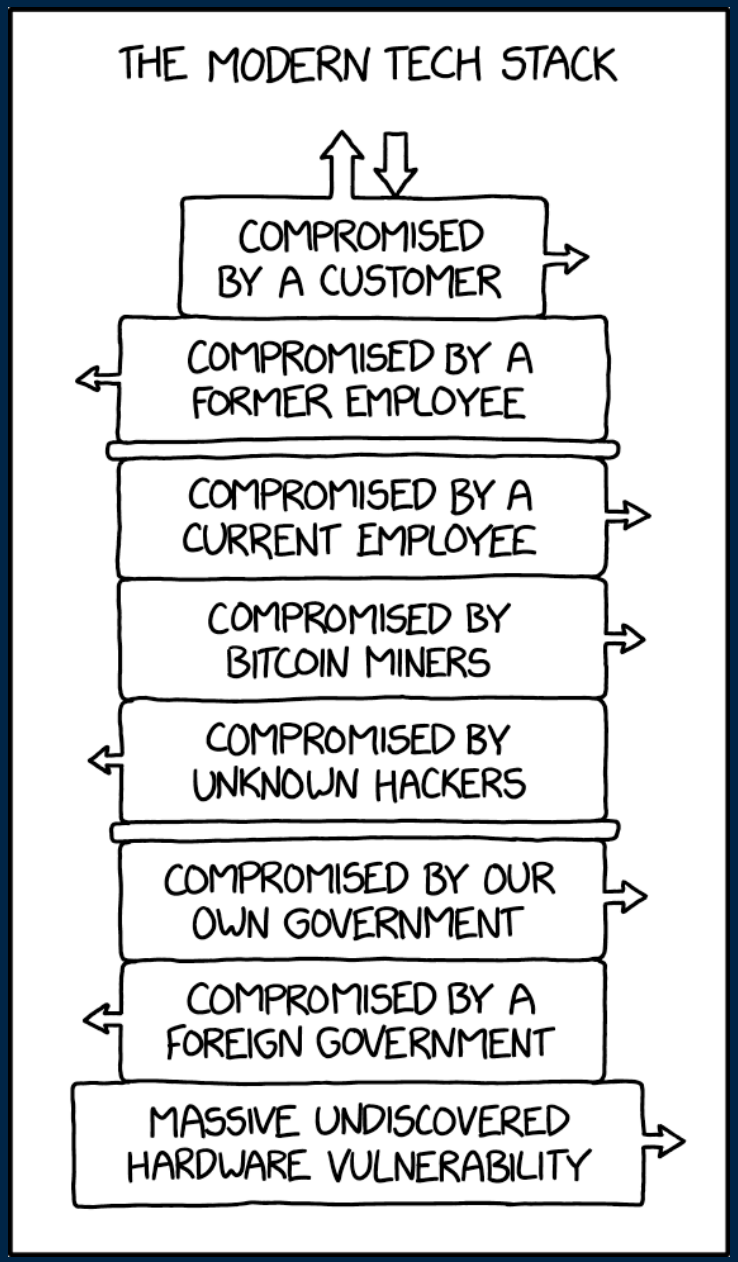
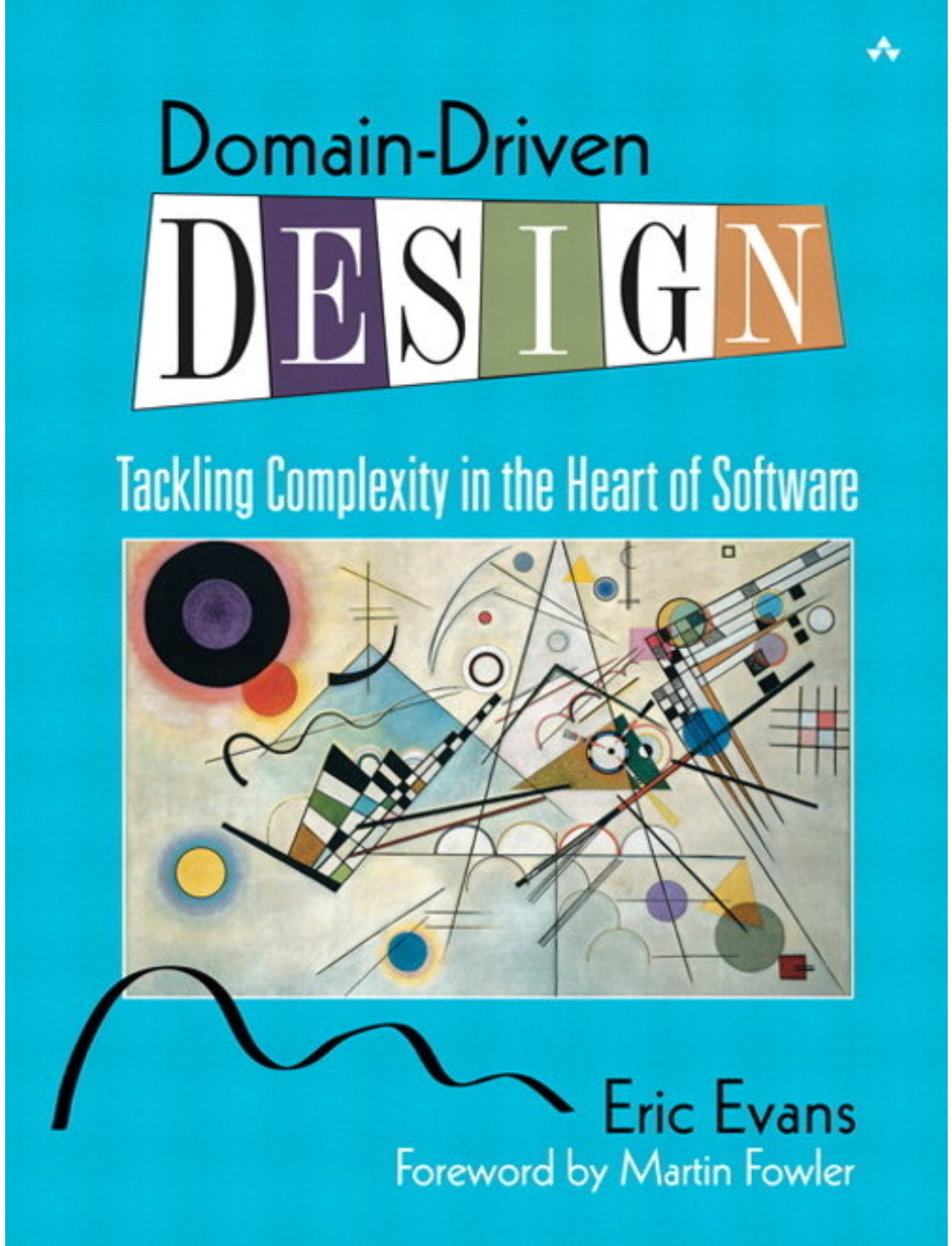
DDD

