Domain-Driven Design: Separating Fact from Fiction

Marek Dominiak

Marek Dominiak

I'm running training in EventStorming, Architecture, DDD, and TDD

17 years of professional experience

Co-owner and instructor at trainitek

Hands-on Architect and Team Lead at MINTRA

https://www.linkedin.com/in/marekdominiak

Do you feel confident in your understanding of the business aspects of your project?

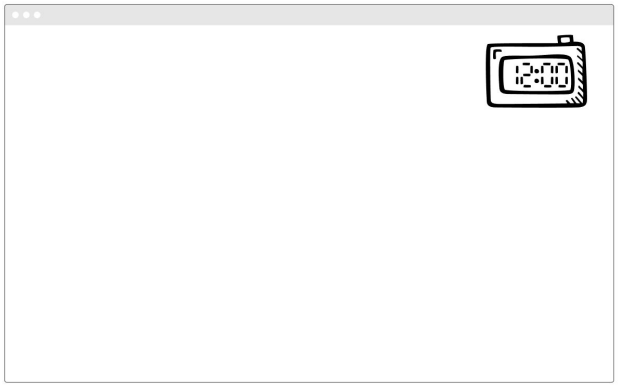
Are the business needs in your project communicated clearly and easy to implement?

Do you often experience headaches when trying to solve bugs or implement new features?

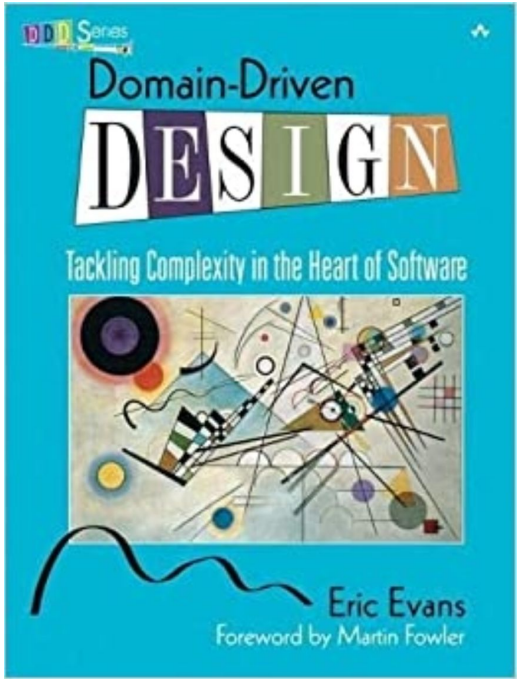
Is the model you work on too large to fully understand?

Would you invest six months of two senior developers' time to implement date-time library supporting timezones?

It depends



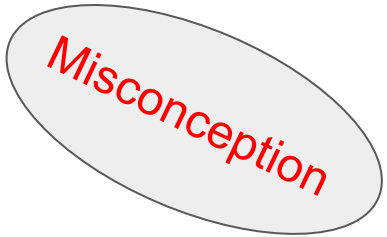
“**Domain-Driven Design**: Tackling Complexity in the Heart of Software”, Eric Evans 2003



Common misconceptions

Using aggregates, repositories, entities, events, value objects and services in your code

