Microscope
Boilerplate

Opiniated solution boilerplate & guidelines for product engineering teams

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

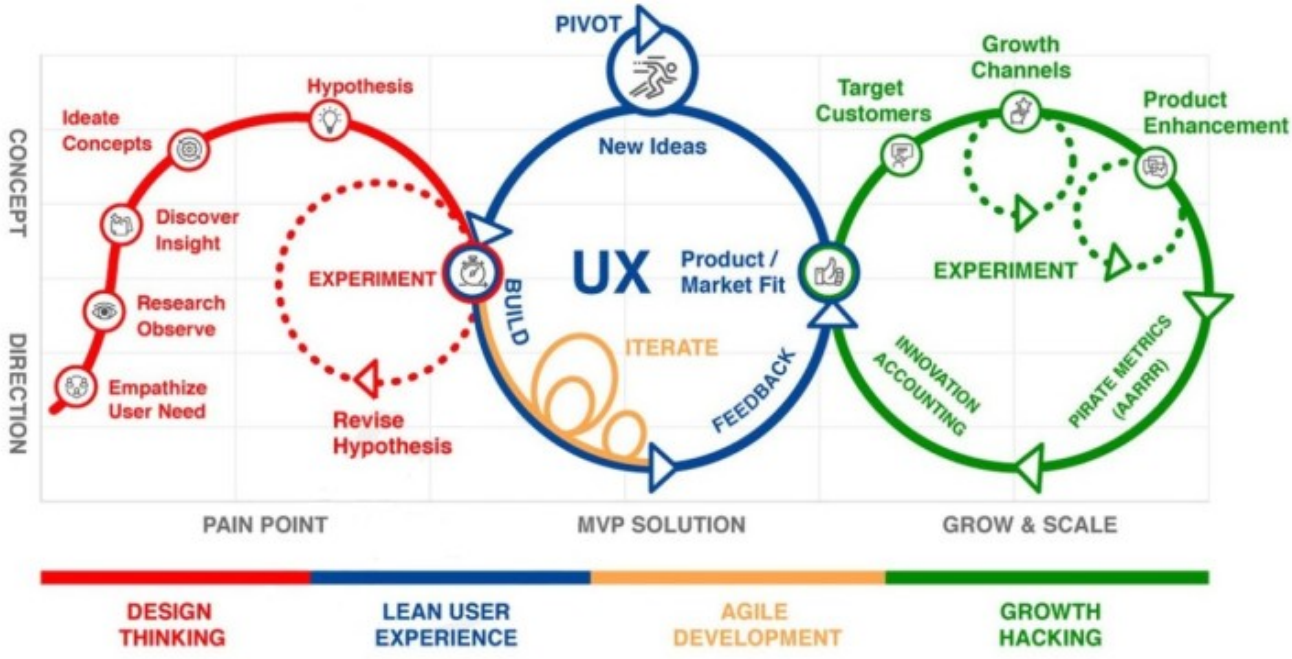
Docs

Vitepress as documentation static website

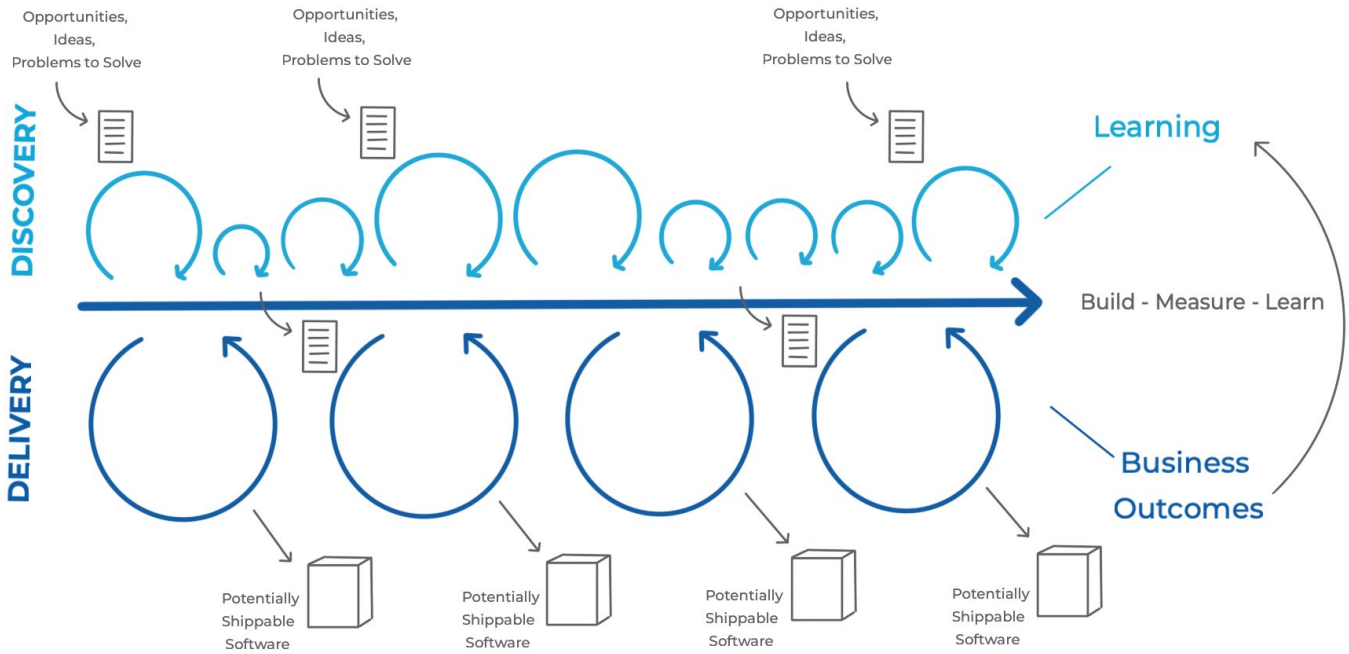
Getting started

Opiniated solution boilerplate for software engineers

Product discovery



Product discovery & delivery

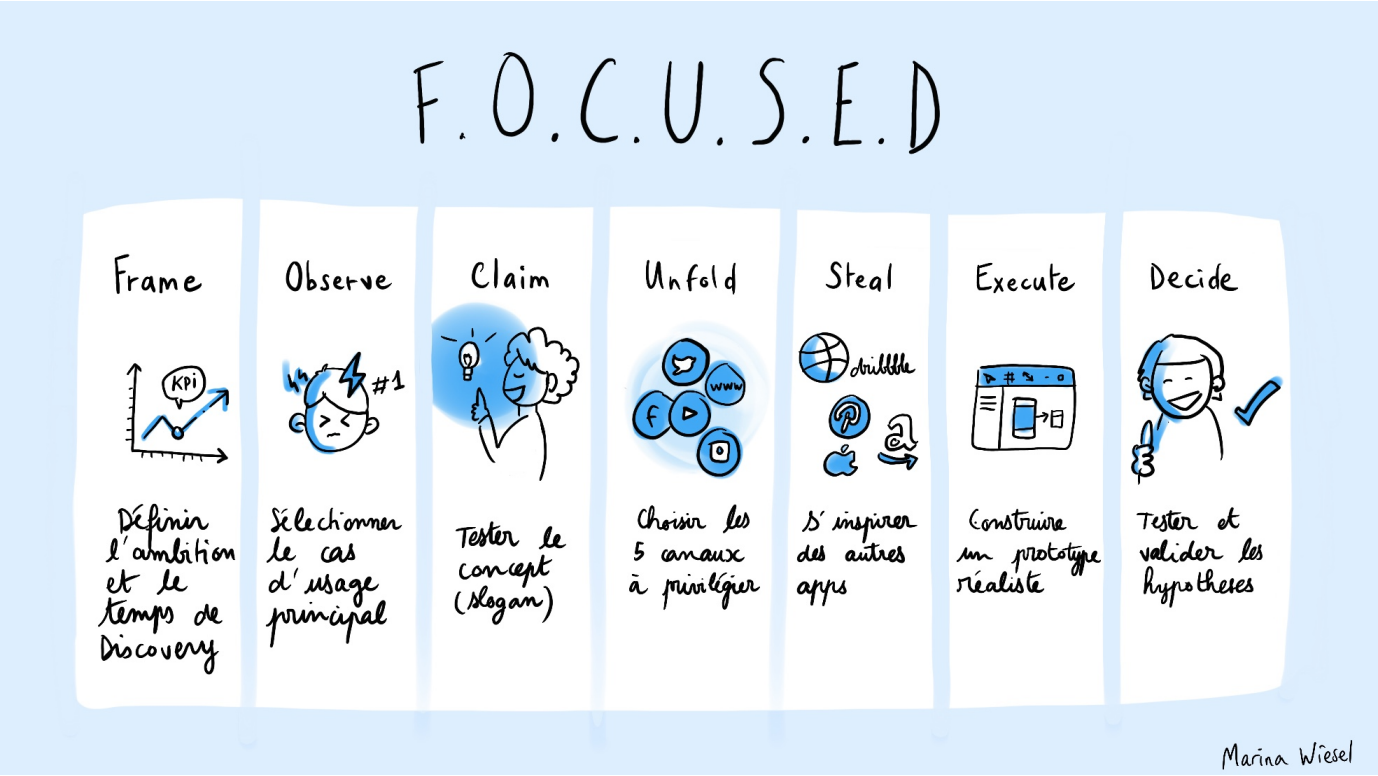


Product discovery

Product discovery

