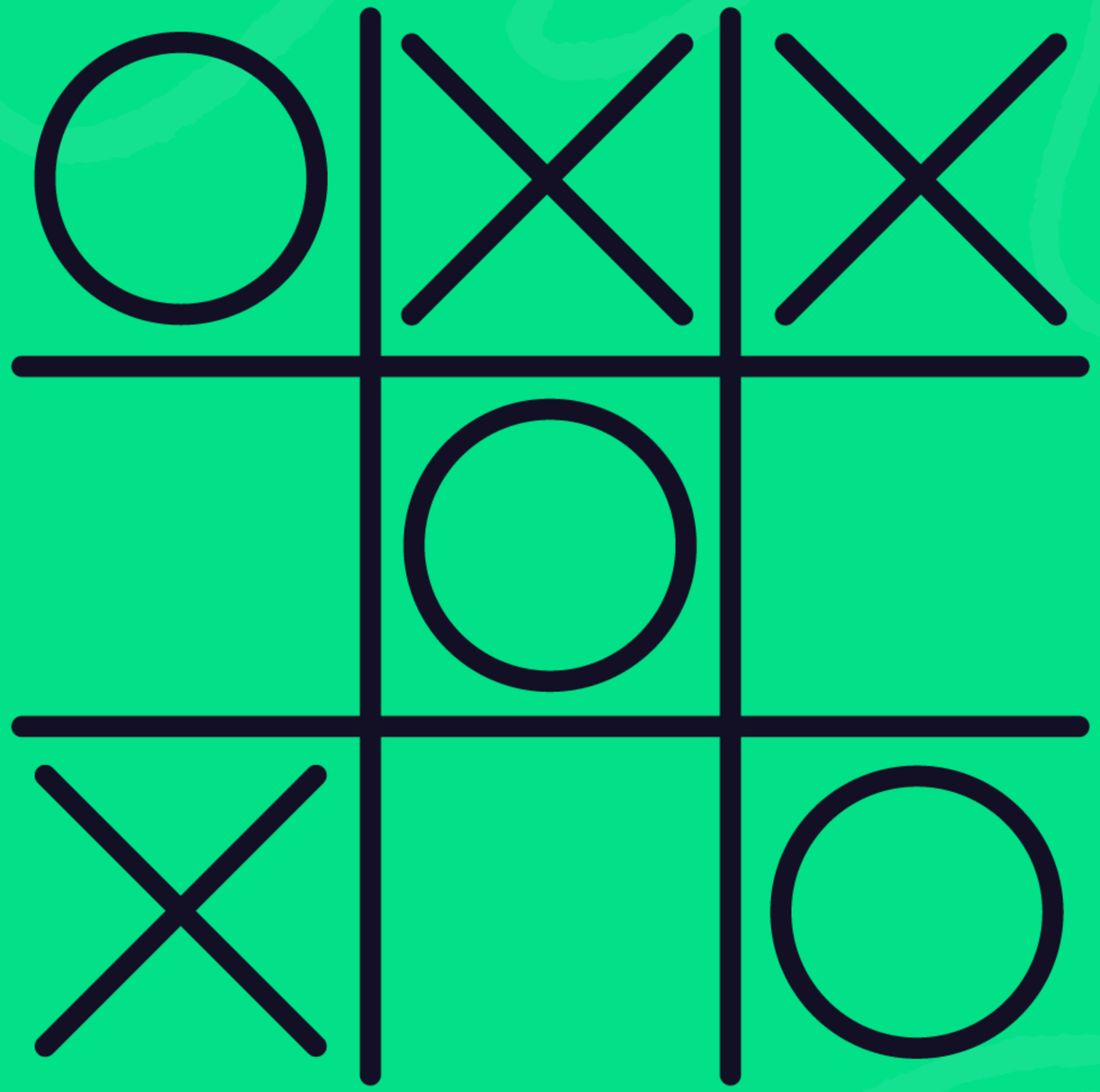
New processes

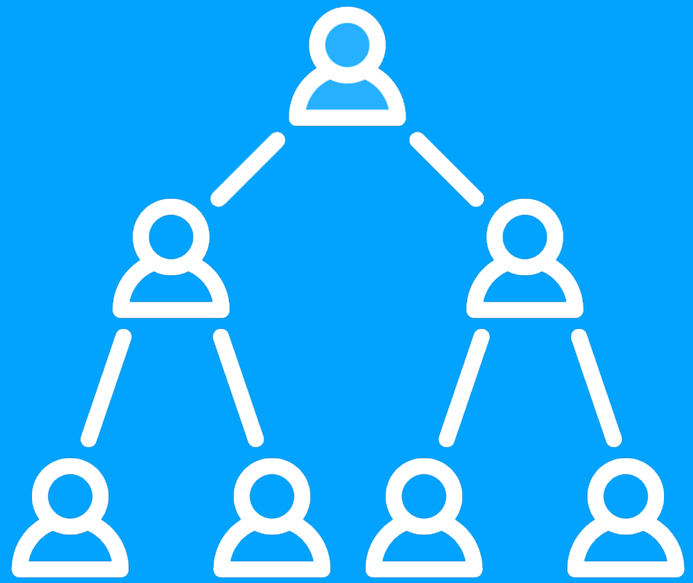
Be Thoughtful About Possible Overuse