==

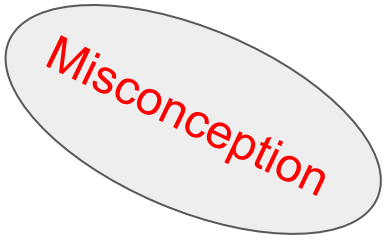


Domain-Driven Design

Using Anemic Model

!=

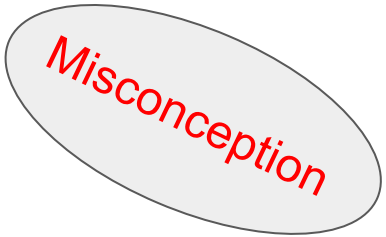
Domain-Driven Design



Ref: https://martinfowler.com/bliki/AnemicDomainModel.html

Using Microservices

==



Domain-Driven Design

Domain-Driven Design is **not** about
elegant code or any particular methodology

it is about

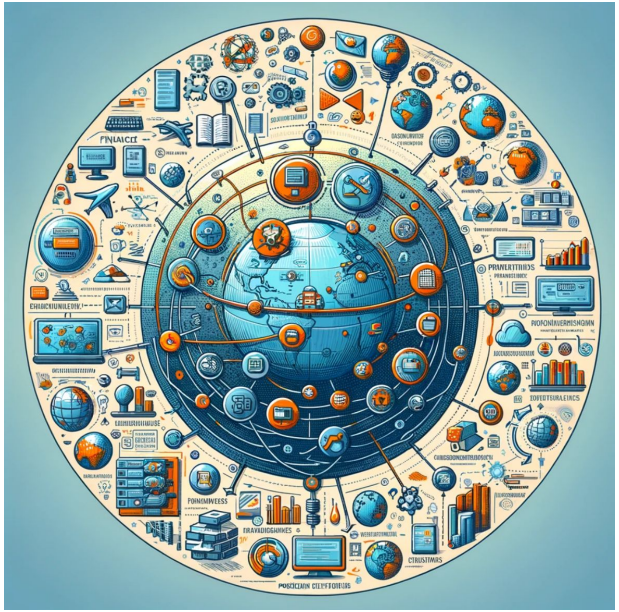
providing value

Why Domain-Driven Design matters?

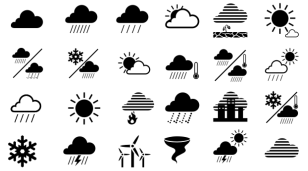
- To set focus on what really matters - solving real business problems for companies
- To make communication between people / departments / teams more effective
- To distill the core of a complex domain
- To have tools/patterns to handle multiple models on the same project (integration between different systems)
- To have tools on the code level - Tactical DDD

What is a domain?

- A domain is a sphere of knowledge, representing the main business of a company.
- Domain refers to an area, sector, or process within the company.
- Domain provides the context for **the Problem Space**

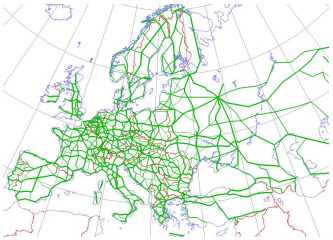


Examples of domains:



- Meteorology, atmospheric science, and data related to weather patterns

- Warehouse management, inventory levels, and supply chain processes



- Transportation systems, road networks, and traffic patterns

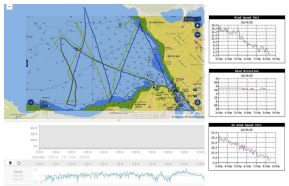
- Banking, financial transactions, and patterns of fraud

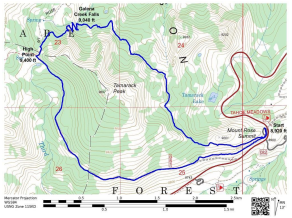
Example	Domain	Problem	Possible Models - Possible solutions
Route Planning and Navigation	Cartography, geospatial data, transportation networks	Assist users in efficient moving from one place to another (A->B)	??
Fraud Detection in Banking Transactions	Banking, financial transactions, patterns of fraud	Detect and prevent fraudulent transactions in real-time	Supervised machine learning, unsupervised machine learning, rule-based systems
Smart Traffic Control System	Transportation systems, road networks, traffic patterns	Optimize traffic flow and reduce congestion at peak hours	Traffic simulation models, machine learning, real-time data analysis
Inventory Management in a Warehouse	Warehouse management, inventory levels, supply chain processes	Maintain optimal inventory levels to minimize costs and avoid stockouts	Economic order quantity (EOQ) model, just-in-time (JIT) inventory management, safety stock models

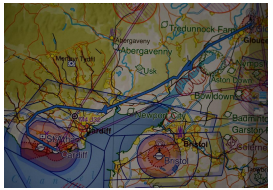
Let’s talk Models

- **Domain:** Cartography, Geospatial Data, Transportation Networks
- **Problem:** Route Planning and Navigation: Assist users in efficiently moving from one place to another (A -> B)
- **Possible Models:**
  - Google Maps, OpenStreetMap (in cities or between cities)
  - Marine Navigation Systems (e.g., Navionics, iNavX)
  - Mountain Navigation Systems (e.g., Gaia GPS, AllTrails)
  - Air Navigation Systems (e.g., ForeFlight, SkyDemon)
  - Classic physical Maps