Domain Driven Design



AARHUS UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

HENRIK BITSCH KIRK
ASSOCIATE PROFESSOR


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

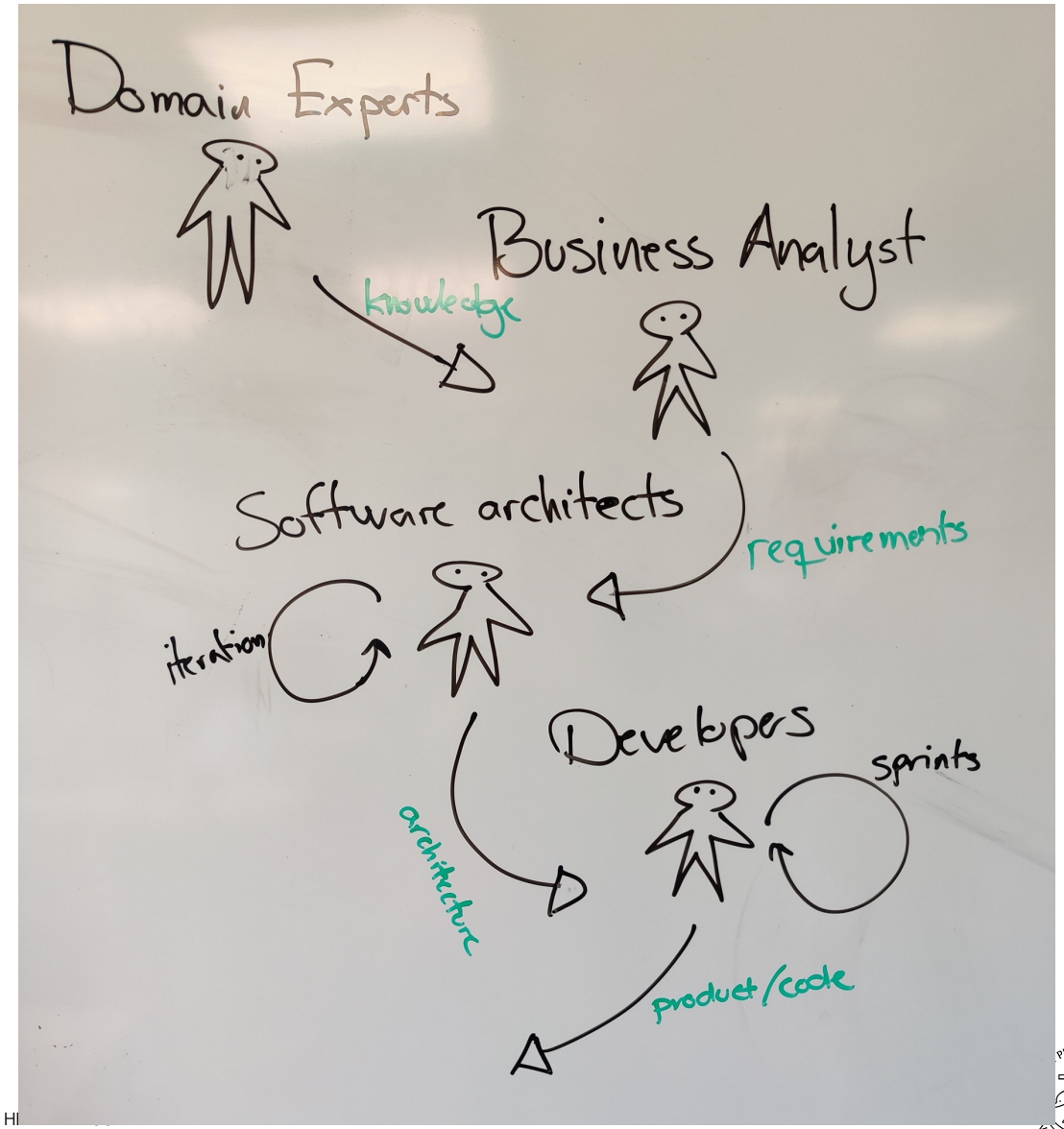
DOMAIN KNOWLEDGE

Developers don't know/talk to domain export – architects and salespersons do.