DDD is for handling complexity in business problems

Not just CRUD or data-driven applications

Not just technical complexity without business domain complexity



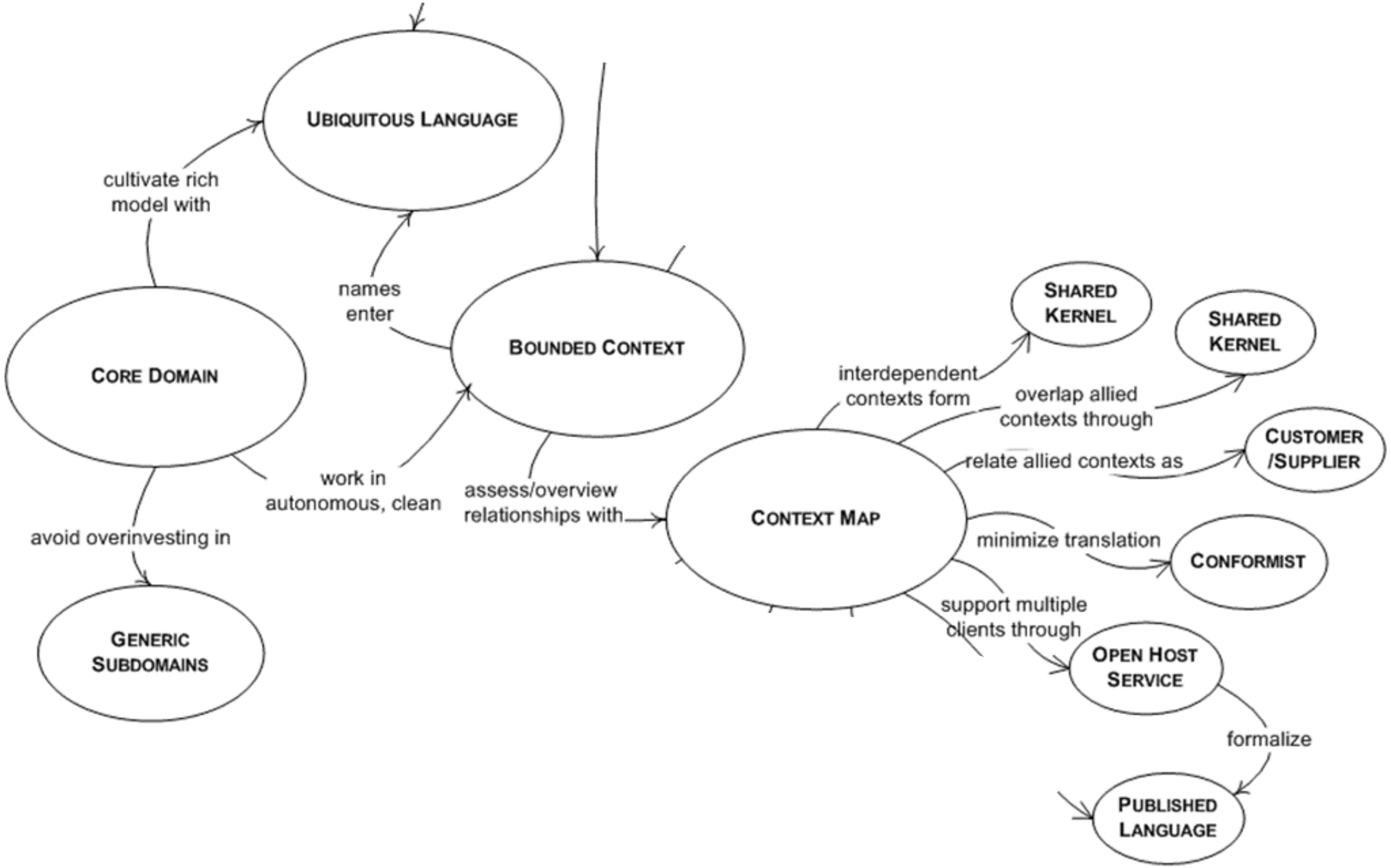


Requires Team and Business Buy-In

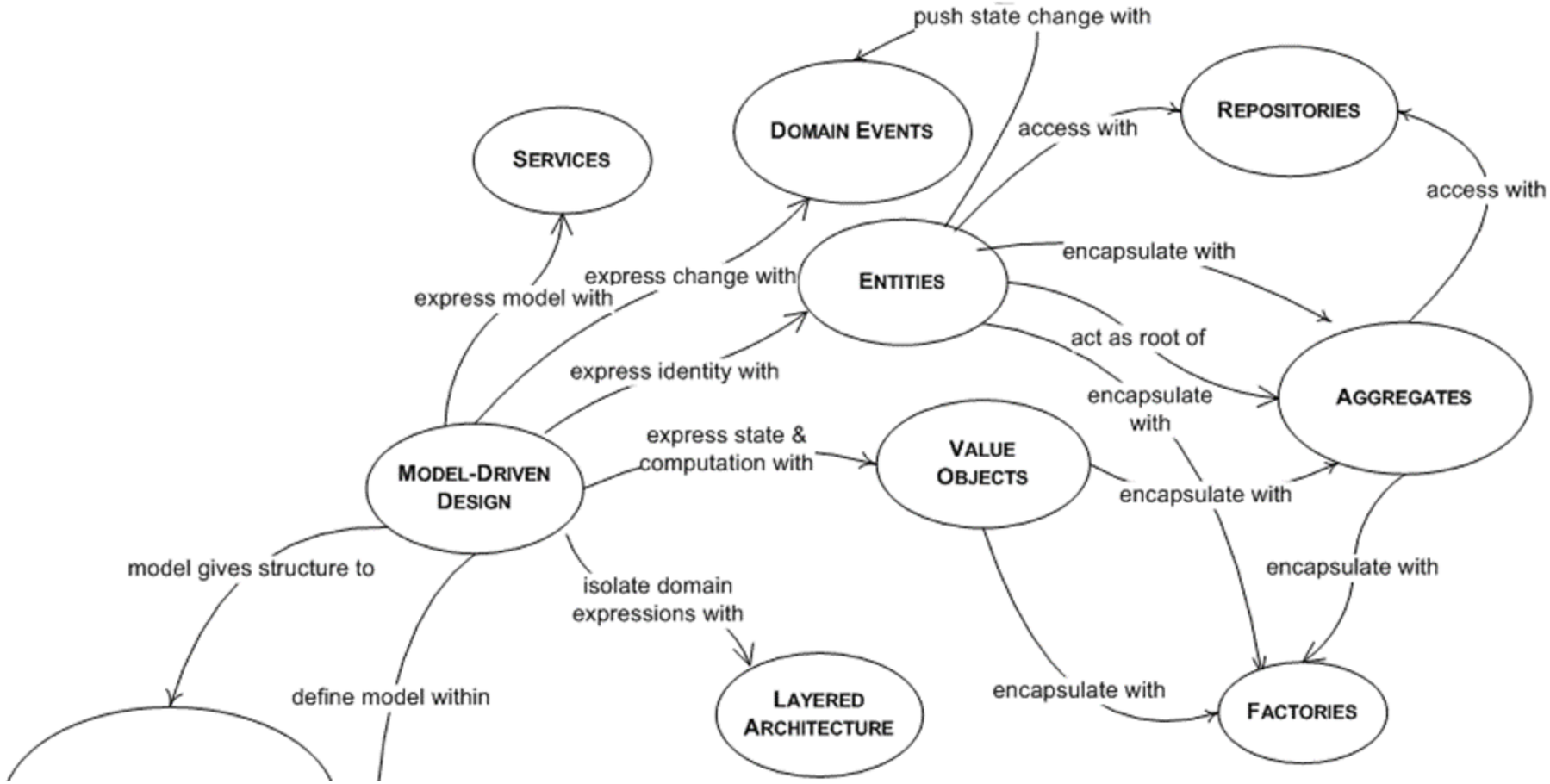
Inspecting a Mind Map of Domain-Driven Design