Google Maps







Models:

- A model is a **simplified** representation of a domain, focusing on essential elements to **address specific problems**

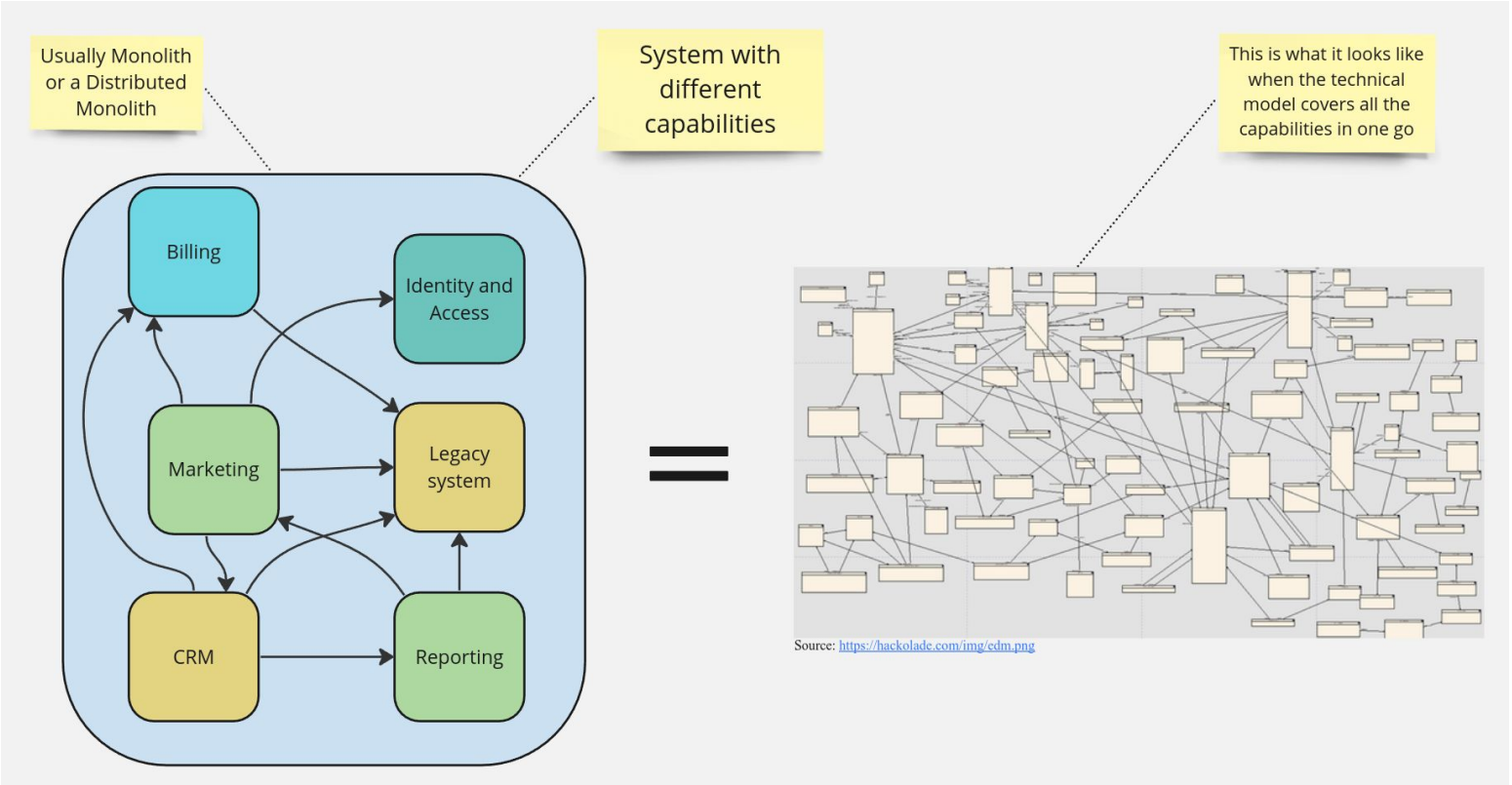
- **All models are "wrong"** as they cannot capture every aspect of reality, yet some are useful for problem-solving