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

Il s'agit d'une méthodologie de cadrage orientée utilisateur. C'est une approche pragmatique. Sur un temps court, l'équipe investigue une fonctionnalité et teste son intérêt pour l'utilisateur. Cela permet de diminuer les coûts de développement d'une fonctionnalité peu utilisée. La product discovery est donc complémentaire au product delivery.



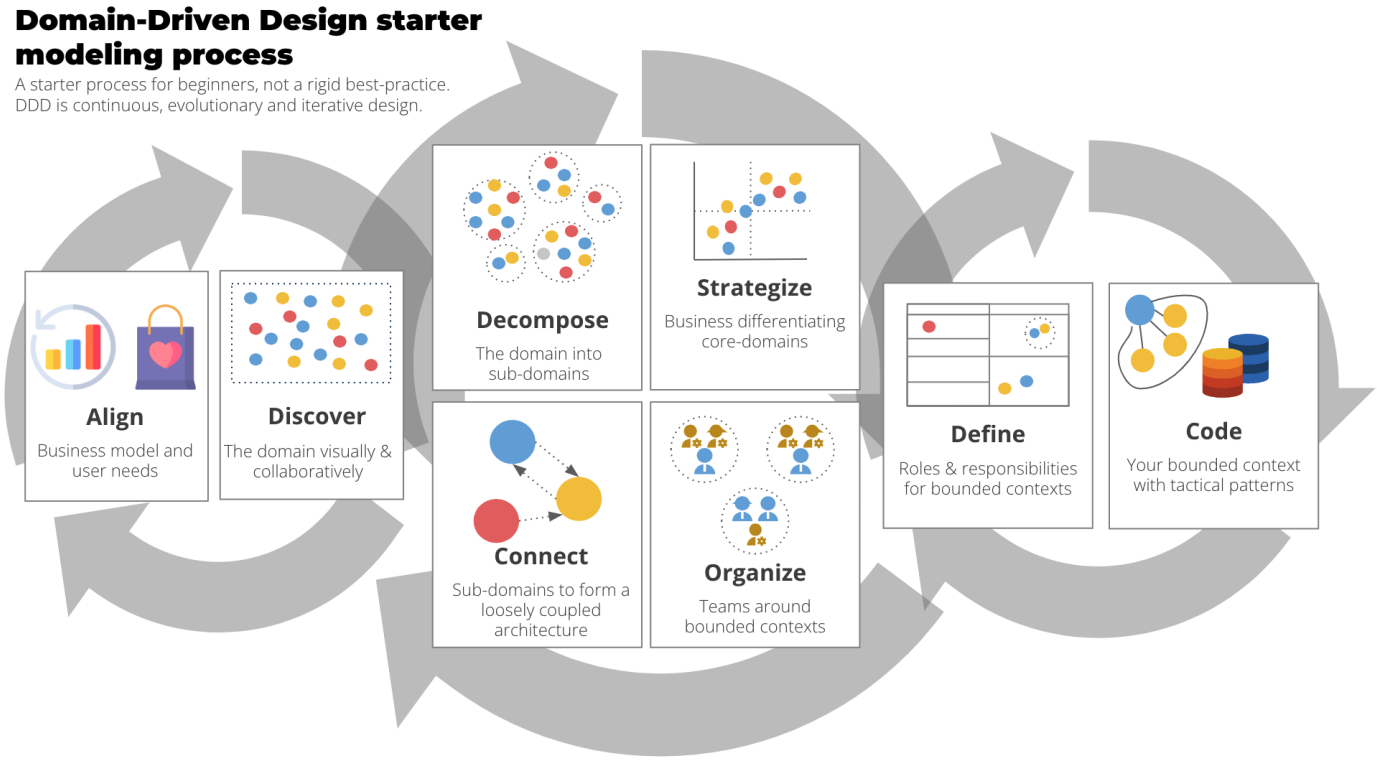
Domain-Driven Design Starter Modelling Process