- Requirements are just given in a document
- Misunderstandings can occur

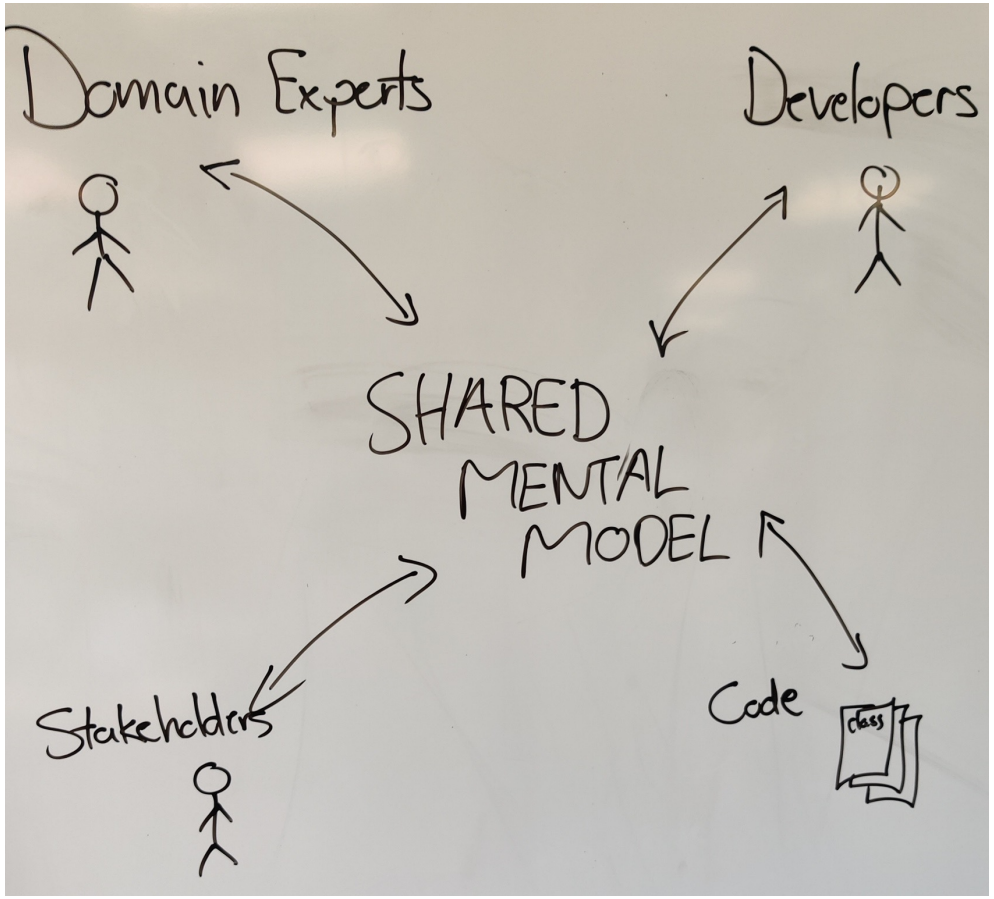
It is important to

- Understand the domain
- Have a shared knowledge about the domain



WHAT IS DOMAIN DRIVEN DEVELOPMENT

- The primary focus should be on the core and domain logic
- Collaboration between technical and domain experts
  - Build a common understanding of domain problems
- Domain experts, developers, stakeholders, and (most importantly) code
  - Must share the same model



Align domain and software

EXERCISE (10 MINUTES)

Idea based on:

WasteNoFood

We will try to model a domain for an application that helps save food from being thrown out.

The general idea is:

- Supermarkets, restaurants, etc. (*Sellers*) can sign up to sell food to households
- Households (*Customers*) can buy food that would otherwise be thrown out

Create functional requirements for our new unicorn seen from different perspectives:

- Economy department, customers, sellers, PR (2-3 persons together)
- Fill in the requirements in Padlet: https://aarhusuni.padlet.org/henrikbitschkirk/leave-no-food-behind-8uisy08duunbumdo

STRATEGIC DESIGN

BENEFITS OF SHARED MODEL

Build a model – mental and in code – that is shared between developer, domain experts, architect, testers …

- Faster time to market
- More business value
- Less waste
- Easier maintenance

DOMAIN EVENTS

- Focus on transforming data rather than static data
  - static data do not add value
- ‘Every’ piece of work is triggered by an event (outside or inside)

- ‘*Order placed*, ‘*Flight booked*, ‘*Patient arrived*, etc.

