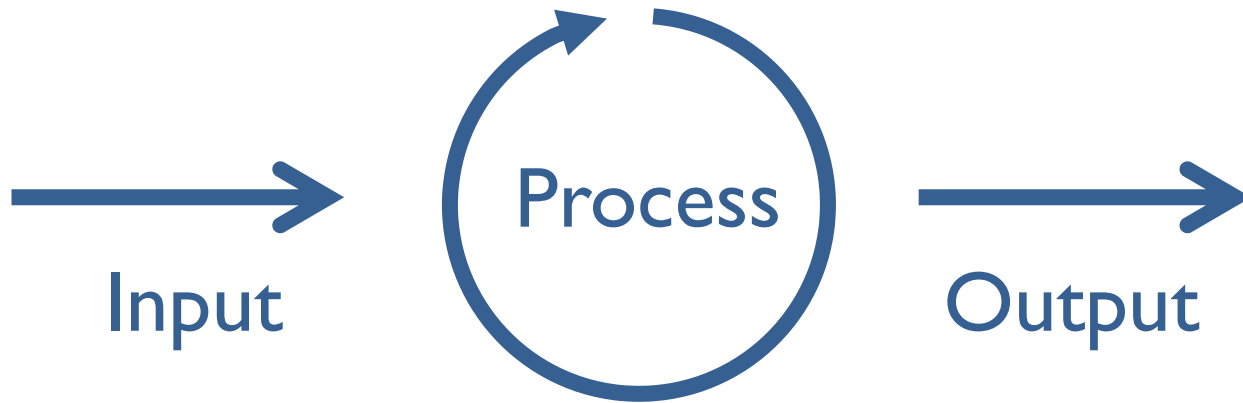
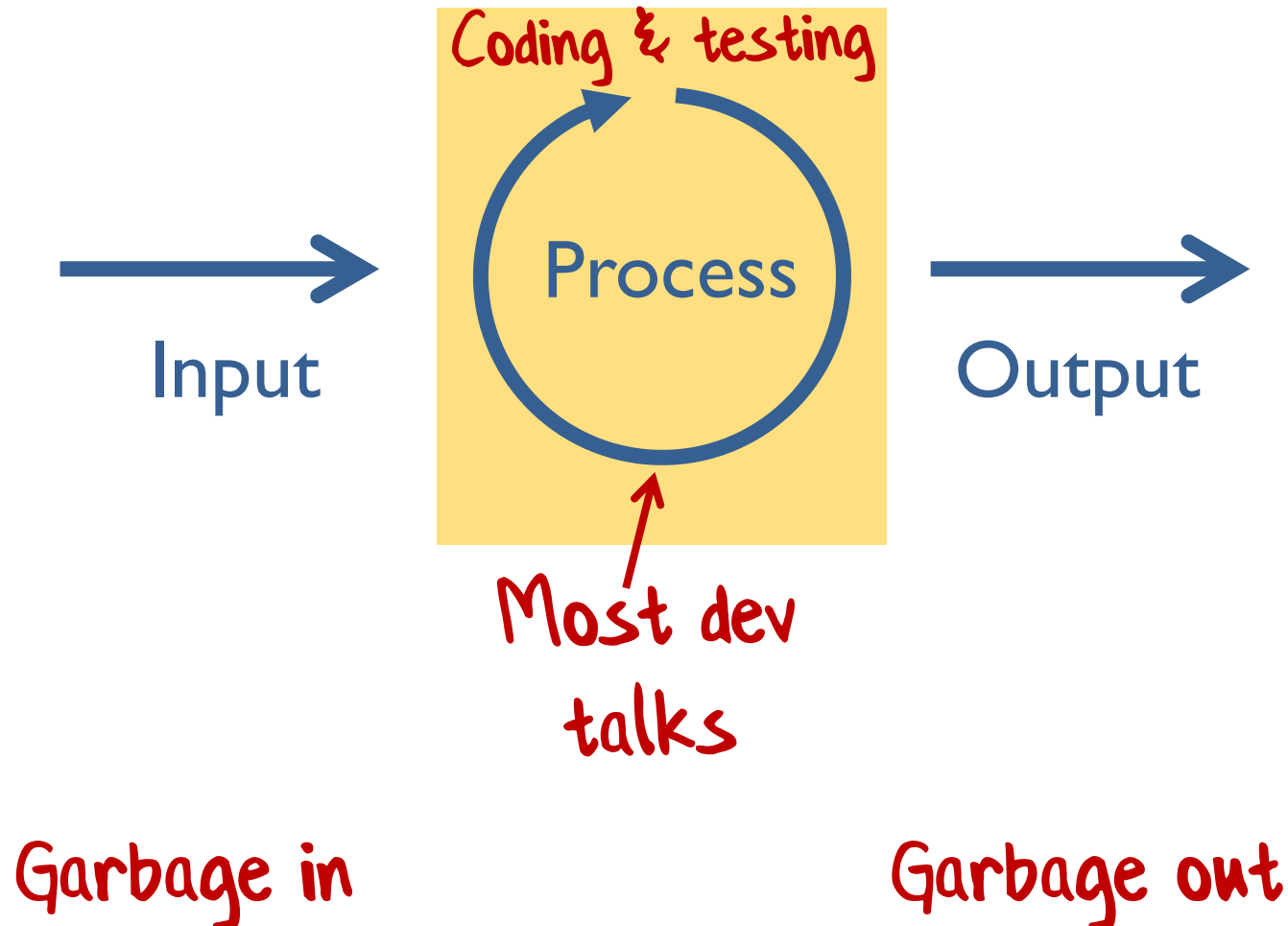