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.

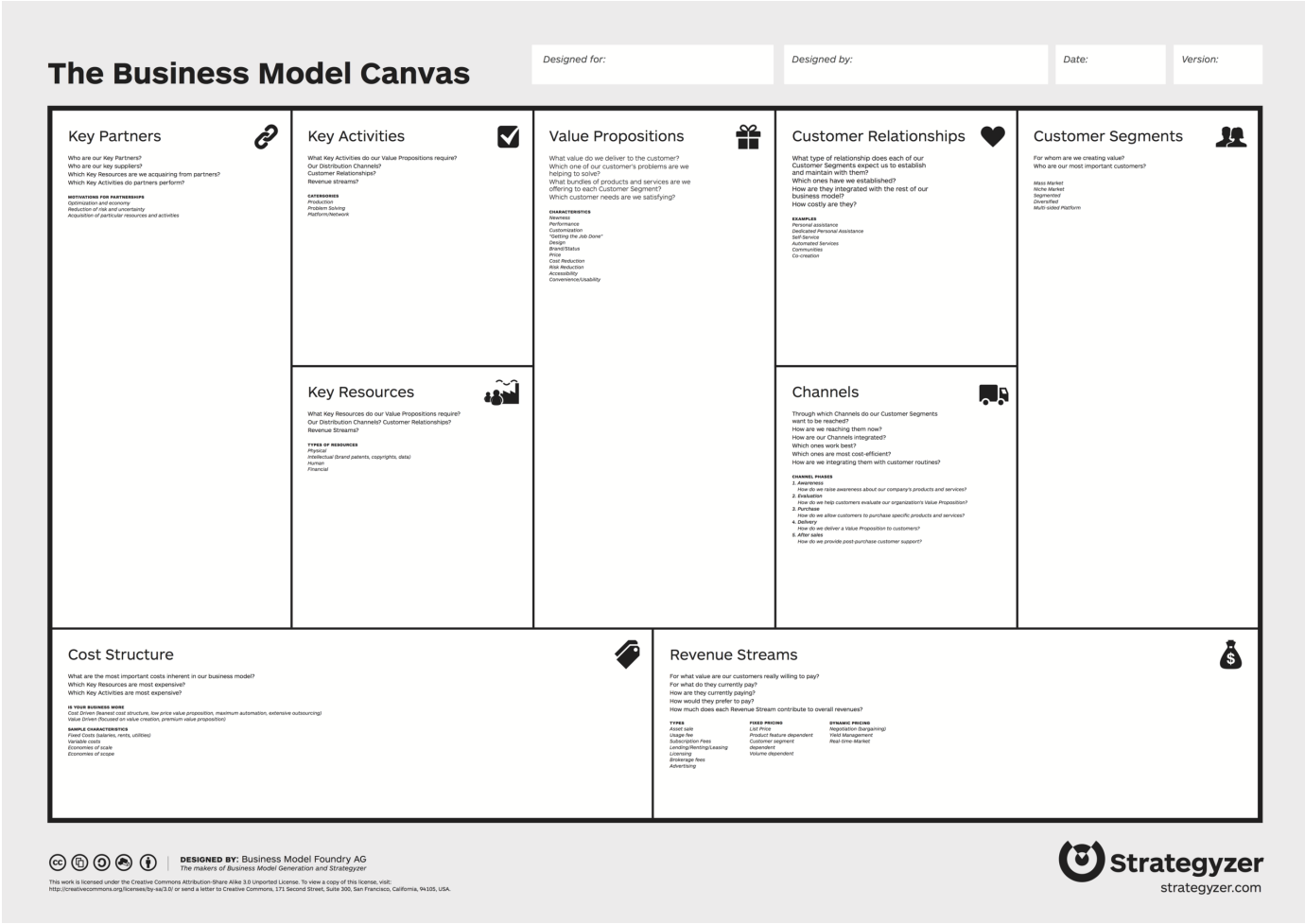
Understand

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

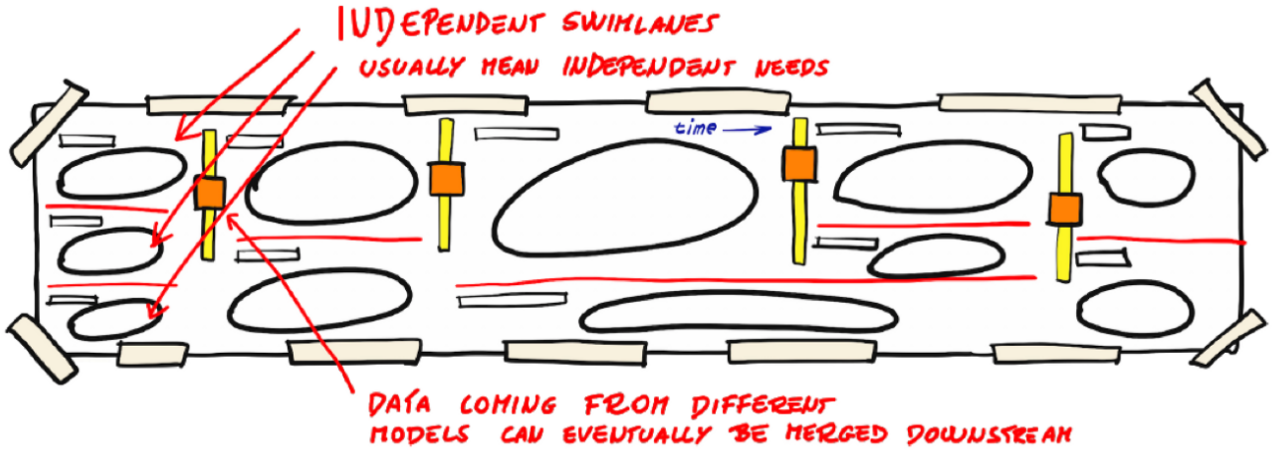
Decompose

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

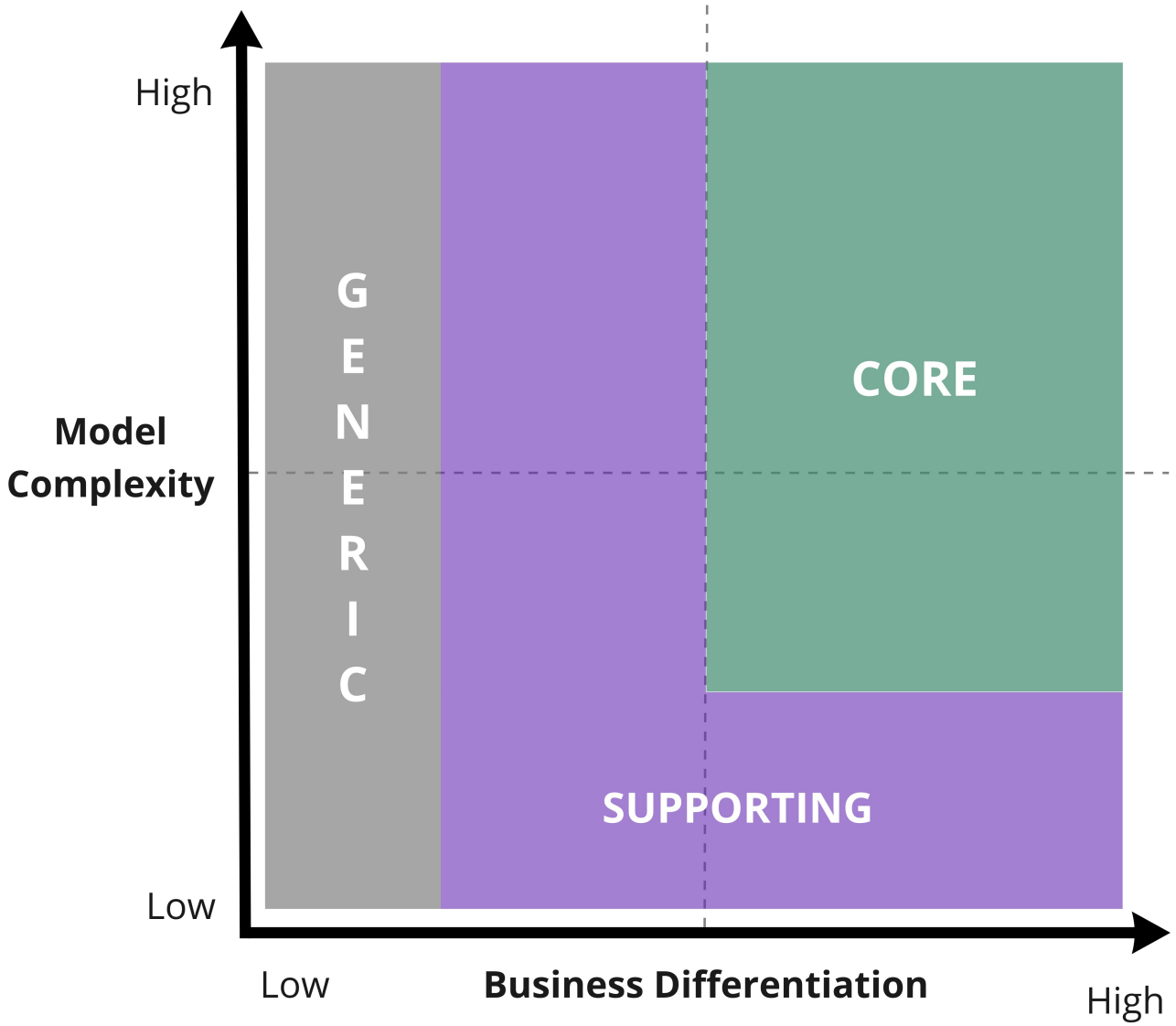
Strategize

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

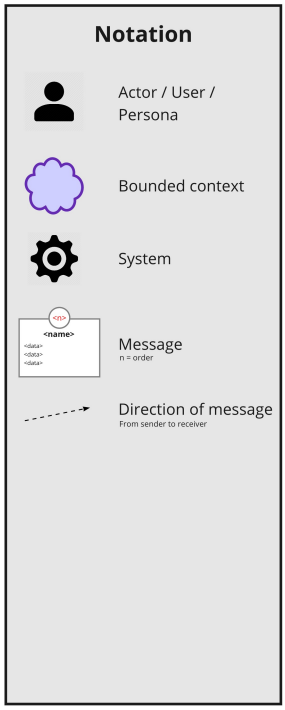
- People who understand product and business strategy
- People who design, build, test software
- People who have domain knowledge

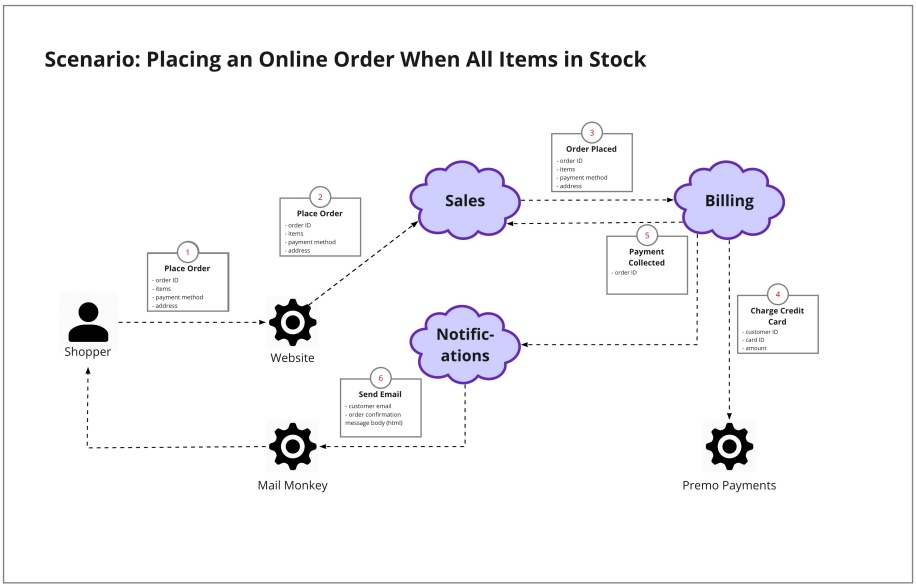
Connect

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

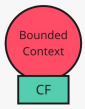
Context Map Patterns

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.

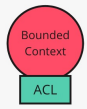
Conformist

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.

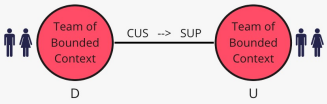


Shared Kernel

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

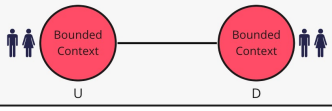
Two software artifacts or systems in two bounded contexts need to be delivered together to be successful and work. There is often a close, reciprocal link between data and functions between the two systems.

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

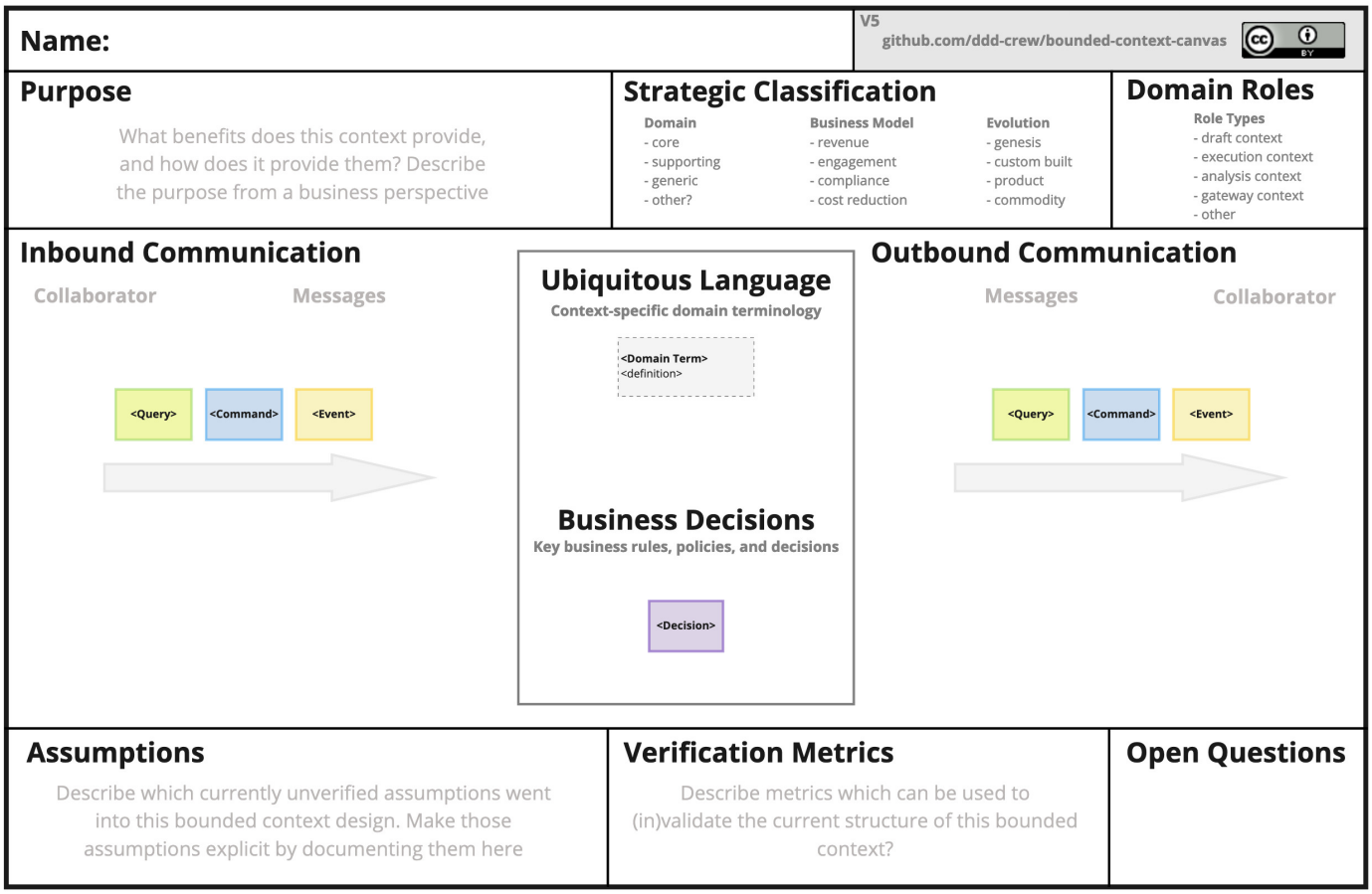
Define

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

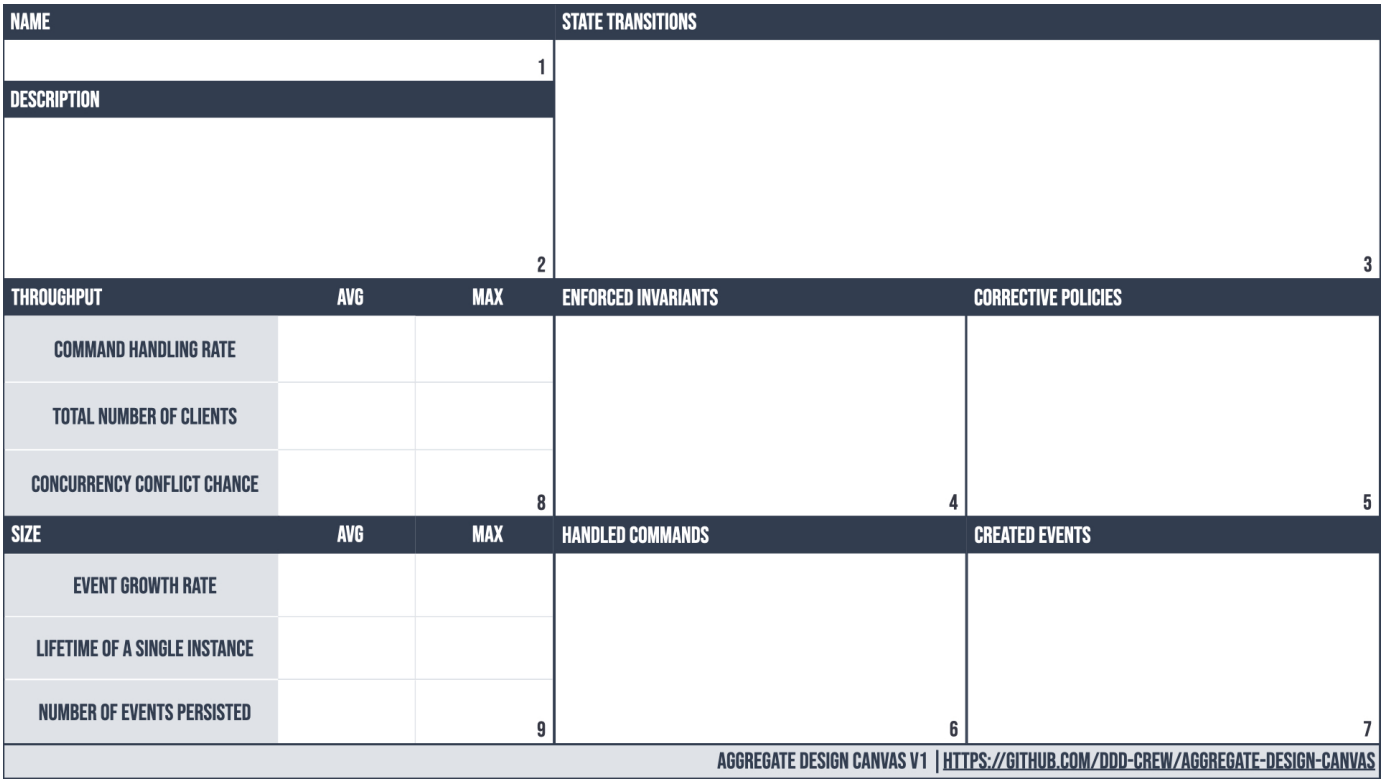
- People who design, build, test software
- People who have domain knowledge
- People who are responsible for the product

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

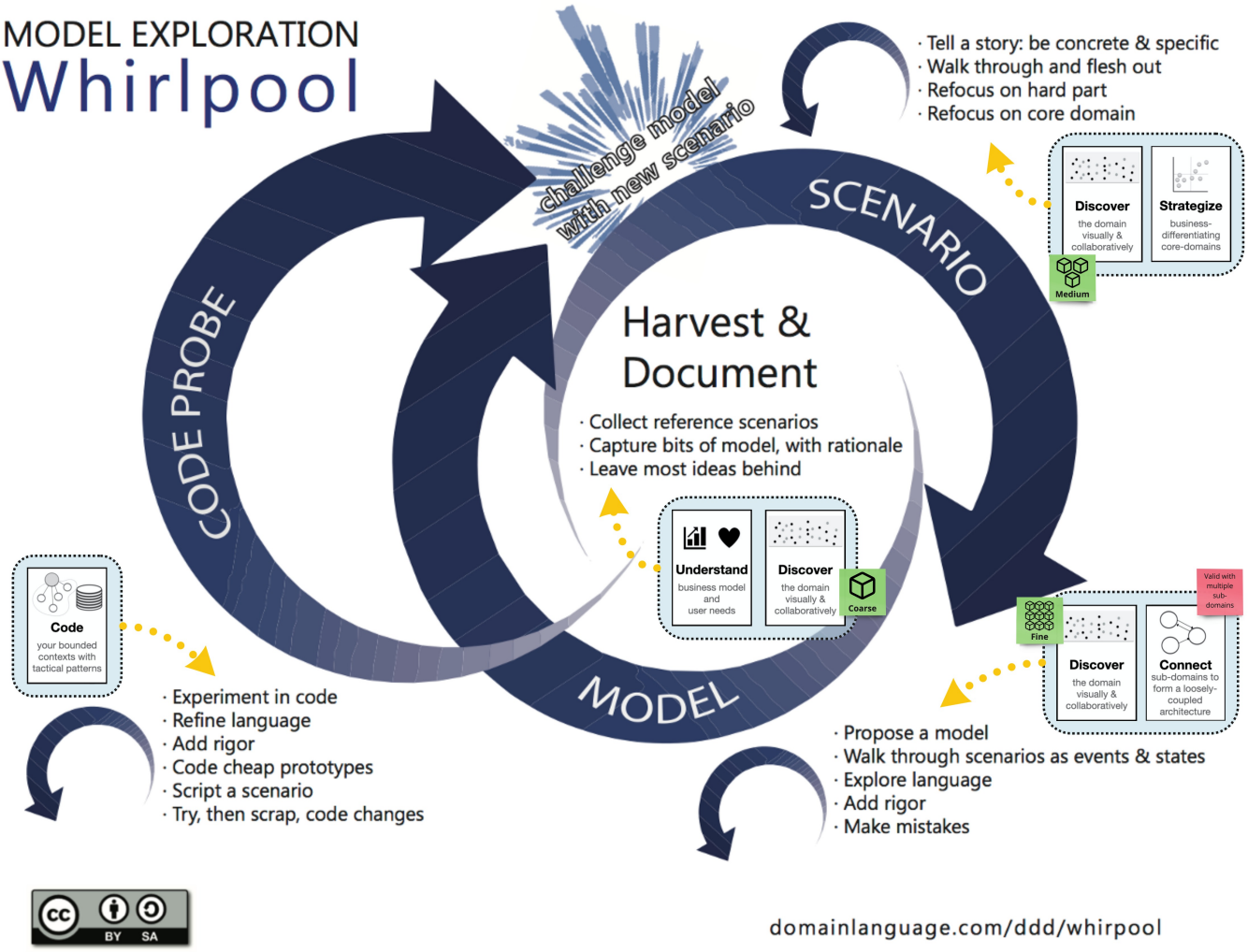
- People who design, build, test software

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

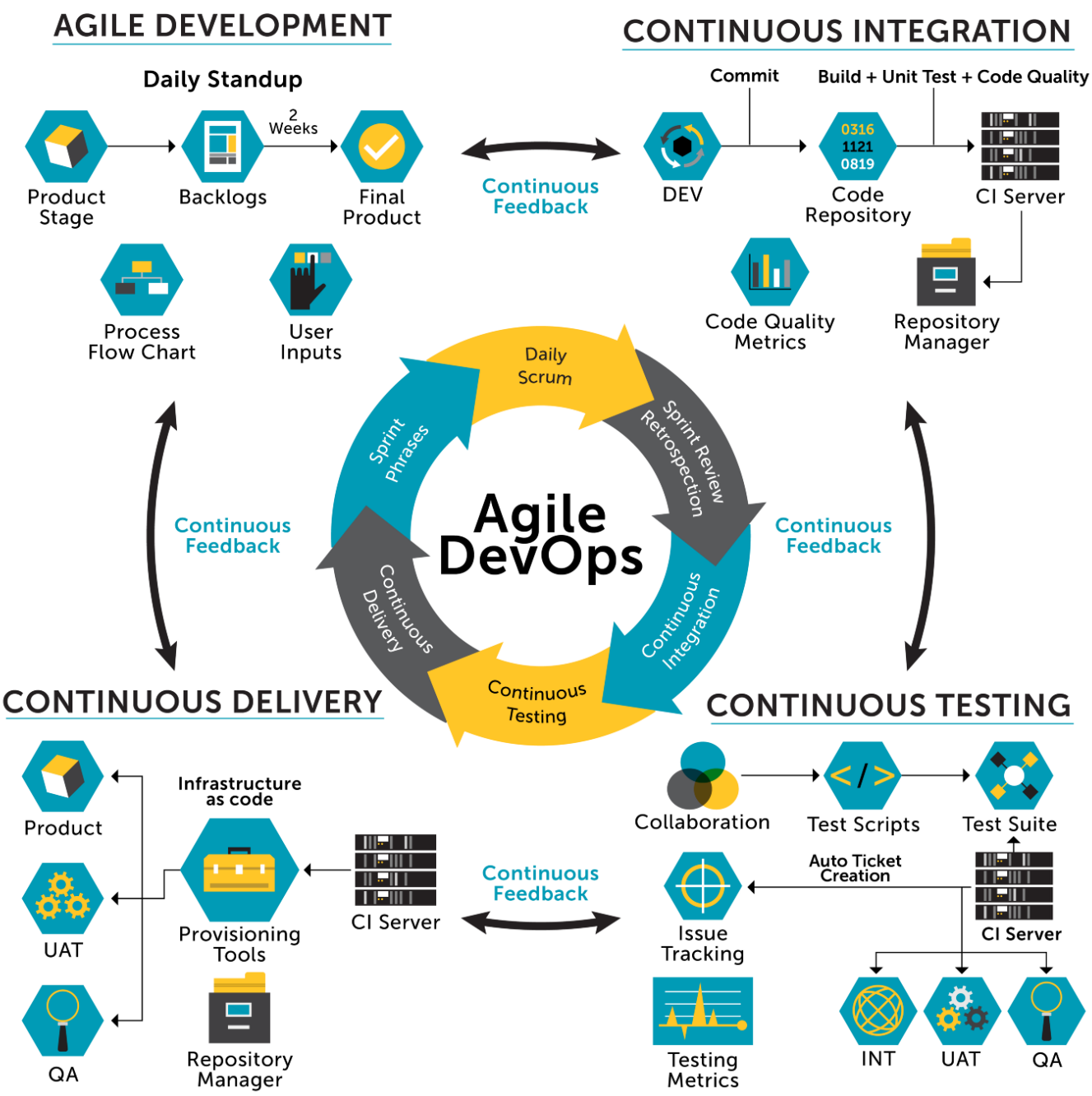
The Domain-Driven Design Starter Modelling Process is freely available for you to use. In addition, your feedback and ideas are welcome to improve the technique or to create alternative versions.

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

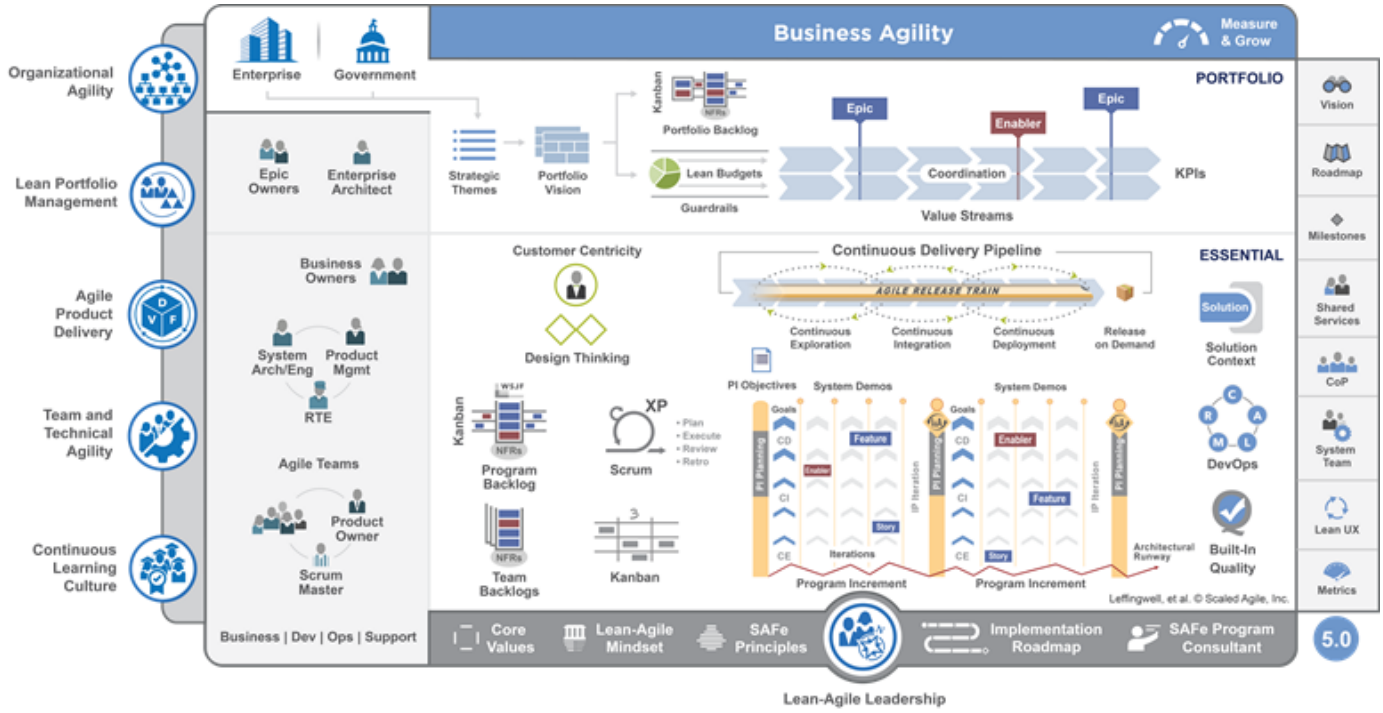
Feel free to also send us a pull request with your examples or experience reports.

Delivery process

Devops delivery process



Agile at scale delivery process



Roadmap

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

V1

Initial setup of the boilerplate

Template

- [ ] Setup dotnet template

Clients

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

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
    - [ ] Mappings
    - [ ] Todolist handlers
  - [ ] Feature
    - [ ] Todolist features
      - [ ] Commands
      - [ ] Queries
      - [ ] Policies
  - [ ] Infrastructure
    - [ ] Persistence
      - [ ] Entity framework
        - [ ] EF Entities configuration
        - [ ] EF Migration
      - [ ] MartenDB
    - [ ] External systems implementation
      - [ ] Storage
      - [ ] User
      - [ ] AI Prompting
      - [ ] Mail

        - [ ] PDF
    - [ ] Distributed tracing
        - [ ] OpenTelemetry
- [ ] Interface
    - [ ] Authentication
    - [ ] Authorization
    - [ ] GraphQL API
    - [ ] REST API
        - [ ] OpenAPI contract
    - [ ] HealthCheck
    - [ ] Feature management
- [ ] Tests
    - [x] Unit tests
        - [x] Setup Unit tests
            - [x] Todolist tests
    - [ ] Integration tests

Storage (optional) ?

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

Workflow (optional)

- [ ] Elsa core

Scheduled Jobs (optional)

- [ ] Hangfire

Cross cutting

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

Building blocks

- [x] Reverse proxy
  - [x] Setup YARP reverse proxy
- [ ] IAC
  - [ ] docker-compose
    - [ ] postgres
    - [ ] keycloak
  - [ ] Azure biceps / ARM
  - [ ] K8S

Docs

- [x] Setup vitepress
- [x] Enable task list markdown plugin
- [x] Home page
- [ ] Getting started
  - [x] Roadmap page
- [ ] Guidelines
  - [ ] Discovery & delivery dual track
  - [ ] Product & Tech Discovery
    - [ ] FOCUSED product discovery
    - [ ] DDD Tech Discovery
  - [ ] Agile Delivery
    - Agile guidelines

    - Application lifecycle management
- [ ] Boilerplate
  - [ ] Architecture (C4)
    - [ ] Clients
    - [ ] Services
    - [ ] Building blocks
    - [ ] Docs
    - [ ] Cross cutting
  - [ ] Testing
    - [ ] Unit tests
    - [ ] Architecture tests
    - [ ] Integration tests
    - [ ] E2E tests
  - [ ] Deploy