EVENT STORMING

Workshop developer by Alberto Brandolini for DDD

Source: https://deravesoftware.com/what-is-event-storming/

EVENT STORMING

Workshop-based method to events in process

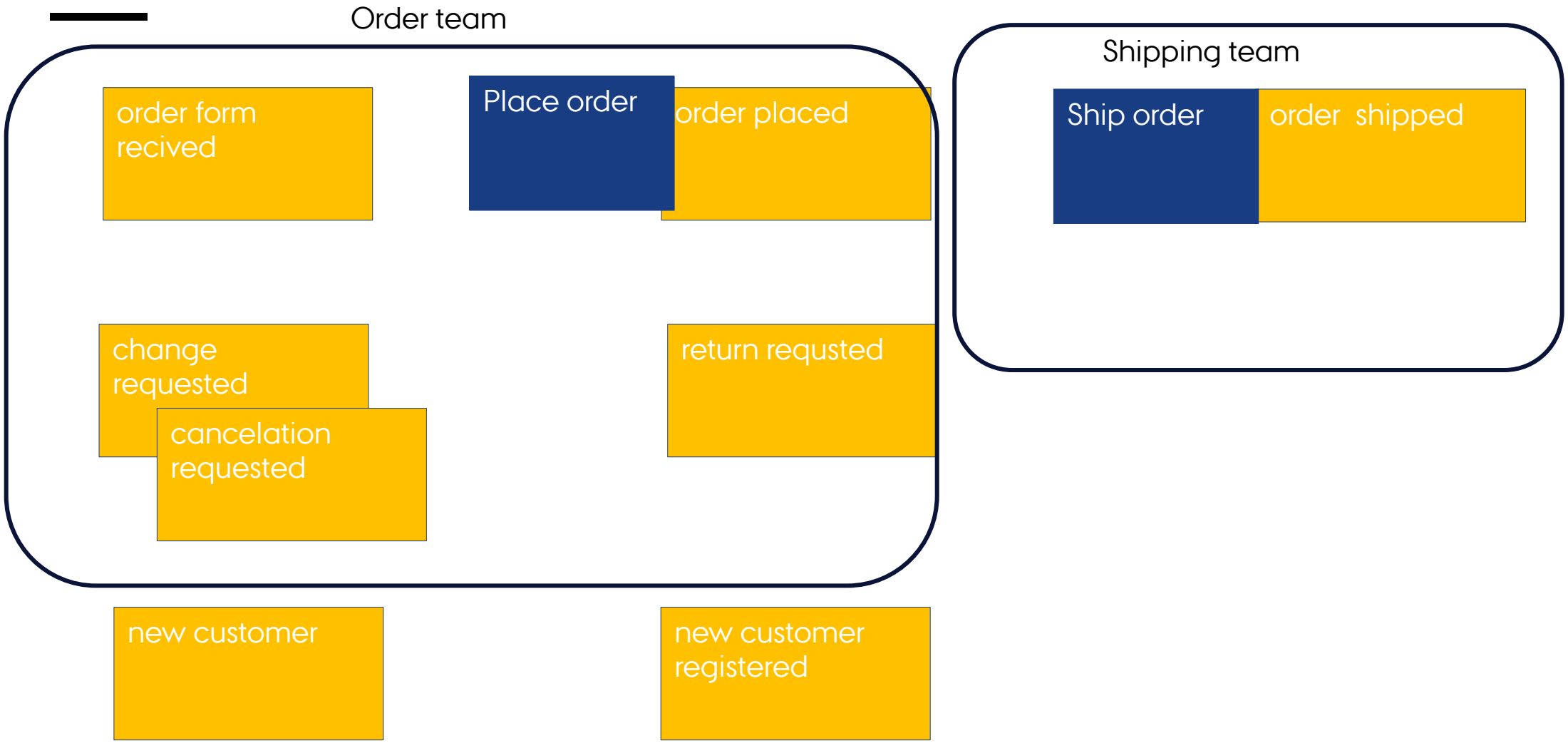
Steps:

1. Create Domain events (orange)
2. Add commands that cause an event (blue with orange)
    1. Add an actor that executes the command (yellow)
3. Add corresponding aggregate (yellow)
4. Figure out Business processes (Purple)