    - “*The Map is not the Territory”, Alfred Korzybski*

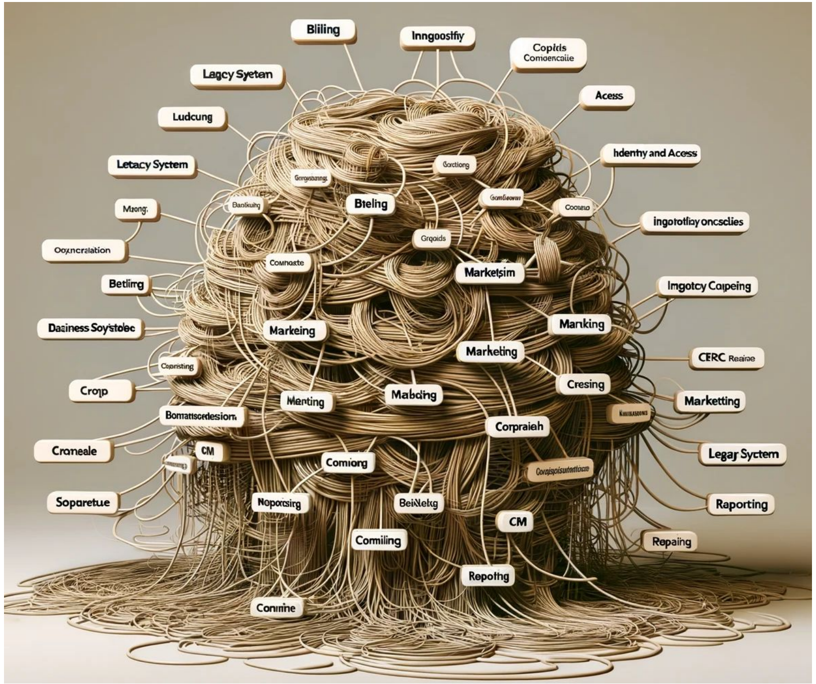
- **Models live in the solution space**, representing potential approaches to tackle challenges within a problem context

Ref: https://conceptually.org/concepts/the-map-is-not-the-territory

A word on Big Models

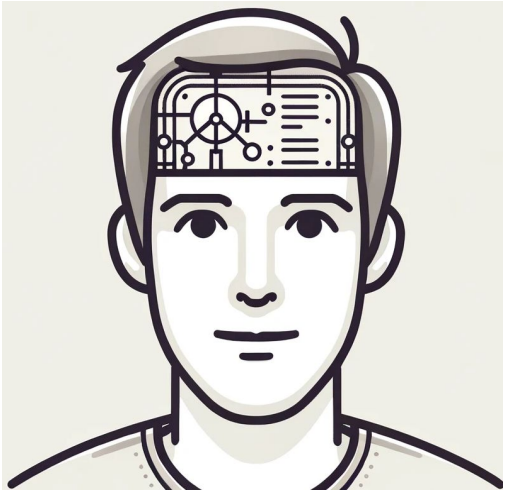


Big models aren't that fun to work with



Having many smaller and focused models works better

- Smaller cognitive load
- It’s easier to maintain smaller models
  - Less bugs
  - Better quality
- It’s more fun to work with



So, what’s in the Heart of Domain-Driven Design?

- Focus on the solving business problems - Domain problems
- Ubiquitous Language
- Bounded Contexts
- Context Maps

Ubiquitous Language

Eases communication between developers, architects, and business experts by **using the same language across departments**.