Domain-Driven Design and Domain Modelling

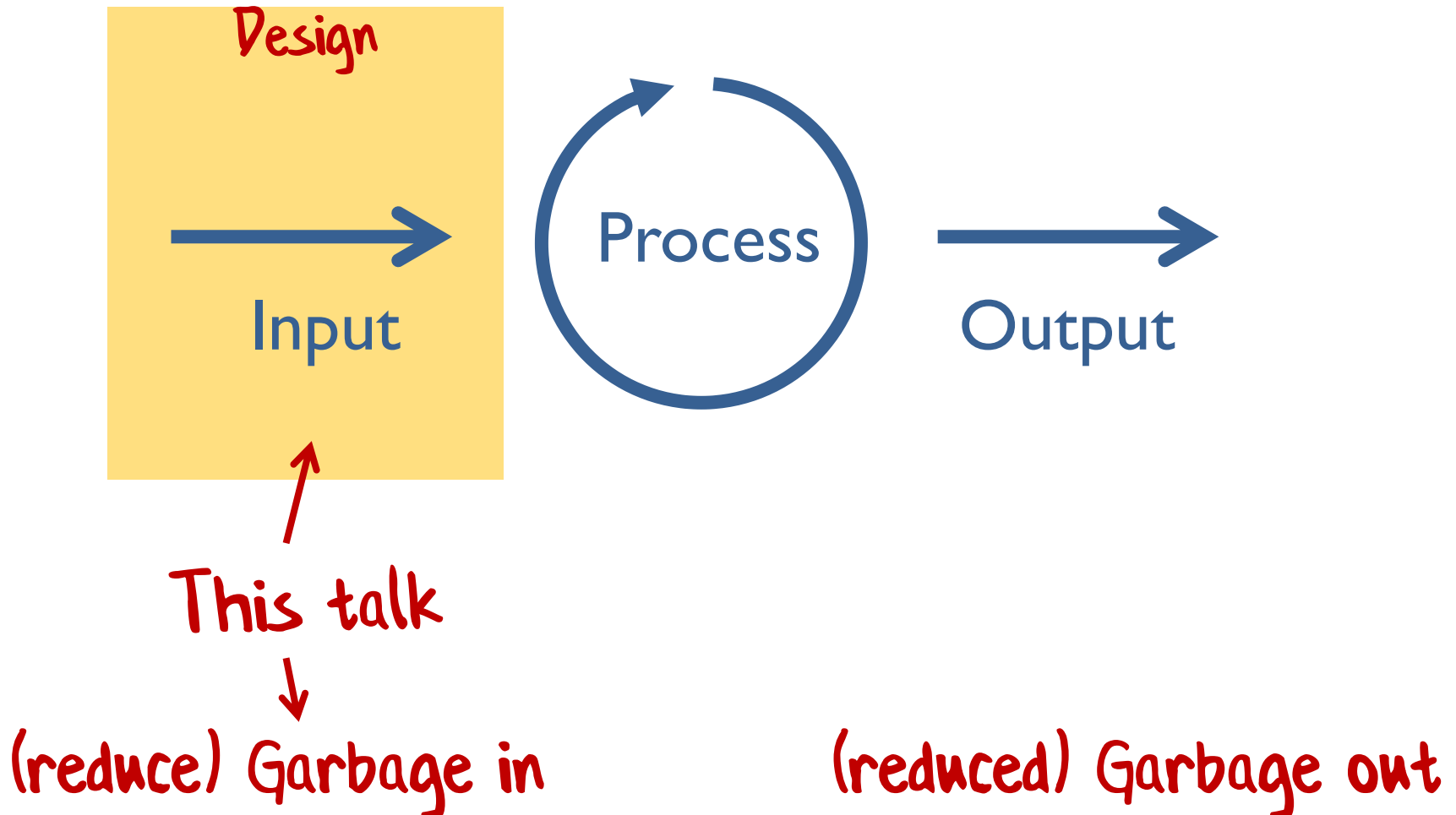
The software development process

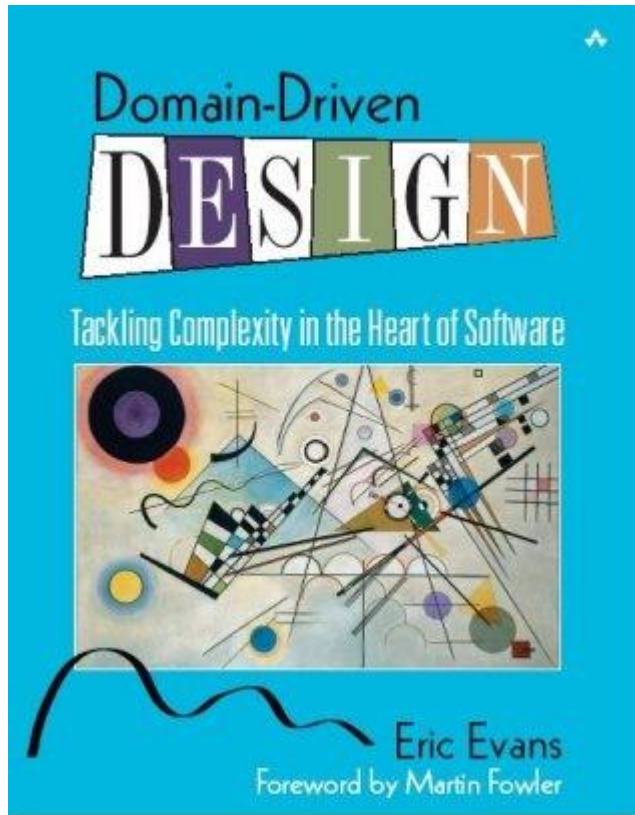


The software development process



The software development process

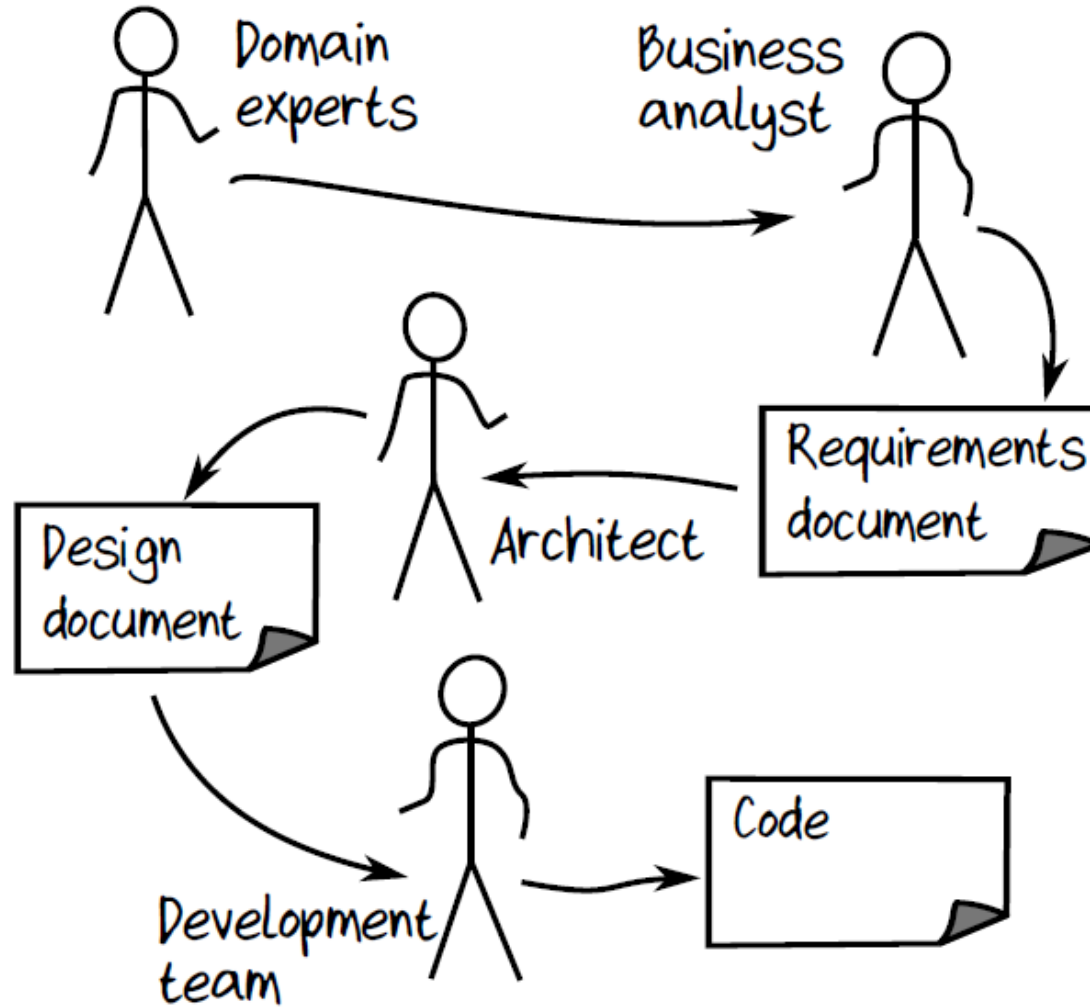




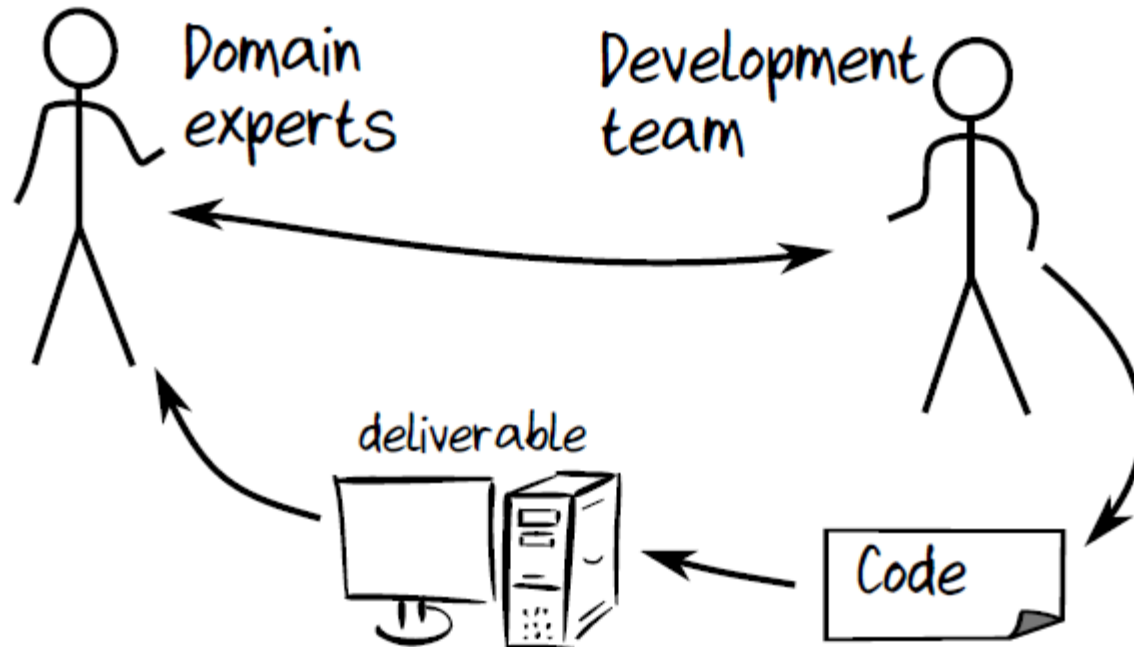
"Focus on the domain and
domain logic rather than
technology"
-- Eric Evans

Why Domain-Driven Design?