DDD Mind Map

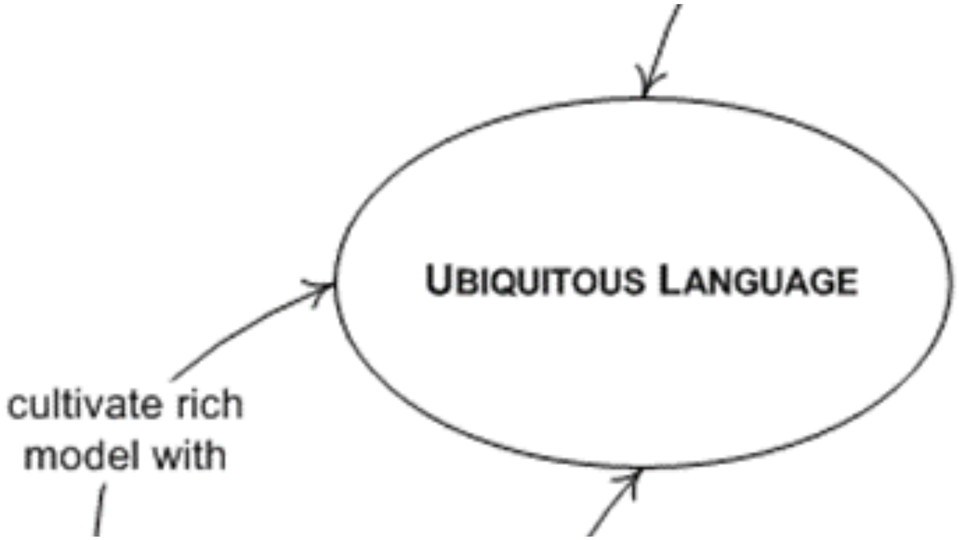
DDD Mind Map: Modeling



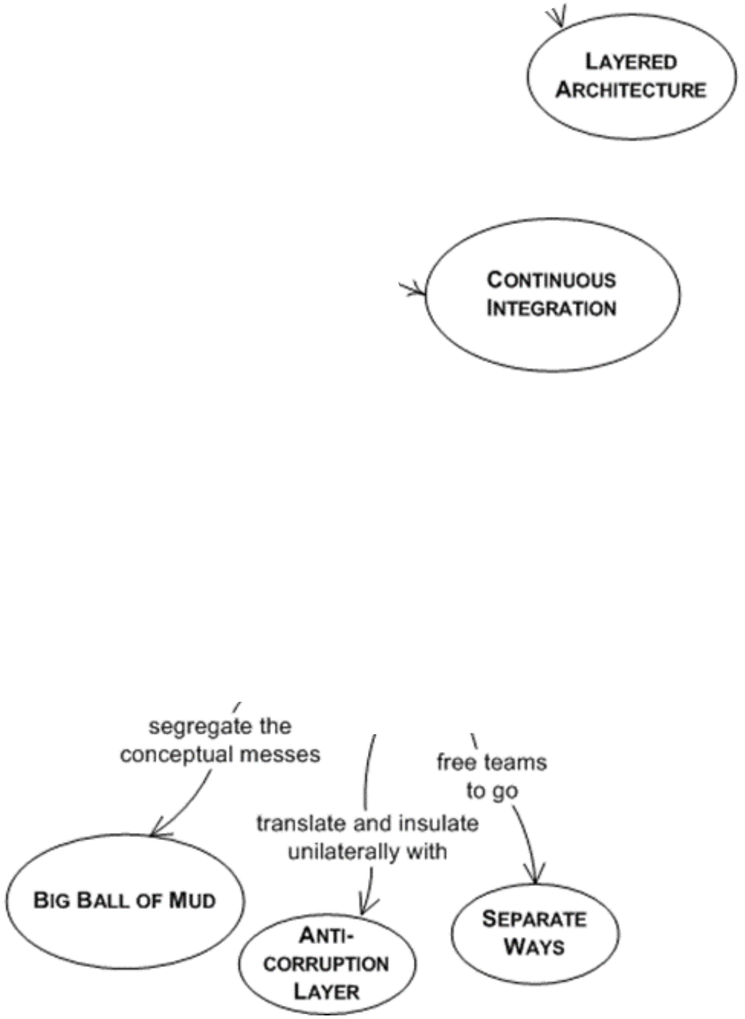
DDD Mind Map: Software Implementation



DDD Mind Map: Communication



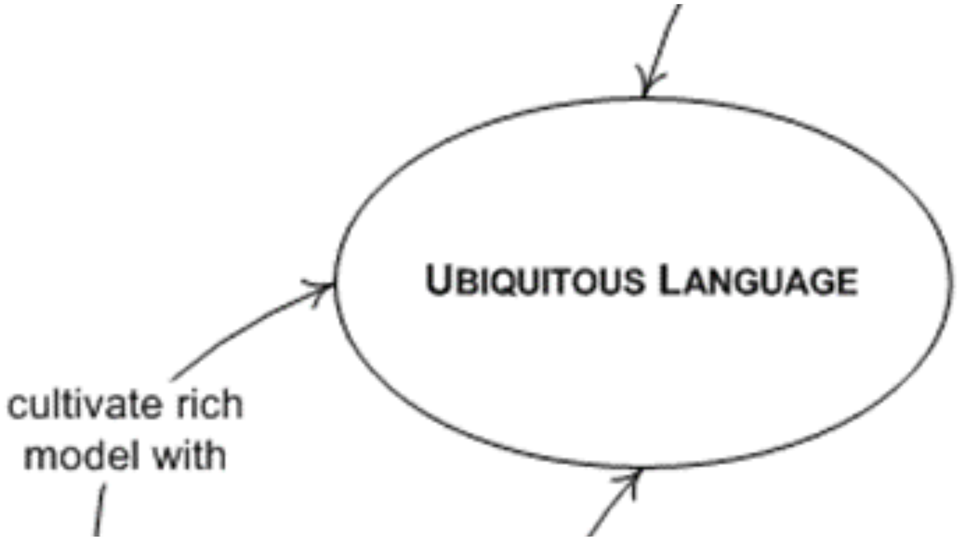
DDD Mind Map: Development Process



DDD Mind Map


push state change with
DOMAIN EVENTS
access with
REPOSITORIES
access with
SERVICES
express model with
express change with
ENTITIES
encapsulate with
act as root of
express identity with
encapsulate with
AGGREGATES
express state & computation with
MODEL-DRIVEN DESIGN
VALUE OBJECTS
encapsulate with
model gives structure to
isolate domain expressions with
encapsulate with
define model within
LAYERED ARCHITECTURE
encapsulate with
FACTORIES
UBIQUITOUS LANGUAGE
cultivate rich model with
keep model unified by
CONTINUOUS INTEGRATION
names enter
SHARED KERNEL
SHARED KERNEL
BOUNDED CONTEXT
CORE DOMAIN
interdependent contexts form
overlap allied contexts through
CUSTOMER /SUPPLIER
relate allied contexts as
work in autonomous, clean
assess/overview relationships with
CONTEXT MAP
minimize translation
CONFORMIST
avoid overinvesting in
support multiple clients through
segregate the conceptual messes
free teams to go
OPEN HOST SERVICE
GENERIC SUBDOMAINS
translate and insulate unilaterally with
loosely couple contexts through
formalize
BIG BALL OF MUD
ANTI-CORRUPTION LAYER
SEPARATE WAYS
PUBLISHED LANGUAGE