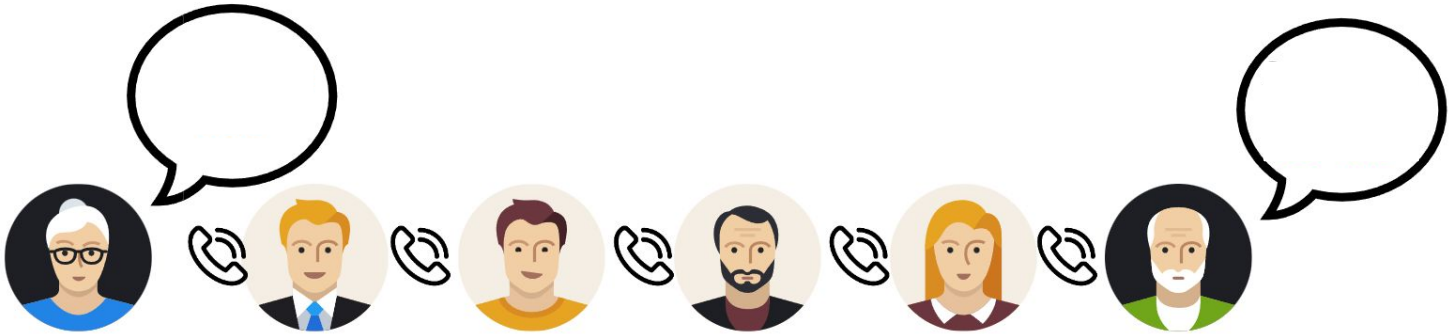
- Perfect for preventing miscommunication.
- Helps avoid the corporate version of the "broken telephone" game.

Bounded Context

The part of the system in which we use **the same Ubiquitous Language**.

- User / Visitor vs Customer (CRM and WebApp)
- Jagoda (or in corporate world e.g. Product)

Telephone game



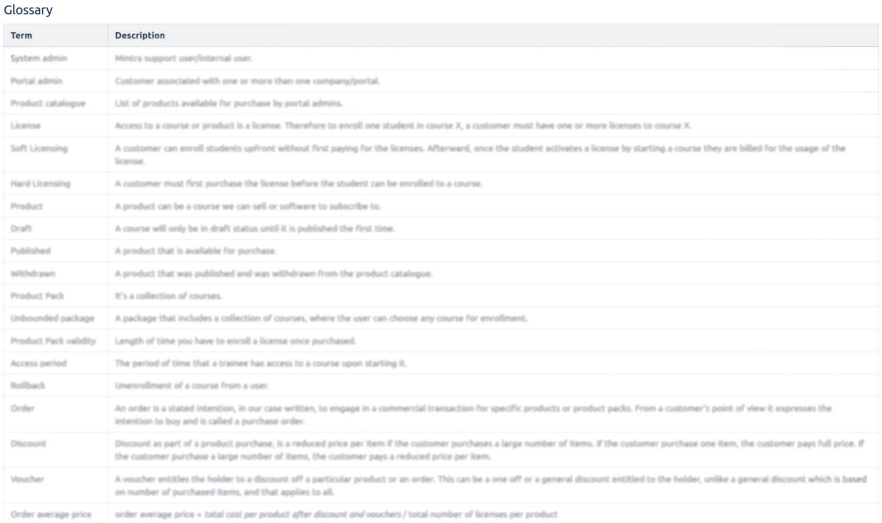
Why are we continuing this madness?

There is one Ubiquitous Language per bounded context.

How to document Ubiquitous Language

- Ubiquitous Language should be visible everywhere: in the code, UI, documentation, requirements etc.