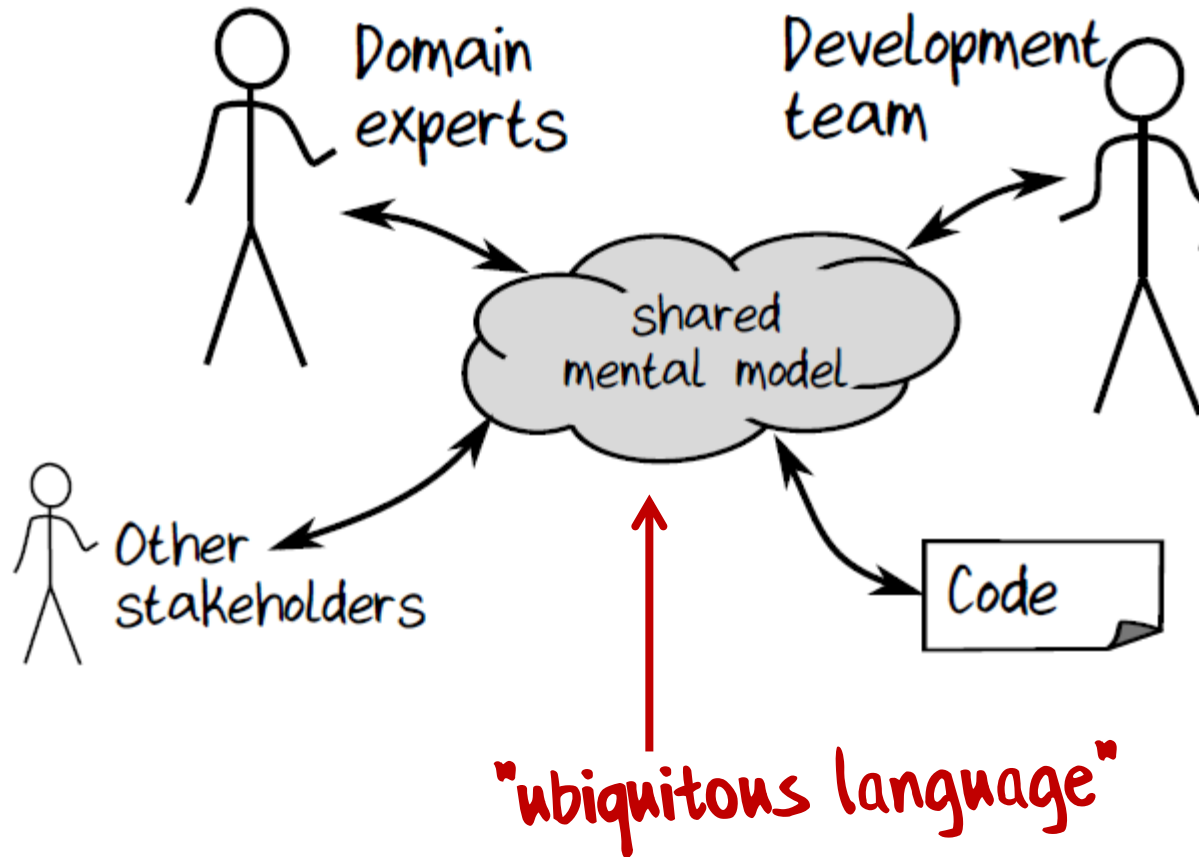
Waterfall



Agile



DDD



Understanding the domain
through business events

Why events are important

- A business doesn't just *have data*, it *transforms it somehow*
 - The value of the business is created in this process of transformation
- Data that is sitting there unused is not contributing anything
- What causes an employee/process to start working with that data and adding value?
 - An event!

Finding the events with Event Storming

order form
received

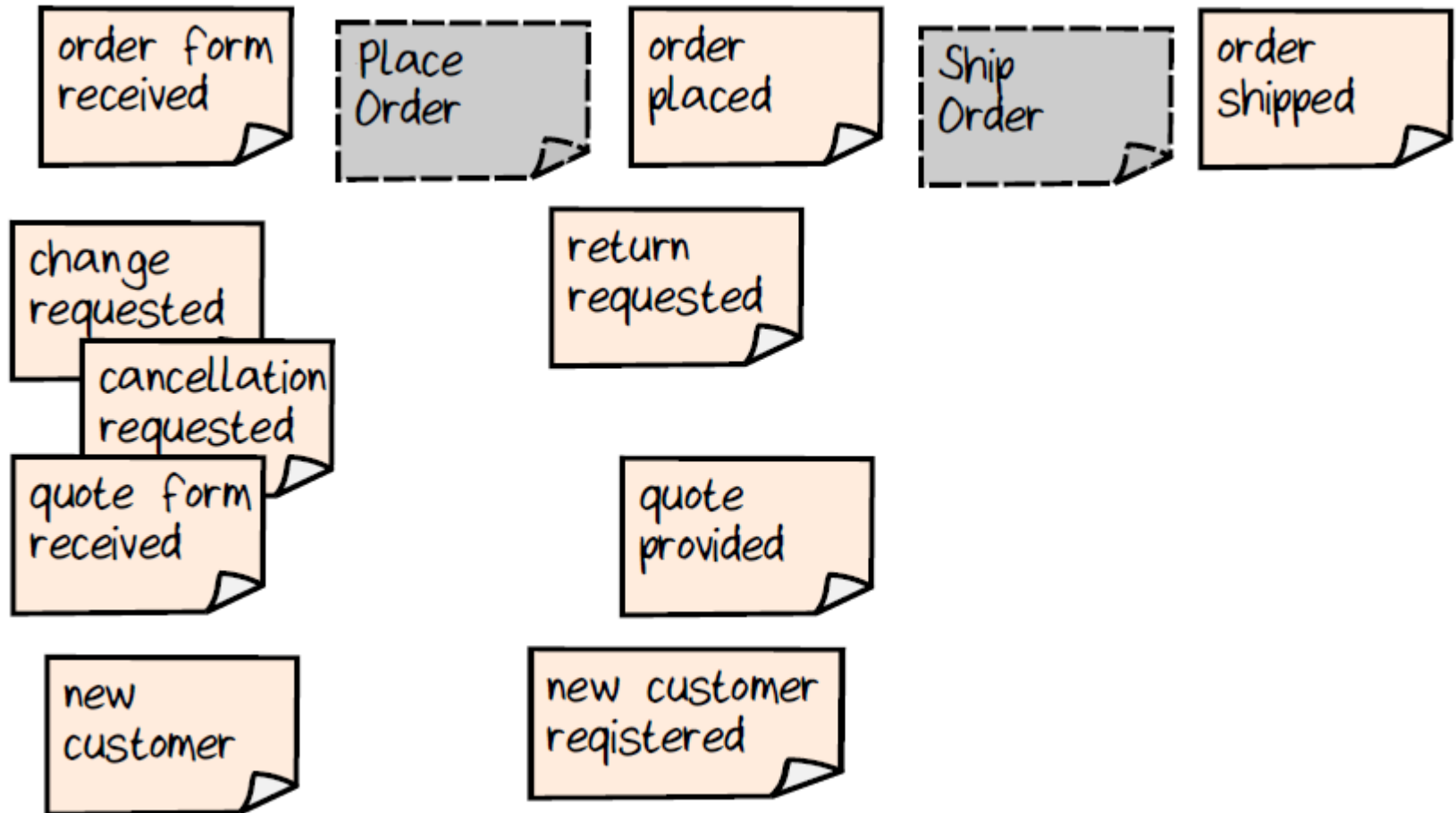
change
requested

cancellation
requested

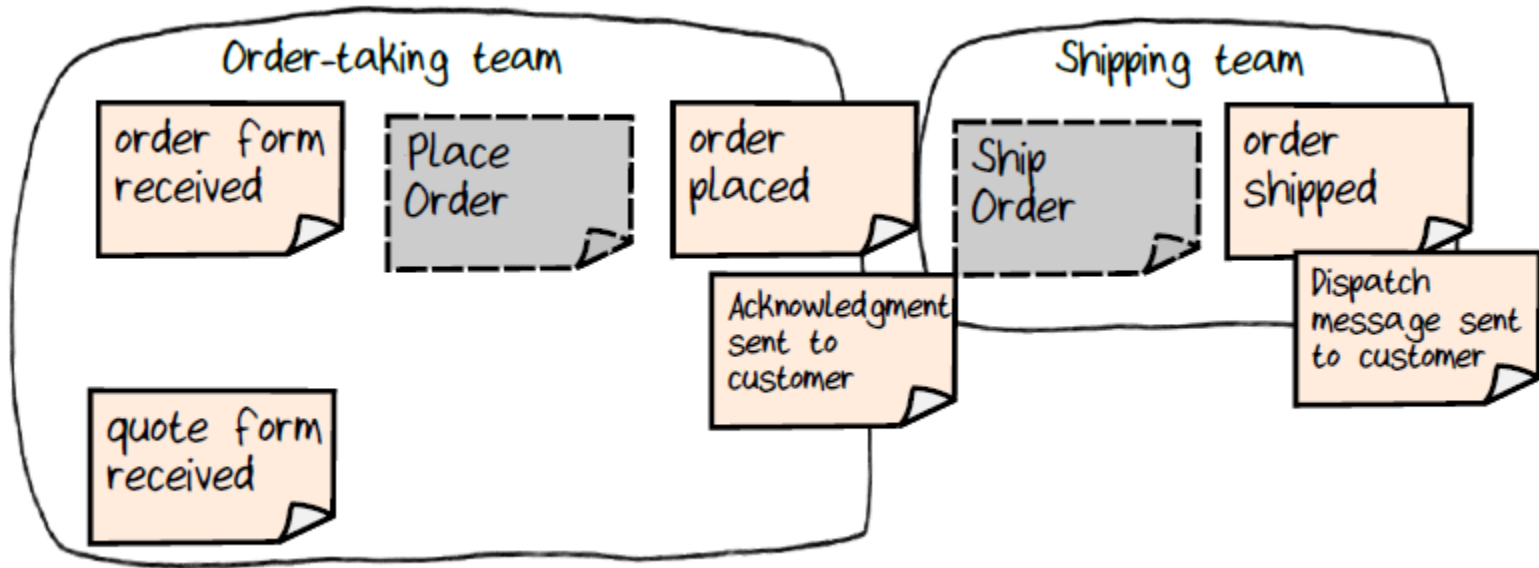
quote form
received

new
customer

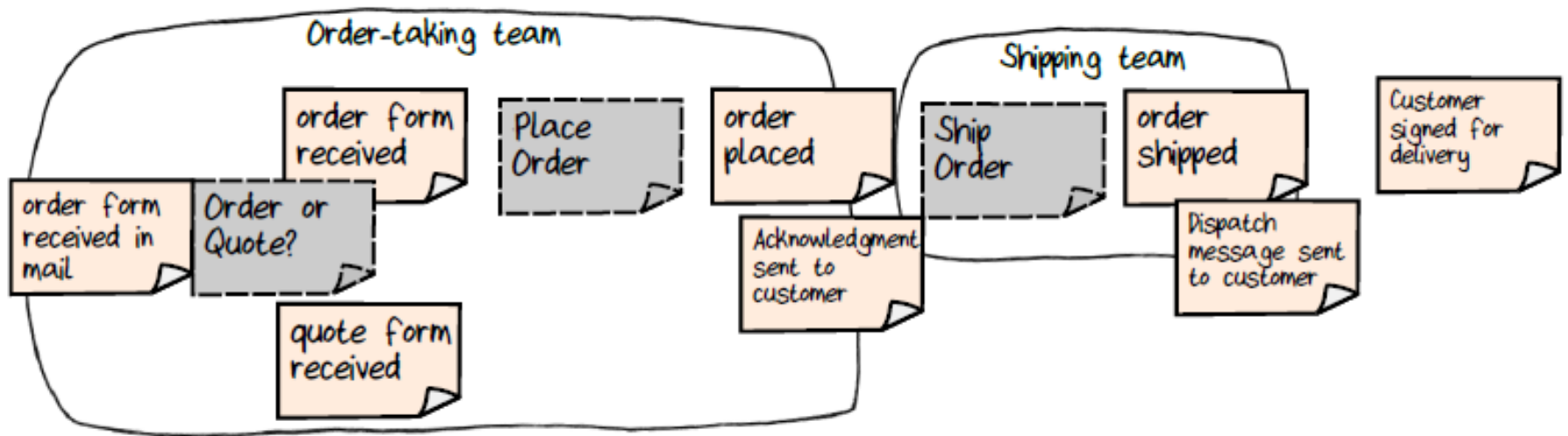
Finding the events with Event Storming



Then group the events

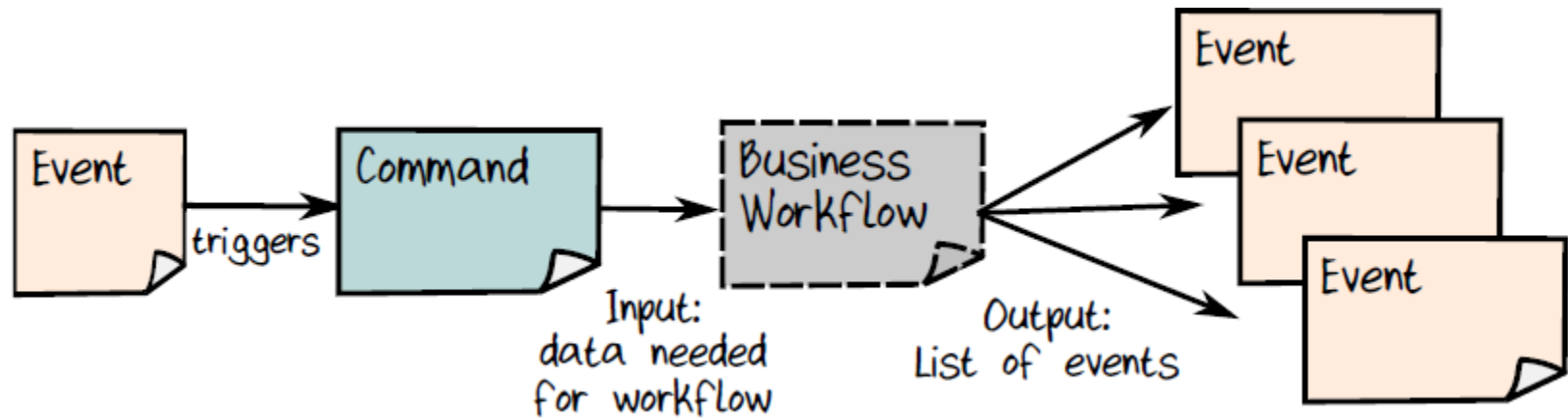


And extend to the edges



Introducing "workflows"

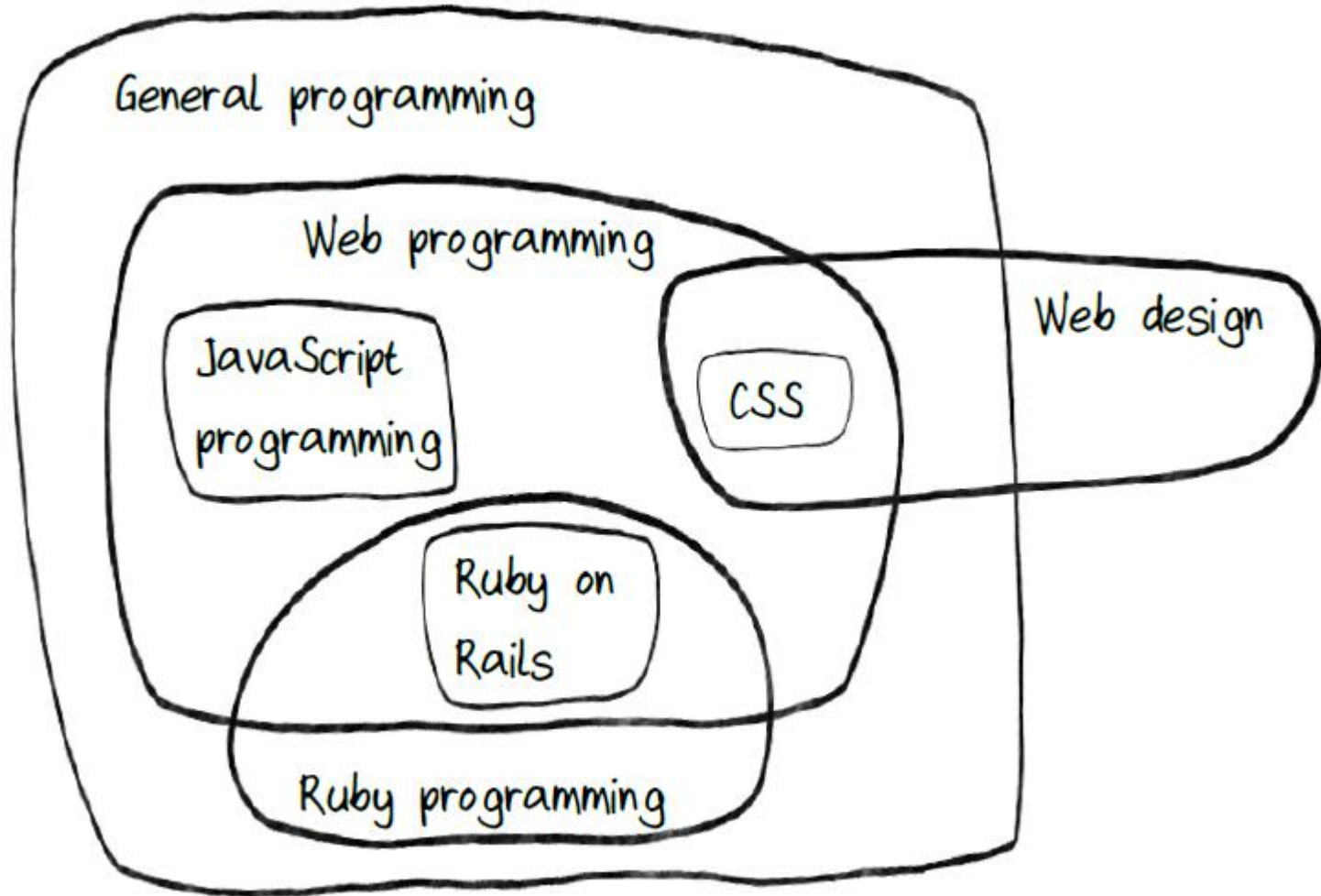
Events, commands, workflows



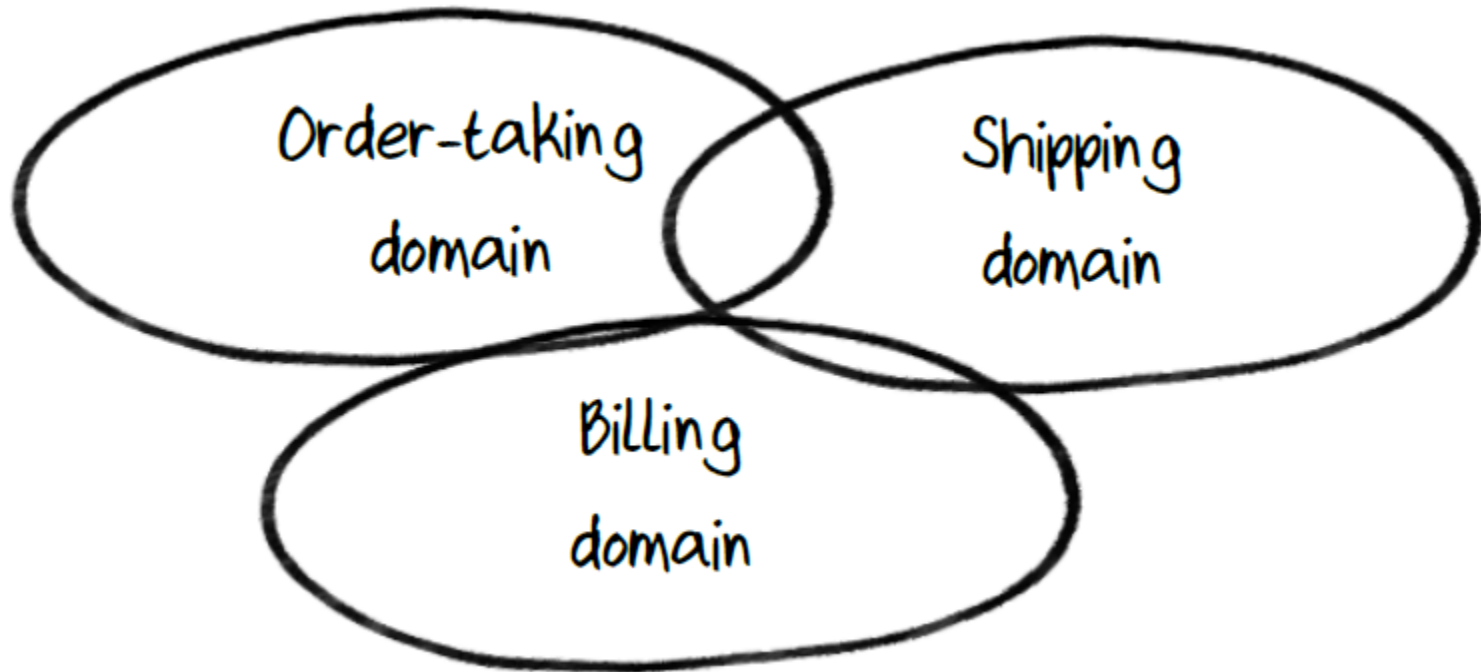
These are the units of work when coding

Introducing "subdomains"

What is a subdomain?



What is a subdomain?

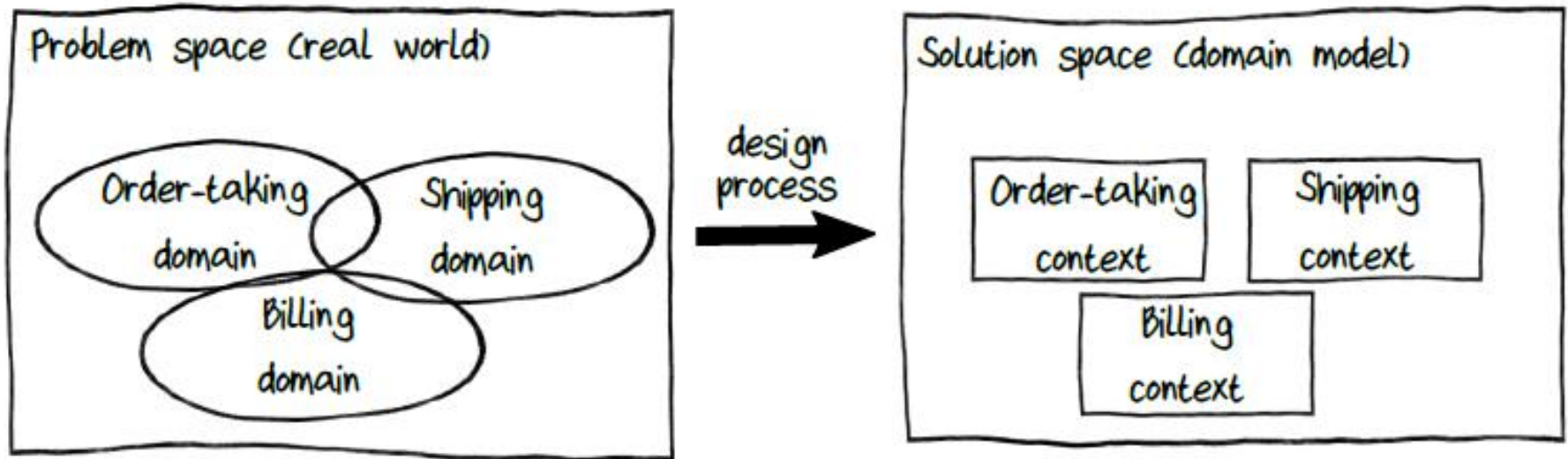


Introducing "bounded contexts"

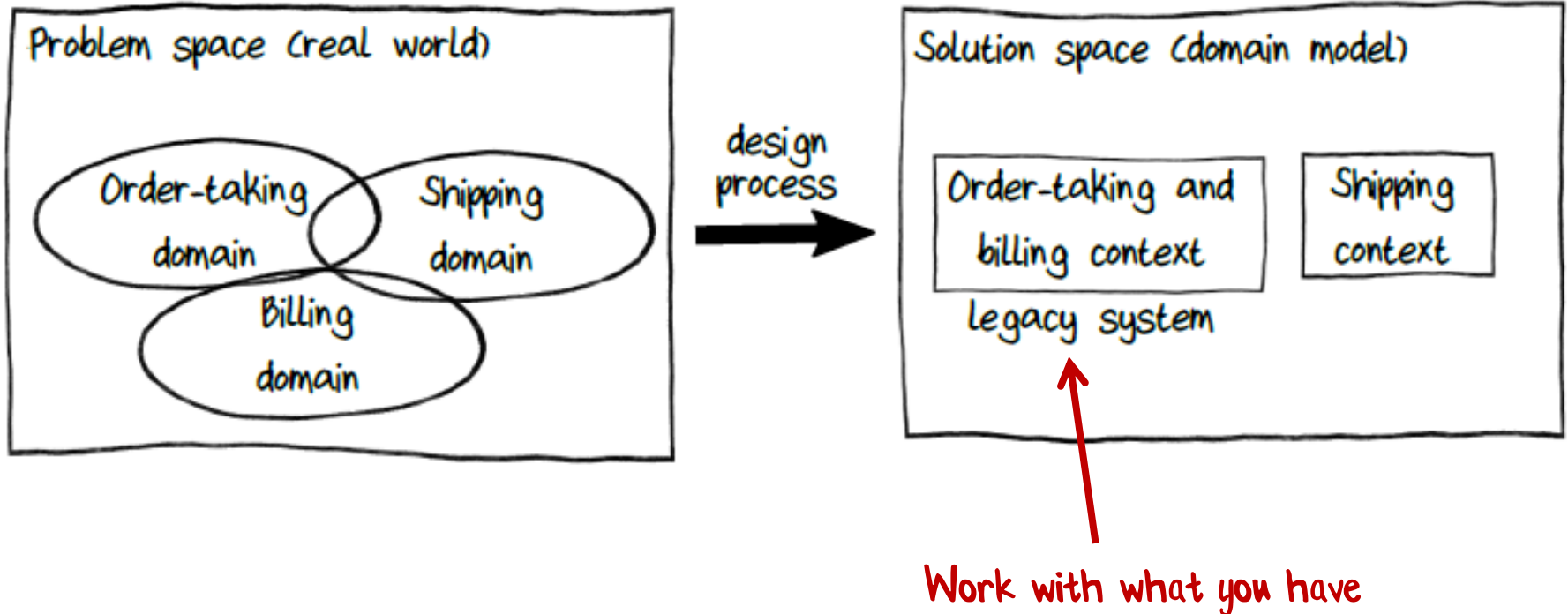
"Problem space" vs. "Solution space"

- The solution is a *model* of the problem domain
 - Only contains aspects of the domain that are relevant!
- A "Subdomain" is in the problem space
- A "Bounded context" is in the solution space

"Problem space" vs. "Solution space"



"Problem space" vs. "Solution space"



Why "Bounded Context"?

- Focus on what is important
 - being aware of the context
 - being aware of the boundaries.
- "*Context*"
 - Specialized knowledge and common language
 - Information taken out of context is confusing or unusable
- "*Bounded*"
 - We want to reduce coupling
 - Contexts must evolve independently!

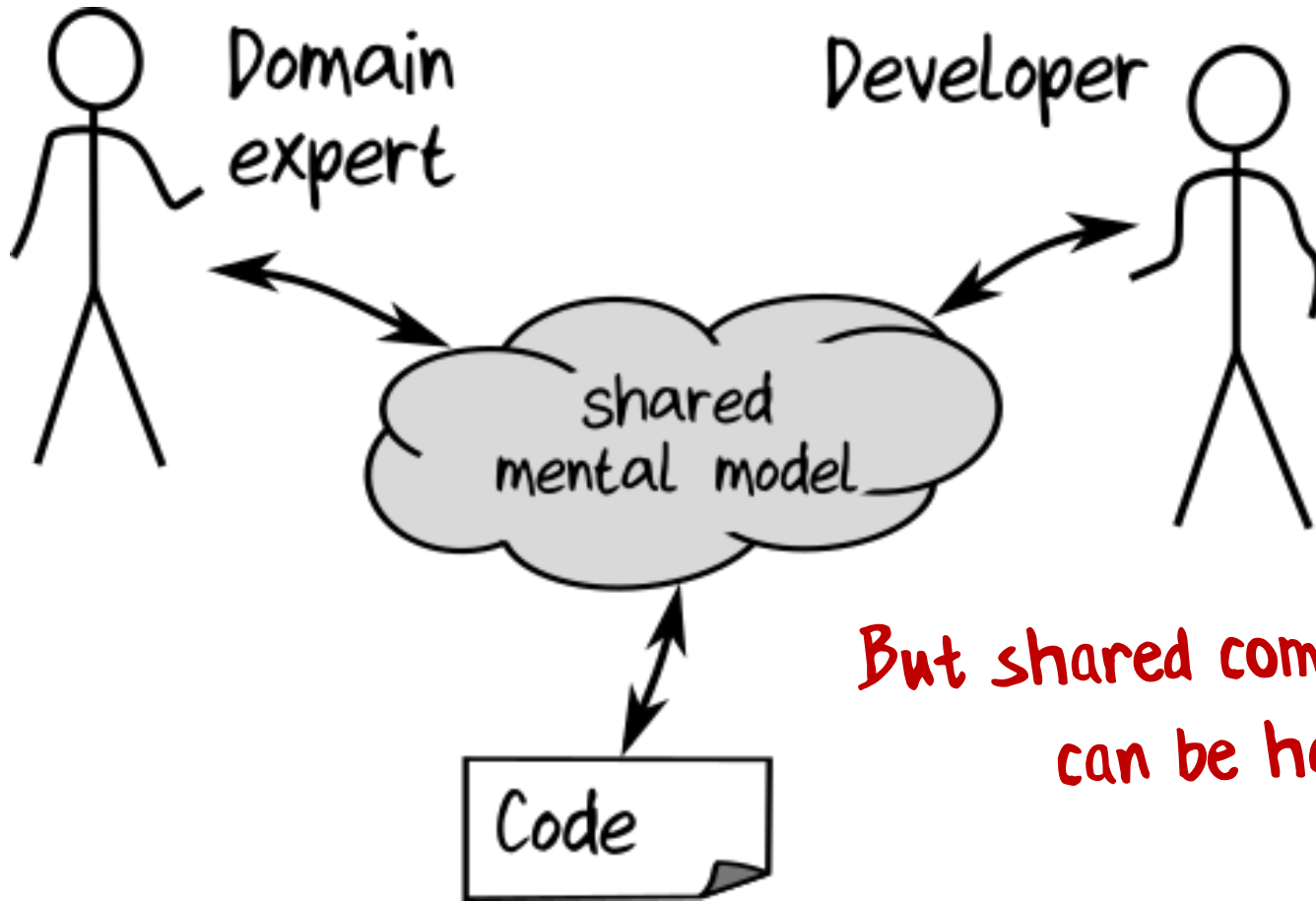
How to get the contexts right

- Listen to the domain experts!
 - Pay attention to existing team and department boundaries
- Don't forget the “bounded” part of a bounded context
 - Watch out for scope creep when setting boundaries
- Design for autonomy
 - If two groups contribute to the same bounded context, they might end up pulling the design in different directions as it evolves
 - Better to have separate bounded contexts that can evolve independently than one mega-context that tries to make everyone happy

Introducing "ubiquitous language"

Ubiquitous Language

- The Ubiquitous Language is a set of concepts and vocabulary associated with the domain and is shared by
 - Domain experts
 - Development team
 - The source code
- The "everywhere language"

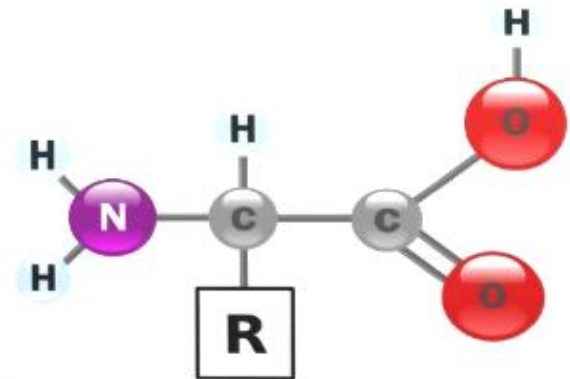


**But shared communication
can be hard!**

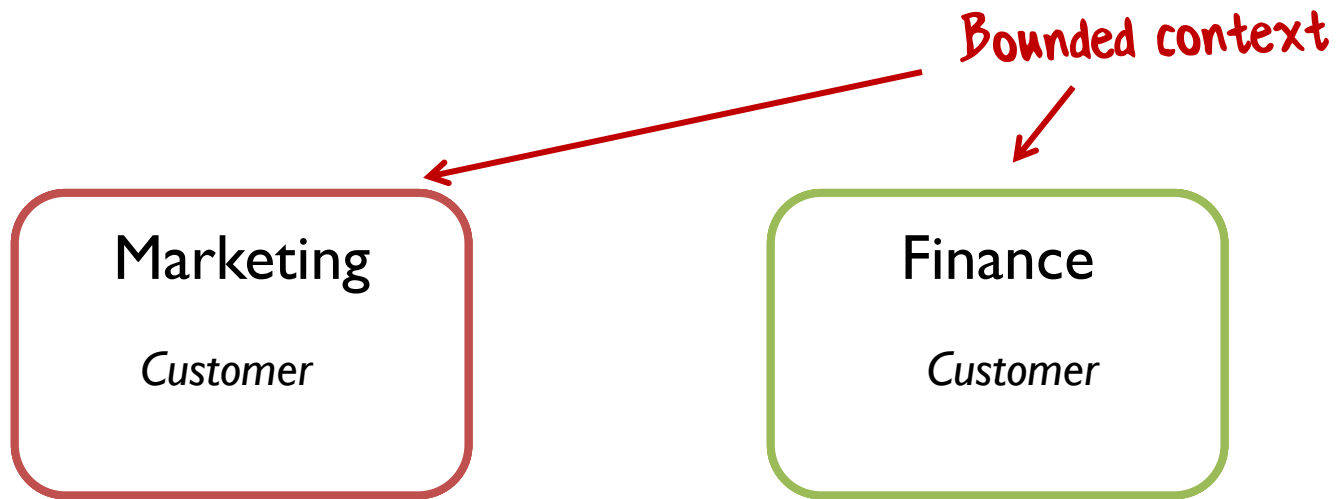
U-N-I-O-N-I-Z-E



α **AMINO ACID**



IN ITS UN-IONIZED FORM



Warehouse

Product Stock Transfer Depot Tracking

Ubiquitous Language

← Bounded context
module **CardGame** =

type **Suit** = Club | Diamond | Spade | Heart

type **Rank** = Two | Three | Four | Five | Six | Seven | Eight
| Nine | Ten | Jack | Queen | King | Ace

type **Card** = Suit * Rank

type **Hand** = Card list

type **Deck** = Card list

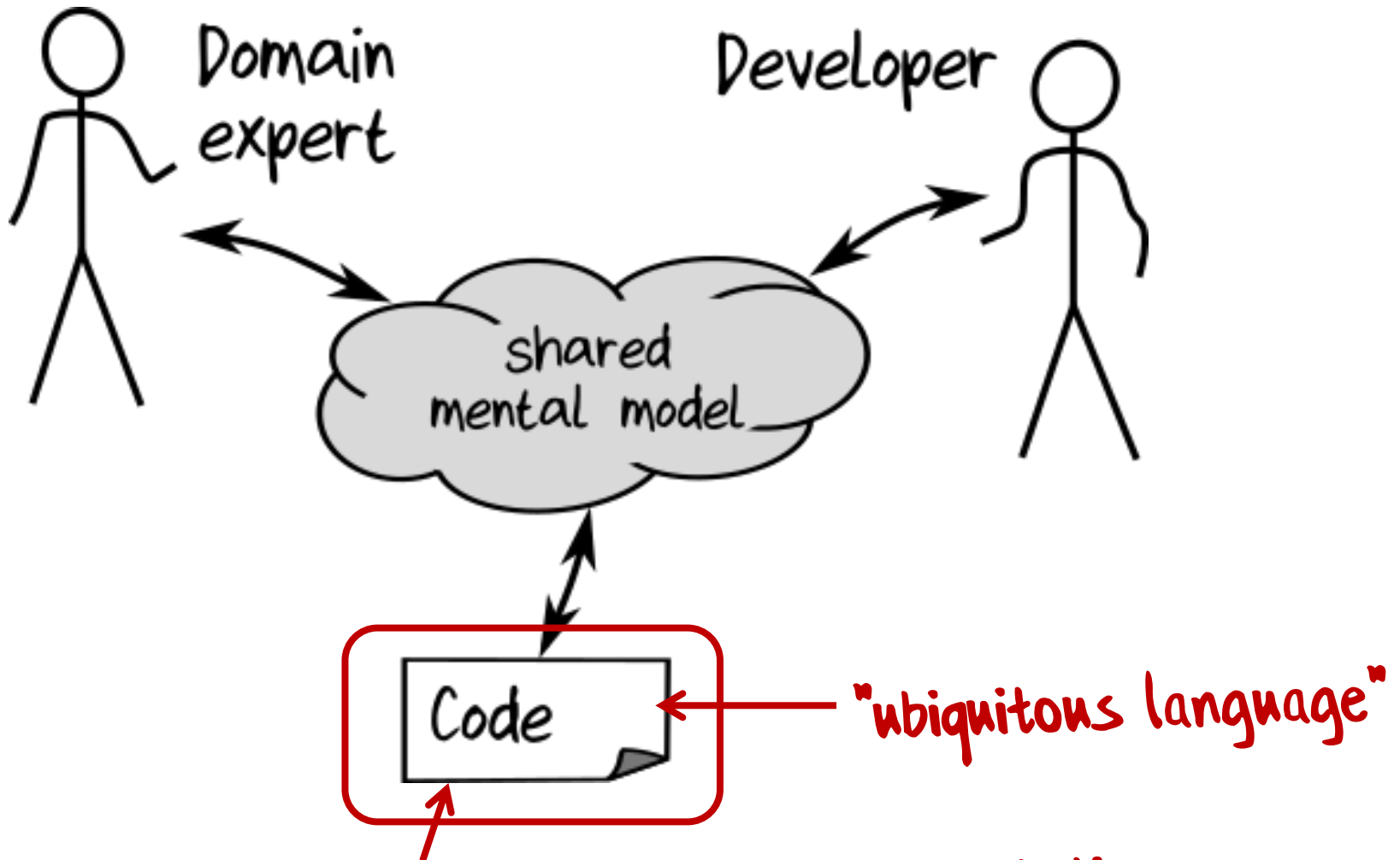
type **Player** = {Name:string; Hand:Hand}

type **Game** = {Deck:Deck; Players: Player list}

type **Deal** = Deck → (Deck * Card)

type **PickupCard** = (Hand * Card) → Hand

Ubiquitous language



Domain model == F# code == documentation

module **CardGame** =

type **Suit** = Club | Diamond | Spade | Heart

type **Rank** = Two | Three | Four | Five | Six | Seven | Eight
| Nine | Ten | Jack | Queen | King | Ace

type **Card** = Suit * Rank

type **Hand** = Card list

type **Deck** = Card list

type **Player** = {Name:string; Hand:Hand}

type **Game** = {Deck:Deck; Players: Player list}

type **Deal** = Deck → (Deck * Card)

type **PickupCard** = (Hand * Card) → Hand

'|' means a choice -- pick one from the list

'*' means a pair. Choose one from each type
list type is built in

X → Y means a function
- input of type X
- output of type Y

module **CardGame** =

*Do you think this is a reasonable amount
of code to write for this domain?*

type **Suit** = Club | Diamond | Spade | Heart

type **Rank** = Two | Three | Four | Five | Six | Seven | Eight
| Nine | Ten | Jack | Queen | King | Ace

type **Card** = Suit * Rank

*Do you think a non-
programmer could
understand this?*

type **Hand** = Card list

type **Deck** = Card list

type **Player** = {Name:string; Hand:Hand}

type **Game** = {Deck:Deck; Players: Player list}

type **Deal** = Deck → (Deck * Card)

type **PickupCard** = (Hand * Card) → Hand

module **CardGame** =

type **Suit** = Club | Diamond | Spade | Heart

type **Rank** = Two | Three | Four | Five | Six | Seven | Eight
| Nine | Ten | Jack | Queen | King | Ace

type **Card** = Suit * Rank

"persistence ignorance"

type **Hand** = Card list

type **Deck** = Card list

*"The design is the code,
and the code is the design."*

type **Player** = {Name:string; Hand:Hand}

*This is not pseudocode —
this is executable code!*

type **Game** = {Deck:Deck; Players: Player list}

type **Deal** = Deck → (Deck * Card)

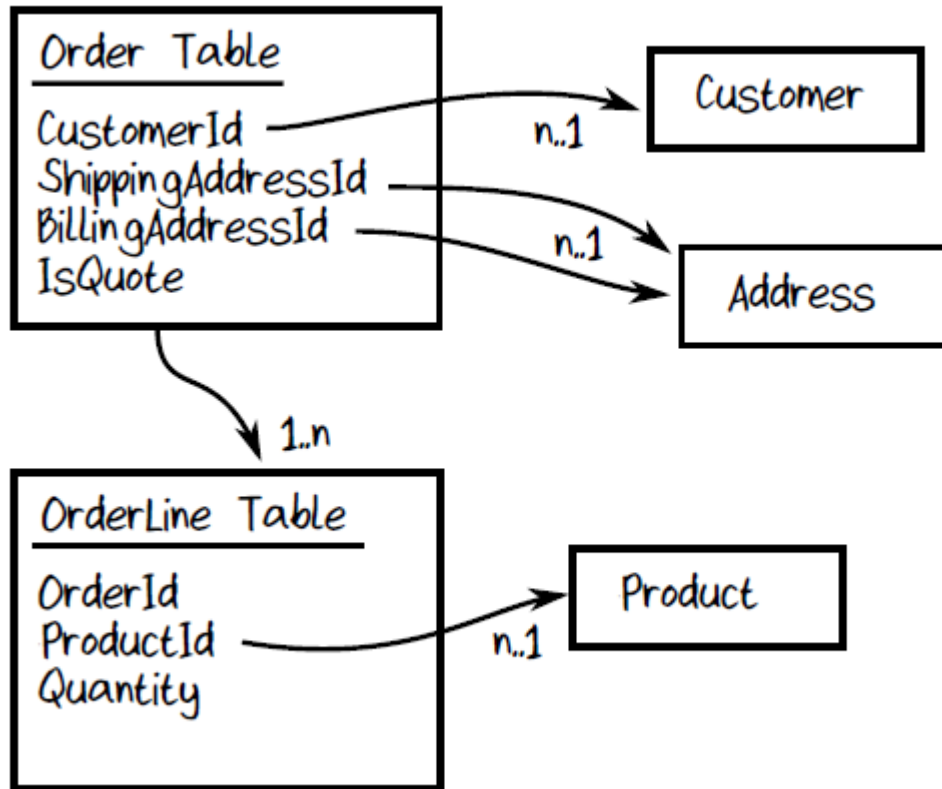
type **PickupCard** = (Hand * Card) → Hand

Summary of DDD concepts

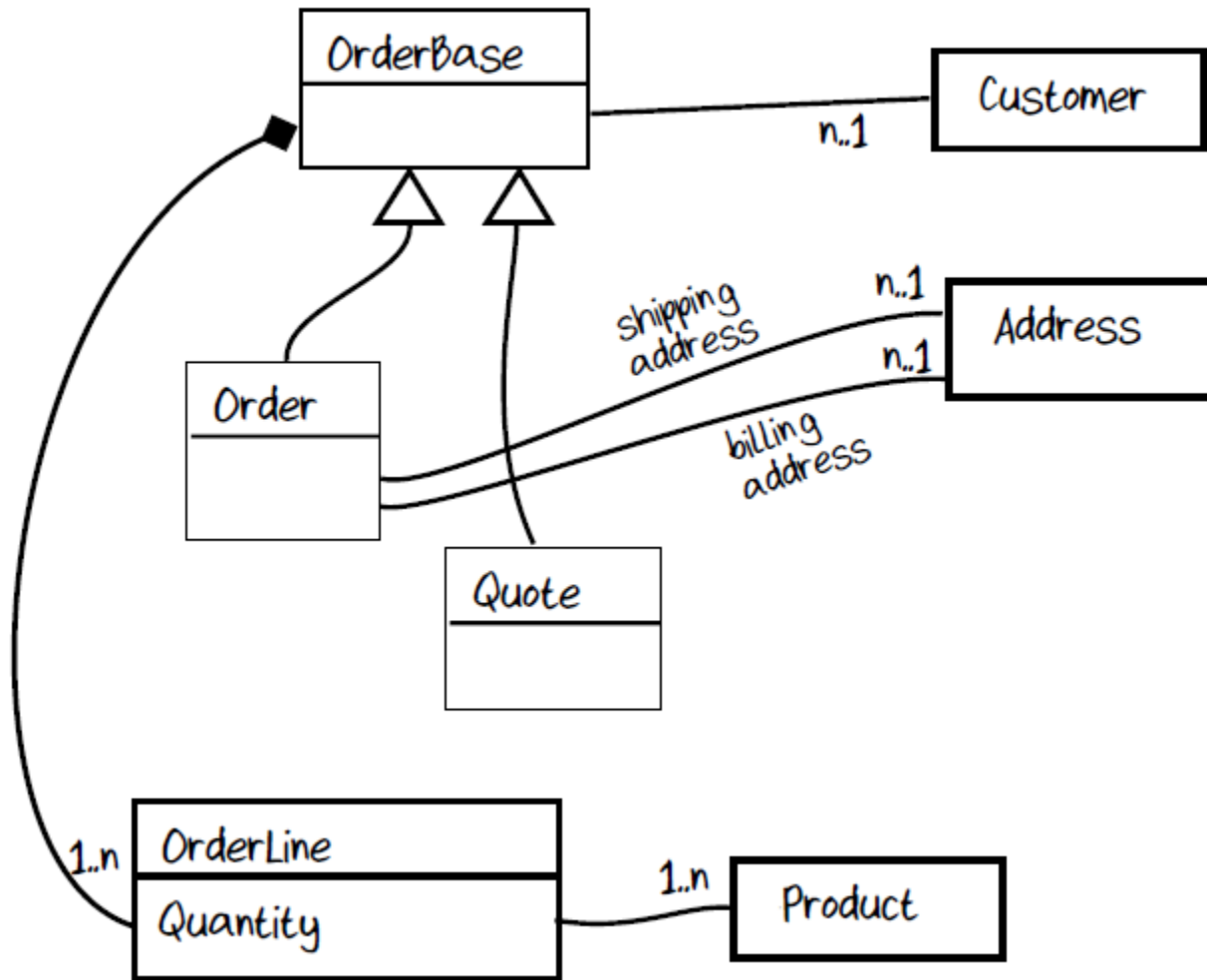
- Domain Model
 - A set of simplifications that represent those aspects of a domain that are relevant to a particular problem.
 - The domain model is part of the solution space
- Bounded context
 - A subsystem in the solution space with clear boundaries that distinguish it from other subsystems.
- Ubiquitous Language
 - A set of concepts and vocabulary associated with the domain and is shared by both the team members and the source code.
- Domain Event
 - A record of something that happened in the system. It is always described in the past tense. An event often triggers additional activity.
- Command
 - A request for some process to happen, triggered by a person or another event. If the command succeeds, the state of the system changes and one or more Domain Events are recorded.

Getting started with domain modeling

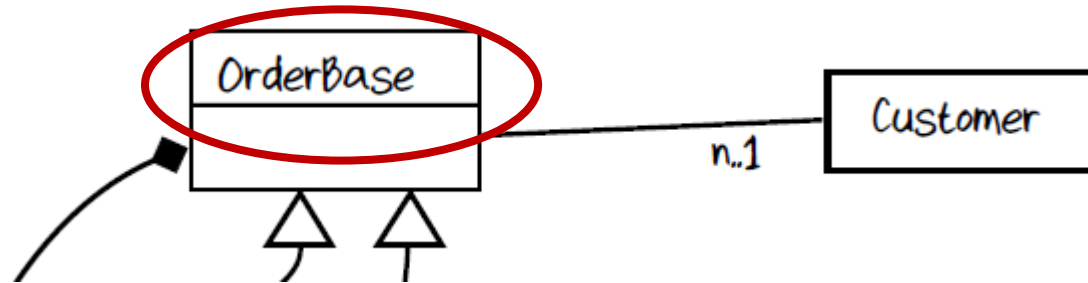
It's not database design!



It's not OO design!



It's not OO design!



Pro tip: If you have a "base", "factory", "manager", or "helper" class then you're doing it wrong!

A domain expert wouldn't know what you meant by these words.

My recommended way of
domain modeling

Start with an event and workflow

Bounded context: Order-Taking

Workflow: "Place order"

triggered by:

"Order form received" event

primary input:

An order form

other input:

Product catalog

output events:

"Order Placed" event

side-effects:

An acknowledgment is sent to the customer,
along with the placed order

Document the data with AND

```
data Order =  
  CustomerInfo  
  AND ShippingAddress  
  AND BillingAddress  
  AND list of OrderLines  
  AND AmountToBill
```

```
data OrderLine =  
  Product  
  AND Quantity  
  AND Price
```

```
data CustomerInfo = ??? // don't know yet  
data BillingAddress = ??? // don't know yet
```

Never use primitive types in a domain model

```
data Customer = string AND string
```

```
data OrderLine = int AND int
```



Important! "int" and "float"
are not domain concepts

```
data Customer = FirstName AND LastName
```

```
data OrderLine = ProductId AND Quantity
```



Document choices with OR

```
data ContactInfo =  
  EmailAddress  
  OR PhoneNumber
```

```
data OrderQuantity =  
  UnitQuantity  
  OR KilogramQuantity
```

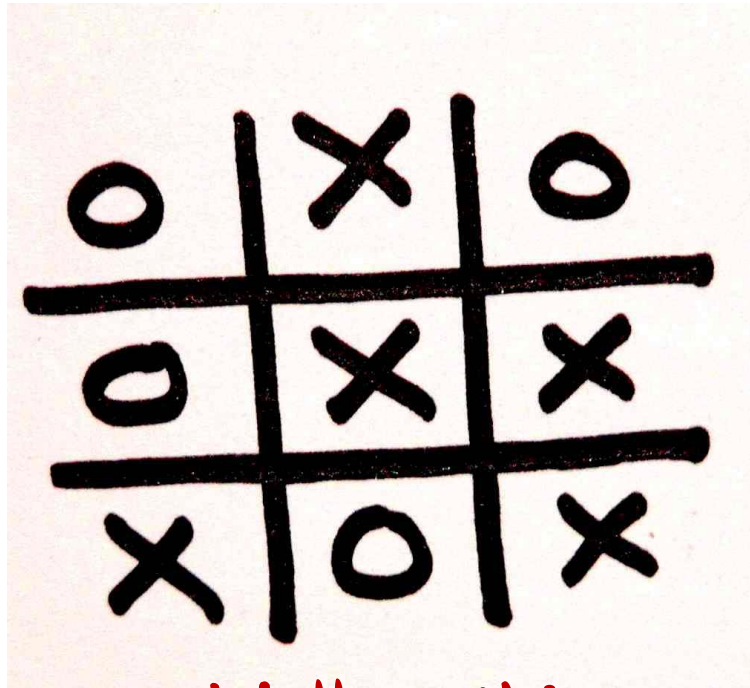
```
data UnitQuantity = integer between 1 and ?  
data KilogramQuantity = decimal between ? and ?
```

Exercises

For each of these domains, document the events and the associated data.
Then paste your domain models into the GDocs file.

Exercise:

Tic-Tac-Toe / Noughts and Crosses / Kruisje-rondje



We'll do this one together

Exercise: ATM / Cash machine



Exercise: Microwave



Exercise: Self-service coffee maker



Exercise: Amazon shopping

Checkout (1 item)

1 Shipping address

Edit

Address - 20 locations near this address

2 Payment method

Edit

Billing address: Same as shipping address. [Change](#)

+ Add a gift card or promotion code

3 Review items and shipping

Get a \$5.99 credit for Prime Pantry
Skip the trip and receive a \$5.99 Prime Pantry credit once this order ships when you choose FREE No-Rush Shipping.

Guaranteed delivery date: Mar. 19, 2016 If you order in the next 8 hours and 28 minutes ([Details](#))
Items shipped from Amazon.com

Pink and White 100 Ft Pennant Stringer w/48 Large Flags by Pudgy Pedro's Party Supplies
\$8.49 ✓Prime
Qty: 1 ±
Sold by: E-Brands

and see other gift options

Choose your Prime delivery option:

- ☐ \$3.99 One-Day Shipping — get it tomorrow, Mar. 18
- ☒ FREE Two-Day Shipping — get it Saturday, Mar. 19
- ☐ FREE Standard Shipping — get it Wednesday, Mar. 23
- ☐ FREE No-Rush Shipping — get it Thursday, Mar. 24

Get a \$5.99 credit for Prime Pantry. [Details](#)

Order total: \$8.49

By placing your order, you agree to Amazon.com's [privacy notice](#) and conditions of use.

Place your order

By placing your order, you agree to Amazon.com's [privacy notice](#) and conditions of use.

Order Summary

Items:	\$8.49
Shipping & handling:	\$0.00
Total before tax:	\$8.49
Estimated tax to be collected:	\$0.00

Order total: \$8.49

[How are shipping costs calculated?](#)

Prime shipping benefits have been applied to your order. (Why didn't I qualify for Prime FREE Same/One Day?)

*Why has sales tax been applied? See [tax](#) and [seller information](#).

Need help? Check our [Help pages](#) or [contact us](#)

For an item sold by Amazon.com: When you click the "Place your order" button, we'll send you an email message acknowledging receipt of your order. Your

Exercise: Ebay-style bidding

[← Back to list of items](#) Listed in category: [Toys & Hobbies](#) > [TV, Movie, Character Toys](#) > [Smurfs](#)

1970's PROMOTIONAL SMURF B/P - Pair on Base

Bidder or seller of this item? [Sign in](#) for your status



[↓ Go to larger picture](#)

Current bid: **US \$81.00** ([Reserve not met](#))

[Place Bid >](#)

Time left: **15 hours 36 mins**
7-day listing
Ends Oct-21-03 19:16:18 PDT

History: [9 bids](#) (US \$9.99 starting bid)

High bidder: [pbrown-pfau](#) ([326](#) ★)

Exercise: Your own favourite domain



End