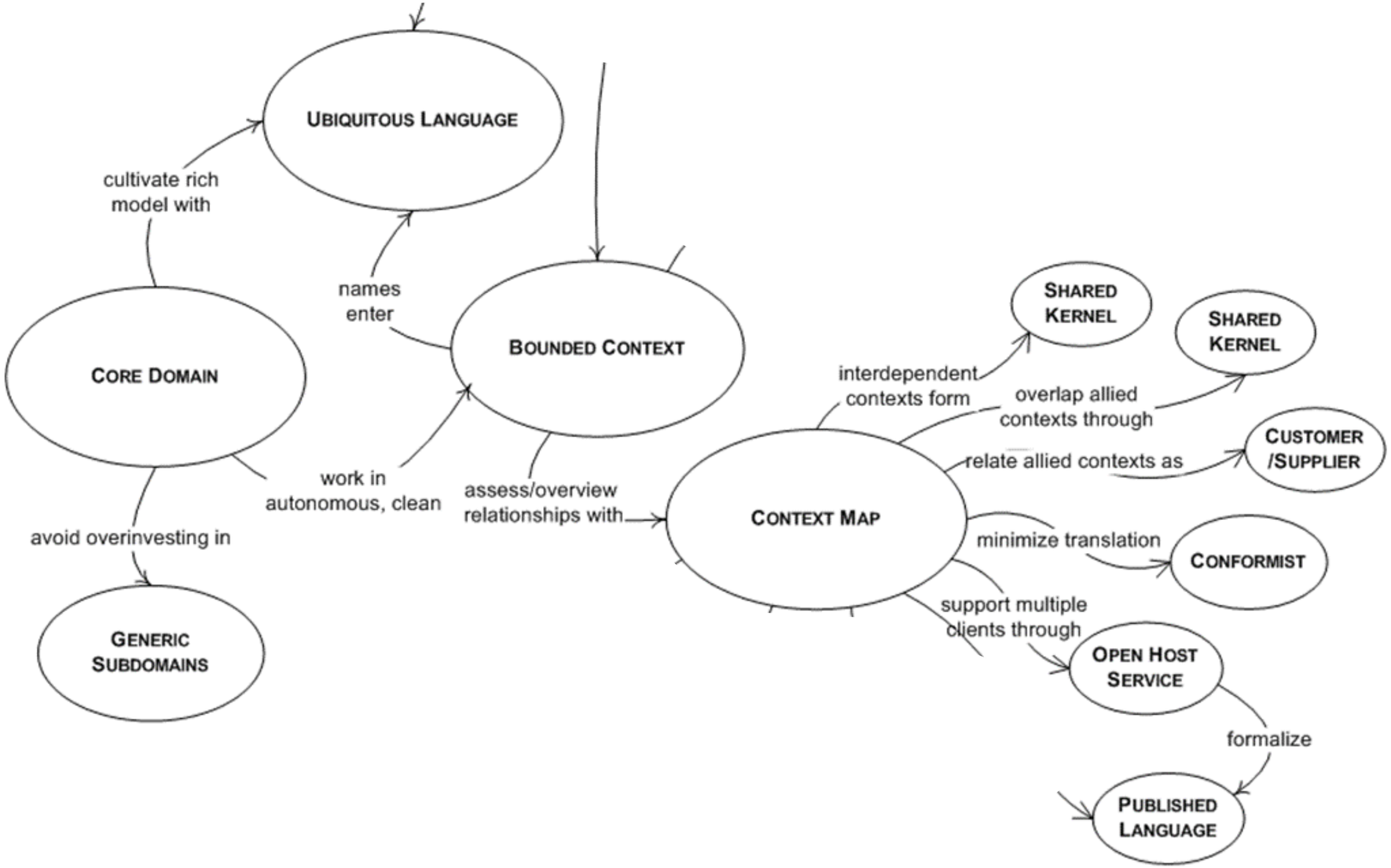
DDD Mind Map: Communication



DDD Mind Map


push state change with
DOMAIN EVENTS
access with
REPOSITORIES
access with
SERVICES
express model with
express change with
ENTITIES
encapsulate with
act as root of
express identity with
encapsulate with
AGGREGATES
express state & computation with
MODEL-DRIVEN DESIGN
VALUE OBJECTS
encapsulate with
model gives structure to
isolate domain expressions with
encapsulate with
define model within
LAYERED ARCHITECTURE
encapsulate with
FACTORIES
UBIQUITOUS LANGUAGE
cultivate rich model with
keep model unified by
CONTINUOUS INTEGRATION
names enter
SHARED KERNEL
SHARED KERNEL
BOUNDED CONTEXT
CORE DOMAIN
interdependent contexts form
overlap allied contexts through
CUSTOMER /SUPPLIER
relate allied contexts as
work in autonomous, clean
assess/overview relationships with
CONTEXT MAP
minimize translation
CONFORMIST
avoid overinvesting in
support multiple clients through
segregate the conceptual messes
free teams to go
OPEN HOST SERVICE
GENERIC SUBDOMAINS
translate and insulate unilaterally with
loosely couple contexts through
formalize
BIG BALL OF MUD
ANTI-CORRUPTION LAYER
SEPARATE WAYS
PUBLISHED LANGUAGE