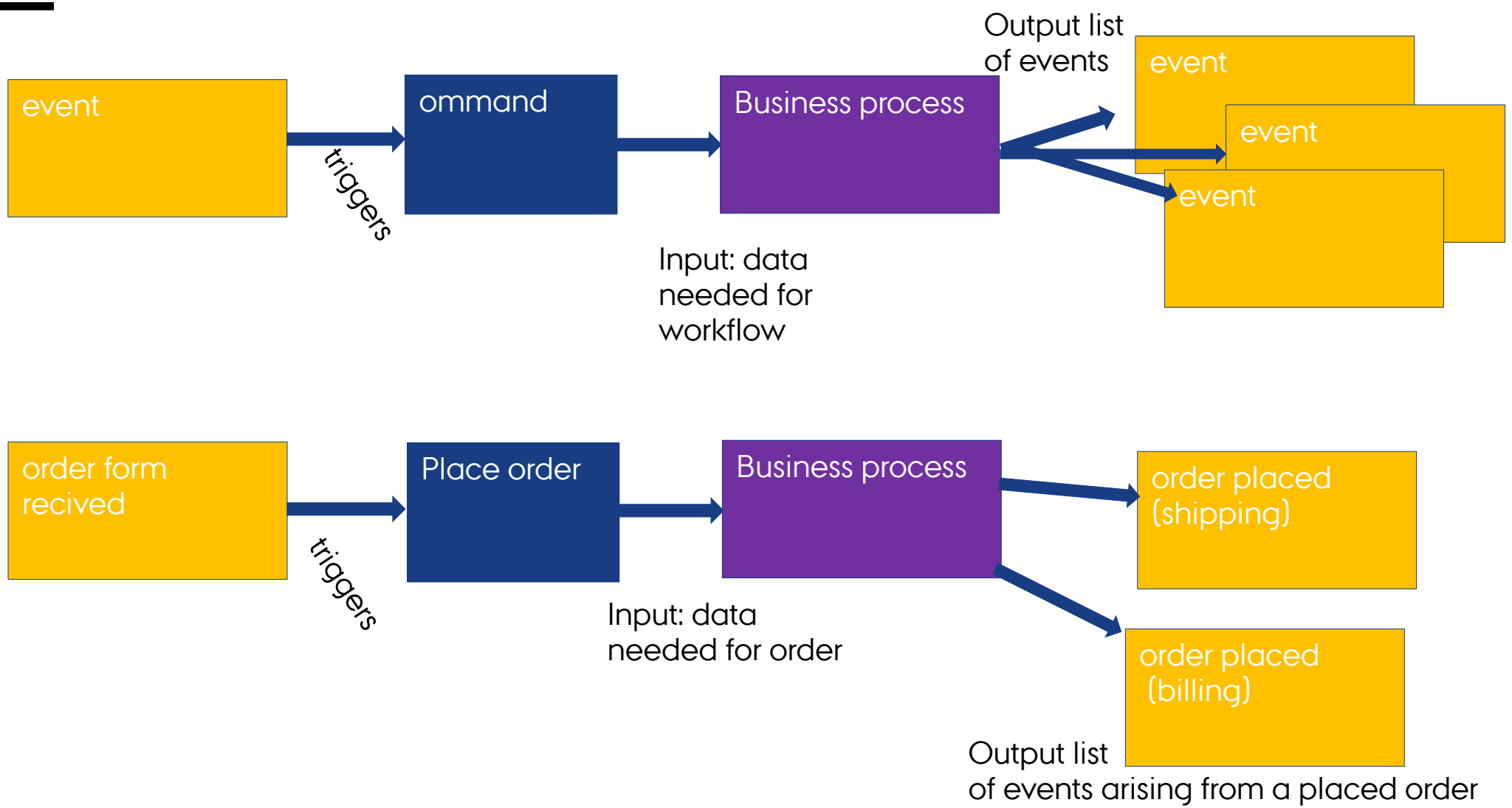
Participants: Domain experts, developers, and other stakeholders

- Event storming contains more elements; 'External systems', 'Views', and 'Errors'

EXAMPLES


Order team
order form recived
Place order
order placed
change requested
cancelation requested
return requsted
new customer
new customer registered
Shipping team
Ship order
order shipped

DOCUMENTING COMMANDS



EXERCISE 20 MINUTES

In the same groups

1) Fill in as many events as you can – try putting them in order **(7 minutes)**
   - Use functional requirements as input

2) Refine events (Find missing events, remove duplicates, look for order) **(5 min)**

3) What triggers an event and who (actor) with what (command) – or other domain events **(8 min)**
   - Add Commands in blue and actors in pale yellow

Use **Draw.io** or **paper**

UBIQUITOUS LANGUGE

- Building a common language between domain experts and developers
  - Only contain things represented in the domain
  - Technical terms (factory, helper, manager, controller, etc.) should not be part of the design

