Aggregate Design Canvas

⚡ Name

Todo list

🎯 Description

> A generic todolist business object, filled with todo items collection & description tags collection

🎯 State transition

```mermaid
flowchart LR
Created --> Completed
```

📈 Enforced invariant

- if todo list have items and all of this items are marked as completed, so the todo list item is completed too

📈 Corrective policies

- only creator of todolist can update (manage tags & items or delete it include)

📈 Handled commands

- Create todo list
- Update todo list
- Remove todo list
- Add todo item
- Remove todo item
- Toggle todo item

📈 Created events

- On todolist completed

📈 Throughput

Use cases	AVG	MAX
Command handling rate	100/d	400/d
Total number of client	15	lorem
Concurrency conflict chance	5%	10%

📈 Size

Use cases	AVG	MAX
Event growth rate	10%/m	10%/m

Use cases	AVG	MAX
Lifetime of single instance	1m	4m
Number of events persisted	20/d	100/d

Additional resources

NAME			STATE TRANSITIONS	
1				
DESCRIPTION				
2			3	
THROUGHPUT	AVG	MAX	ENFORCED INVARIANTS	CORRECTIVE POLICIES
COMMAND HANDLING RATE				
TOTAL NUMBER OF CLIENTS				
CONCURRENCY CONFLICT CHANCE		8	4	5
SIZE	AVG	MAX	HANDLED COMMANDS	CREATED EVENTS
EVENT GROWTH RATE				
LIFETIME OF A SINGLE INSTANCE				
NUMBER OF EVENTS PERSISTED		9	6	7

AGGREGATE DESIGN CANVAS V1 | HTTPS://GITHUB.COM/DDD-CREW/AGGREGATE-DESIGN-CANVAS

Bounded Context Canvas

⚡ Name

TodoApp

🎯 Purpose

> Give a generic action plan / todo app capabilities through api

📈 Strategic classification

```mermaid
quadrantChart
    title Strategize bounded context
    x-axis Low business differentiation --> High business differentiation
    y-axis Low model complexity --> High model complexity
    quadrant-1 CORE
    quadrant-2 GENERIC to SUPPORTING
    quadrant-3 GENERIC to SUPPORTING
    quadrant-4 SUPPORTING
    Todo service: [0.3, 0.2]
```

Domain

- [ ] core
- [ ] supporting

- [x] generic
- [ ] other

Business model

- [ ] revenue
- [ ] compliance
- [ ] engagement
- [x] cost reduction

Evolution

- [ ] genesis
- [x] custom build
- [ ] product
- [ ] commodity

💬 Ubiquitous language

> A "todo list" is a managed list of "todo items" with a label that can be marked as "completed" by domain user, and a managed list of tags

---

🖋️ Business decisions & policies

- only creator of todolist can update (manage tags & items or delete it
- if todo list have items and all of this items are marked as completed, so the todo list item is completed too

---

📩 Inbound communication

Queries

- Get todo lists by user
- Get todo list by id

Commands

- Create todo list
- Update todo list
- Remove todo list
- Add todo item
- Remove todo item
- Toggle todo item

Events

- On todolist completed

📤 Outbound communication

Queries

Commands

Events

- Publish "OnTodolistCompleted" event on bus

!? Assumptions

> Can envolve in to adding users (owners) management

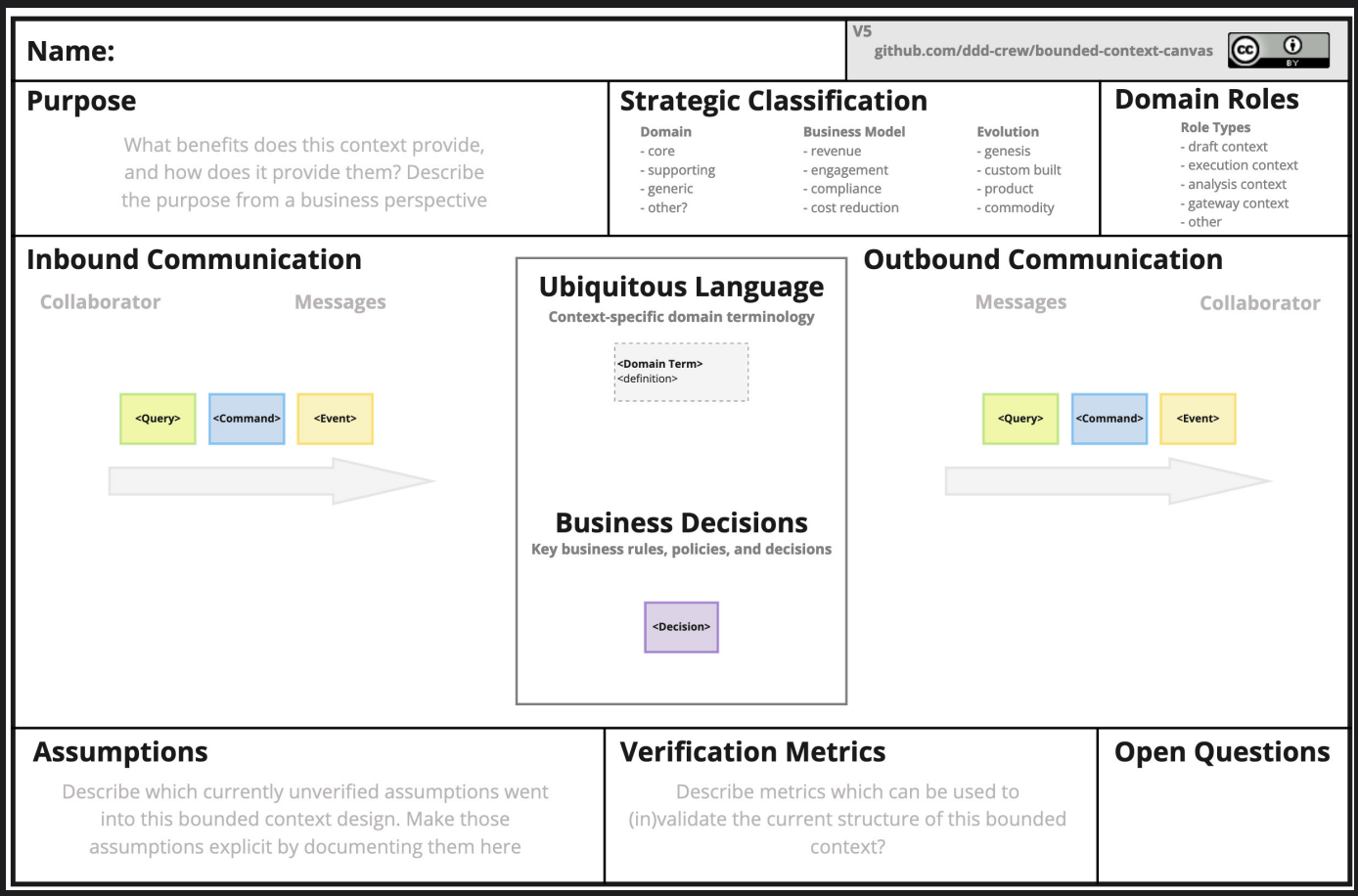
📊 Verification metrics

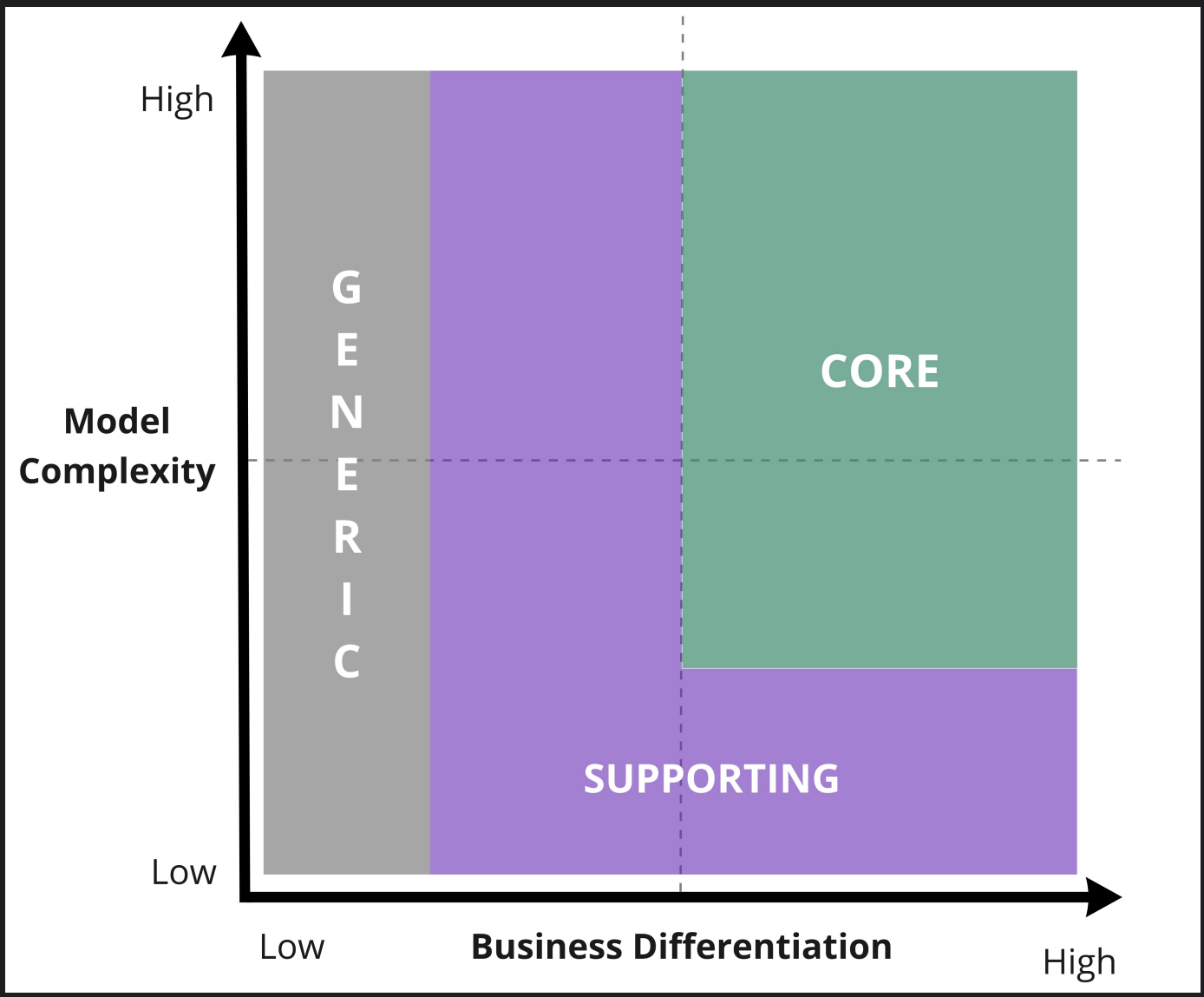
- lorem

? Open questions

- lorem

Additional resources




High
Model Complexity
Low
GENERIC
CORE
SUPPORTING
Low
Business Differentiation
High

Solution Bounded contexts

- Service : Todo app
  - Aggregate : TodoList

Event storming notes

- Todo app

TodoApp Event storming

Todo list created

mermaid

```
graph TD;

classDef aggregate fill:#fdfd9d
classDef command fill:#45abef
classDef readModel fill:#77dd77
classDef event fill:#ffb853
classDef policy fill:#c14bc0
classDef external fill:#f8b1f5
classDef actor fill:transparent

Actor[User]:::actor --> Command:::command
Command[Create todolist]:::command --> Aggregate:::aggregate
Aggregate[Todolist]:::aggregate --> Event:::event
Event[Todolist created]:::event --> ReadModel:::readModel
ReadModel[Todolist read model]:::readModel --> Actor:::actor
```

Todo list updated

mermaid

```
graph TD;

classDef aggregate fill:#fdfd9d
classDef command fill:#45abef
classDef readModel fill:#77dd77
classDef event fill:#ffb853
```

```
classDef policy fill:#c14bc0
classDef external fill:#f8b1f5
classDef actor fill:transparent

Actor[User]:::actor --> Command:::command
Command[Update todolist]:::command --> Aggregate:::aggregate
Aggregate[Todolist]:::aggregate --> Event:::event
Event[Todolist updated]:::event --> ReadModel:::readModel
Event:::event -- only if --> Policy[Created by requirement]:::policy
ReadModel[Todolist read model]:::readModel --> Actor:::actor
```

Todo item added

mermaid

```
graph TD;

classDef aggregate fill:#fdfd9d
classDef command fill:#45abef
classDef readModel fill:#77dd77
classDef event fill:#ffb853
classDef policy fill:#c14bc0
classDef external fill:#f8b1f5
classDef actor fill:transparent

Actor[User]:::actor --> Command:::command
Command[Add item]:::command --> Aggregate:::aggregate
Aggregate[Todolist]:::aggregate --> Event:::event
Event[Todo item added]:::event --> ReadModel:::readModel
Event:::event -- only if --> Policy[Created by requirement]:::policy
ReadModel[Todolist complete read model]:::readModel --> Actor:::actor
```

Todo item removed

mermaid

```
graph TD;
```

```
classDef aggregate fill:#fdfd9d
classDef command fill:#45abef
classDef readModel fill:#77dd77
classDef event fill:#ffb853
classDef policy fill:#c14bc0
classDef external fill:#f8b1f5
classDef actor fill:transparent

Actor[User]:::actor --> Command:::command
Command[Remove todo item]:::command --> Aggregate:::aggregate
Aggregate[Todolist]:::aggregate --> Event:::event
Event[Todolist completed]:::event --> ReadModel:::readModel
Event:::event -- only if --> Policy[Created by requirement]:::policy
Event:::event -- only if --> Policy2[all items completed]:::policy
Event:::event --> External[Send mail]:::external
ReadModel[Todolist complete read model]:::readModel --> Actor:::actor
```

Toggle todo item

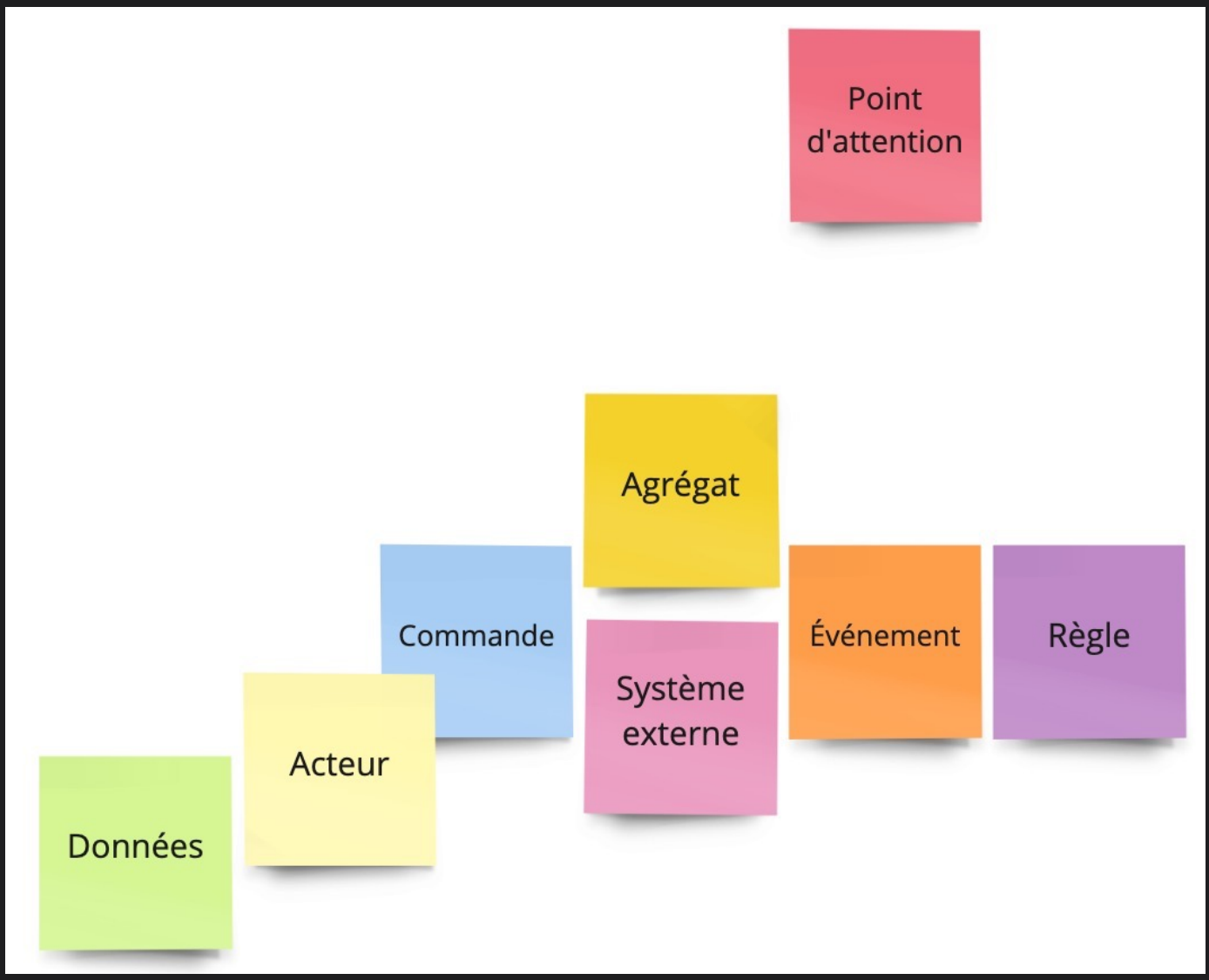
mermaid

```
graph TD;

classDef aggregate fill:#fdfd9d
classDef command fill:#45abef
classDef readModel fill:#77dd77
classDef event fill:#ffb853
classDef policy fill:#c14bc0
classDef external fill:#f8b1f5
classDef actor fill:transparent

Actor[User]:::actor --> Command:::command
Command[Toggle todo item]:::command --> Aggregate:::aggregate
Aggregate[Todolist]:::aggregate --> Event:::event
Event[Todolist completed]:::event --> ReadModel:::readModel
Event:::event -- only if --> Policy[Created by requirement]:::policy
Event:::event -- only if --> Policy2[all items completed]:::policy
Event:::event --> External[Send mail]:::external
ReadModel[Todolist complete read model]:::readModel --> Actor:::actor
```

Additional resources



Solution architecture schema

> Add your own architecture schema

mermaid

```
flowchart TB

subgraph DomainUser[Domain user]
        direction LR
        h1[-Person-]:::type
        d1[A generic internal domain user]:::description
end
DomainUser:::person

subgraph TodoApp[Todo application]
    subgraph Client[Todo web client]
        subgraph pwa[Progressive Web App]
            direction LR
            h2[Container: ASP .NET 7 Blazor]:::type
            d2[Web client for todoo application and others]:::description
        end
        pwa:::internalContainer

        subgraph sdk[SDK]
            direction LR
            h12[Container: Typescript / dotnet]:::type
            d12[HTTP SDK for GraphQL / REST protocols]:::description
        end
        sdk:::internalContainer

        subgraph bff[Backend for frontend]
            direction LR
            h3[Container: ASP .NET 7 and YARP]:::type
            d3[Backend for frontend hosting blazor PWA and routing API Services]:::de
        end
        bff:::internalContainer
    end
```

```
    Client:::groupInternalContainer

    subgraph TodoAPI[Todo microservice]
        direction LR
        h4[Container: ASP .NET 7]:::type
        d4[Todo app microservice exposing API]:::description
    end
    TodoAPI:::internalContainer

    subgraph TodoData[Todo service database]
        direction LR
        h5[Container: postgres]:::type
        d5[Relational database]:::description
    end
    TodoData:::database

    subgraph OtherAPI[microservice x]
        direction LR
        h6[Container: xxx]:::type
        d6[xxx microservice exposing API]:::description
    end
    OtherAPI:::internalContainer

    subgraph OtherData[Other service database]
        direction LR
        h7[Container: xxx]:::type
        d7[Relational xxx]:::description
    end
    OtherData:::database

    subgraph emailSystem[Email System]
        h99[-Software System-]:::type
        d99[The external email system provided by xxx]:::description
    end
    emailSystem:::externalSystem

    subgraph serviceBusSystem[Service Bus System]
        h10[-Software System-]:::type
        d10[The external service bus system provided by xxx]:::description
    end
    serviceBusSystem:::externalSystem
```

```
end
TodoApp:::internalComponent

subgraph identityAccessManagement[Identity & Access Management]
    h11[-Software System-]:::type
    d11[The IAM system provided by xxx]:::description
end
identityAccessManagement:::externalSystem

DomainUser--Use-->TodoApp
TodoAPI--Use-->emailSystem
TodoAPI--Use-->serviceBusSystem
pwa--use-->sdk
bff--Host blazor PWA-->pwa
bff--proxying microservices-->TodoAPI
bff--proxying microservices-->OtherAPI
TodoAPI--Use-->TodoData
OtherAPI--Use-->OtherData
TodoApp--Use-->identityAccessManagement

classDef application fill:#26A69A
classDef person fill:#01579B, color:#ffffff, stroke: #ffffff
classDef internalContainer fill:#006064, color:#ffffff, stroke: #ffffff
classDef groupInternalContainer fill:#00838F, color:#ffffff, stroke: #ffffff
classDef internalComponent fill:#ededed, stroke: #ffffff
classDef externalSystem fill:#607D8B, color:#ffffff, stroke: #ffffff
classDef database fill:#757575, color:#ffffff, stroke: #ffffff
```

Getting started

Start with cloning the repository :

```
git clone https://github.com/bhtz/microscope-boilerplate.git
```

Run solution :

```
docker-compose up
```

Development

Install dependencies :

```
dotnet restore
```

Build solution :

```
dotnet build
```

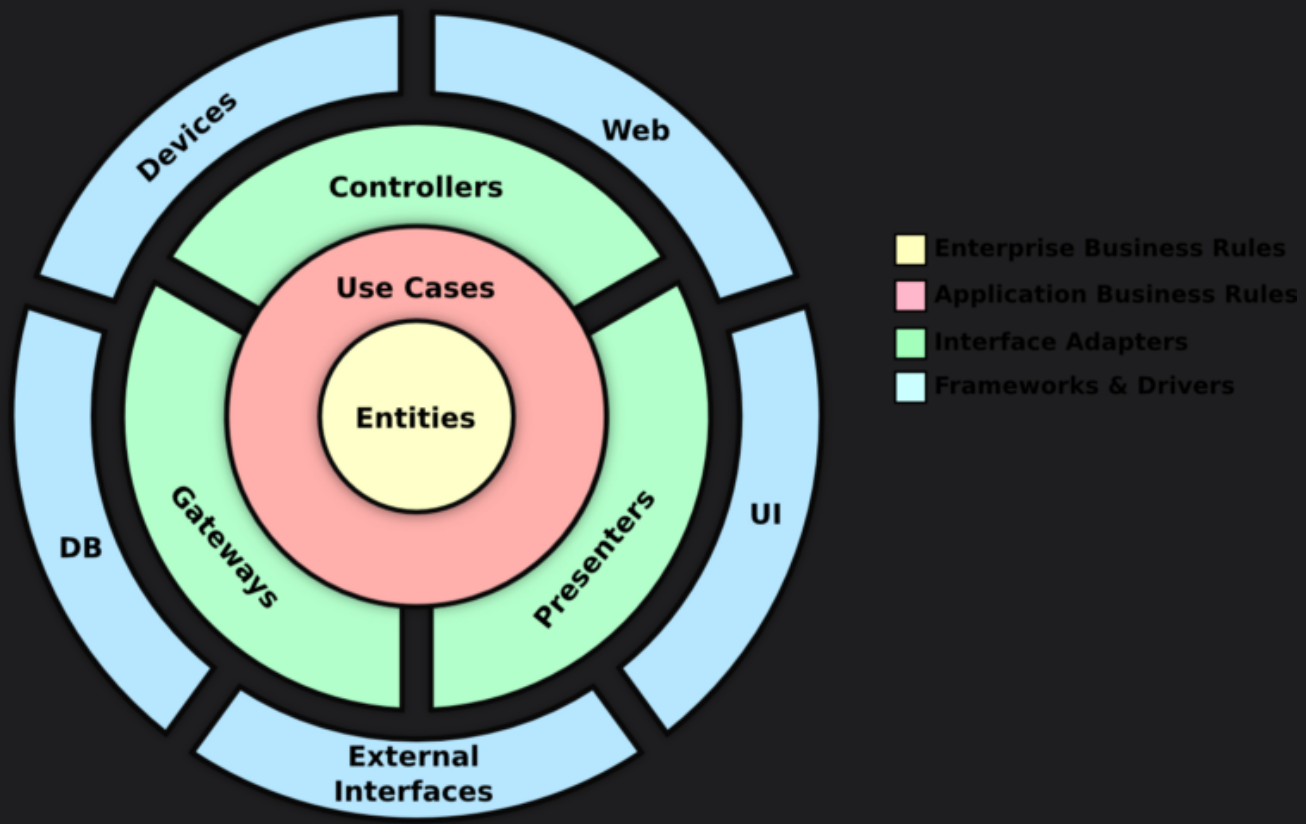
Solution structure

Services

> Each service is a bounded context implemented by layered architecture solution

Clean architecture concept

This repository follow the Clean / Hexagonal architecture pattern :



Domain Layer:

> The Domain Layer is the core and most critical layer in Domain-Driven Design. It contains the heart of the business logic and represents the business domain. This layer focuses on encapsulating the business rules, behaviors, and concepts of the application. It consists of the following components:

- **Domain Entities:** Objects representing the core business concepts.
- **Value Objects:** Immutable objects representing descriptive aspects of the domain.
- **Aggregates:** Clusters of related objects treated as a single unit of consistency and transactional boundaries.
- **Domain Services:** Encapsulate domain logic that doesn't naturally fit within a single entity.
- **Repositories:** Provide an abstraction over data storage, allowing the application to interact with the persistence mechanism without knowing the underlying implementation.

Application Layer:

> The Application Layer sits on top of the Domain Layer and acts as an orchestrator of domain objects to fulfill user requests. It is responsible for coordinating the application's business workflows and handling user interactions. This layer contains application services that provide coarse-grained use cases, often mapped one-to-one with the use case requirements defined by the business. The application services collaborate with domain entities, value objects, and domain services from the Domain Layer to execute business logic, enforce business rules, and perform transactions.

- **Mappings:**
- **Behaviour:**
- **Policies:**
- **Features:**
- **Application services:**

Infrastructure Layer:

> The Infrastructure Layer is responsible for providing technical implementations that support the Domain and Application Layers. It contains components that handle concerns like data persistence, messaging, caching, external integrations, and any other

> infrastructure-related services. Examples of infrastructure components are databases, data access repositories, message queues, email services, file storage systems, etc. This layer abstracts the underlying technologies from the rest of the application, allowing easy replacement or adaptation of technical components without impacting the core domain logic.

- **Persistence configurations:**
- **Repository implementation:**
- **Migration scripts:**
- **External services implementation:**

Interface Layer:

> The Interface Layer is the outermost layer of the application and is responsible for handling communication with external systems and users. It provides various interfaces to interact with the application, such as user interfaces (web UI, mobile app UI), APIs (RESTful, GraphQL), command-line interfaces (CLI), and messaging interfaces. The Interface Layer translates user input into application requests and presents the application's output to users or external systems. This layer should be as thin as possible, delegating most of the business logic and processing to the Domain and Application Layers.

- **Services configurations:**
- **REST API:**
- **GraphQL API:**
- **Custom middlewares:**
- **Interface services implementation:**

Building blocks

IAC

ESB

IAM

APIM

Reverse proxy

Clients

SDK

> The SDK (Software Development Kit) layer represents a boundary between the clients layer and the services exposing REST and/or GraphQL APIs. Its primary objective is to encapsulate the complexities of interacting with these APIs, offering a simplified, uniform, and consistent interface to the clients (web, console, desktop, mobile, ...).

- TodoApp GraphQL SDK
- TodoApp REST SDK
  - Typescript
  - dotnet

Web

Web Application (SPA / PWA):

> The web application represents the user interface layer of the client-side. Whether it is a Single Page Application or a Progressive Web App, the focus is on delivering a rich and responsive user experience in the browser. The SPA architecture allows for dynamic content updates without reloading the entire page, while PWAs enable native-like capabilities such as offline access and push notifications.

- **Configuration:**
- **Pages:**
- **Shared components:**
- **Pages:**

- **Settings:**

Backend for Frontend (BFF):

> The Backend for Frontend is a design pattern where a dedicated backend service is created for each specific client application or type of client. It acts as an intermediary between the web application and the microservices, serving as a tailored API gateway for the client. The BFF pattern enables customizing data and functionalities to suit the exact requirements of the client, reducing unnecessary data transfers and minimizing the risk of over-fetching (aggregation call). By hosting the PWA, the BFF serves the static files and resources required for the client-side application to run in the user's browser. This hosting can be done using web servers or serverless infrastructure, depending on the application's needs and scalability requirements. Hosting the PWA on the BFF allows for better control over caching strategies, content delivery, and server-side configurations to enhance the application's performance and reliability. As the BFF interacts with various microservices to fetch data for the client application, it acts as a reverse proxy for the microservice API. This means that instead of the client directly communicating with the microservices, all API requests from the client are directed to the BFF, which then forwards those requests to the appropriate microservices.

- **Configuration:**

CLI

Mobile

Documentation

Documentation as code

> The "Documentation Layer" revolutionizes the way we approach software project documentation. By writing solution documentation as code using Markdown, incorporating Mermaid for visualizations, and using VitePress to generate a web documentation portal, developers can foster a culture of up-to-date, accessible, and

> engaging documentation that grows alongside the project. Embrace the power of code-driven documentation and unlock the true potential of your software projects.

Living documentation

Product Discovery

- Discovery discipline notes template

Architecture

- Getting started guidelines
- Event storming catalog
- Architecture schema
- Bounded context canvas
  - Aggregate canvas
- Technology matrix guidelines

Organization & Governance

- Governance process templates
  - Weekly meeting notes
  - Architecture Decision record
- Product engineering organization
  - product engineering squad template

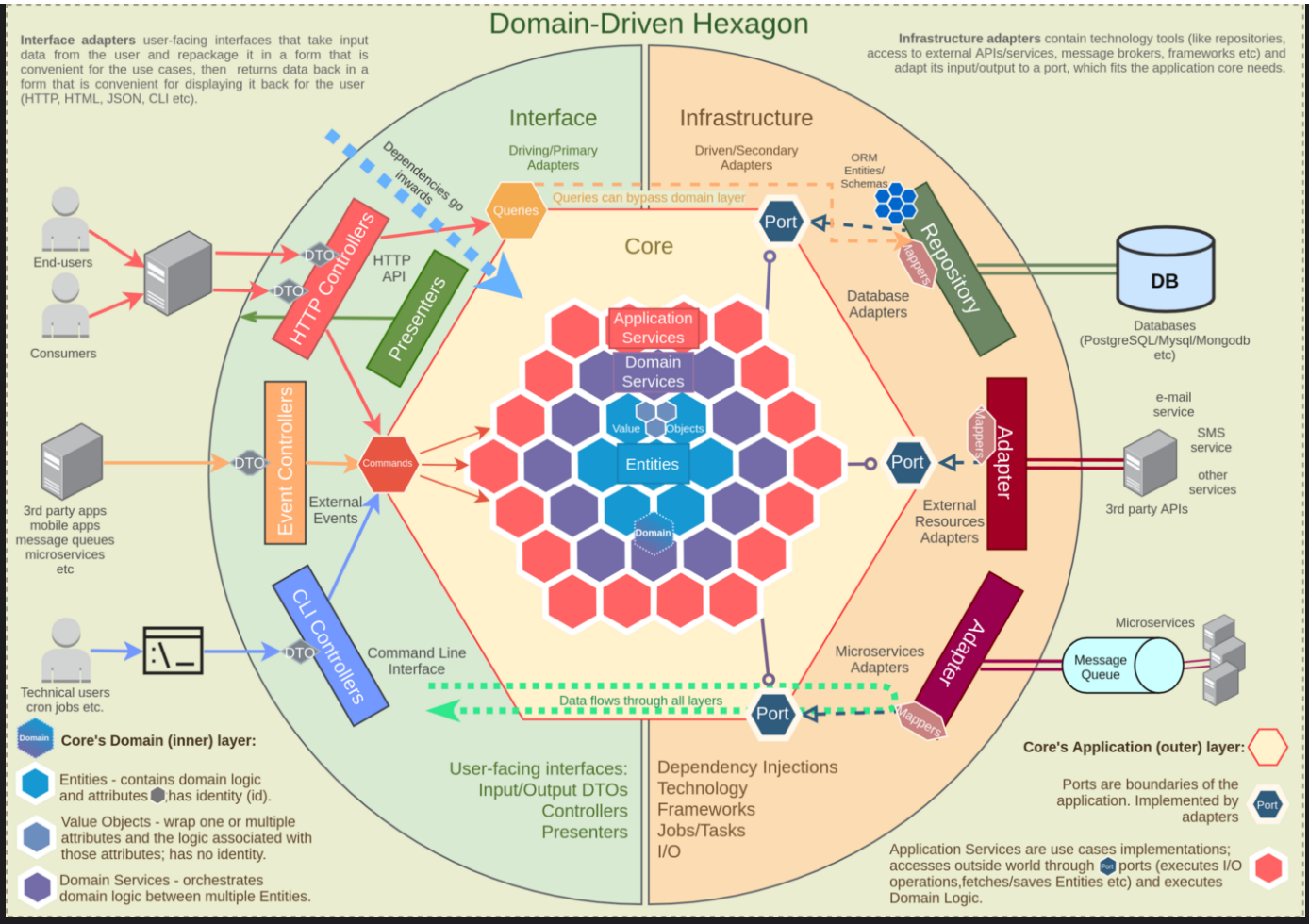
Guidelines

- Generic product engineering guidelines & must-read you want to provide to the teams

---

Additional resources

Domain driven hexagon


Domain-Driven Hexagon
Interface adapters user-facing interfaces that take input data from the user and repackage it in a form that is convenient for the use cases, then returns data back in a form that is convenient for displaying it back for the user (HTTP, HTML, JSON, CLI etc).
Infrastructure adapters contain technology tools (like repositories, access to external APIs/services, message brokers, frameworks etc) and adapt its input/output to a port, which fits the application core needs.
Interface
Infrastructure
Driving/Primary Adapters
Driven/Secondary Adapters
Dependencies go inwards
Queries
Queries can bypass domain layer
Port
ORM Entities/ Schemas
Repository
Mappers
DB
Databases (PostgreSQL/Mysql/Mongodb etc)
Database Adapters
End-users
Consumers
DTO
HTTP Controllers
HTTP API
Presenters
Core
Application Services
Domain Services
Value Objects
Entities
Domain
3rd party apps mobile apps message queues microservices etc
Event Controllers
Commands
External Events
Port
Adapter
External Resources Adapters
e-mail service
SMS service
other services
3rd party APIs
Technical users cron jobs etc.
CLI Controllers
Command Line Interface
Microservices Adapters
Microservices
Message Queue
Data flows through all layers
Core's Domain (inner) layer:
Entities - contains domain logic and attributes has identity (id).
Value Objects - wrap one or multiple attributes and the logic associated with those attributes; has no identity.
Domain Services - orchestrates domain logic between multiple Entities.
User-facing interfaces:
Input/Output DTOs
Controllers
Presenters
Dependency Injections
Technology
Frameworks
Jobs/Tasks
I/O
Core's Application (outer) layer:
Ports are boundaries of the application. Implemented by adapters
Application Services are use cases implementations; accesses outside world through ports (executes I/O operations,fetches/saves Entities etc) and executes Domain Logic.

Technology matrix

Use cases	Name	Actual Version	Target Version
Web API	asp net	7.x	x.x
Web frontend	asp net blazor 7.x	7.x	x.x
Relational Database	postgres	13	x.x
BFF	asp net + YARP	7.x	x.x
GraphQL	HotChocolate	13	x.x
Mediator pattern	Mediatr	11.x	x.x
Service Bus abstraction	MassTransit	x.x	x.x
Storage	Microscope.Storage	x.x	x.x
Identity & Access Management	Keycloak	18.x	x.x
Workflow engine	Elsa core	x.x	x.x
Scheduled jobs	Hangfire	x.x	x.x
xxxxxxxx	xxxxxxxx	x.x	x.x

Career path

🪜 Different ladders :

- **Developer:** role also known as programmer or software engineer, requires a deep level of technical expertise
- **Tech Lead:** role also known as dev lead, is the owner of the system and requires a unique balance between hands-on development, architecture knowledge and production support
- **Technical Program Manager:** role responsible for coordinating and driving to completion initiatives that span multiple teams
- **Engineering Manager:** role also known as dev manager, is responsible for the consistent delivery, career growth and level of happiness of the team

📊 Levels

Technology

- **Adopts:** actively learns and adopts the technology and tools defined by the team
- **Specializes:** is the go-to person for one or more technologies and takes initiative to learn new ones
- **Evangelizes:** researches, creates proofs of concept and introduces new technologies to the team
- **Masters:** has very deep knowledge about the whole technology stack of the system
- **Creates:** designs and creates new technologies that are widely used either by internal or external teams

System

- **Enhances:** successfully pushes new features and bug fixes to improve and extend the system
- **Designs:** designs and implements medium to large size features while reducing the system's tech debt
- **Owns:** owns the production operation and monitoring of the system and is aware of its SLAs
- **Evolves:** evolves the architecture to support future requirements and defines its SLAs
- **Leads:** leads the technical excellence of the system and creates plans to mitigate outages

People

- **Learns:** quickly learns from others and consistently steps up when it is required
- **Supports:** proactively supports other team members and helps them to be successful
- **Mentors:** mentors others to accelerate their career-growth and encourages them to participate
- **Coordinates:** coordinates team members providing effective feedback and moderating discussions
- **Manages:** manages the team members' career, expectations, performance and level of happiness

Process

- **Follows:** follows the team processes, delivering a consistent flow of features to production
- **Enforces:** enforces the team processes, making sure everybody understands the benefits and tradeoffs
- **Challenges:** challenges the team processes, looking for ways to improve them
- **Adjusts:** adjusts the team processes, listening to feedback and guiding the team through the changes
- **Defines:** defines the right processes for the team's maturity level, balancing agility and discipline

Influence

- **Subsystem:** makes an impact on one or more subsystems
- **Team:** makes an impact on the whole team, not just on specific parts of it
- **Multiple Teams:** makes an impact not only his/her team but also on other teams
- **Company:** makes an impact on the whole tech organization
- **Community:** makes an impact on the tech community

Additional resources

- engineeringladders.com

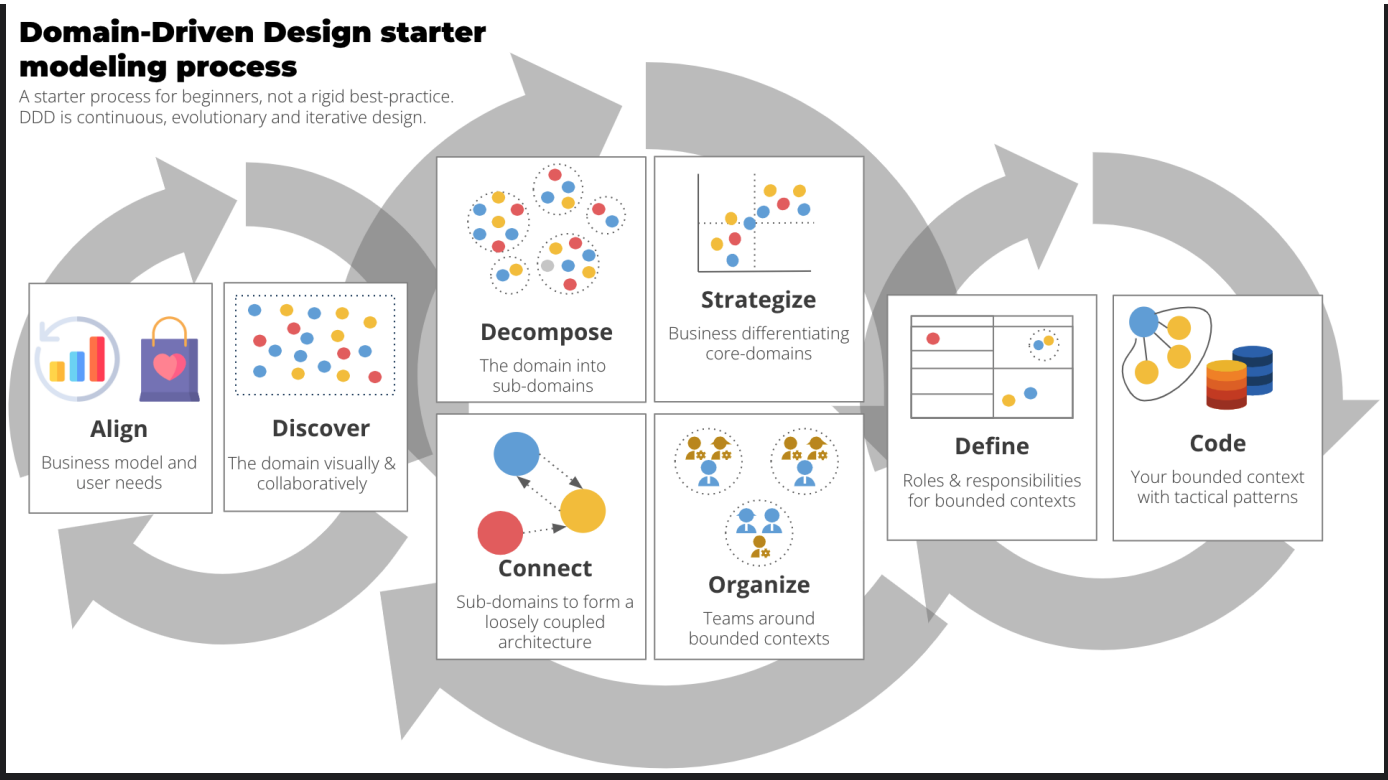
Domain-Driven Design Starter Modelling Process

This process gives you a step-by-step guide for learning and practically applying each aspect of Domain-Driven Design (DDD) - from orienting around an organisation's business model to coding a domain model.

Using this process will guide you through each of the essential steps in designing a software system with the DDD mindset, so you can focus on your business challenges and not be overwhelmed by learning DDD at the same time.

Once you have been through a few iterations of the process you will have the foundational DDD theory and practical experience to go deeper into DDD. Then you will be able to adapt and improve the process to suit your needs in any context. On a real project you'll often be jumping back and forth between these steps.

> This process is for beginners. It is not a linear sequence of steps that you should standardise as a best practice. Domain-Driven Design is an evolutionary design process which necessitates continuous iteration on all aspects of knowledge and design.



Navigation:

When to use the DDD Starter Modelling Process?

If you're new to DDD or just not sure where to start, this process can reduce your cognitive load. It will guide you through following scenarios, and possibly others:

Kicking Off a Greenfield Project

At the start of a new project the number of things you need to think about can be overwhelming. One or two iterations of this process can help you put the foundations in place.

Beginning a Brownfield Migration

Before getting to work on modernising your legacy system, a few iterations of this process can help you to uncover essential information needed to create a vision for your target architecture.

Kicking Off a Major Program of Work

When starting a new initiative involves a significant investment across many teams, it is essential to cover the 8 steps in the process. This process can guide you through the first few iterations.

Explore Your Domain for New Learning Opportunities

Software development is a learning process. You can apply the DDD Starter Modelling Process at any time to uncover new insights, identify new opportunities, or simply share knowledge around the team.

Assess Current State of Your Project

This process can be the foundation for assessing how well your current system is aligned to the domain and business model.

Re-organising Teams

A loosely-coupled architecture enables teams to work in parallel without being blocked. A loosely-coupled architecture also must be aligned to coupling in the domain. This process will help you to design a software architecture, and a team structure aligned with your domain.

Practicing or Learning DDD

This process is ideal when you are new to DDD and want to practice, or you want to teach others the different aspects of modelling a domain. It's important to communicate that this linear process is not a realistic process. It's just a starting point to reduce cognitive load until you are confident with DDD.

How to Adapt the Process?

This process can be customised in many ways. On a real project, you'll be switching between all 8 steps based on the new insights you gain or need to gain.

Below are a few reasons for deciding when to change the order or switch between steps.

Start with Collaborative Modelling

If you want to get your whole team collaborating immediately, modelling the domain which they are familiar with might be more comfortable than talking about business models and

strategy which they are less comfortable with.

Start by Assessing IT Landscape

Before looking forward to the business vision and going deep into the domain, it might be better to visualise the existing architecture first. Start with step 5 and map out your strategic portfolio to see what the major constraints you will face are.

Code Before Confirming Architecture and Team Boundaries

On some projects it makes sense to start by writing code sooner. Perhaps you need to deliver an MVP or the domain is so complex that creating a model in code is necessary before you can consider the architecture.

Repeat Steps 2 (Discover) - 6 (Organise) Before Moving to 7 (Define)

Before you dive into the definition of individual bounded contexts, it may be beneficial to model the domain multiple times and look for different ways to decompose your system into sub-domains and teams.

Organise Teams Before Designing Contexts

For a great deal of projects there are organisational constraints that we need to take into account. If this is the case, you should consider identifying possible team structures before designing architectures that you will never be able to implement.

Blending Definition and Coding

Steps 7 (Define) and 8 (Code) can occur concurrently. This may happen when you are coding a bounded context, and the insights you get from writing code make you change the high-level design.

The Process

The modelling process is composed of 8 steps which are introduced below.

A good talk that gives an overview of the process in the context of typical phases of designing sociotechnical architectures is "Sociotechnical Architecture: co-designing technical & organizational architecture to maximize impact" by Eduardo da Silva. Eduardo groups the activities of the process and its 8 steps in four distinct phases, namely:

1. Align & Understand
2. Strategic Architecture
3. Strategy & Org Design
4. Tactical Architecture.

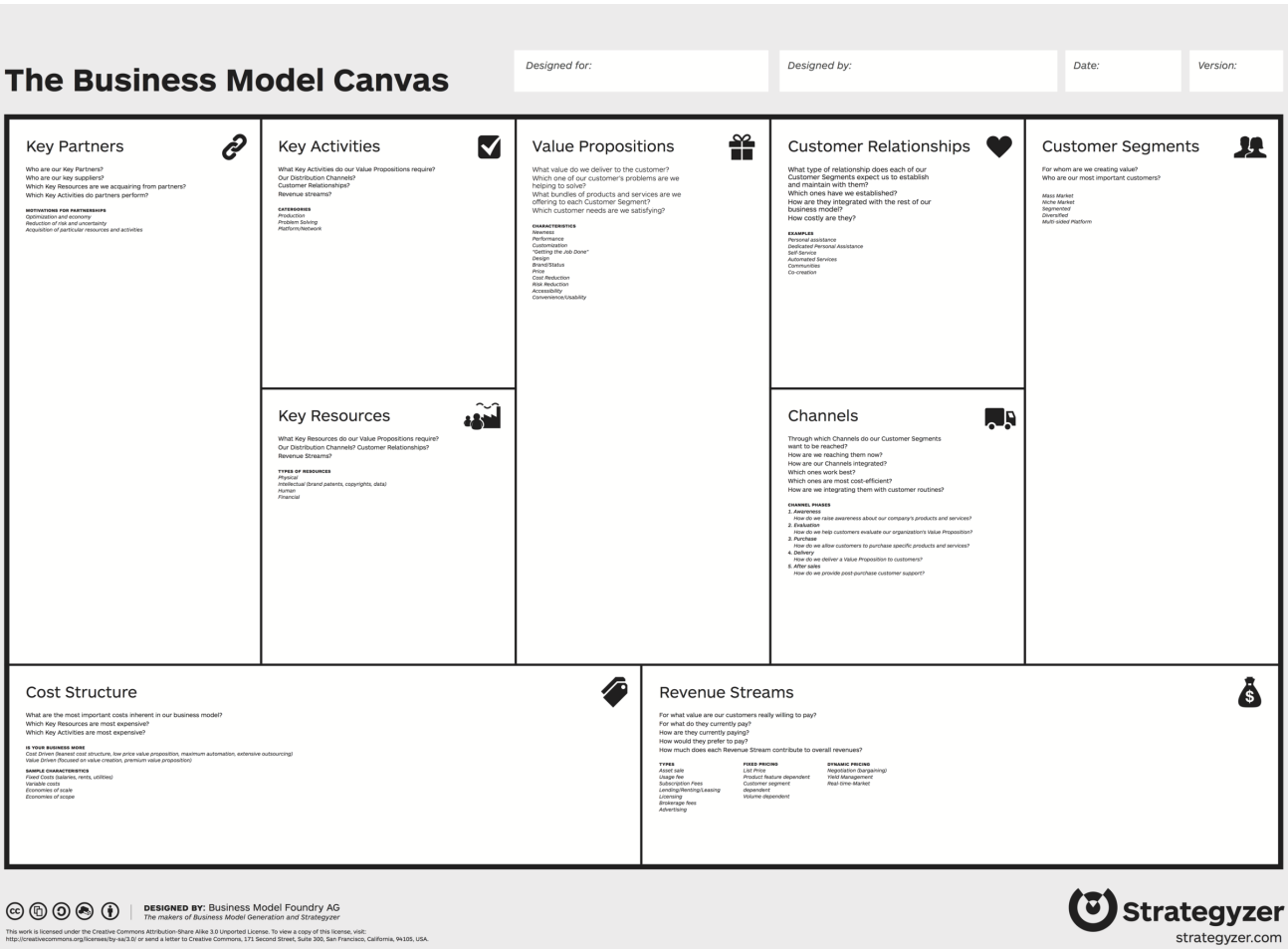
Understand

Align our focus with the organisation's business model, the needs of its users, and its short, medium, and long-term goals.

Every decision we take regarding the architecture, the code, or the organisation has business and user consequences. In order to design, build, and evolve software systems most effectively, our decisions need to create the optimal business impact, which can only be achieved if we are aligned to the business goals, as well as supporting the users current and potential future needs.

Badly designed architecture and/or boundaries can have a negative impact or even make it impossible to achieve these goals.

As a starting point, we recommend The Business Model Canvas for the business perspective, User Story Mapping for understanding the user vantage point.



Tools

- Impact Mapping
- The Business Model Canvas
- The Product Strategy Canvas
- Wardley Mapping
- User Story Mapping

Who to Involve

- People who design, build, test software
- People who have domain knowledge
- People who understand the product and business strategy
- Real end users, not only their representatives in your organisation

Discover

Discover the domain visually and collaboratively.

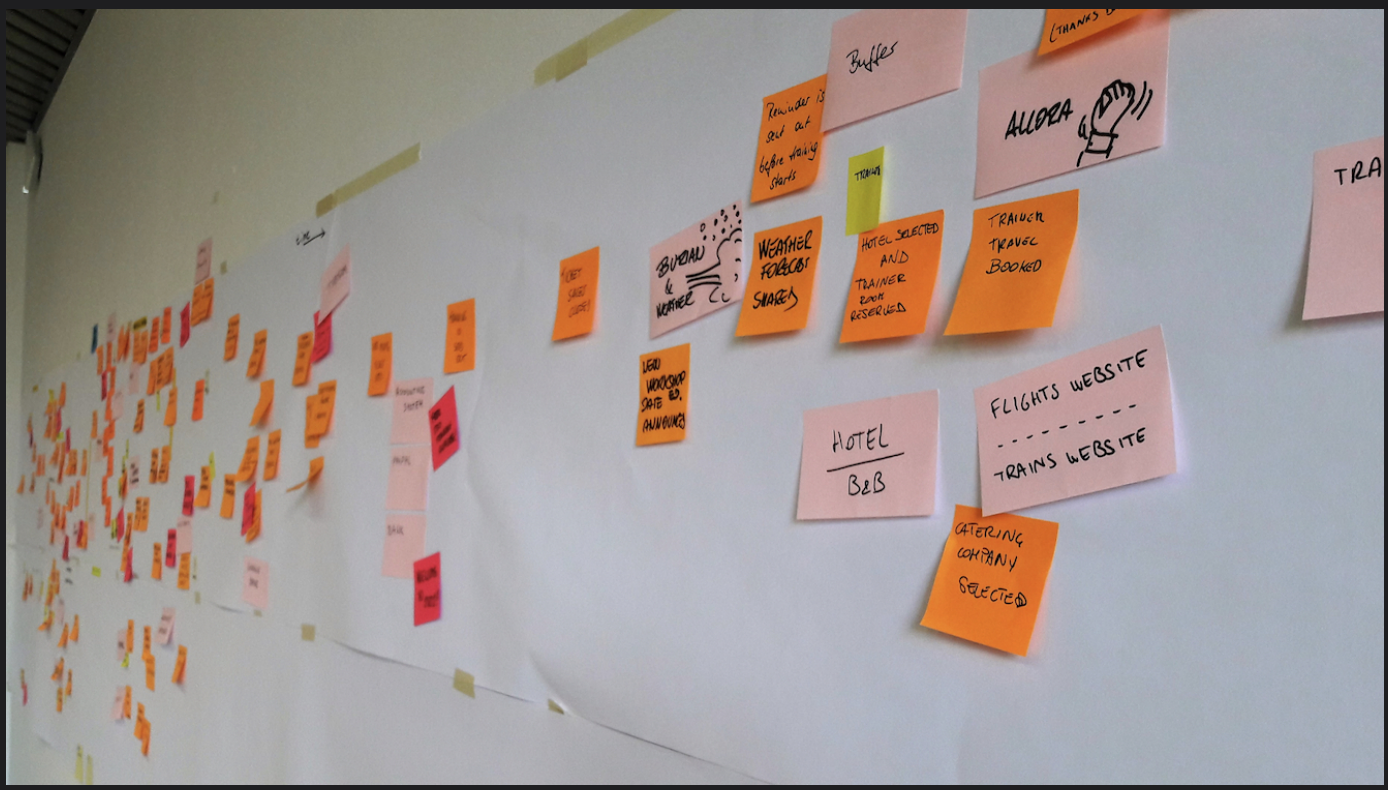
This is the most crucial aspect of DDD. You cannot skip discovery. If your whole team doesn't build up a good understanding of the domain, all software decisions will be misguided.

Spreading domain knowledge through the whole team will create a shared understanding. It enables the developers to build a software system aligned to the domain which can be more flexible to incorporate future business changes.

Ensuring that domain knowledge is spread across the whole team enables its members to contribute with ideas for improving the product.

> **Discovery is Continuous**
> Teams who are successful with DDD are practicing discovery techniques on a frequent basis. There is always more to learn about the domain.
> When first attempting discovery, a facilitator who is experienced with techniques like EventStorming can help a team to see the true benefits of discovery beyond a superficial level.
> We strongly encourage you to check out Visual Collaboration Tools.

As a starting point, we recommend EventStorming.



Tools

- Domain Storytelling

- Example Mapping
- EventStorming
- User Journey Mapping
- User Story Mapping

Who to Involve

- People who design, build, test software
- People who have domain knowledge
- People who understand product and business strategy
- People who understand the customers' needs and problems
- Real end users

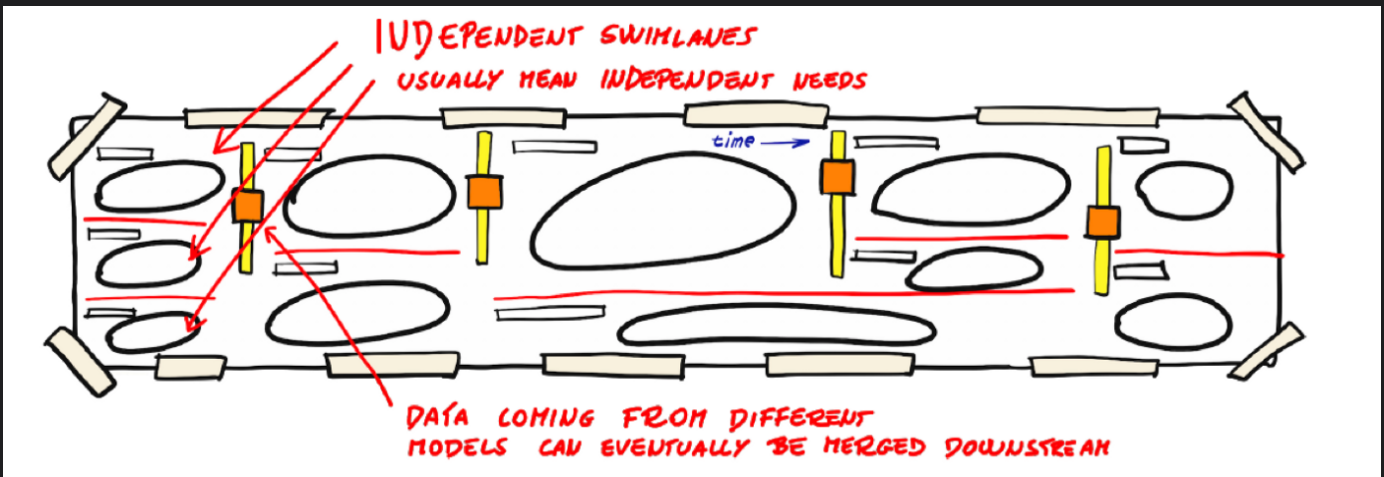
Decompose

Decompose the domain into sub-domains - loosely-coupled parts of the domain.

We decompose a large problem domain into sub-domains for a few key reasons:

- reduced cognitive load, so that we can reason about parts of the domain independently,
- give development teams autonomy, so that they can work on separate parts of the solution,
- identifying loose-coupling and high-cohesion in the domain which carries over to our software architecture and team structure.

As a starting point, we recommend carving up your event storm into sub-domains and Context Maps.



Credit: Alberto Brandolini

Tools

- Business Capability Modelling
- Design Heuristics
- EventStorming with sub-domains
- Independent Service Heuristics
- Visualising Sociotechnical Architecture with Context Maps

Who to Involve

- People who design, build, test software
- People who have domain knowledge

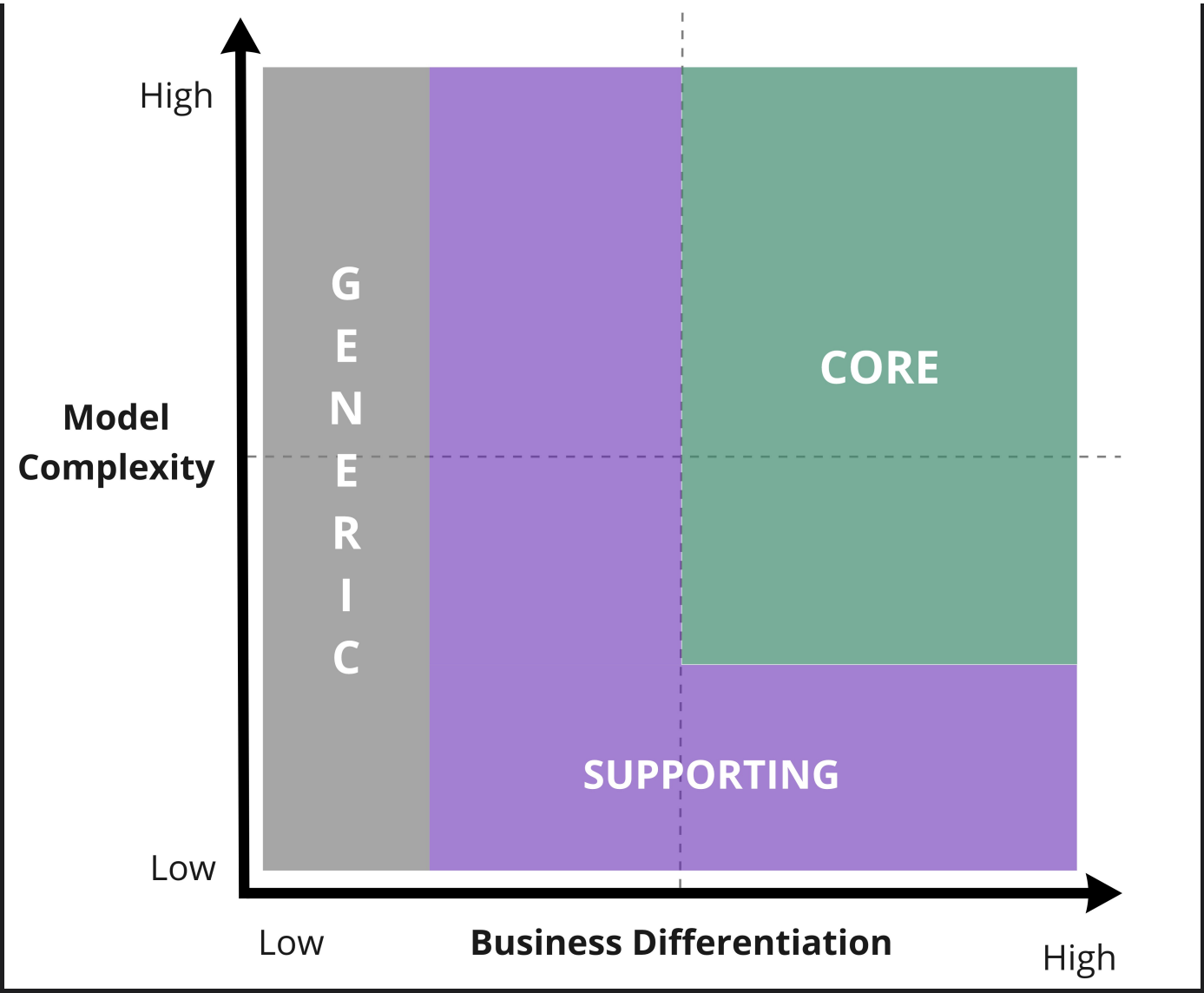
Strategize

Strategically map out your sub-domains to identify core domains: the parts of the domain which have the greatest potential for business differentiation or strategic significance.

Time and resources are limited, so understanding which parts of the domain to focus on is critical to delivering optimal business impact.

By analysing what your core domains are, you will have a better idea of how much quality and rigour is required to build each part of your system, and you'll be able to make highly-educated build vs buy vs outsource decisions.

As a starting point, we recommend Core Domain Charts.



Tools/Resources

- Core Domain Charts
- Purpose Alignment Model
- Wardley Mapping
- Revisiting the Basics of Domain-Driven Design

Who to Involve

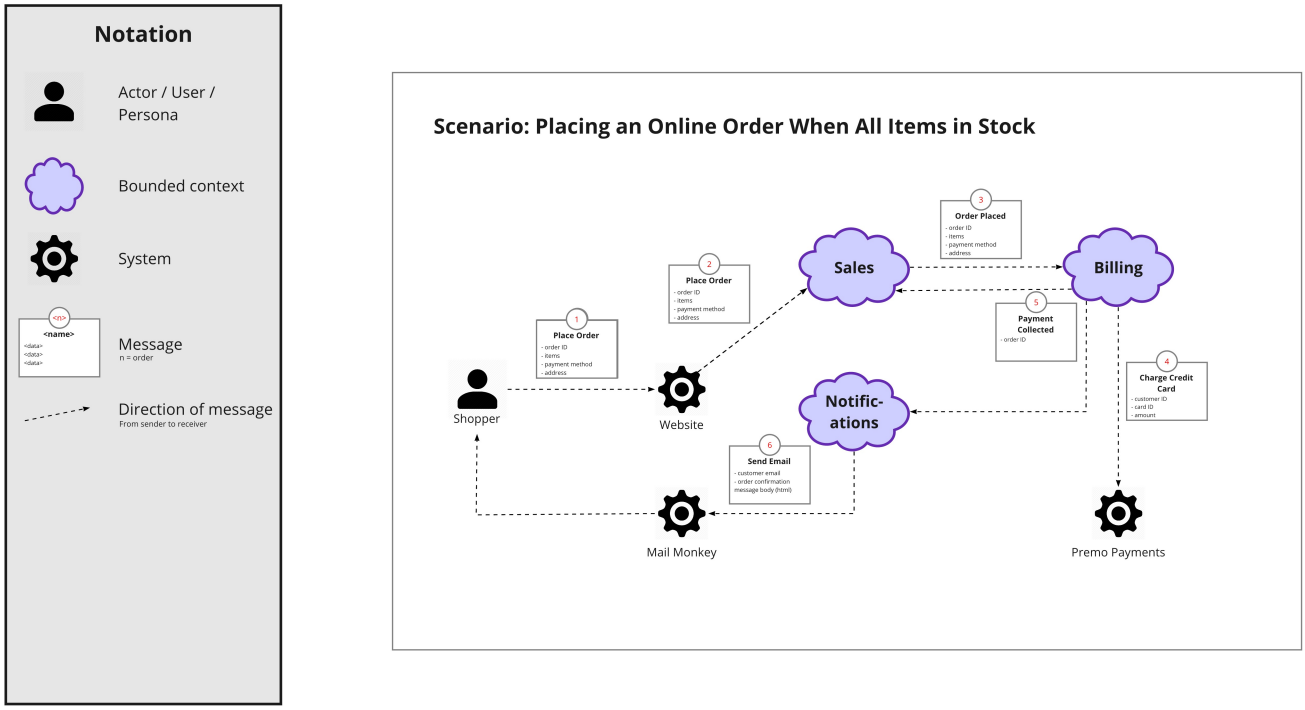
- People who understand product and business strategy
- People who design, build, test software
- People who have domain knowledge

Connect

Connect the sub-domains into a loosely-coupled architecture which fulfills end-to-end business use-cases.

It is imperative to not only decompose a large domain into parts but to also carefully design the interactions between those parts to minimise unwanted coupling and complexity. It is necessary to challenge the initial design by applying concrete use-cases to uncover hidden complexity.

As a starting point, we recommend Domain Message Flow Modelling.



Tools

- Business Process Model and Notation
- Domain Message Flow Modelling
- Process Modelling EventStorming
- Sequence Diagrams

Who to Involve

- People who design, build, test software
- People who have domain knowledge

Organise

Organise autonomous teams that are optimised for fast flow and aligned with context boundaries.

Teams need to be organised to have autonomy, clear goals and sense of purpose. In order to do that we need to take into account organisational constraints, so that teams organise themselves for fast flow.

> **Team Self-organisation**
> Organisation is not something that is done to teams, rather teams should be involved in the process of defining their boundaries, interactions, and responsibilities.
> Some companies like Red Gate Software empower and trust their teams to fully organise themselves.

We can optimise how people collaborate with each other if we align teams with context boundaries. In order to right-size the teams we need to take into account available talent, cognitive load, communication overhead, and bus factor.

As a starting point, we recommend visualising sociotechnical architecture with the Context Maps. A brief overview of the most important patterns can be found under the context-mapping GitHub Project.

Context Map Cheat Sheet

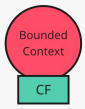
Context Map Patterns

Open / Host Service

A Bounded Context offers a defined set of services that expose functionality for other systems. Any downstream system can then implement their own integration. This is especially useful for integration requirements with many other systems. Example: public APIs.

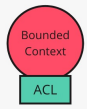
Conformist

The downstream team conforms to the model of the upstream team. There is no translation of models. Couples the Conformist's domain model to another bounded context's model.



Anticorruption Layer

The anticorruption layer is a layer that isolates a client's model from another system's model by translation. Only couples the integration layer (or adapter) to another bounded context's model but not the domain model itself.

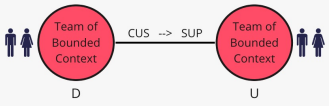


Shared Kernel

Two teams share a subset of the domain model including code and maybe the database. Typical examples: shared JARs, DLLs or a shared database schema. Teams with a Shared Kernel are often mutually dependent and should form a Partnership.

Customer / Supplier

There is a customer / supplier relationship between teams. The downstream team is considered to be the customer. Downstream requirements factor into upstream planning. Therefore, the downstream team gains some influence over the priorities and tasks of the upstream team.



Partnership

Partnership is a cooperative relationship between two teams. These teams establish a process for coordinated planning of development and joint management of integration.

Published Language

A Published Language is a well documented shared language between Bounded Contexts which can translate in and out from that language. Published Language is often combined with Open Host Service.Typical examples are iCalendar or vCard.

Separate Ways

Bounded Contexts and their corresponding teams have no connections because integration is sometimes too expensive or it takes very long to implement. The teams chose to go separate ways in order to focus on their specific solutions.

Big Ball Of Mud

A (part of a) system which is a mess by having mixed models and inconsistent boundaries. Don't let this lousy model propagate into the other Bounded Contexts. Big Ball Of Mud is a demarcation of a bad model or system quality.

Team Relationships

Mutually Dependent

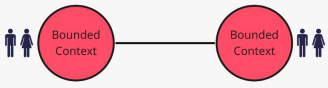
Two software artifacts or systems in two bounded contexts need to be delivered together to be successful and work. There is often a close, reciprocal link between data and functions between the two systems.

Free

Changes in one bounded context do not influence success or failure in other bounded contexts. There is, therefore, no organizational or technical link of any kind between the teams.

Upstream / Downstream

Actions of an upstream team will influence the downstream counterpart while the opposite might not be true. This influence can apply to code but also on less technical factors such as schedule or responsiveness to external requests.



Credit: Michael Plöd

Tools

- Dynamic Reteaming
- Pioneers, Settlers & Town Planners
- Team Topologies
- Visualising Sociotechnical Architecture with Context Maps

Who to Involve

- People who design, build, test software
- People who have domain knowledge
- People who understand the product and business strategy

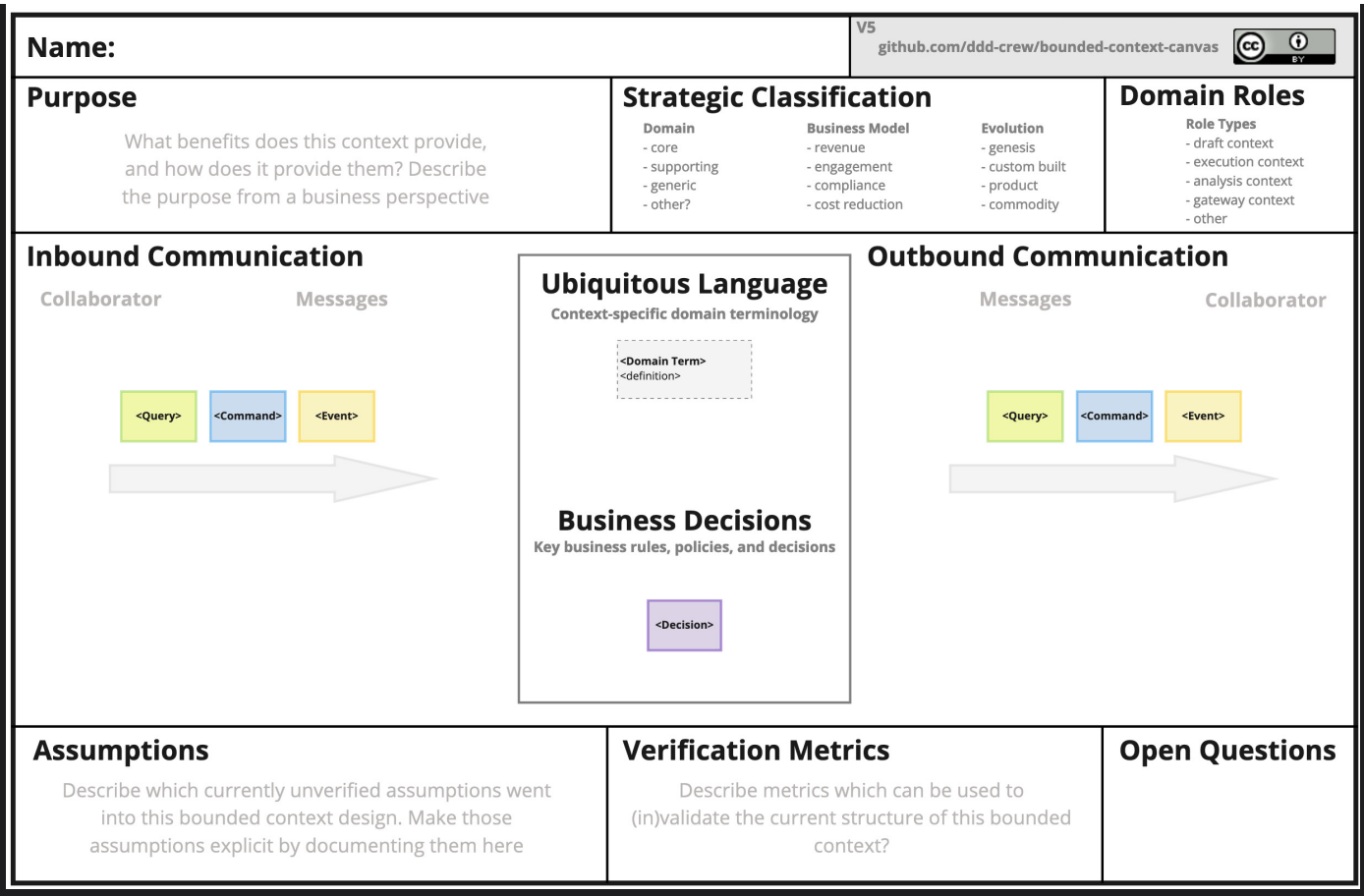
Define

Define the roles and responsibilities of each bounded context.

Before committing to a design, make explicit decisions about the choices which can have a significant impact on the overall design. Have these conversations early while it is still easy to change your mind and explore alternative models.

Design collaboratively and visually, and start to consider the technical limitations so that you can uncover constraints or opportunities.

As a starting point, we recommend the Bounded Context Canvas.



Tools

- Bounded Context Canvas
- C4 System Context Diagram
- Quality Storming

Who to Involve

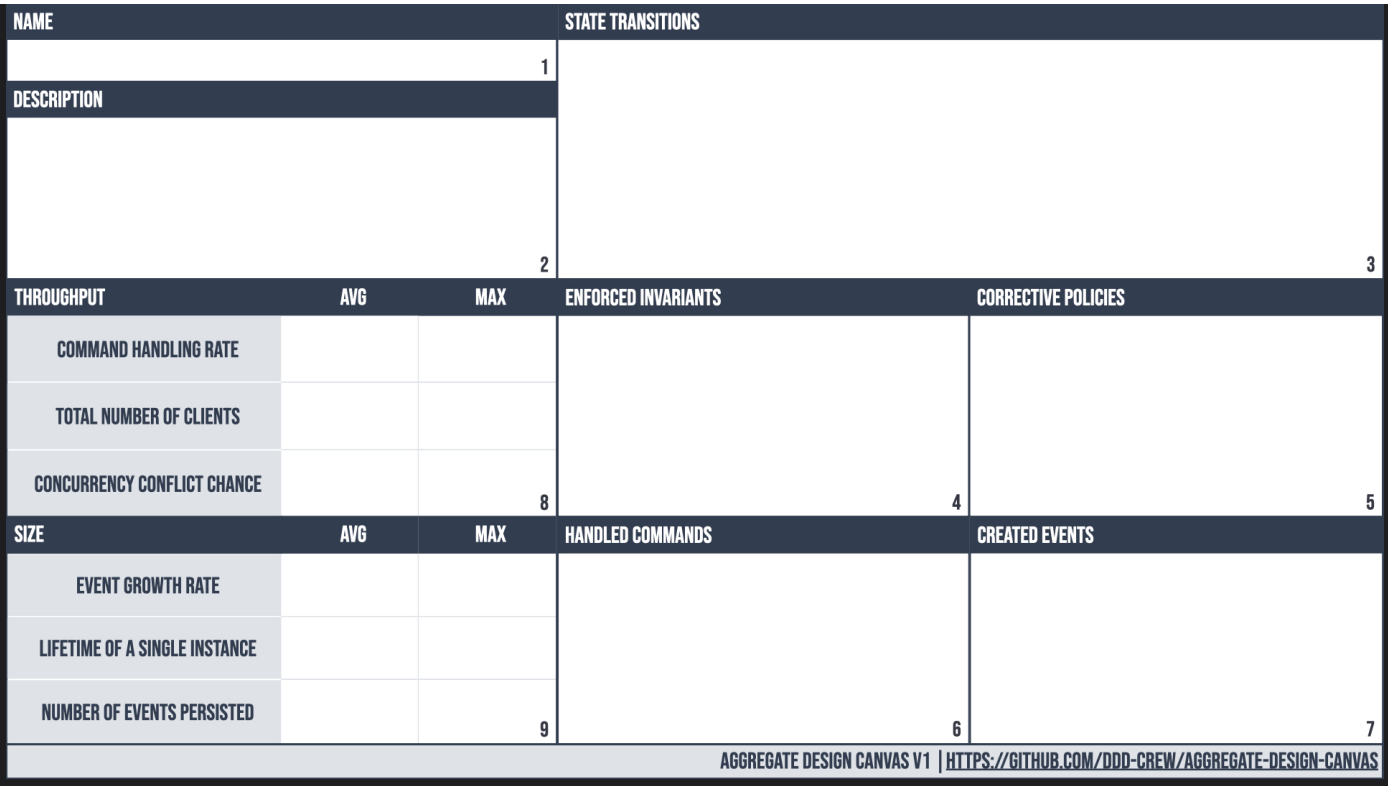
- People who design, build, test software
- People who have domain knowledge
- People who are responsible for the product

Code

Code the domain model.

Aligning the code to the domain makes it easier to change the code when the domain changes. By collaboratively modelling the problem space with experts, the developers have a chance to learn about the domain and minimise misunderstandings.

As a starting point, we recommend the Aggregate Design Canvas.



Tools

- Aggregate Design Canvas
- C4 Component Diagrams
- Design-Level EventStorming
- Event Modeling
- Hexagonal Architecture
- Mob Programming
- Model Exploration Whirlpool
- Onion Architecture
- Unified Modelling Language

Who to Involve

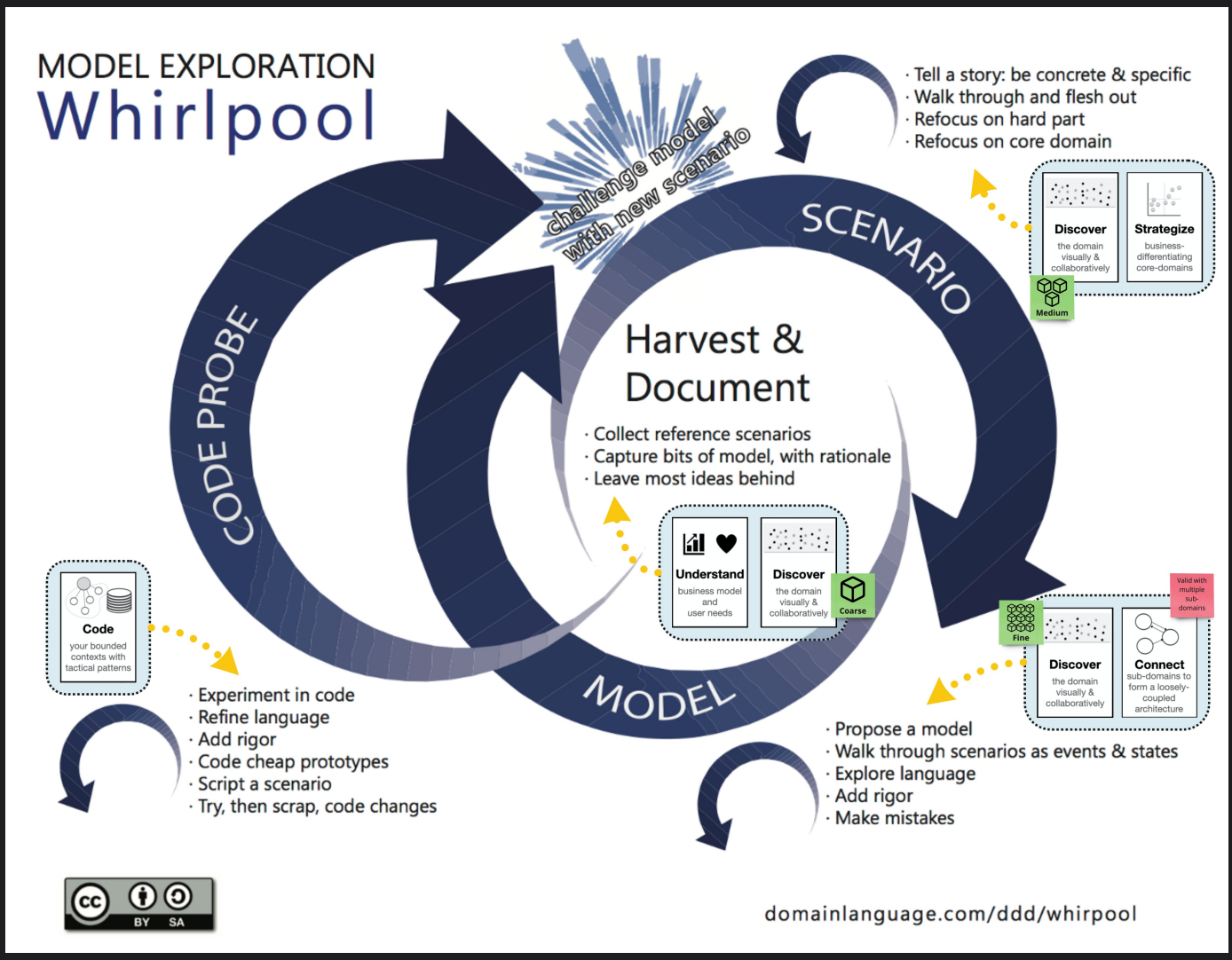
- People who design, build, test software

How the DDD Starter Modelling Process relates to the Whirlpool Process

Some of you might have noticed some similarities with Eric Evans' Whirlpool Process. And indeed, both are guides and not rigid best-practices. They're also both continuous and iterative.

But the DDD Starter Modelling Process covers more than the Whirlpool process by aiming

at building a socio-technical architecture.
The picture below shows a possible overlap between the two processes.



Needless to say that Eric Evan's Whirlpool process remains totally relevant today and gives people highly valuable insights and guidance on how to explore models.

Contributors

Thanks to all existing and future contributors and to the following individuals who have all contributed to the DDD Starter Modelling Process:

- Ciaran McNulty
- Eduardo da Silva
- Gien Verschatse
- James Morcom

- Maxime Sanglan-Charlier

Contributions and Feedback

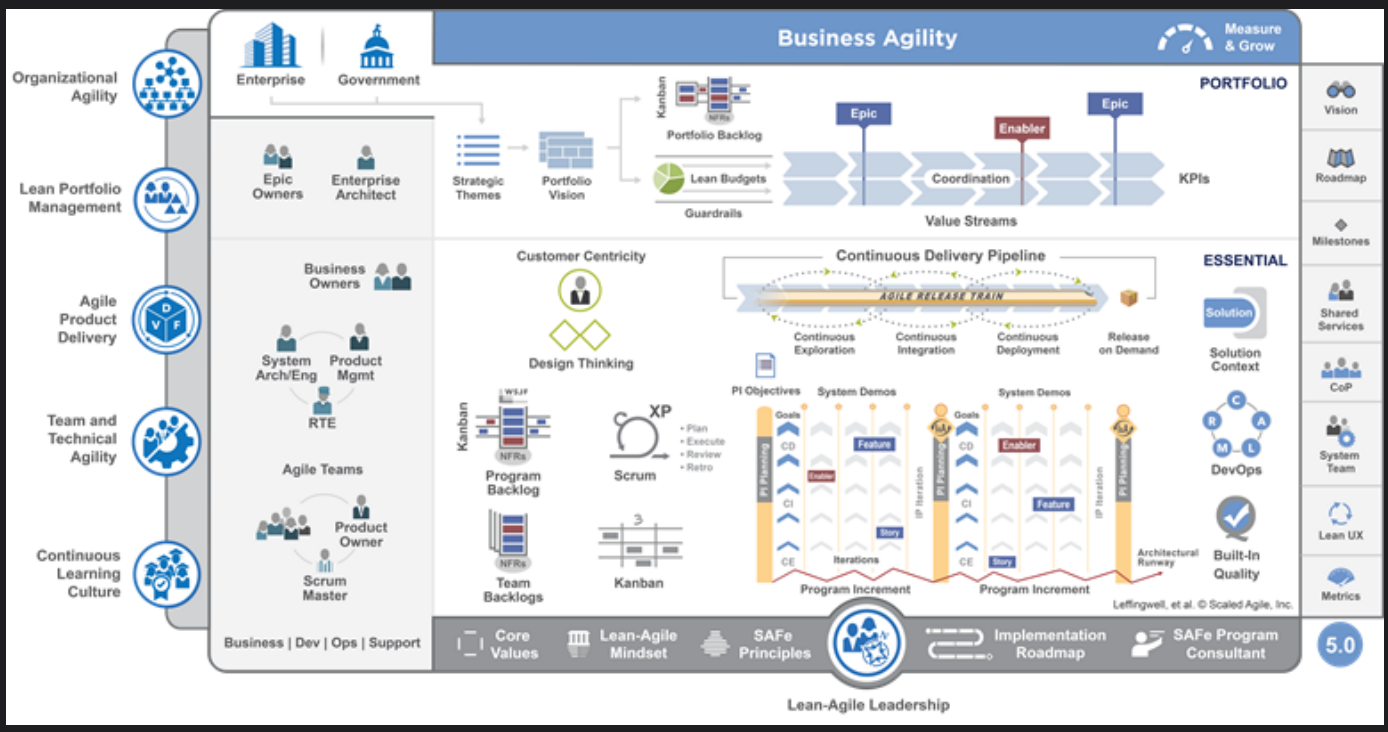
The Domain-Driven Design Starter Modelling Process is freely available for you to use. In addition, your feedback and ideas are welcome to improve the technique or to create alternative versions.

If you have questions you can ping us or open an Issue.

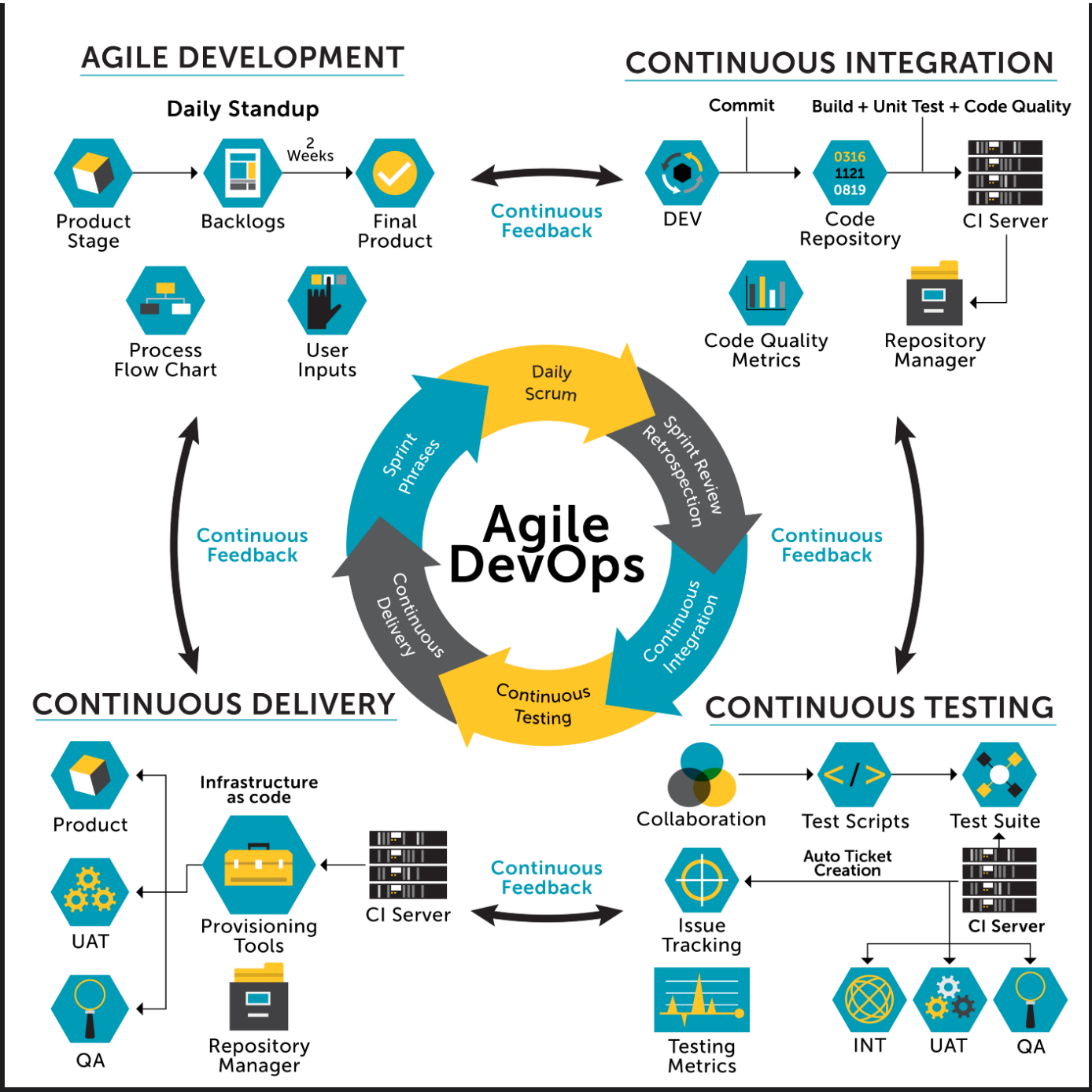
Feel free to also send us a pull request with your examples or experience reports.

Delivery process

Agile at scale



Devops

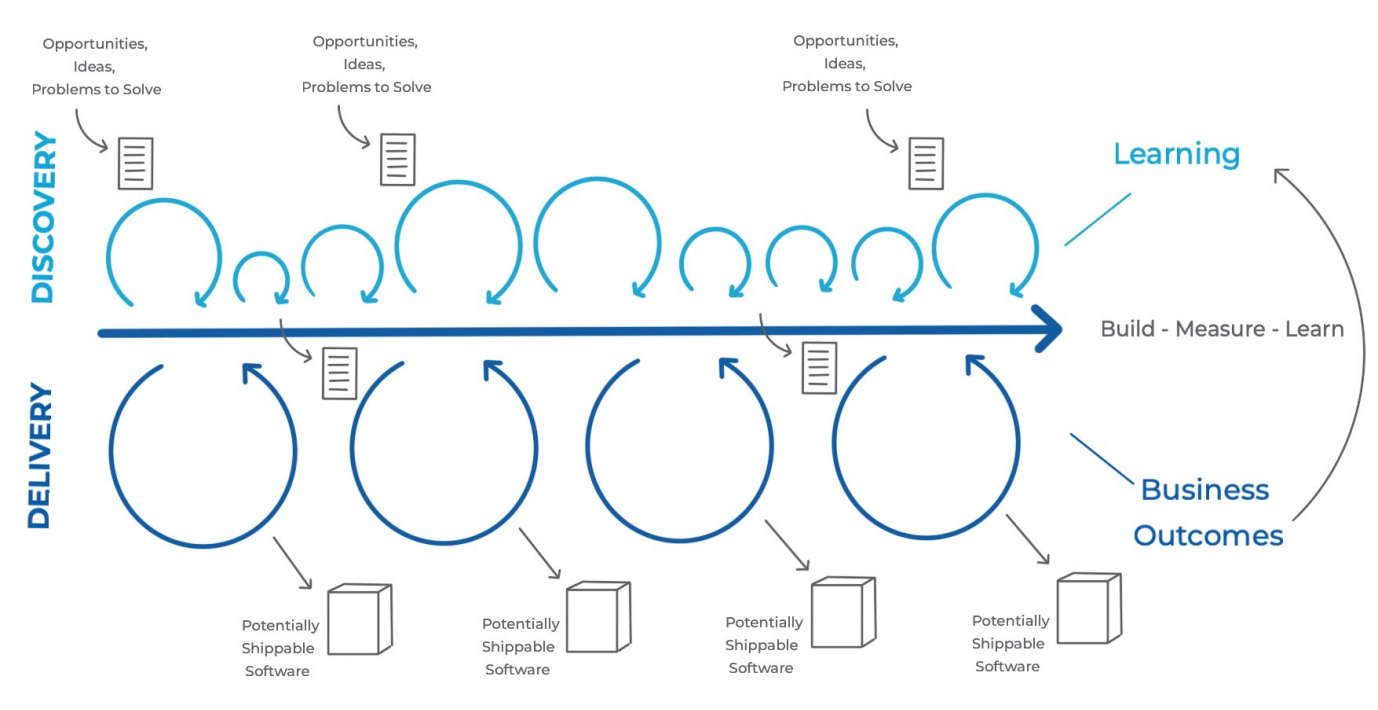


Additional resources

Product discovery & delivery

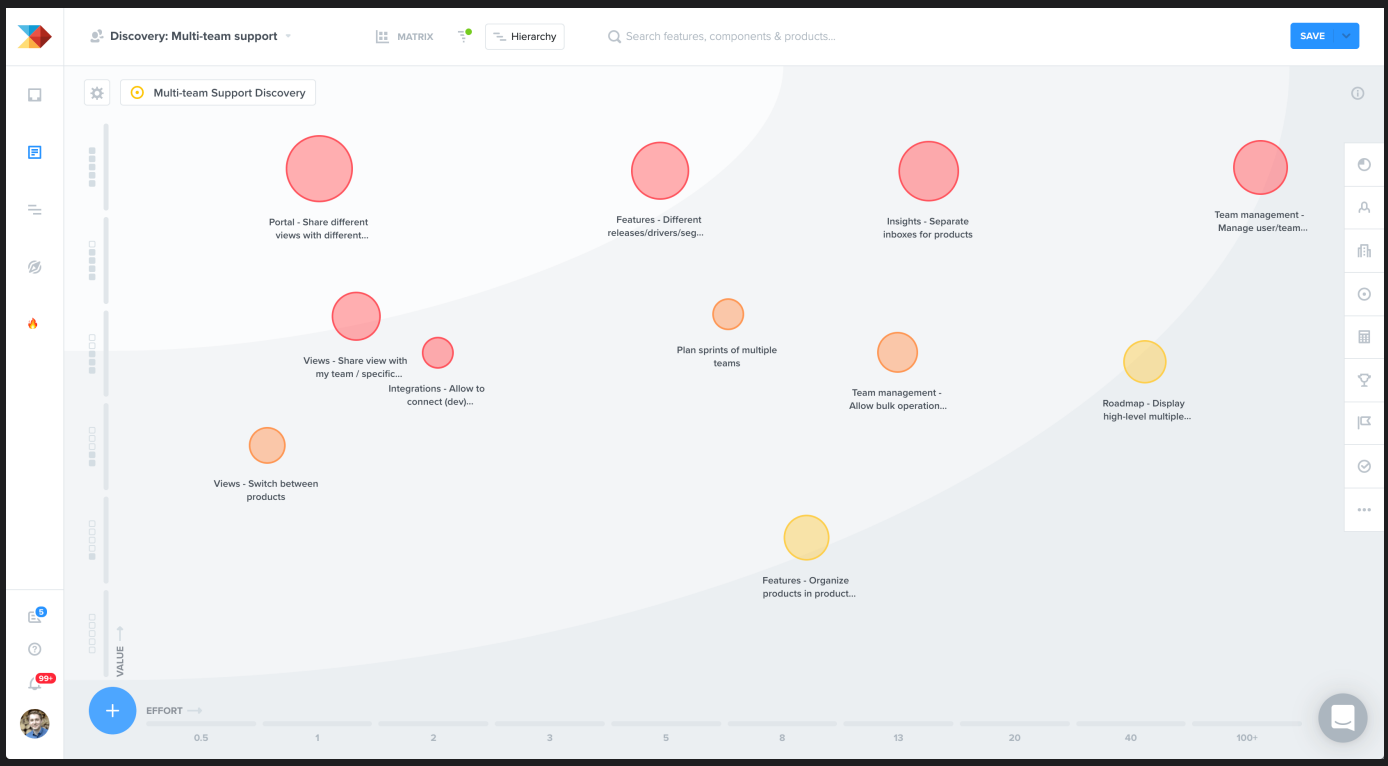
Discovery & delivery dual track

> Product discovery enables you to proactively research and prioritize the needs of your users. It also validates your ideas before you spend time building them. Implementing a discovery cycle removes the need for assumptions. It ensures that you work on the best possible features for your product. This means you don't waste resources on features your users don't need. It means you can put all your time and effort into the right features. Dual-track agile lets you balance both the discovery and delivery tracks. As the development team work on one set of improvements, your discovery team can start figuring out what the next set should be. It takes the guesswork out of product management. Marty Cagan is a major proponent of the dual-track agile framework. He says, "our higher order objective is to validate our ideas the fastest, cheapest way possible". Dual-track agile is the solution to that objective.



Prioritization

> Prioritization in product management is the disciplined process of evaluating the relative importance of work, ideas, and requests to eliminate wasteful practices and deliver customer value in the quickest possible way, given a variety of constraints.

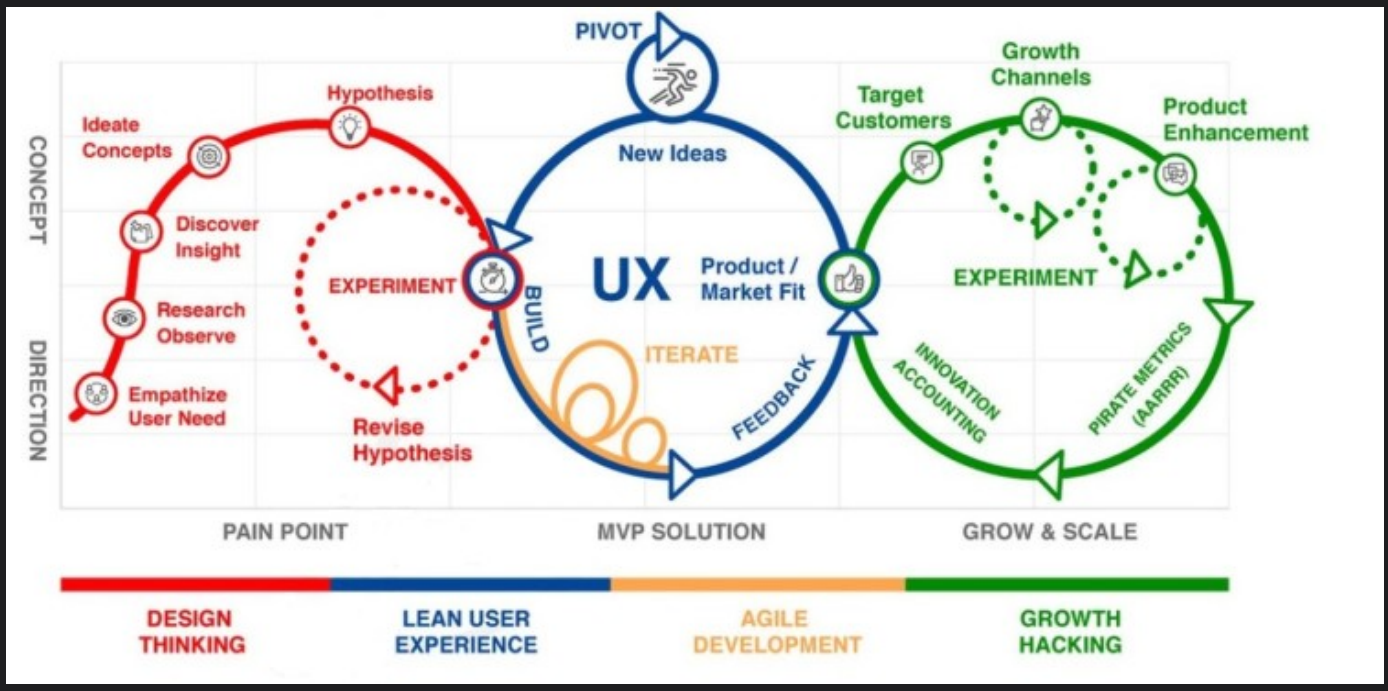


Additional resources

Dual track concept

Lean startup

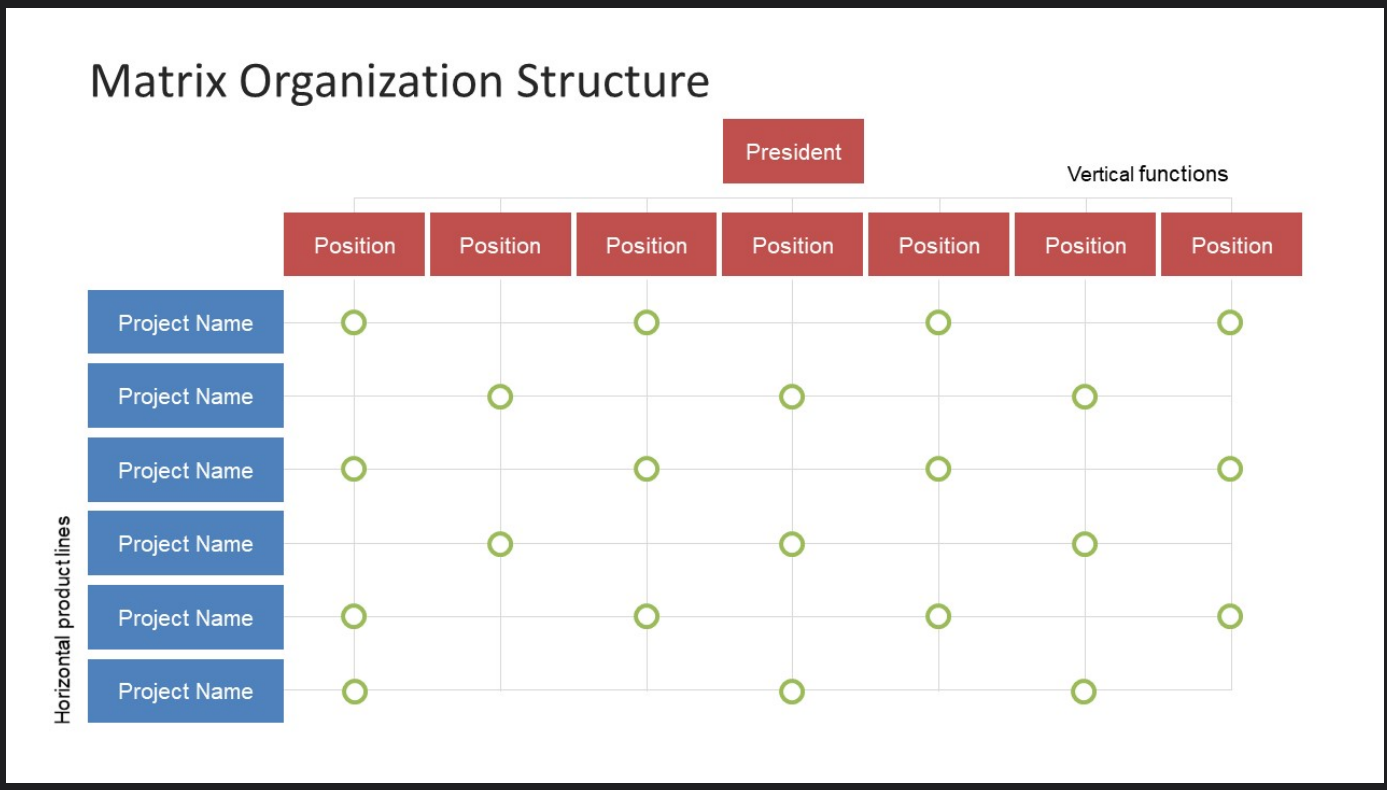
> Lean startup is a methodology for developing businesses and products that aims to shorten product development cycles and rapidly discover if a proposed business model is viable; this is achieved by adopting a combination of business-hypothesis-driven experimentation, iterative product releases, and validated learning. Lean startup emphasizes customer feedback over intuition and flexibility over planning. This methodology enables recovery from failures more often than traditional ways of product development. Central to the lean startup methodology is the assumption that when startup companies invest their time into iteratively building products or services to meet the needs of early customers, the company can reduce market risks and sidestep the need for large amounts of initial project funding and expensive product launches and financial failures. While the events leading up to the launch can make or break a new business, it is important to start with the end in mind. This means thinking about the direction in which you want your business to grow and how to put all the right pieces in place to make this possible



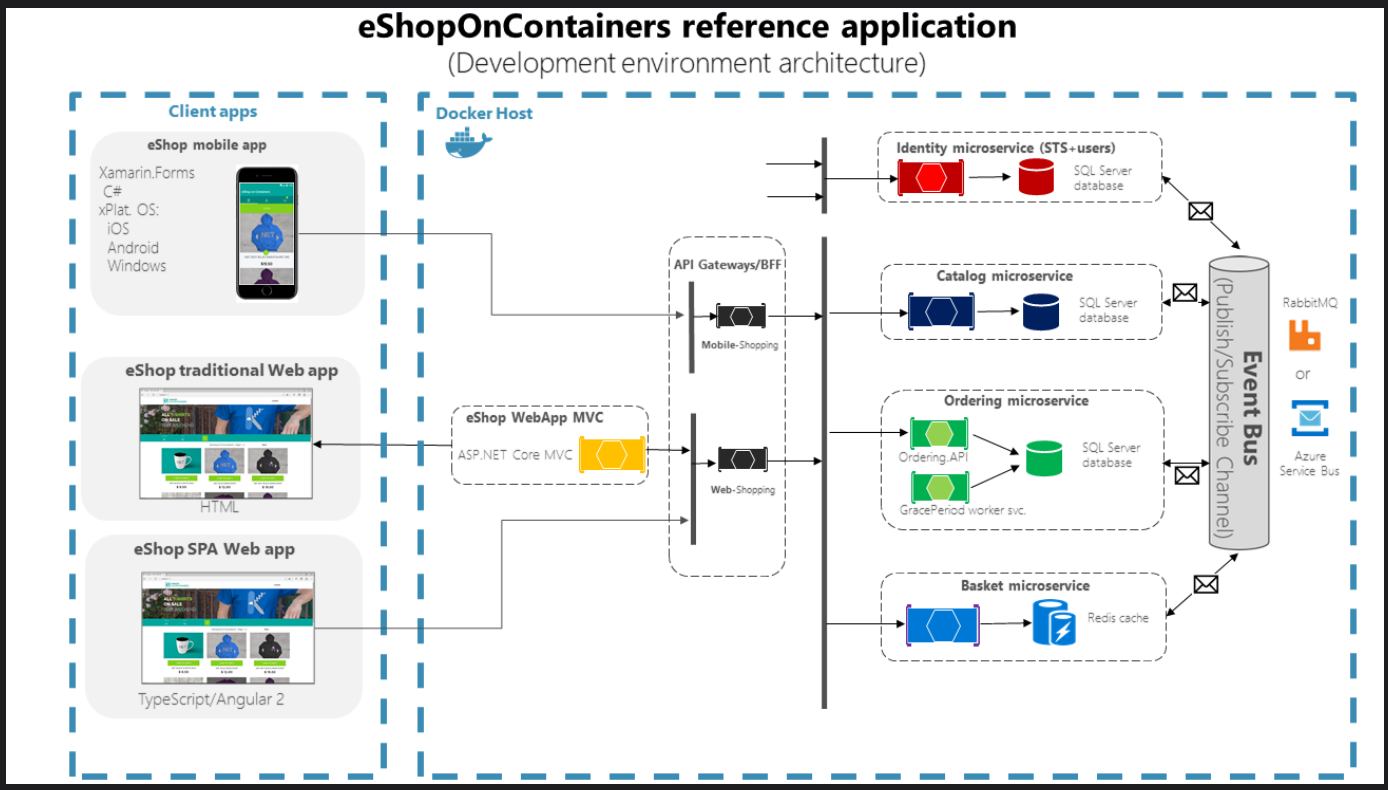
Additional resources

- Définition Lean startup

👨‍👩‍👧 matrix management

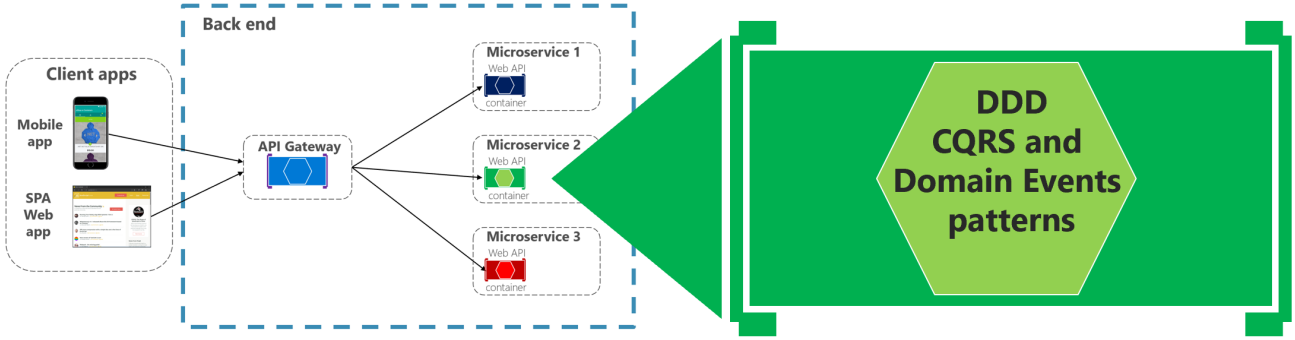


Microservices oriented architecture



External architecture per application

Internal architecture per microservice

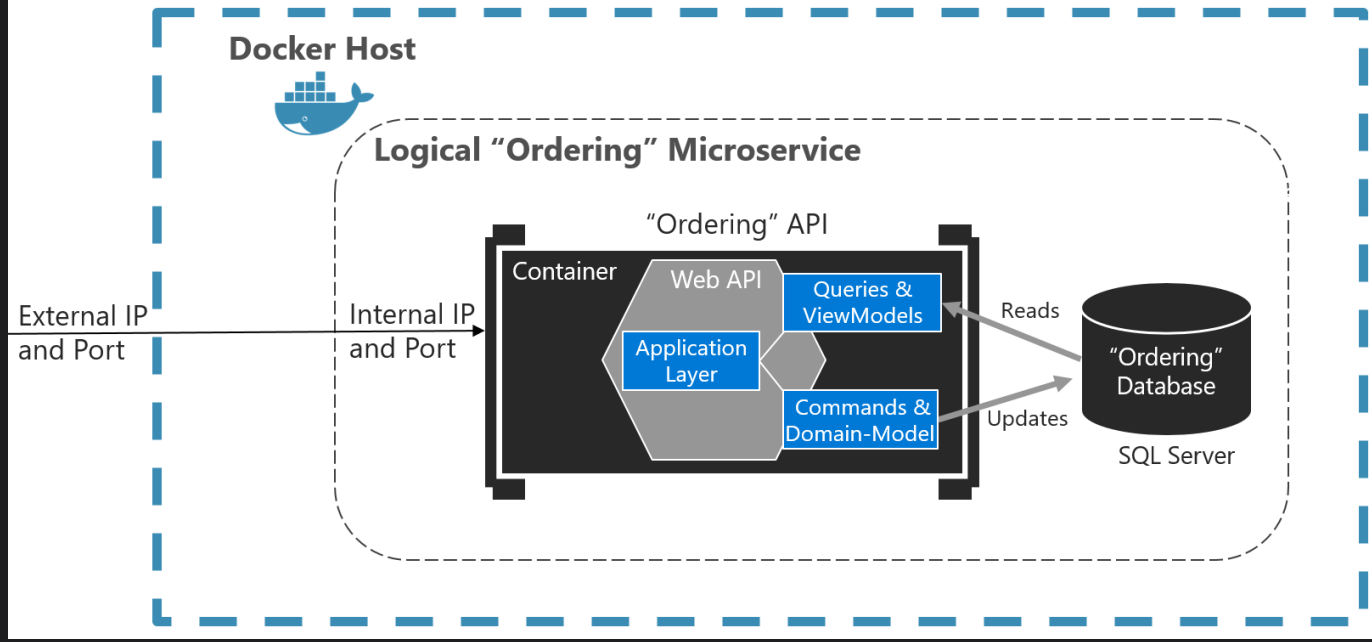


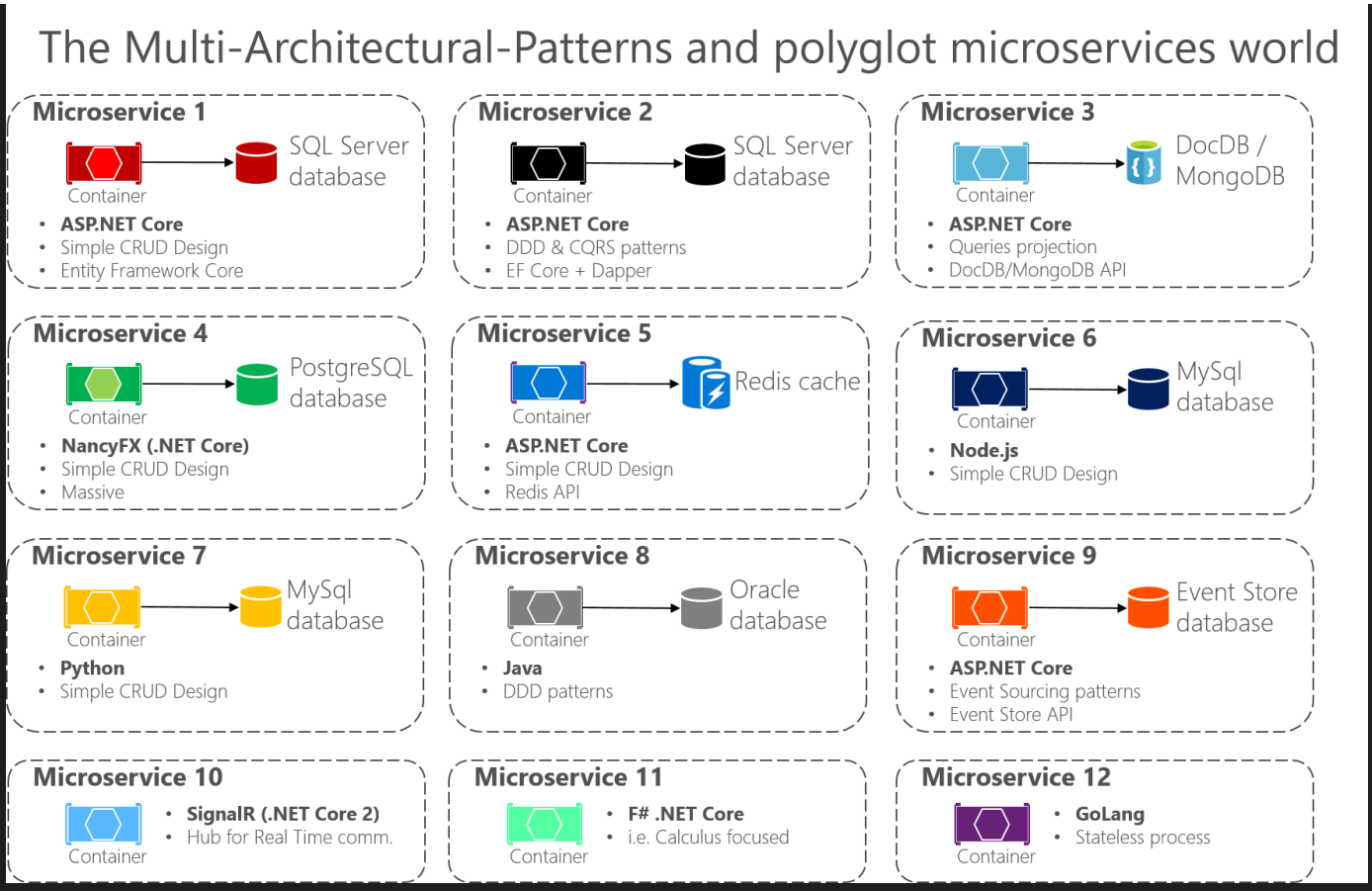
- External microservice patterns
- API Gateway
- Resilient communication
- Pub/Sub and event driven

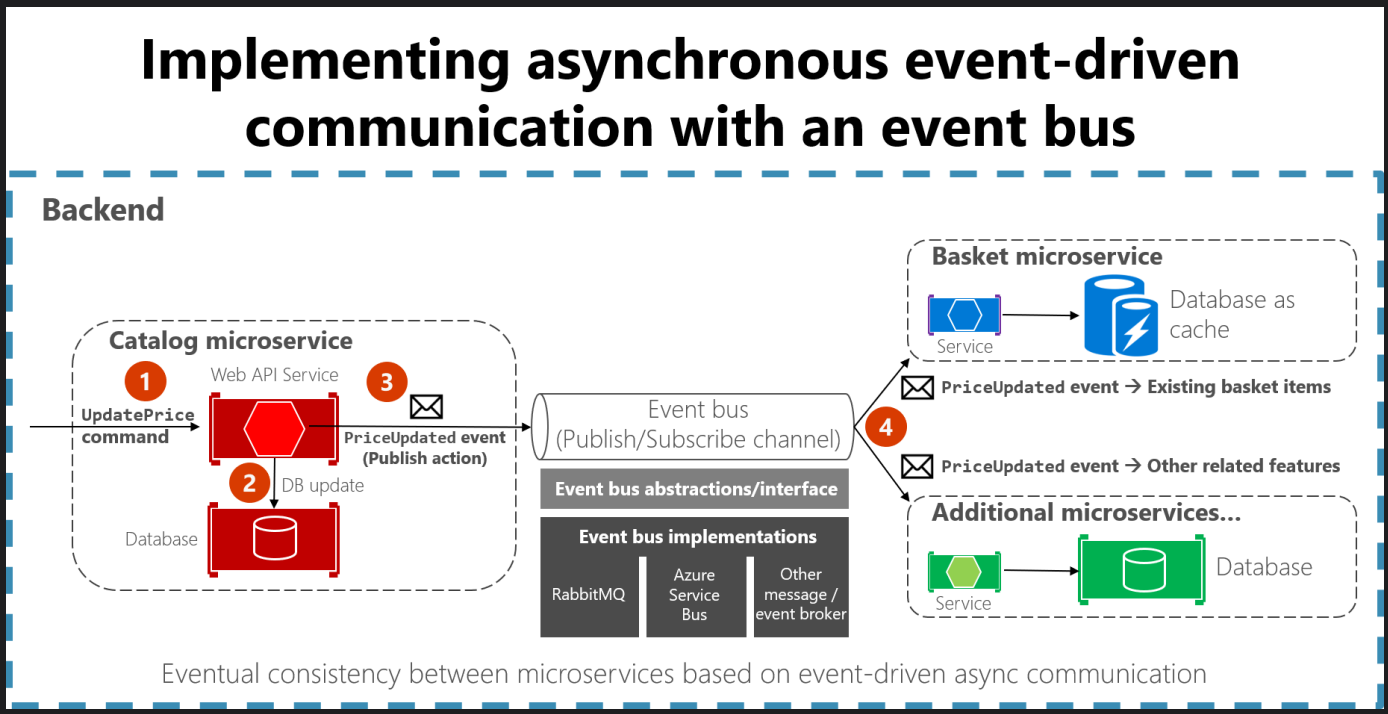
Internal DDD patterns in addition to SOLID principles and Dependency Injection

Simplified CQRS and DDD microservice

High level design







Additional resources

- https://learn.microsoft.com/en-us/dotnet/architecture/microservices/

No estimate delivery process

Introduction

The "No Estimate" delivery process is an alternative approach to traditional project management, where the focus shifts from estimating task durations to delivering value incrementally and iteratively. In this approach, the emphasis is on continuous improvement, adaptability, and maximizing productivity. Here are some guidelines to implement the "No Estimate" approach in a product engineering organization, along with associated practices and key performance indicators (KPIs) to measure success:

Embrace Agile Methodologies

Adopt Agile methodologies, such as Scrum or Kanban, to facilitate iterative development and quick feedback loops. Embrace the Agile principles of collaboration, responding to change, and delivering working software regularly.

Prioritize User Value

Focus on delivering features and improvements that provide tangible value to end-users. Collaborate closely with product managers and stakeholders to understand customer needs and align development efforts accordingly.

Predictability

"NoEstimates" bases its approach on the notion of forecasting and suggests that a team will be able to predict its speed of progress, in terms of US produced per capacity planning, thanks to data from 3 to 5 two-week iterations, including when it these are early project iterations, not very representative of the team's future cruising speed.

Continuous Delivery and Integration

Invest in robust continuous integration and continuous delivery (CI/CD) pipelines to ensure a smooth and automated release process. Frequent integration and delivery of smaller changes promote faster feedback and reduced risk.

Short Iterations and Incremental Releases

Adopt short development iterations, such as two weeks or less, to deliver incremental improvements regularly. These iterations allow for quick course correction and adaptability to changing requirements.

Focus on Cycle Time

Monitor cycle time, which is the time taken from idea to production deployment. Shortening cycle time is a key goal in the "No Estimate" approach as it emphasizes rapid delivery and faster feedback.

Measure Customer Satisfaction

Use customer satisfaction surveys, Net Promoter Score (NPS), or other customer feedback mechanisms to measure the impact of delivered features on user satisfaction and loyalty.

Monitor Lead Time

Lead time measures the time taken from the inception of a task or feature request to its completion. Keep track of lead time metrics to identify bottlenecks and optimize development processes.

Team Velocity

While avoiding time-based estimates, measure team velocity in terms of completed work items per iteration. This can help teams in planning future iterations and managing workloads.

Continuous Improvement

Encourage a culture of continuous improvement within the organization. Conduct regular retrospectives to identify areas of improvement, implement changes, and adapt to evolving needs.

Celebrate Small Wins

Recognize and celebrate the achievements of the team, even for small incremental improvements. Positive reinforcement fosters a motivated and engaged team environment.

Conclusion

The "No Estimate" delivery process is a value-driven and adaptive approach to software development that empowers product engineering organizations to focus on delivering customer value rather than meeting time-based estimates. By embracing Agile methodologies, using relative sizing, prioritizing user value, and employing continuous

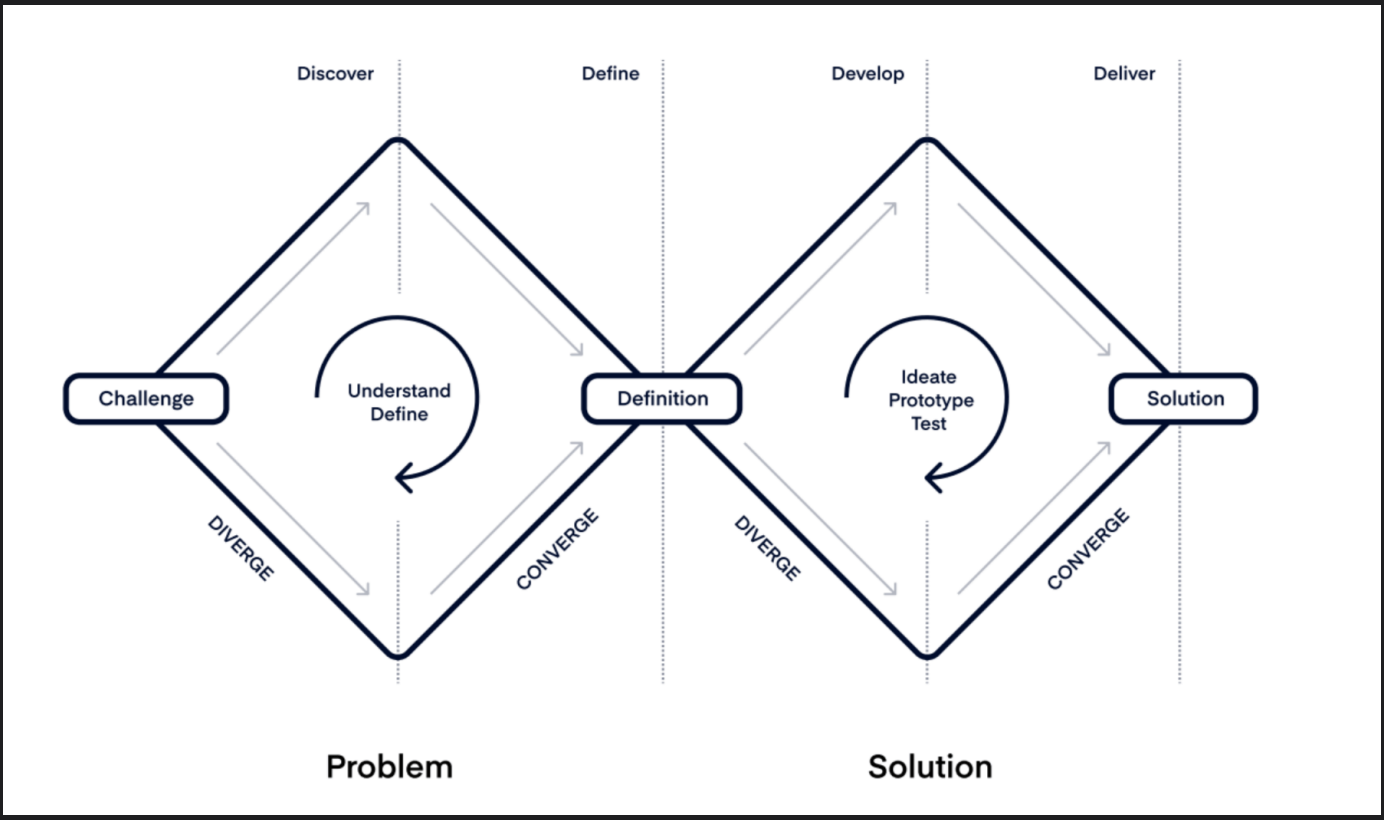
delivery practices, teams can achieve greater productivity and responsiveness. Monitoring KPIs such as cycle time, customer satisfaction, lead time, and team velocity will provide valuable insights to further enhance the organization's performance and deliver successful outcomes.

Product discovery

Frameworks

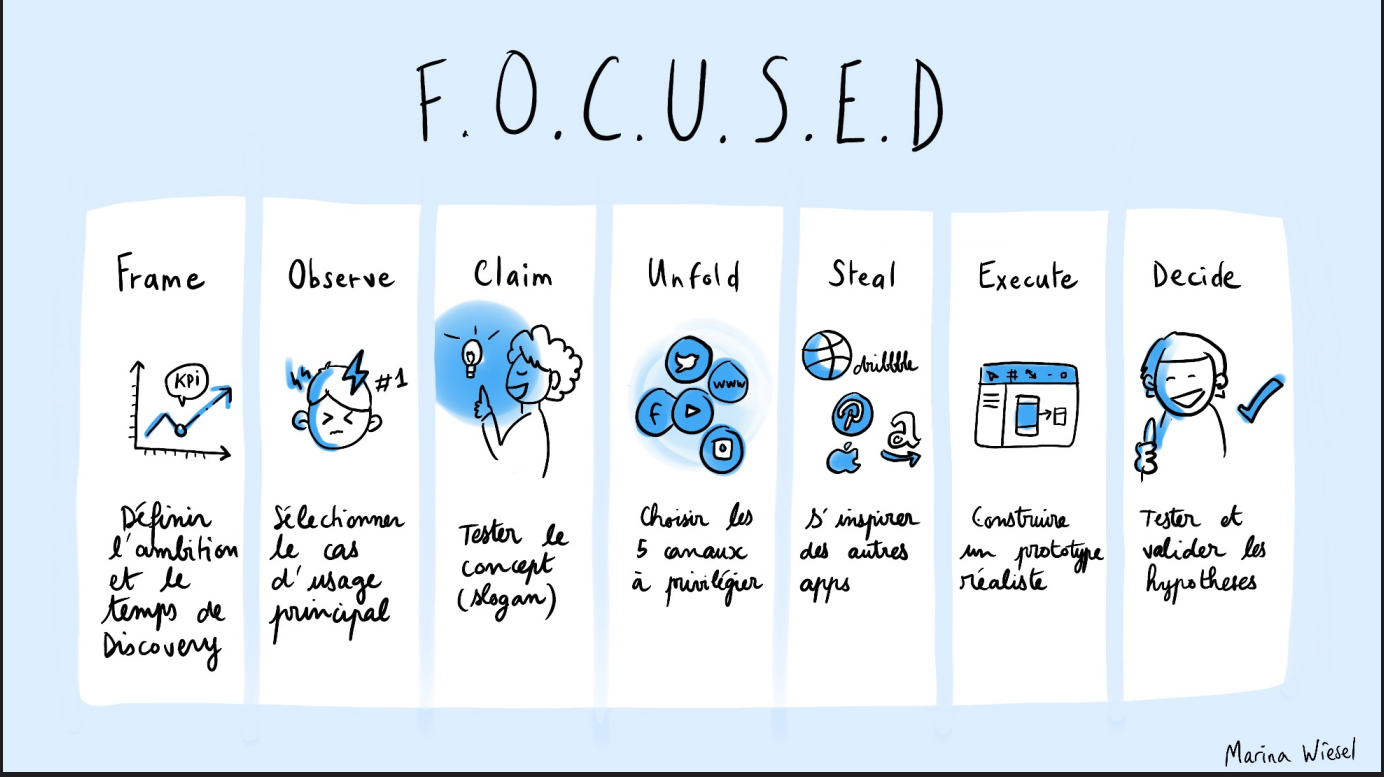
Double diamond

> The double diamond framework encourages product teams to approach problems and solutions by diverging (starting broad to enable creative collaboration) and then converging (narrowing down to one or two key areas of focus). It also provides a structure for understanding and defining user problems before jumping straight into a solution.



F.O.C.U.S.E.D

> FOCUSED is a user-oriented scoping methodology. It is a pragmatic approach. Over a short period of time, the team investigates a feature and tests its interest for the user. This reduces the development costs of a little-used feature. Product discovery is therefore complementary to product delivery.



Frame

🎯 Project ambition

- What will make us say that the project will be a success?
- Specific indicator "The Target”
- Level to be reached for the "Ambition" indicator
- Current level of this indicator "The Situation”

💥 Damage Control

- The risk we are willing to take
- Indicator
- Level to be reached
- Current level

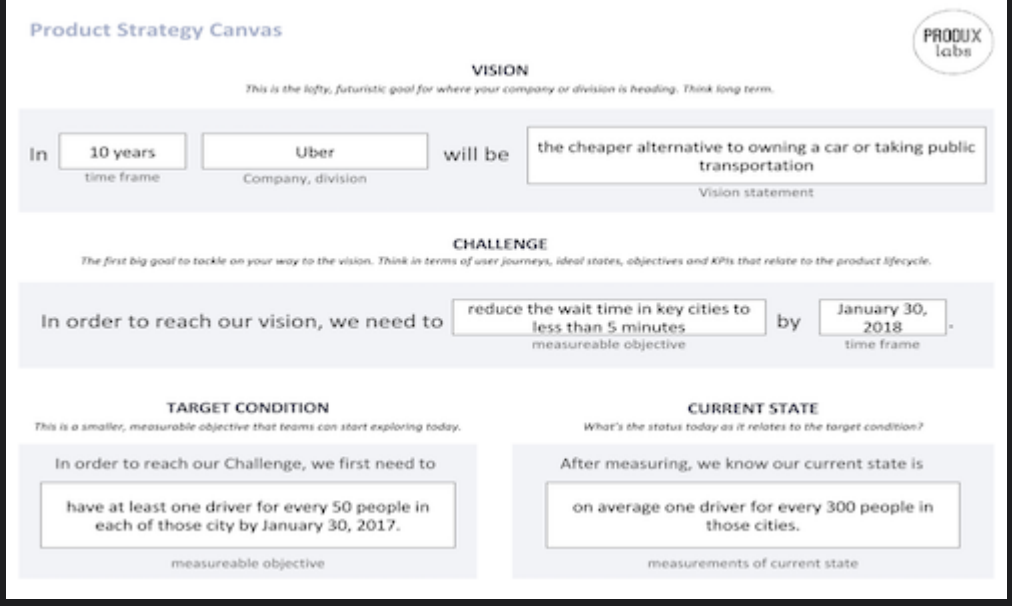
⏰ Time Box

> - How long can we spend on the discovery ?

Workshops & Tools

- Vision-project connection
- Reverse Brainstorm
- Observation of metrics over time
- Analysis of the history

Deliverables



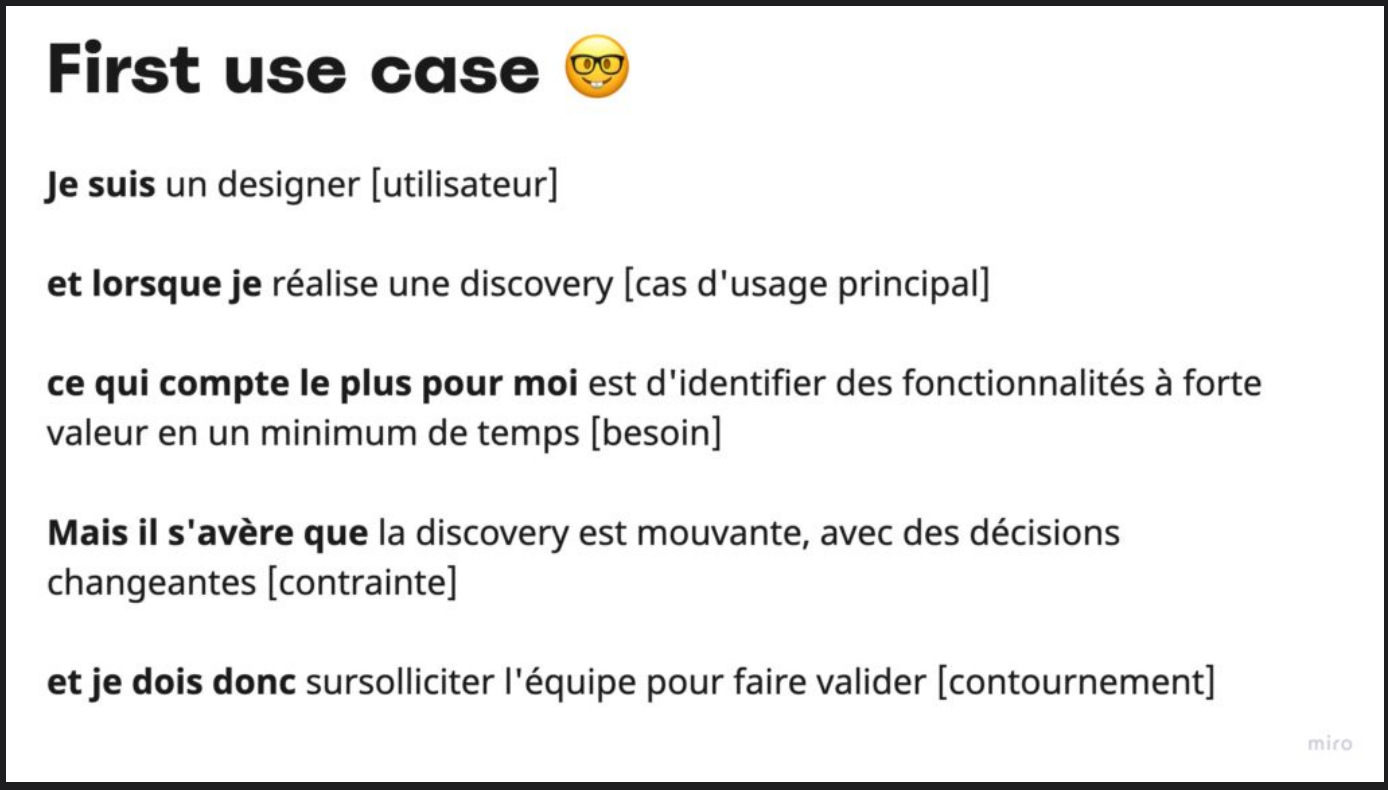
Observe

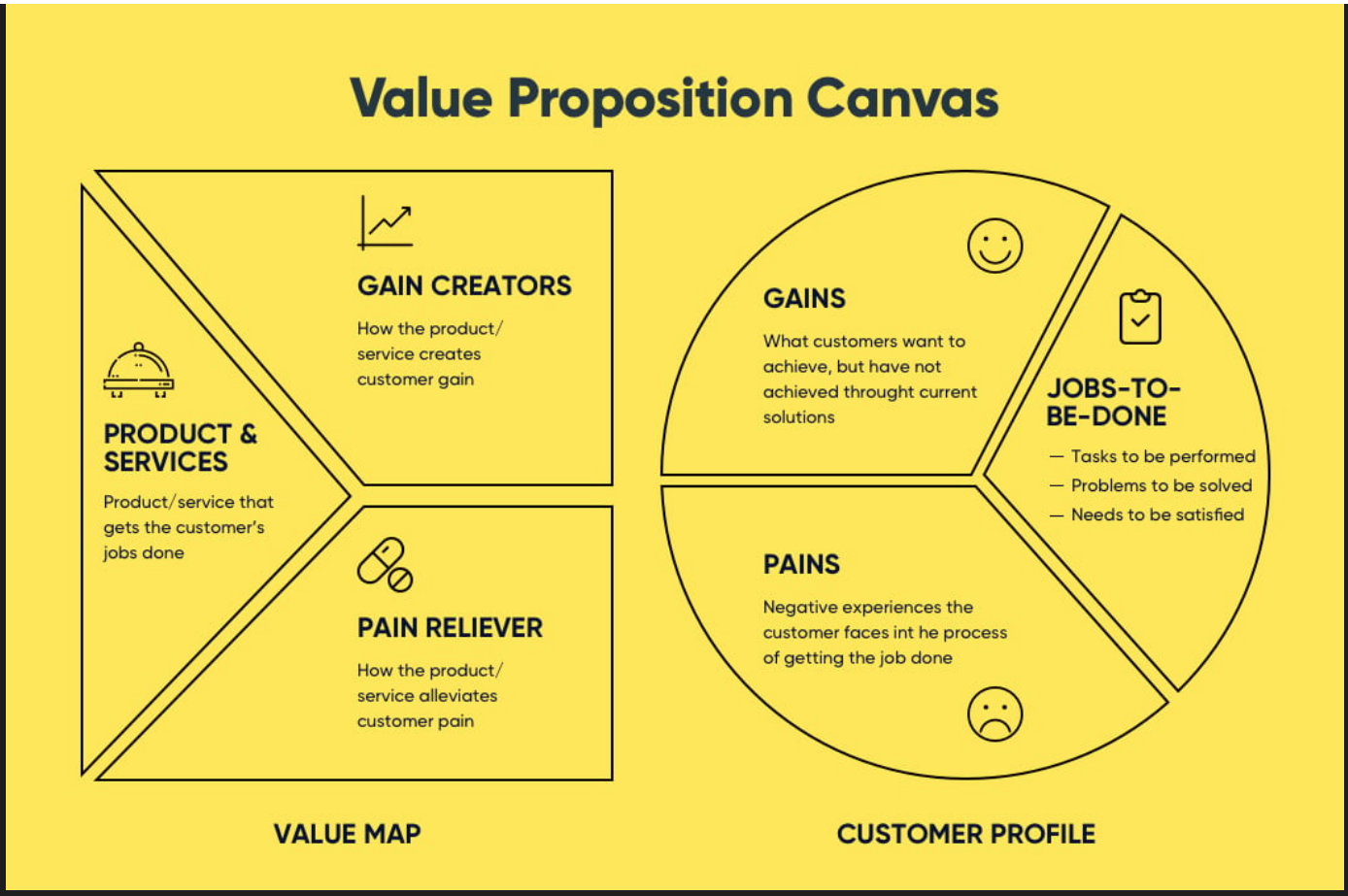
> Objectives of observe

Workshops & Tools

- Setting in situation
- Telephone interview
- Social networks and app stores
- Online questionnaire
- Customer service
- Home visit
- Empathy card
- Behavioral data

Deliverables





Claim

> Objectives of claim

Workshops & Tools

- Internal Benchmark
- External benchmark
- Brainstorming
- Press release
- Focus group

Deliverables

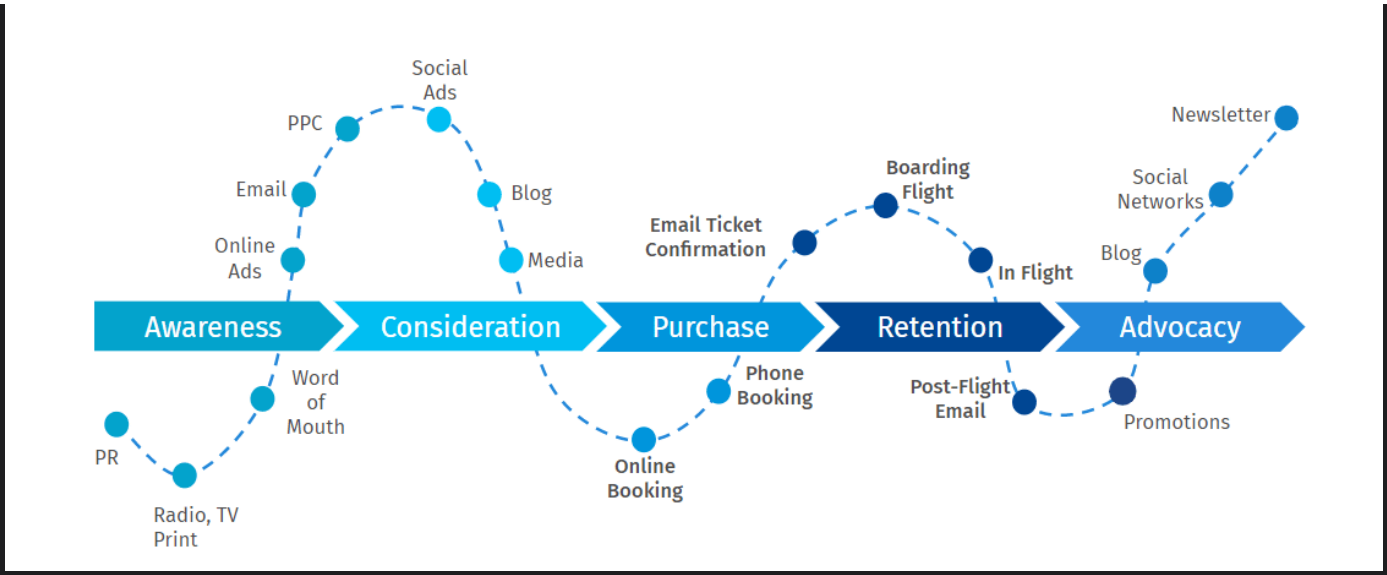
Unfold

> Objectives of unfold

Workshops & Tools

- Architecture
- Multi-channel mapping
- Before/after
- Customer journey

Deliverables



Steal

Execute

Decide

Additional resources

- F.O.C.U.S.E.D discovery process

Well Architecture Framework

5 pillars of Well Architected Framework

- Not something to balance, or trade-offs, they're a synergy
  - Help you track your performance and evolve your architecture.
- 5 pillars summary:
  1. **Operational Excellence**
     - **Description**: Run and monitor systems & continually improve supporting processes and procedures
     - **Design principles**
       - **Perform operations as code**: Infrastructure as Code
       - **Annotate documentation**: e.g. auto-document after every build)*
       - **Make frequent, small, reversible changes**
       - **Refine operations procedures frequently**: Ensure team members are familiar with it.
       - **Anticipate failure**: Learn from all operational failures
     - **Example AWS Services**
       - **Selection**: Auto Scaling, Lambda, EBS, S3, RDS
       - **Review**: AWS CloudFormation, AWS News Blog
       - **Monitoring**: CloudWatch, Lambda
       - **Tradeoffs**: Amazon RDS (vs Aurora), ElastiCache (read performance but stale), Snowball (a lot of data but takes a week), CloudFront (cache but stale)
  2. **Security**
     - **Description**: Protect information, systems, and assets through risk assessments and mitigation strategies
     - **Design principles**

    - **Implement a strong identity foundation**: Least privilege, IAM, centralize privilege management, eliminate reliance on long-term credentials
    - **Enable traceability**: Automatically respond to logs and metrics.
    - **Apply security at all layers**: Every instance, OS, and app, e.g. VPC, subnet, LB.
    - **Automate security best practices**: Encryption, tokenization, and access control.
    - **Protect data in transit and at rest**
    - **Keep people away from data**: Reduce the need for direct access or manual data processing
    - **Prepare for security**: Run incident response simulations and use tools with automation to increase your speed for detection, investigation, and recovery
  - **Example AWS Services**
  - **Identity and access management**: IAM, AWS-STS, MFA token, AWS Organizations
  - **Detective controls**: AWS Config, AWS CloudTrail, Amazon CLoudWatch
  - **Infrastructure Protection**
    - **Amazon CloudFront**: Defense against DDoS attack
    - Amazon VPC
    - **AWS Shield**: DDoS protection of AWS account
    - **AWS WAF**: Web Application Firewall
    - **Amazon Inspector**: for security of EC2 instance
  - **Data protection**: KMS, S3 (SSE, SSE-KMS, SSE-C, bucket policies etc.)
    - And other managed services such as Elastic Load Balancing (e.g. only HTTPS), Amazon EBS & RDS (SSL Capability)
  - **Incident Response**
    - **IAM**: Delete account or give it zero privilege
    - **CloudFormation**: If someone deletes entire structure, how to get back?
    - Amazon CloudWatch Events
3. **Reliability**
  - **Description**:
    - Ability of a system to recover from infrastructure or service disruptions
    - Dynamically acquire computing resources to meet demand
    - Mitigate disruptions such as misconfigurations or transient network issues.
  - **Design principles**

        - **Test recovery producers**: Use automation to simulate different failures or to recreate scenarios that led to failures before
        - **Automatically recover from failure**: Anticipate and remediate failures before they occur.
        - **Scale horizontally to increase aggregate system availability**: Distribute requests across multiple, smaller resources to ensure that they don't share a common point of failure
        - **Stop guessing capacity**: Maintain the optimal level to satisfy demand without over or under provisioning, use auto-scaling.
        - **Manage change in automation**: Use automation to make changes to infrastructure
    - **Example AWS Services**
        - **Foundations**
            - **IAM**: No one has too many rights to damage
            - Amazon VPC
            - **Service limits**: monitor limits so you don't get disruption.
            - **AWS Trusted Advisor**: e.g. look at service limits
        - **Change Management**: AWS Auto Scaling, Amazon CloudWatch, AWS CloudTrail, AWS Config
        - **Failure Management**: Backups, AWS CloudFormation (recreate whole infrastructure at once), Amazon S3, Amazon S3 Glacier, Amazon Route 53 (e.g. if application fails you redirect to another application)
4. **Performance**
    - **Description**: Adopt & provide best performance
    - **Design principles**
        - Democratize advanced technologies *(track new services)*
        - Go global in minutes *(easy multi-region deployment)*
        - Use serverless architectures *(avoid burden of managing servers)*
        - Experiment more often *(easy to carry out comparative testing)*
        - Mechanical sympathy *(be aware of all AWS services)*
    - **Example AWS Services**
        - **Prepare**: AWS CloudFormation, AWS Config
        - **Operate**: AWS CloudFormation, AWS Config, AWS CloudTrail, Amazon CloudWatch, AWS X-Ray

    - **Evolve**: AWS CloudFormation, CI/CD: CodeBuild, CodeCommit, CodeDeploy, CodePipeline
5. **Cost Optimization**
    - **Description**: Can run systems to deliver business value at the lowest price point.
    - **Design Principles**
        - **Adopt a consumption mode**: Pay only for what you use
        - **Measure overall efficiency**: Use CloudWatch
        - **Analyze and attribute expenditure**: Use tags, Accurate identification of system usage and costs → helps measure return on investment (ROI)
        - **Use managed and application level services to reduce cost of ownership**: As managed services operate at cloud scale, they can offer a lower cost per transaction or service
    - **Example AWS Services**
        - **Expenditure awareness**: AWS Budgets, AWS Cost and Usage Report, AWS Cost Explorer, Reserved Instance Reporting
        - **Cost-effective resources**: Spot instance, Reserved instance, Amazon S3 Glacier
        - **Matching supply and demand**: AWS Auto Scaling, AWS Lambda
        - **Optimizing over time**: AWS Trusted Advisor, AWS Cost and Usage Report, AWS News Blog

---

Well Architected Tool

- Tool
- Flow:
    1. Define a workload
    2. Do a review
        - Answer questions for each pillar.
        - E.g. for operational excellence
            - Question: How do you determine what your priorities are?
            - Answers: customer needs / compliance requirements / ... / none
    3. Optionally generate reports, milestones, improvement plans (with risks, e.g. "understand business needs").

Introduction

Template for product engineering teams documentation website

Product Roadmap

Microscope boilerplate - Opiniated solution boilerplate & guidelines for product engineering teams

V1

Initial setup of the boilerplate

Template

- [x] Setup dotnet template
- [x] Make docs a choice

Clients

- [x] Setup mudblazor UI
- [x] Setup Preferences & Settings
  - [x] Dark / Light mode
  - [x] Drawer open
  - [x] Language support
- [x] Setup PWA
- [x] Setup Authentication
- [x] Setup Feature management
- [x] BFF

    - [x] blazor hosted & SSR
    - [x] reverse proxy APIs (YARP)
- [ ] Setup Globalization

Services

TodoList

- [ ] Core
    - [x] Domain
        - [x] Setup aggregate root
        - [x] Setup entities
        - [x] Setup domain events
        - [x] Setup repository interface
        - [x] Setup exceptions
    - [ ] Application
        - [x] Common behaviours
        - [x] Mappings
        - [ ] App settings check
        - [ ] Features
            - [x] Todolist
                - [x] Commands
                    - [x] Create todo list
                    - [x] Delete todo list
                    - [x] Update todo list
                    - [x] Create todo item
                    - [x] Delete todo item
                    - [x] Toggle todo item
                - [x] Queries
                    - [x] Get all todo lists by user
                    - [x] Get todo list by id

        - [x] Policies
            - [x] Todolist created by policy requirement
        - [x] Events
            - [x] SendMailOnTodoListCompleted
            - [x] OnTodoListCompletedIntegrationEvent
    - [ ] Upload
- [ ] Infrastructure
    - [ ] Persistence
        - [x] Entity framework
            - [x] EF Entities configuration
            - [x] EF Migration
        - [ ] MartenDB
    - [ ] External systems implementation
        - [x] Storage
        - [x] User
        - [x] Mail
        - [ ] AI Prompting
        - [ ] PDF
    - [ ] Bus
        - [x] MassTransit
            - [ ] OpenTelemetry
- [ ] Interface
    - [x] GraphQL API
    - [x] REST API
    - [x] Authentication
        - [x] OPENID JWT
        - [x] MASTER KEY
    - [x] Authorization
    - [x] HealthCheck
    - [x] Feature management
    - [x] Auto migration option

    - [ ] SignalR websocket use case
    - [ ] OpenTelemetry
- [ ] Tests
    - [x] Unit tests
        - [x] Setup Unit tests
            - [x] Todolist tests
    - [ ] Integration tests
    - [x] Architecture tests

Storage (optional) ?

- [x] Azure blob storage
- [x] Minio
- [x] AWS S3
- [x] File system

Workflow (optional)

- [ ] Elsa core

Scheduled Jobs (optional)

- [ ] Hangfire

Cross cutting

- [x] SharedKernel
    - [x] use mediatr contract only

IAC

- [ ] docker-compose
    - [x] postgres

    - [x] keycloak
        - [ ] import realms configuration - bug
    - [x] Service Bus RabbitMQ
    - [ ] Internal services
- [ ] Azure biceps / ARM
- [ ] K8S

Docs

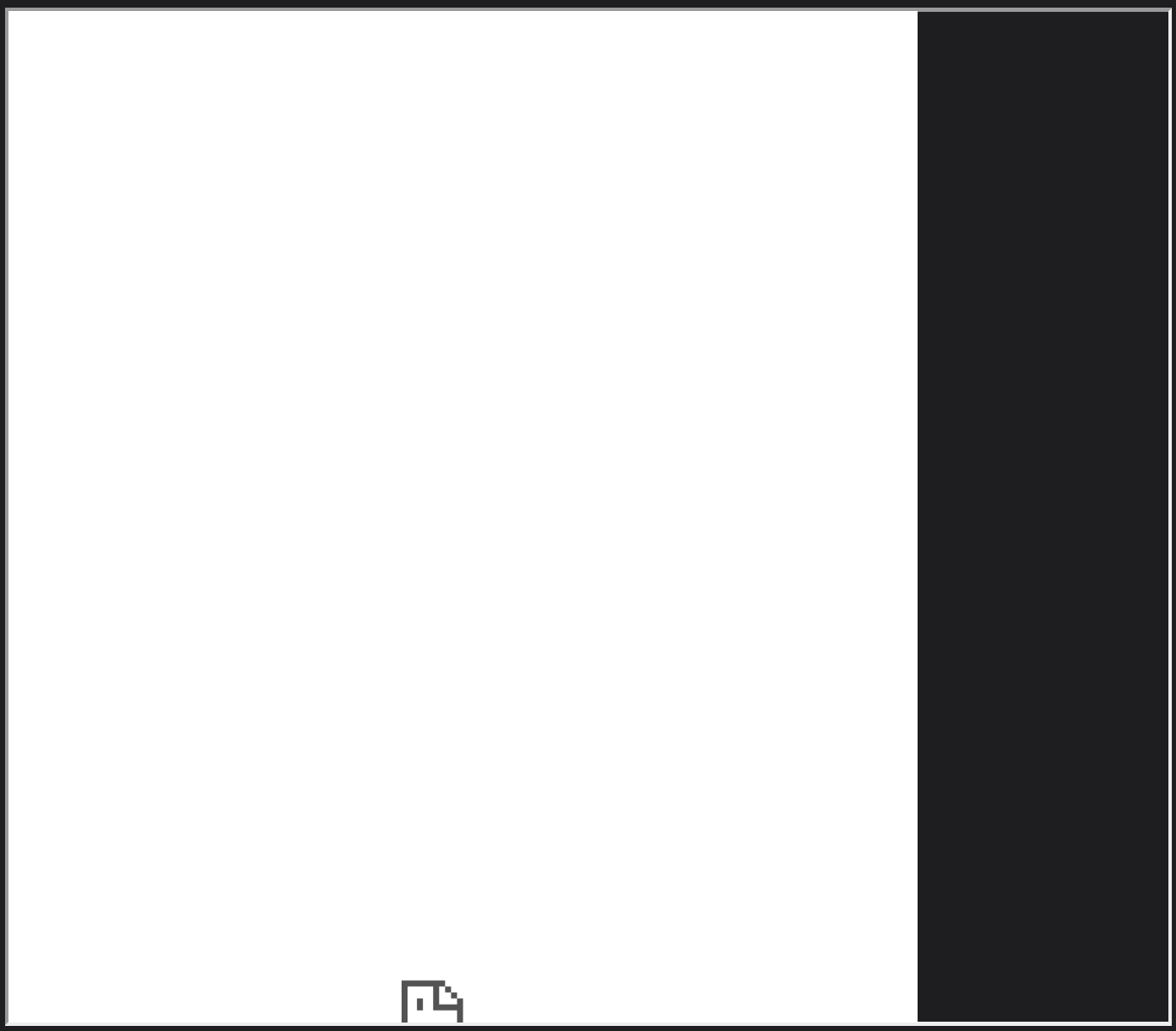
- [x] Setup vitepress
    - [x] Mermaid
    - [x] PDF export
    - [x] Task list
- [x] Setup revealjs slides

Roadmap (sample)

mermaid

```
gantt
    title Product roadmap
    dateFormat YYYY-MM-DD
    section B2C
        Feature 1          :a1, 2023-01-01, 30d
        Feature 2    :after a1, 20d
    section B2B
        Feature 1 :2023-01-12, 12d
        Feature 2    :24d
```

Slides



Code language ADR (sample)

- 📅 Date: 2023/01/11
- 👷 Decision made by: Product Owner, Developers

Context

Business objects have French names, but the execution team speaks English. The question is which language to use in our codebase.

Considered Options 💡

French ✅ Pro: We use the same words as the business team, making code reviews with them easier. 🚫 Con: The execution team does not understand it.

English ✅ Pro: The execution team will understand it. 🚫 Con: The business team will have to translate, and we might need to think a bit more while coding.

Decision 🏆

We choose English because it is easier to translate from English to French when you know both languages than from French to English when you don't know French.

Consequence

We decide to implement a translation dictionary with codes to aid the translation process. The template used for this example can be found on GitHub, and examples of more complex decisions are also provided.

Architecture Decision Record

▶ 2023

▶ 2022

Weekly meeting

▶ 2023

▶ 2022

Weekly

- 📅 Date : 2022/01/11
- 👷 Attendees : John doe, xxx

SQUAD XXX

🏆 Success / Deliverables / Milestones

- prod updated
- new FEATURE implemented
- new EPIC done

🚫 Problems / Blockages

- prod updated
- new FEATURE implemented
- new EPIC done

🛟 Enabling team needs

- Architecture department provisioning container registry
- Design system team implement new design tokens
- Marketing provide analytics platform api key

✓ Action plan

- [x] todo 1
- [ ] todo 2

    - [x] todo 2 bis
    - [ ] todo 2 tier

SQUAD XXX

🏆 Success / Deliverables / Milestones

- prod updated
- new FEATURE implemented
- new EPIC done

🚫 Problems / Blockages

- prod updated
- new FEATURE implemented
- new EPIC done

🛟 Enabling team needs

- Architecture department provisioning container registry
- Design system team implement new design tokens
- Marketing provide analytics platform api key

✓ Action plan

- [x] todo 1
- [ ] todo 2
    - [x] todo 2 bis
    - [ ] todo 2 tier

Squad B2B

🚀 Vision & Mission

> squad mission here lorem ipsum

🎁Products

- B2B Web app
- B2B subscription portal

🗄️Organisation

- Product Roadmap
- Agile governance
  - Daily meeting
  - Weekly tribe

👨‍👧‍👦Team members

- John doe - Product Manager
- John doe - Product Designer
- John doe - Teach Lead
- John DOE - Frontend software Engineer

- John DOE - Backend software Engineer
- John DOE - QA Engineer

📄Documentation

- link here
- link here

🚀On boarding

- link here
- link here

Squad B2C

🚀 Vision & Mission

> squad mission here lorem ipsum

🎁Products

- B2C Web app
- B2C Mobile app

🗄️Organisation

- Product Roadmap
- Agile governance
  - Daily meeting
  - Weekly tribe

👨‍👧Team members

- John doe - Product Manager
- John doe - Product Designer
- John doe - Teach Lead
- John doe - Backend Software Engineer

- John doe - Frontend Software Engineer
- John doe - Mobile Software Engineer
- John doe - QA Engineer

📄Documentation

- link here
- link here

🚀On boarding

- link here
- link here

Squad PLATFORM

🚀 Vision & Mission

> squad mission here lorem ipsum

🗄️Organisation

- Product Roadmap
- Agile governance
  - Daily meeting
  - Weekly tribe

👨‍👧Team members

- John DOE - Principal Architect
- John DOE - Cloud Architect / DevOps Engineer

📄Documentation

- link here
- link here

🚀On boarding

- link here
- link here

Squad MANAGEMENT

🚀 Vision & Mission

> squad mission here lorem ipsum

👨‍👧Team members

- John DOE - CPTO
- John DOE - Engineering manager
- John DOE - Group Product Manager

📄Documentation

- link here
- link here

🚀On boarding

- link here
- link here

Product Engineering Organization

Squads

- Squad B2C
- Squad B2B
- Architecture
- Management

Organization diagram

```mermaid
graph TD;
    CPTO-->ENGINEERING_MANAGER;
    CPTO-->GROUP_PM;

    subgraph SquadB2C
        PM1
        UX1
        TL1
        ENGINEER1
        ENGINEER2
        ENGINEER3
    end

    subgraph SquadB2B
        PM2
        UX2
        TL2
        ENGINEER4
```

```
    ENGINEER5
end

subgraph Architecture
    Principal
    DevOPS
end

CPTO-->UX1;
CPTO-->UX2;
CPTO-->Principal;
CPTO-->DevOPS;
GROUP_PM-->PM1;
GROUP_PM-->PM2;
ENGINEERING_MANAGER-->TL1;
ENGINEERING_MANAGER-->TL2;
ENGINEERING_MANAGER-->ENGINEER1;
ENGINEERING_MANAGER-->ENGINEER2;
ENGINEERING_MANAGER-->ENGINEER3;
ENGINEERING_MANAGER-->ENGINEER4;
ENGINEERING_MANAGER-->ENGINEER5;
```

Product discovery notes & deliverables

- [ ] Todo App discovery

Todo app discovery (sample)

Frame

🎯 Project ambition

> - What will make us say that the project will be a success?

✔️ **The todo-app module will be considered a success if:** 75% of internal users use this system every day for their task management.

💥 Damage Control

> - Level to be reached

its only a sample so ... chill 😉

⏰ Time Box

> - How long can we spend on the discovery ?

1 hour

Observe

🤓 First use case

As a {DOMAIN_USER}, **when** organizing my workdays, **what matters most is** being efficient and thorough. **However, it turns out that** the local tools on my computer are not user-friendly and accessible from any device, **so I have** to rely on my computer to access my schedule.

Claim

> A product launch tweet of max 280 characters

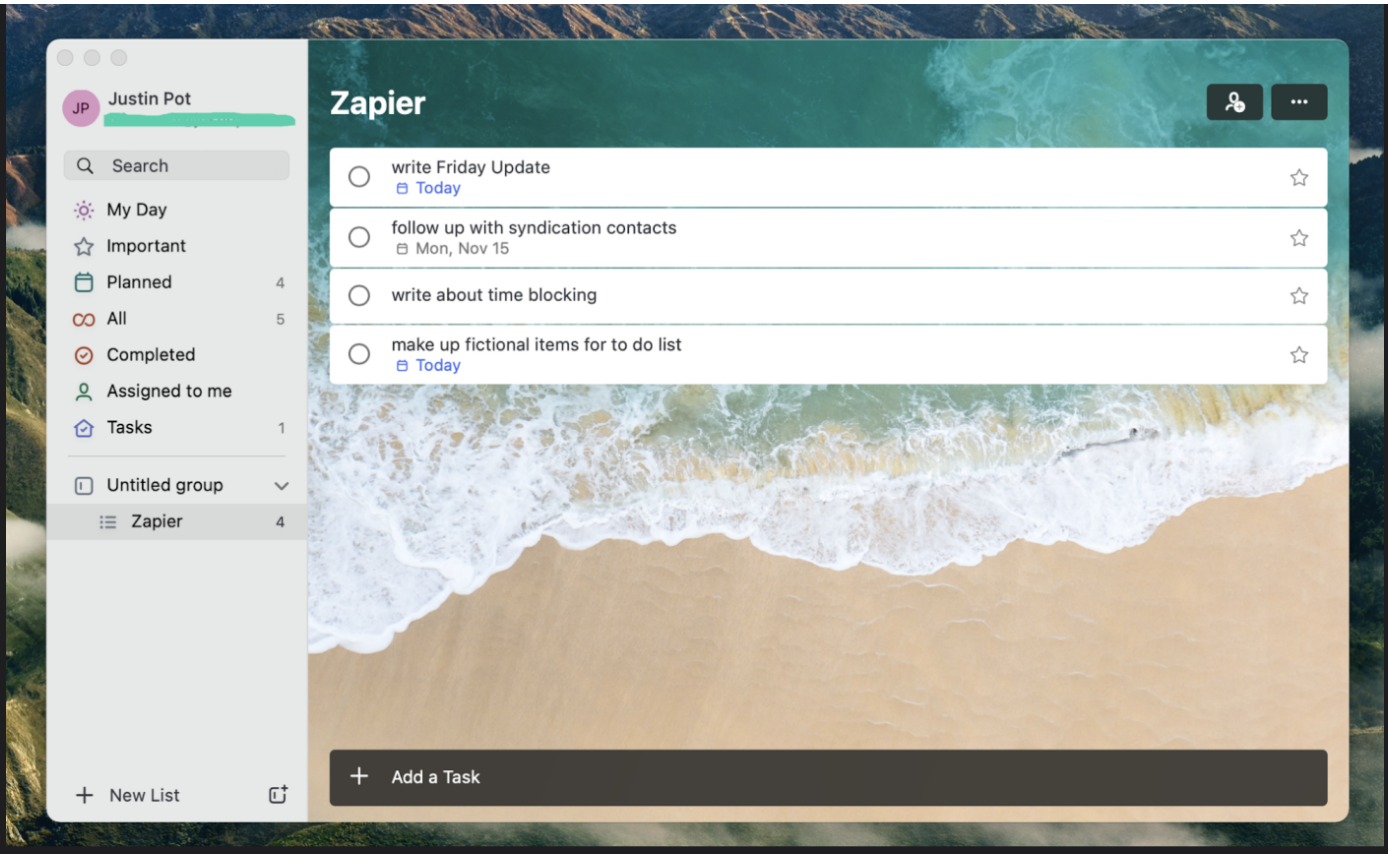
Unfold

> 5 touch points / customer journey

```mermaid
journey
    title My working day
    section Check list of job to be done
      Create todolist: 5: Me, Manager
      Do work: 1: Me
      Update todolist: 3: Me, Manager
    section Go home
      Go downstairs: 5: Me
      Sit down: 5: Me
```

Steal

> Gold nuggets

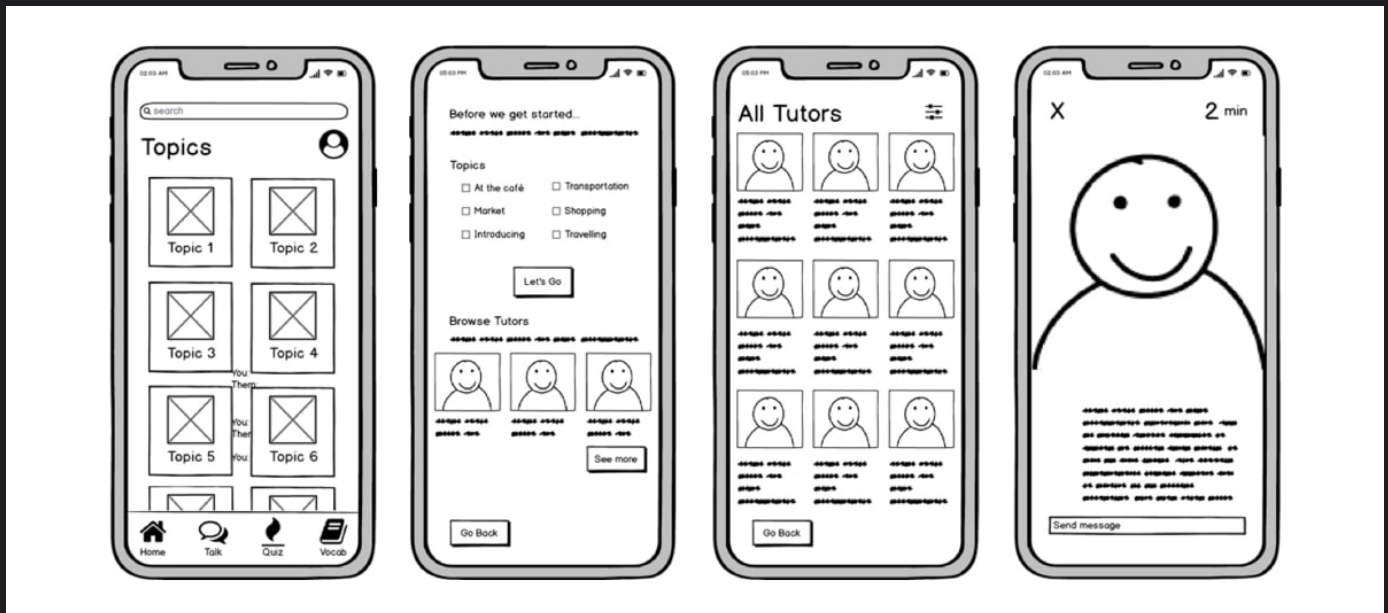


Add 3 more

- Link to golden nuggets image

Execute

> prototyping



- Link to prototype (figma, sketch, xd, ...)

Decide

> Go / NoGo decision

using mermaid.js & C4 template

Architecture diagram

mermaid

```
C4Context
      title System Context diagram for Internet Banking System
      Enterprise_Boundary(b0, "BankBoundary0") {
        Person(customerA, "Banking Customer A", "A customer of the bank, with persona
        Person(customerB, "Banking Customer B")
        Person_Ext(customerC, "Banking Customer C", "desc")

        Person(customerD, "Banking Customer D", "A customer of the bank, <br/> with p

        System(SystemAA, "Internet Banking System", "Allows customers to view informa

        Enterprise_Boundary(b1, "BankBoundary") {

          SystemDb_Ext(SystemE, "Mainframe Banking System", "Stores all of the core b

          System_Boundary(b2, "BankBoundary2") {
            System(SystemA, "Banking System A")
            System(SystemB, "Banking System B", "A system of the bank, with personal
          }

          System_Ext(SystemC, "E-mail system", "The internal Microsoft Exchange e-mai
          SystemDb(SystemD, "Banking System D Database", "A system of the bank, with

          Boundary(b3, "BankBoundary3", "boundary") {
            SystemQueue(SystemF, "Banking System F Queue", "A system of the bank.")
            SystemQueue_Ext(SystemG, "Banking System G Queue", "A system of the bank,
          }
        }
```

```
    }

    BiRel(customerA, SystemAA, "Uses")
    BiRel(SystemAA, SystemE, "Uses")
    Rel(SystemAA, SystemC, "Sends e-mails", "SMTP")
    Rel(SystemC, customerA, "Sends e-mails to")

    UpdateElementStyle(customerA, $fontColor="red", $bgColor="grey", $borderColor="
    UpdateRelStyle(customerA, SystemAA, $textColor="blue", $lineColor="blue", $offs
    UpdateRelStyle(SystemAA, SystemE, $textColor="blue", $lineColor="blue", $offset
    UpdateRelStyle(SystemAA, SystemC, $textColor="blue", $lineColor="blue", $offset
    UpdateRelStyle(SystemC, customerA, $textColor="red", $lineColor="red", $offsetX

    UpdateLayoutConfig($c4ShapeInRow="3", $c4BoundaryInRow="1")
```

Entity Relation Diagram

mermaid

```
---
title: Order example
---
erDiagram
    CUSTOMER ||--o{ ORDER : places
    ORDER ||--|{ LINE-ITEM : contains
    CUSTOMER }|..|{ DELIVERY-ADDRESS : uses
```

Customer journey

mermaid

```
journey
    title My working day
    section Go to work
        Make tea: 5: Me
        Go upstairs: 3: Me
        Do work: 1: Me, Cat
    section Go home
```

```
        Go downstairs: 5: Me
        Sit down: 5: Me
```

Event storming

mermaid

```mermaid
graph TD;

classDef aggregate fill:#fdfd9d
classDef command fill:#45abef
classDef readModel fill:#77dd77
classDef event fill:#ffb853
classDef policy fill:#c14bc0
classDef external fill:#f8b1f5
classDef actor fill:transparent

ReadModel:::readModel --> Actor:::actor
Actor:::actor --> Command:::command
Command:::command --> Aggregate:::aggregate
Command:::command --> ExternalSystem:::external
Aggregate:::aggregate --> Event:::event
ExternalSystem:::external --> Event:::event
Event:::event --> Policy:::policy
Event:::event --> ReadModel:::readModel
Policy:::policy --> Command:::command
```

VitePress

Microscope
Boilerplate

Product Engineering Starter Kit for CPTO

Getting started | Roadmap

Microservices

microservices architecture boilerplate

dotnet 7

dotnet 7 SDK

Blazor

Blazor WASM hosted as web frontend

IAM

Keycloak as Identity & Access Management service

Postgres

Postgres as database

Documentation as code

Vitepress & revealjs as documentation static website & slides

Product discovery templates

F.O.C.U.S.E.D discovery framework notes markdown templates

Organization & governance templates

Weekly meeting, architecture decision record markdown templates

Architecture documentation

Architecture schema as mermaid diagram, bounded context & aggregate canvas markdown templates