- You can publish it to Confluence as e.g. Glossary

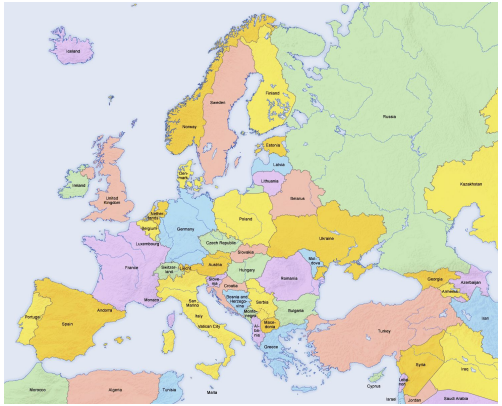


Glossary

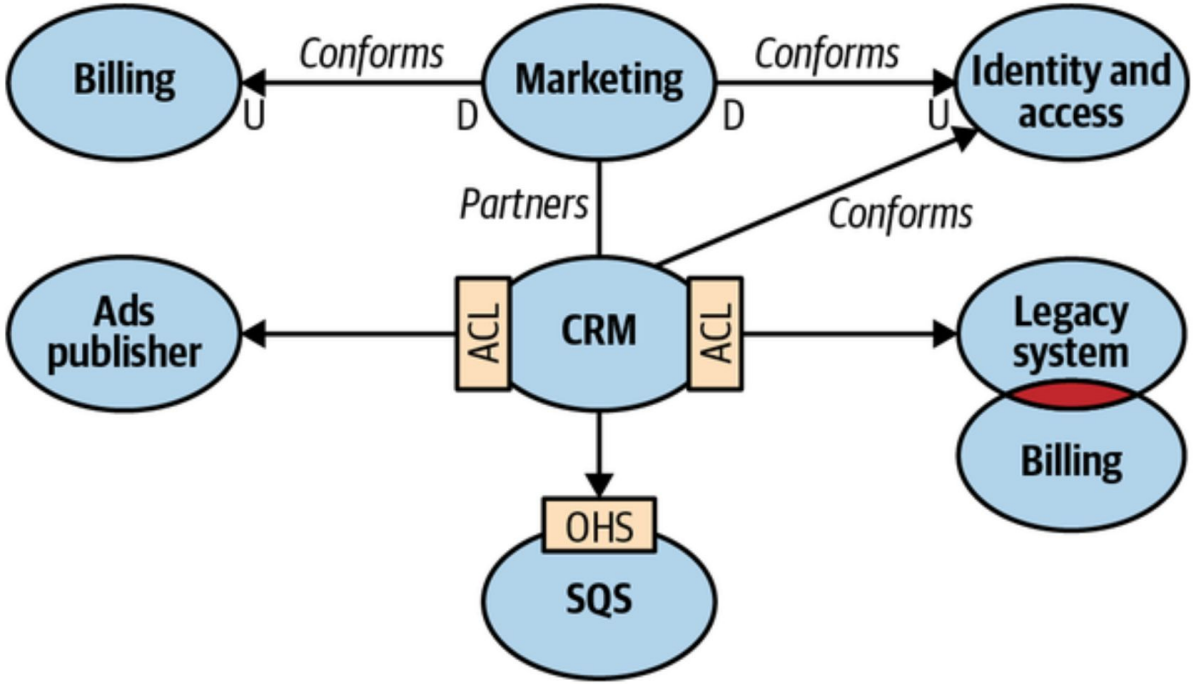
Term	Description
System admin	Mintra support user/internal user.
Portal admin	Customer associated with one or more than one company/portal.
Product catalogue	List of products available for purchase by portal admins.
License	Access to a course or product is a license. Therefore to enroll one student in course X, a customer must have one or more licenses to course X.
Soft Licensing	A customer can enroll students upfront without first paying for the licenses. Afterward, once the student activates a license by starting a course they are billed for the usage of the license.
Hard Licensing	A customer must first purchase the license before the student can be enrolled to a course.
Product	A product can be a course we can sell or software to subscribe to.
Draft	A course will only be in draft status until it is published the first time.
Published	A product that is available for purchase.
Withdrawn	A product that was published and was withdrawn from the product catalogue.
Product Pack	It's a collection of courses.
Unbounded package	A package that includes a collection of courses, where the user can choose any course for enrollment.
Product Pack validity	Length of time you have to enroll a license once purchased.
Access period	The period of time that a trainee has access to a course upon starting it.
Rollback	Unenrollment of a course from a user.
Order	An order is a stated intention, in our case written, to engage in a commercial transaction for specific products or product packs. From a customer's point of view it expresses the intention to buy and is called a purchase order.
Discount	Discount as part of a product purchase, is a reduced price per item if the customer purchases a large number of items. If the customer purchase one item, the customer pays full price. If the customer purchase a large number of items, the customer pays a reduced price per item.
Voucher	A voucher entitles the holder to a discount off a particular product or an order. This can be a one off or a general discount entitled to the holder, unlike a general discount which is based on number of purchased items, and that applies to all.
Order average price	order average price = *total cost per product after discount and vouchers* / total number of licenses per product

You can think about the Bounded Contexts as Countries

- They may or not have similar language (different Ubiquitous Languages)
- They have very specific ways of communication (APIs and contracts)
- There may be no easy way of passing boundaries



Context Maps



Ref: https://www.oreilly.com/library/view/what-is-domain-driven/9781492057802/ch04.html

The hardest work is to find proper boundaries of Bounded Contexts

- EventStorming sessions give a good starting point
- Whatever strategy is used, information should flow from domain experts