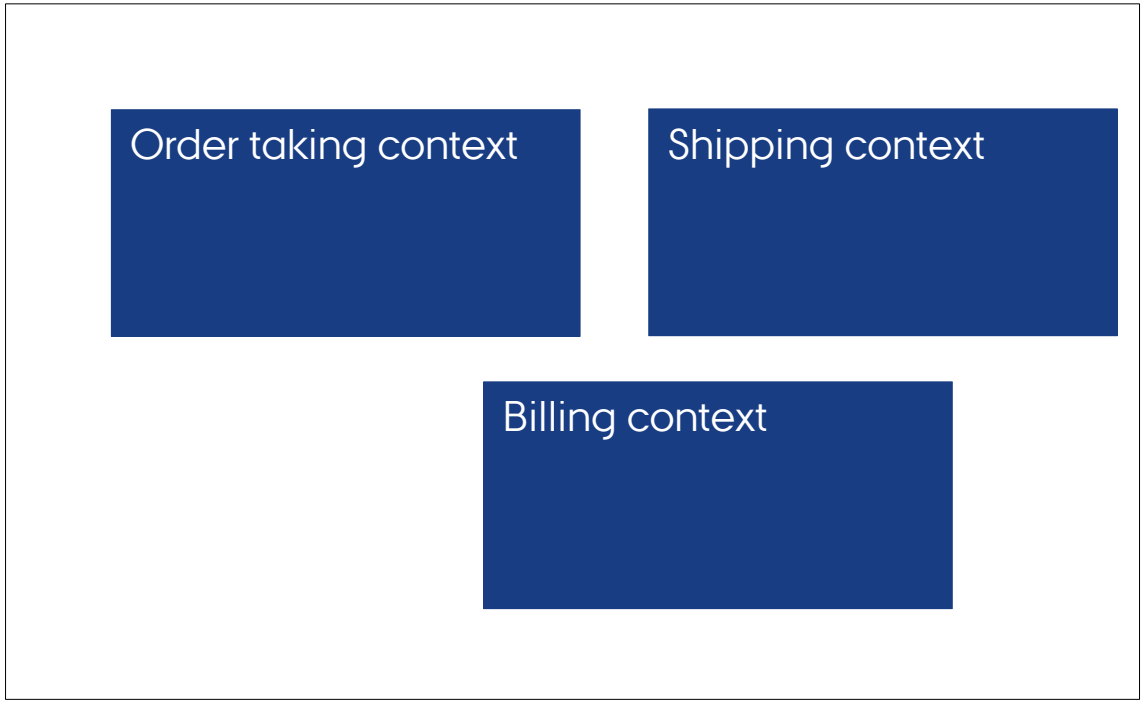
- Defines the shared mental model

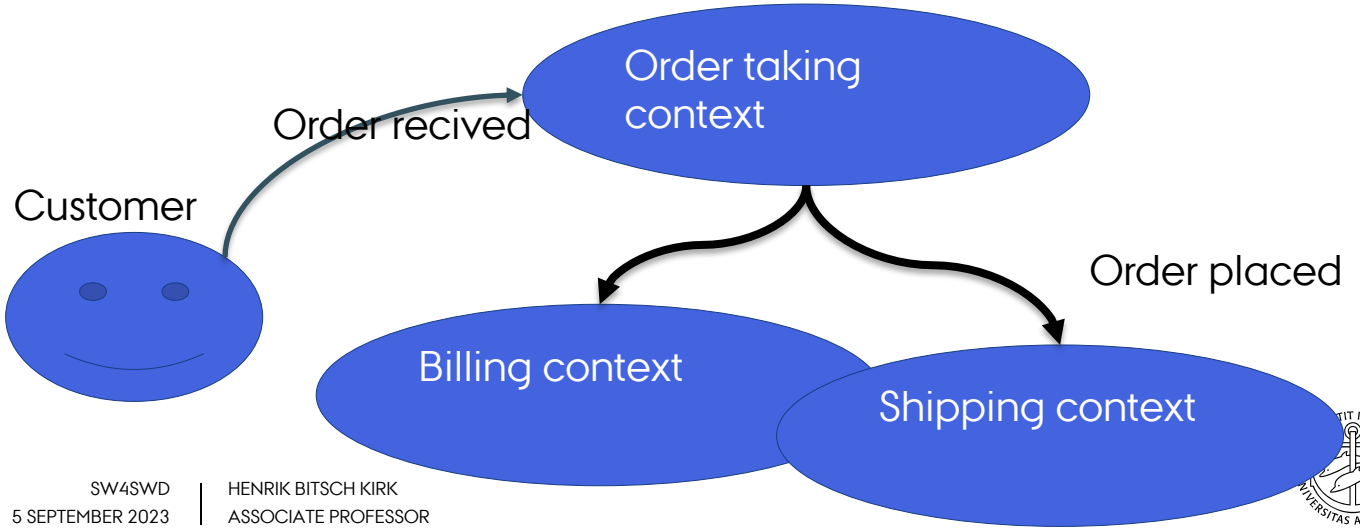
- All stakeholders collaborate on creating the ubiquitous language

- Does not necessarily exists **one** ubiquitous language
  - Dialect for each bounded context

BOUNDED CONTEXT

- Bounded Contexts (DDD for subsystem)
  - ”Mini” domains
- Why Context
  - specialized knowledge
- Why Bounded
  - in software we need subsystems to be decoupled
  - Evolve independently
- Context map
  - Interaction between contexts



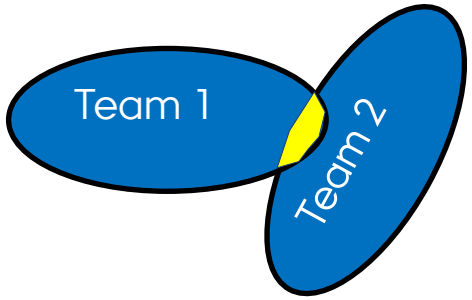


CHOOSING THE RIGHT CONTEXTS

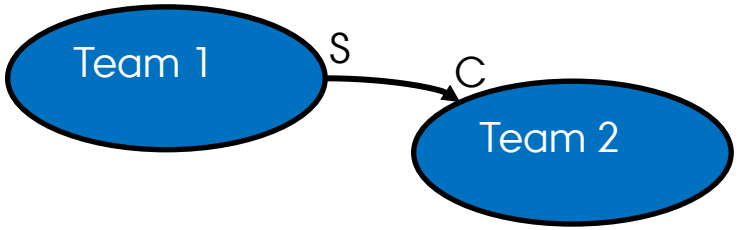
- Domain experts
  - Same language and same problems – properly same domain
- Existing teams and department
- ”Bounded”
- Autonomy
  - Two teams/groups working on the same bounded context – is properly slower than working on two
- Friction-free business workflows
  - Interaction with ‘many’ different bounded contexts

PARTNERSHIPS

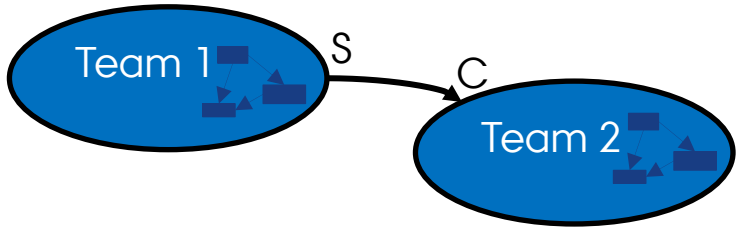
- Shared kernal
  - Share a small common model



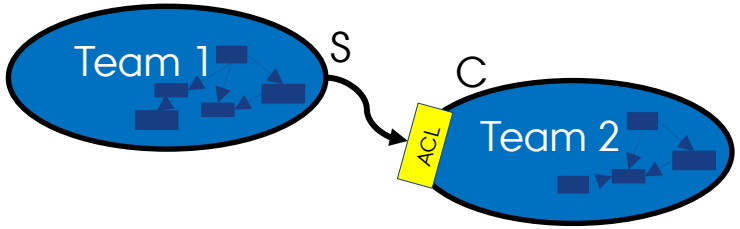
- Customer-supplier
  - Supplier provides what customer needs



- Conformist
  - Customer-supplier – but customer cannot ‘afford’ aford to translate



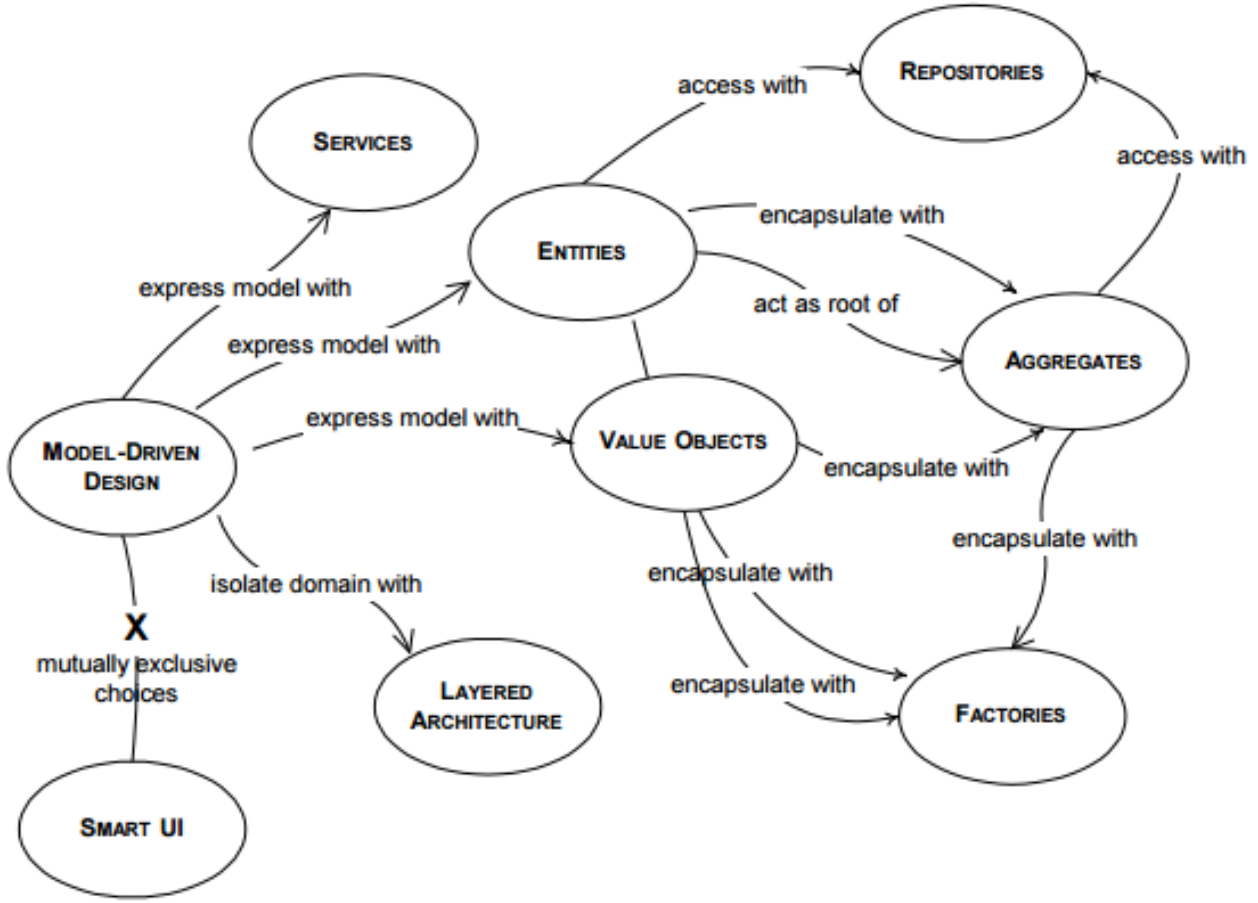
- Anticorrumption layer
  - Customer create a translation layer between suppliers- and own-model



- Other
  - Partnership, Open host service, published language

TACTICAL PATTERNS

Key element from OOP from an DDD perspective



VALUE OBJECTS VS ENTITIES

Value object

- Models a value
- No unique ID
- Immutable
- Comparison is done by attributes/value
- Examples
  - `Address`
  - `PhoneNumber`
  - `Money`

Entity

- Model an individual think
- Has unique ID
- Mutable
- Examples:
  - `OrderItem`
  - `Customer`
  - `Invoice`

AGGREGATES

- Composed of one of more entities and value objects
- Forms a transactional consistent boundary
- One entity is called **aggregate root**
  - Owns all other elements in aggregate
  - Acces to aggregate *must* go through this entity
- Examples:
  - Customer
  - Invoice

AGGREGATE DESIGN CONSIDARATIONS

1. Protect business invariants inside aggregate
2. Design small aggregates
3. Reference other aggregates only by identity
4. Update referenced aggregates using eventual consistency

SERVICES

- Contain domain operations that do not belong to an entity or value object
- Is stateless
- Examples
  - `PriceCalculation(…)`
  - `CurrencyCoversion(…)`

REPOSITORIES

- Retrieve domain objects (aggregates) from data storage or (DAL)
- You will see (have seen) this in SW4BED (Backend development)

FACTORIES

- Create domain objects
- Encapsulates the creation of objects

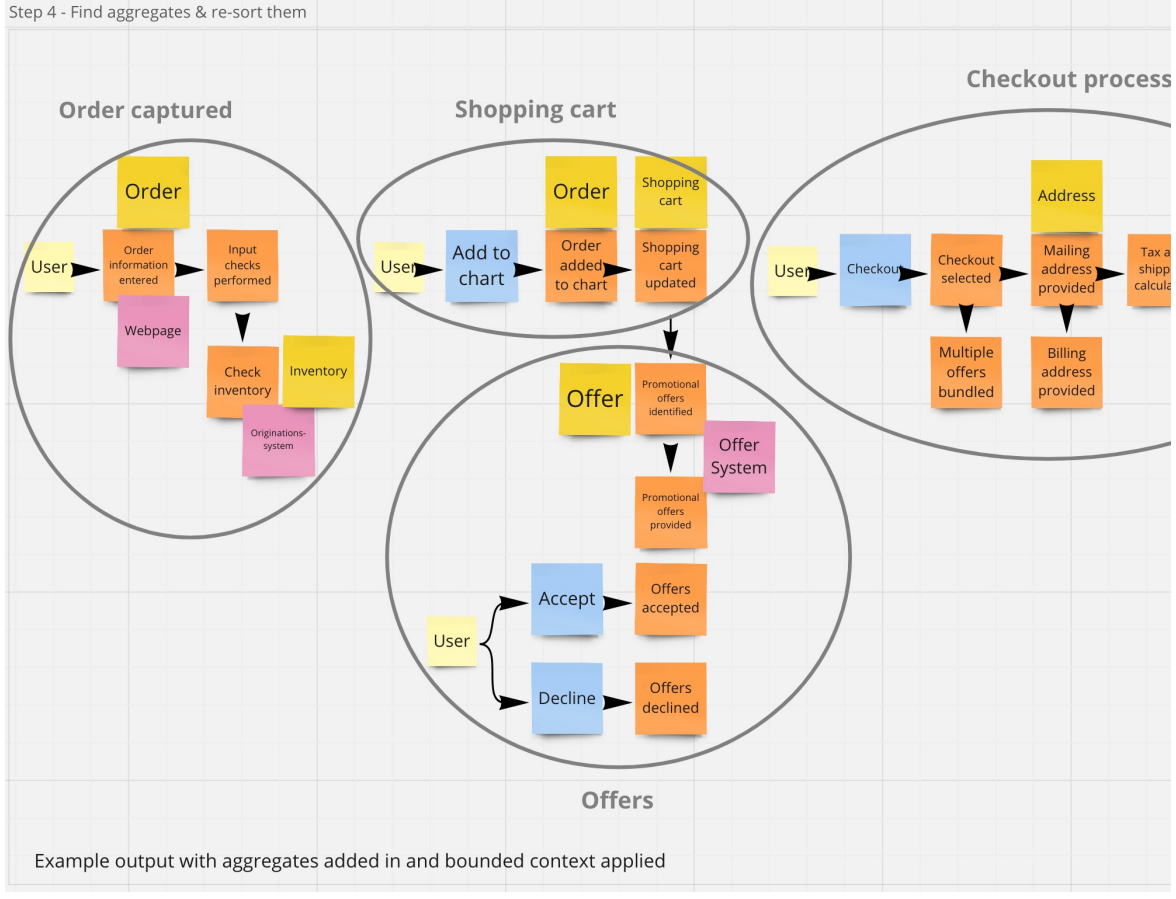
DOMAIN EVENTS

- Represent som business-significant occurrence in a bounded context
- Immutable facts
- Naming: Passed tense – using the ubiquitous language
- Can be used for inter-service communication
- Example
  - `OrderRevieved`
  - `NewCustomerRegisterd`

EXERCISE – BOUNDED CONTEXT

- Find Aggregates and Bounded Contexts (5-10 min)

- Using the elements from the Tactical Patterns to try and describe a design for 1-2 of the aggregates you found.



Example output with aggregates added in and bounded context applied

AARHUS
UNIVERSITY

REFERENCES

- https://xkcd.com/2166/