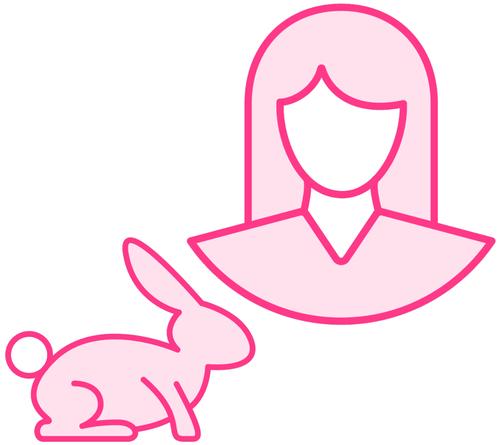
DDD Mind Map: Modeling



Introducing Our Sample Application

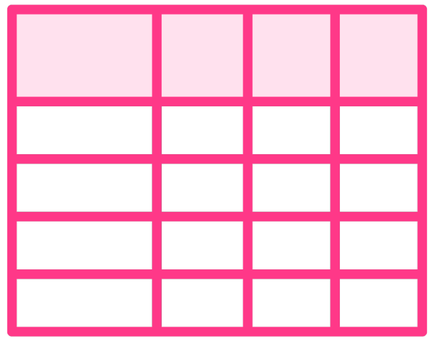
Some Logic Is Complex, Some Is Simple CRUD

Appointment scheduling is complex!



Manage client and patient data

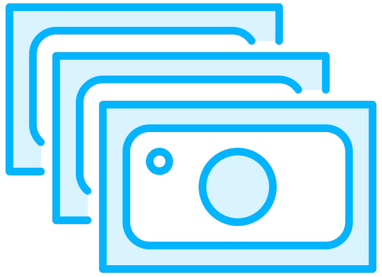
Manage staff and facilities



Other simple data management

The Software Does More Than Scheduling

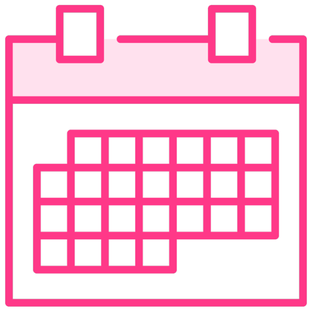
Staff



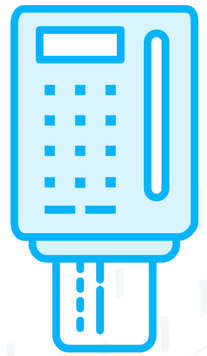
Accounting

Client and patient records

Visit records



Appointment scheduling



Sales

Exploring the Sample App’s High-level Structure

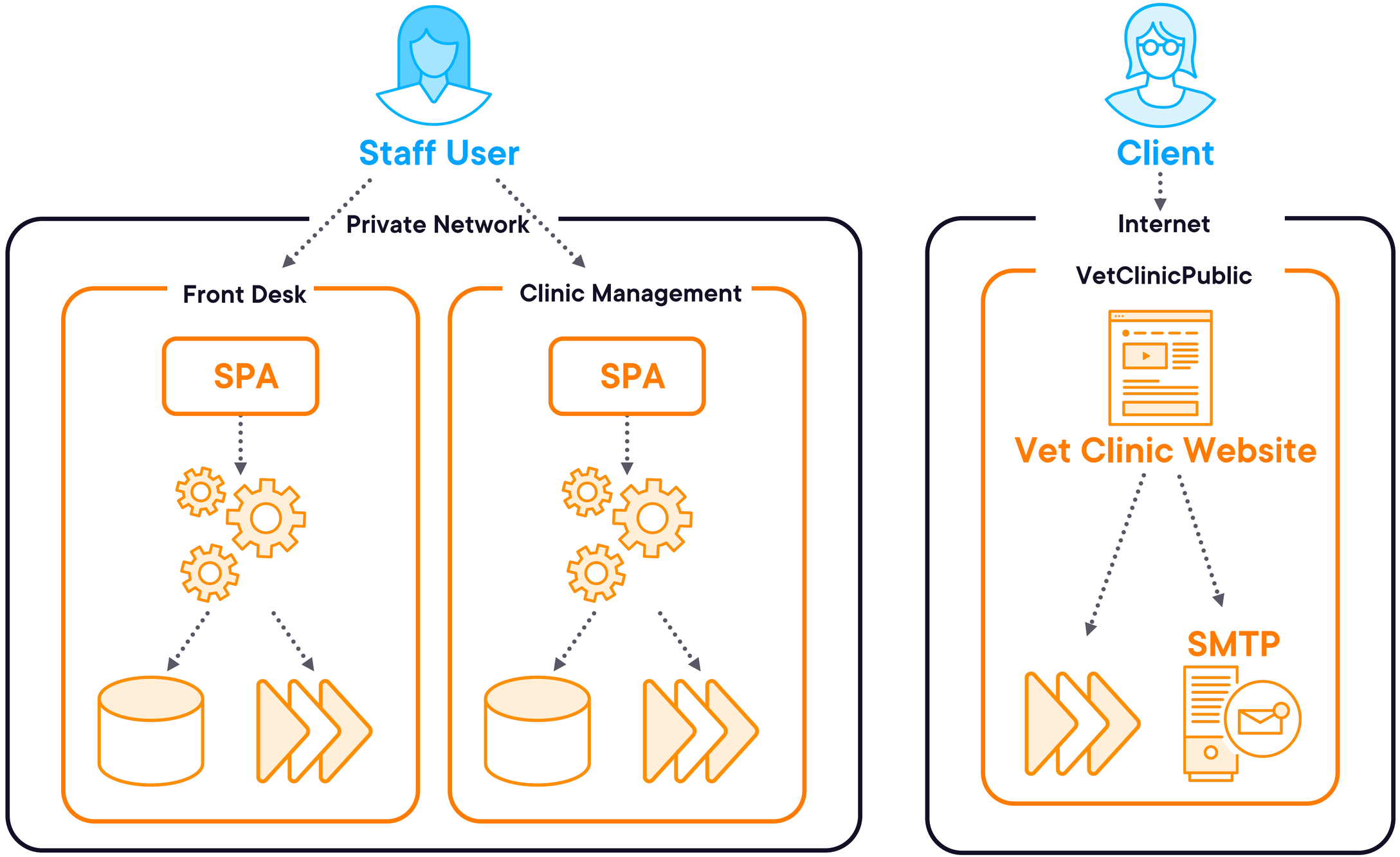
Our Distributed Web Apps

Front End

Single Page App (SPA)
using
Blazor Web Assembly

Back End

APIs
built with
ASP.NET Core


Staff User
Client
Private Network
Internet
Front Desk
Clinic Management
VetClinicPublic
SPA
SPA
Vet Clinic Website
SMTP

DDD Mind Map


push state change with
DOMAIN EVENTS
access with
REPOSITORIES
access with
SERVICES
express model with
express change with
ENTITIES
encapsulate with
act as root of
express identity with
encapsulate with
AGGREGATES
express state & computation with
MODEL-DRIVEN DESIGN
VALUE OBJECTS
encapsulate with
model gives structure to
isolate domain expressions with
encapsulate with
define model within
LAYERED ARCHITECTURE
encapsulate with
FACTORIES
UBIQUITOUS LANGUAGE
cultivate rich model with
keep model unified by
CONTINUOUS INTEGRATION
names enter
SHARED KERNEL
SHARED KERNEL
BOUNDED CONTEXT
CORE DOMAIN
interdependent contexts form
overlap allied contexts through
CUSTOMER /SUPPLIER
relate allied contexts as
work in autonomous, clean
assess/overview relationships with
CONTEXT MAP
minimize translation
CONFORMIST
avoid overinvesting in
support multiple clients through
segregate the conceptual messes
free teams to go
OPEN HOST SERVICE
GENERIC SUBDOMAINS
translate and insulate unilaterally with
loosely couple contexts through
formalize
BIG BALL OF MUD
ANTI-CORRUPTION LAYER
SEPARATE WAYS
PUBLISHED LANGUAGE

Key Takeaways

Solving problems, not just coding

Collaborate with domain experts

Break domain into smaller units

Efficient and effective path to success

Teamwork!

Let’s start exploring the process of discovering and modeling domains.

Resources Referenced in this Module

Domain-Driven Design, Eric Evans, 2003, Addison-Wesley Professional
amzn.to/3k7g948

Patterns, Principles, and Practices of Domain-Driven Design
Scott Millett, Nick Tune, 2015, Wrox https://a.co/d/2Kfska2

Domain-Driven Design, The First 15 Years, Essays from the community
leanpub.com/ddd_first_15_years

VirtualDDD Meetup with recorded sessions: virtualddd.com

Resources Referenced in this Module

Getting to DDD: Pragmatic or Principled
Julie Lerman, KanDDDinsky 2018
youtu.be/CPd2BVR3hEM

Pluralsight DDD Fundamentals Sample for this course
github.com/ardalis/pluralsight-ddd-fundamentals