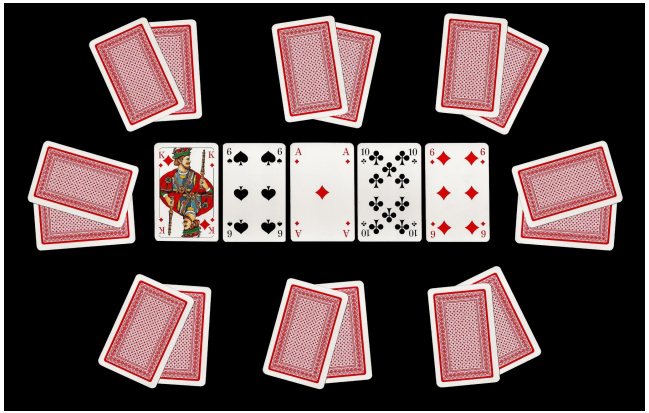
Context mapping - communication - flavours of APIs

- Partnership
- Shared Kernel
- Customer Supplier
- Conformist
- Anticorruption Layer (ACL)
- Open Host Service (OHS)
- Published Language (PL)

Ref: The Blue book - by Eric Evans

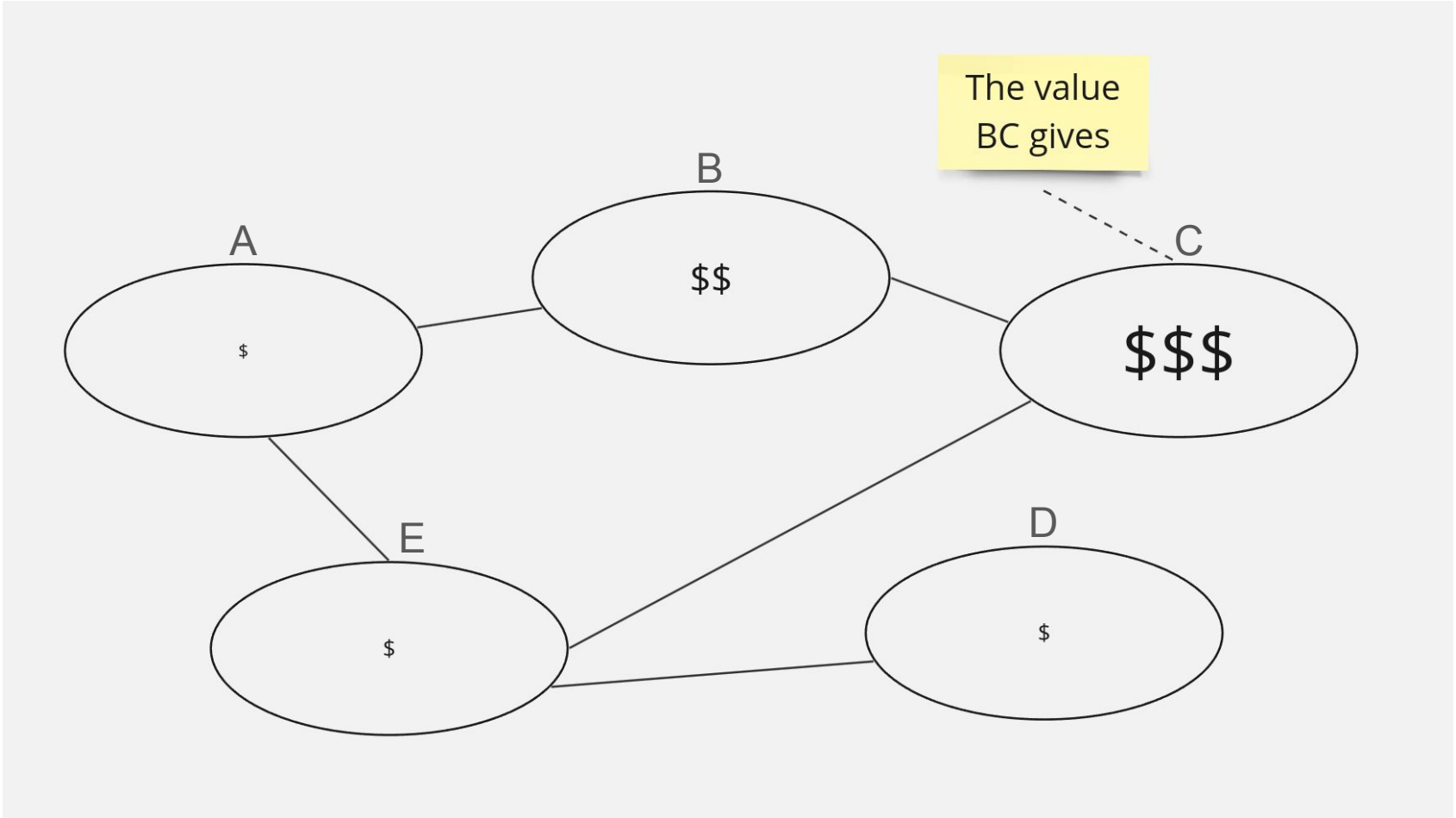
Strategic Design

- Decide which domain is our **core** domain.
- The core domain should be the one we focus on - best people, most effort, etc.
- Not all the Domains are of the same importance! We all have limited budgets and time …



Where should we focus our efforts? T-Shirt sizing

Bounded Context	**Competitive Advantage** (S, M, L)	**Implementation Effort** (S, M, L)	**Value** (Competitive Advantage / Implementation Effort), (XXS, XS, S, M, L, XL, XXL)
BC 1	S	L	XXS
BC 2	L	M	XL
BC 3	M	M	M


The value
BC gives
A
$
B
$$
C
$$$
E
$
D
$

Subdomains - Broken down larger domain into smaller, focused areas

- **Core**: Essential, unique business **differentiators** (e.g., recommendation engine, secret AI model)
- **Supporting**: Important, supplementary/supporting functionalities (e.g., user review system)
- **Generic**: Common features, often standardized (e.g. payment processing, accounting, invoicing)
- We should prioritize resources and efforts based on subdomain types

Splitting Domains into Subdomains - OpenAI

- **Core**: GPT-3.5 -> GPT-4 -> GPT-4o
- **Supporting**: Image generation DALL-E
- **Generic**: Stripe as payment provider

When you know where to put your focus and efforts then use (or not) Tactical DDD

- Aggregates
- Aggregate Roots
- Entities
- Value Objects
- Factories
- Repositories
- Application Services
- Domain Services
- Domain Policies
- Domain Events
- Specifications

Summary of the Benefits of Domain-Driven Design

- Increases the Chances of Project Success
- Enhances Your Professional Value
- Provides Satisfaction in Seeing Effective Results
- Supports Scalable and Flexible Architecture

Some ideas on how to leverage AI tools

- Using “smart” LLMs to act as an expert so you can learn quicker and ask better questions
- Checking your models against AI to get better results
  - Providing suggestions for alternative design approaches or implementation strategies when faced with technical constraints or trade-offs
  - Helping to identify potential anti-patterns or common pitfalls in DDD implementations, and offering recommendations for how to avoid them
- Helping you prepare better notes from meetings with domain experts
- Producing better documentation
- Creating context maps e.g. using Structurizr DSL
- Suggesting acceptance tests to find edge cases, or to discuss with experts
- Helping identify bounded context boundaries
- Finding possible online services that you would otherwise implement yourself, moving them to generic subdomains

References

DDD:

- Domain-Driven Design: Tackling Complexity in the Heart of Software
  https://www.amazon.com/Domain-Driven-Design-Tackling-Complexity-Software/dp/0321125215

- Principles Patterns and Practices of Domain-Driven Design
  https://www.amazon.co.uk/Patterns-Principles-Practices-Domain-Driven-Design/dp/1118714709/ref=sr_1_1?dchild=1&keywords=principles+of+domain+driven+design&qid=1614351049&s=books&sr=1-1

Tactical DDD:

- https://docs.microsoft.com/en-us/azure/architecture/microservices/model/tactical-ddd

Other:

- Anemic domain model - https://martinfowler.com/bliki/AnemicDomainModel.html

Contact / Slides

https://www.linkedin.com/in/marekdominiak

Slides: https://github.com/trainitek/presentations/blob/main/jPrime2024/DDD.pdf

Slides: