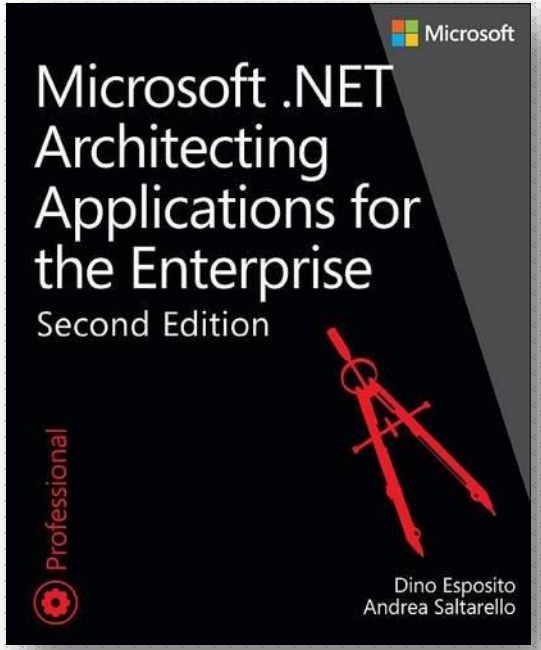
Architecting .NET Solutions for the Enterprise

Dino Esposito
JetBrains

dino.esposito@jetbrains.com
@despos
facebook.com/naa4e

ASP INSIDERS

MVP Microsoft Most Valuable Professional



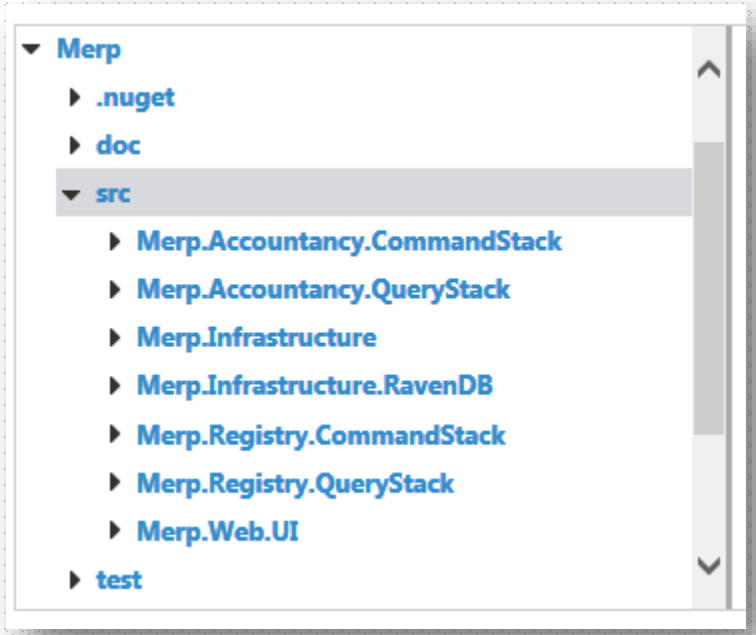
WARNING

This is NOT simply a shameless plug but a truly helpful reference ☺

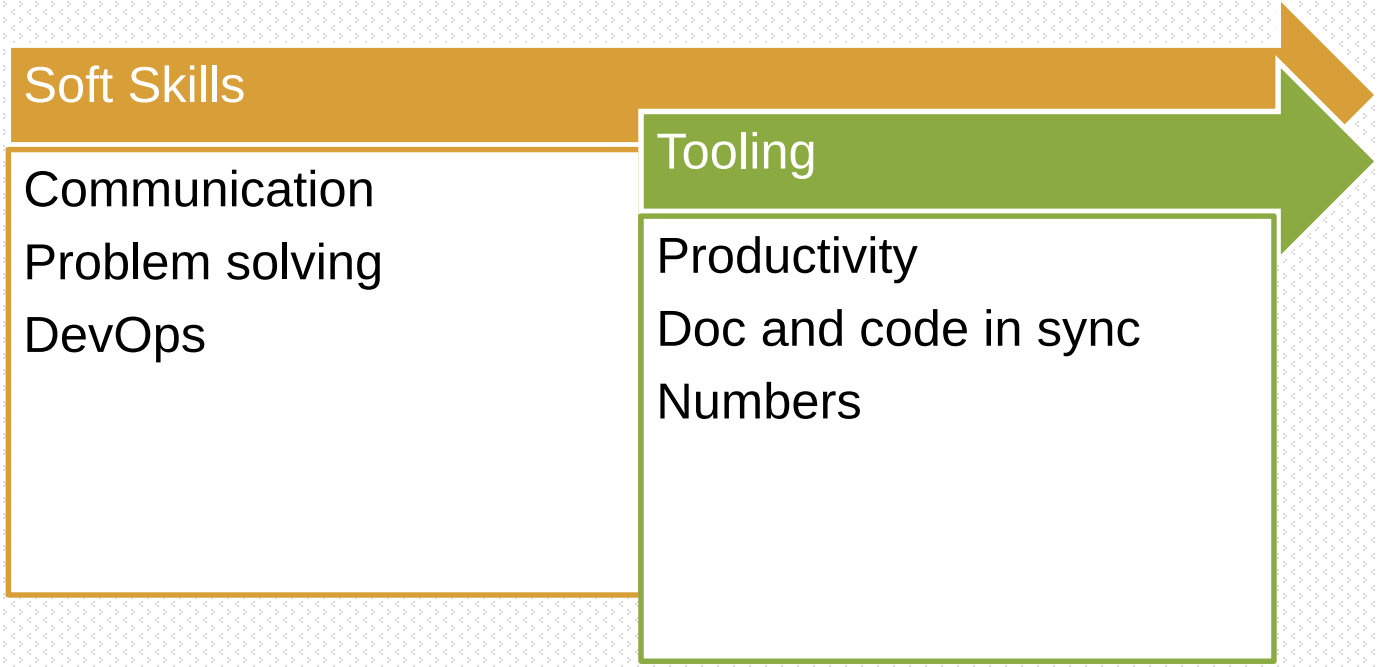
"I will say that in a number of cases, a page from this book erased a mass of confusion I'd acquired from Vaughn Vernon's Implementing Domain-Driven Design. This was written in a much more concise, clear, practical manner than that book."

—(non anonymous) Amazon reviewer

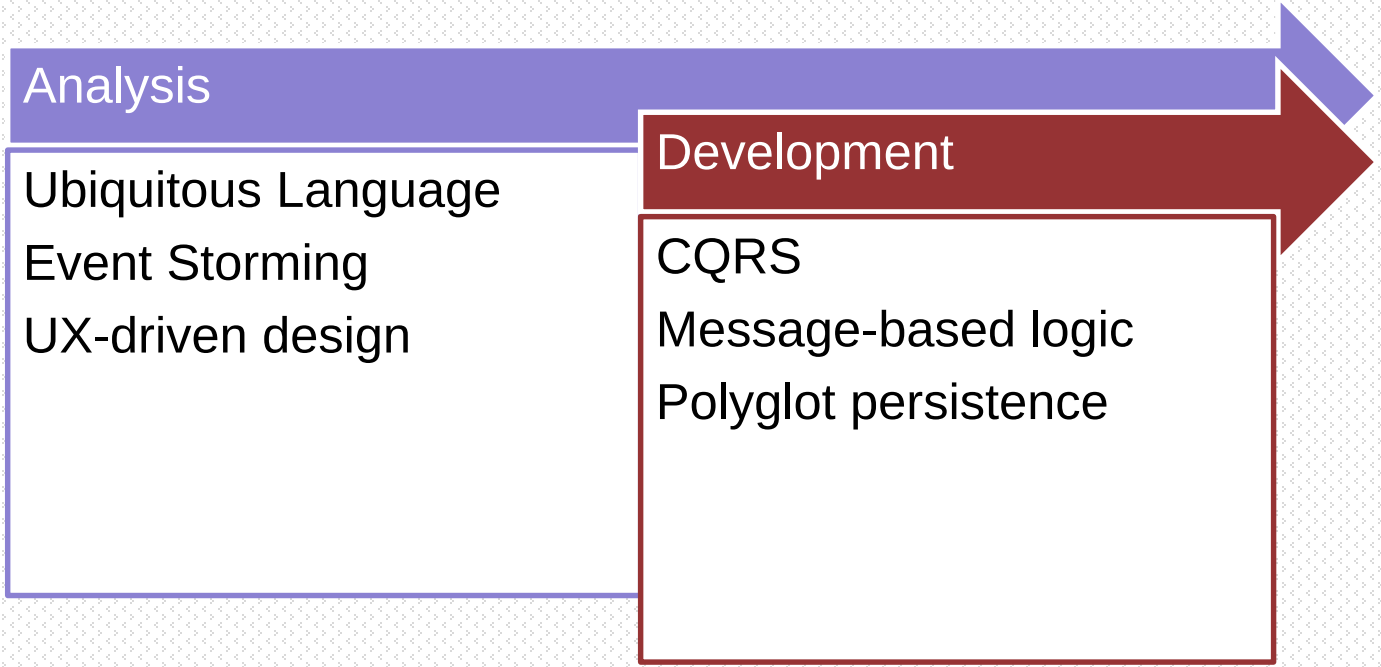
http://naa4e.codeplex.com



Line-of-business



Line-of-business



Conducting Domain Analysis

Why Is DDD So Intriguing?

Captures known elements of the design process

Organizes them into a set of principles

Domain modeling is the focus of development

Different way of building business logic

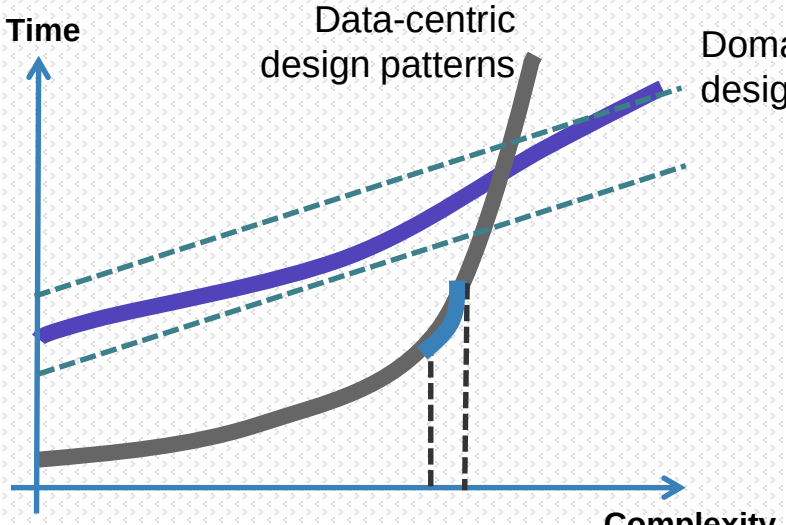
DDD Key Misconception

It's all about using objects and hardcode business behavior in objects.

- Persistence?
- External services
- Cross-objects business logic?
- Business events?

The Secret Dream of Any Developer

An all-encompassing object model describing the entire domain



NOTE: Adapted from Martin Fowler's PoEAA

Give me enough time and enough specs and I'll build the world for you.

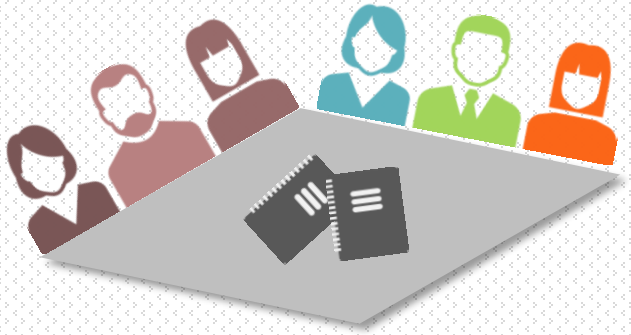
DDD Is Still About Business Logic

Crunch knowledge about the domain

Recognize subdomains

Design a rich domain model

Code by telling objects in the domain model what do to



At Work Defining the Ubiquitous Language

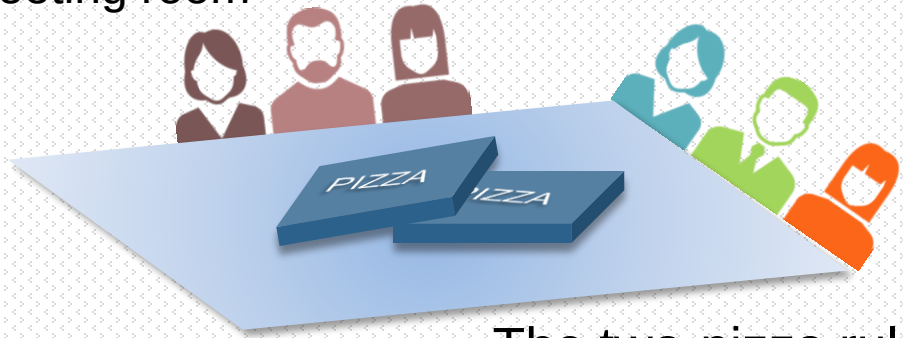
Delete the booking	→	Cancel the booking
Submit the order	→	Checkout
Update the job order	→	Extend the job order
Create the invoice	→	Register/Accept the invoice
Set state of the game	→	Start/Pause the game

Event Storming

Event Storming

Exploring a business domain starting from observable domain events

Developers and domain experts
together in a meeting room

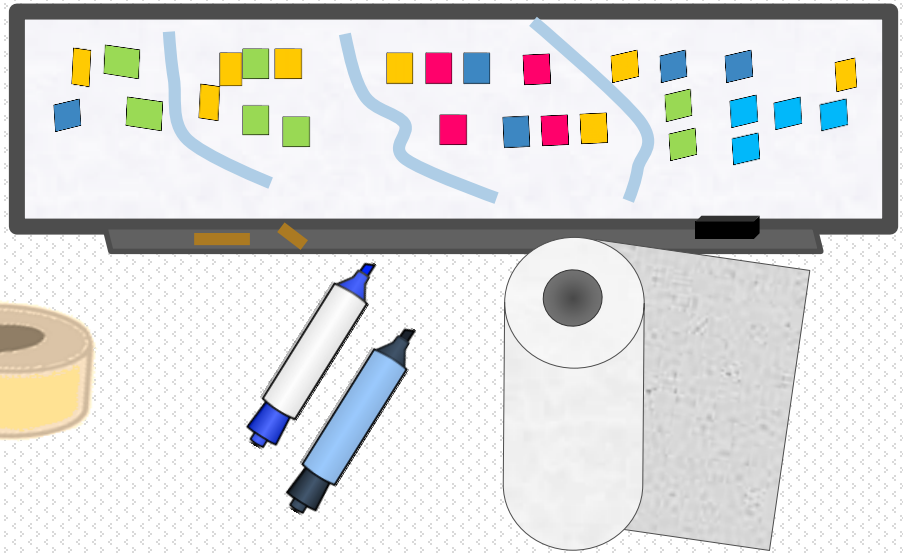


The two-pizza rule sets the right
number of invited people

Event Storming

Exploring a business domain starting from observable domain events

Necessary equipment



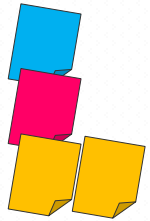
How It Works

Identify relevant domain events

- Use a sticky note of a given color to put events on the wall

Find what causes the event

- **User action**? Add a sticky note of a different color
- **Asynchronous event**? Add a sticky note of a different color
- **Another event**? Add another sticky of same color on top



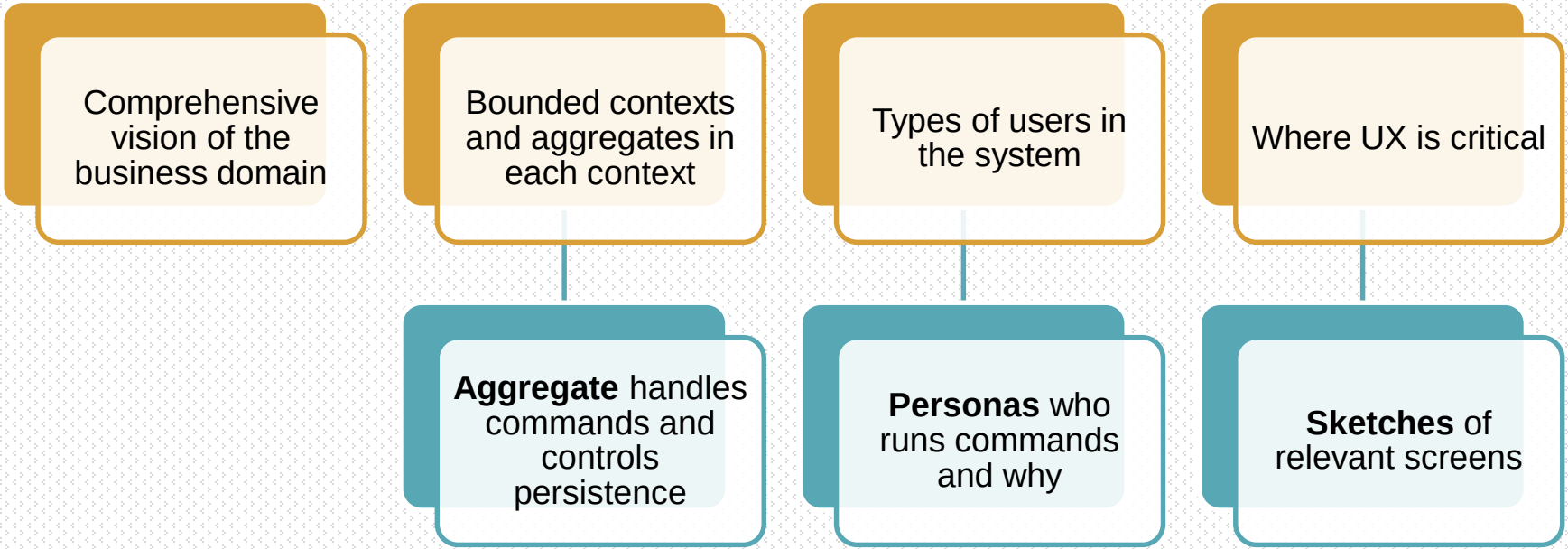
Look at the modeling surface as a timeline

- Add notes with markers

Facilitator

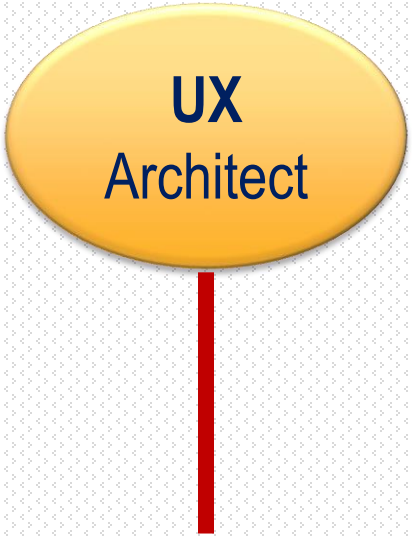
- **Leads the meeting**
  - Starts the meeting asking questions
  - Sticks first notes on the wall to show the way
- **Guides the modeling effort**
  - Asks question to better understand the emerging model
  - Ensures ideas are represented accurately
  - Keep focused and moves ahead

Benefits


Comprehensive vision of the business domain
Bounded contexts and aggregates in each context
Types of users in the system
Where UX is critical
Aggregate handles commands and controls persistence
Personas who runs commands and why
Sketches of relevant screens

UX-driven Design

Two distinct architect roles acting together



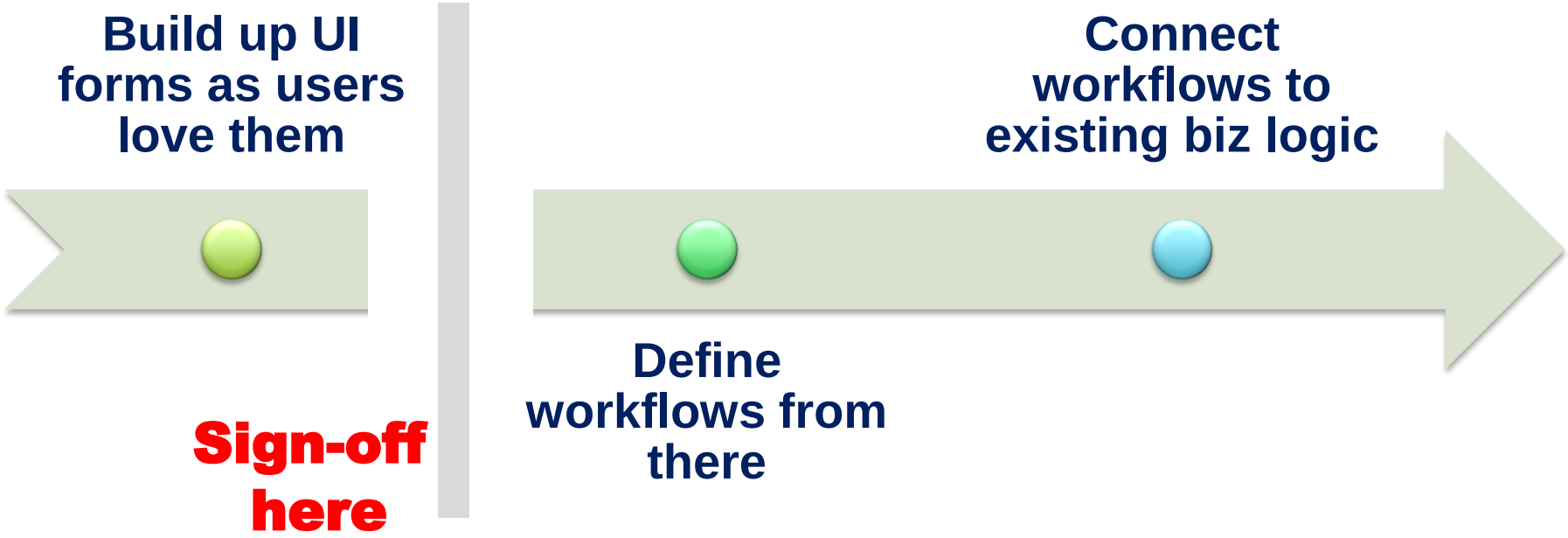
Interviews to collect requirements and
business information to build the domain layer

Interviews to collect usability requirements data
and build the ideal UX for the presentation layer

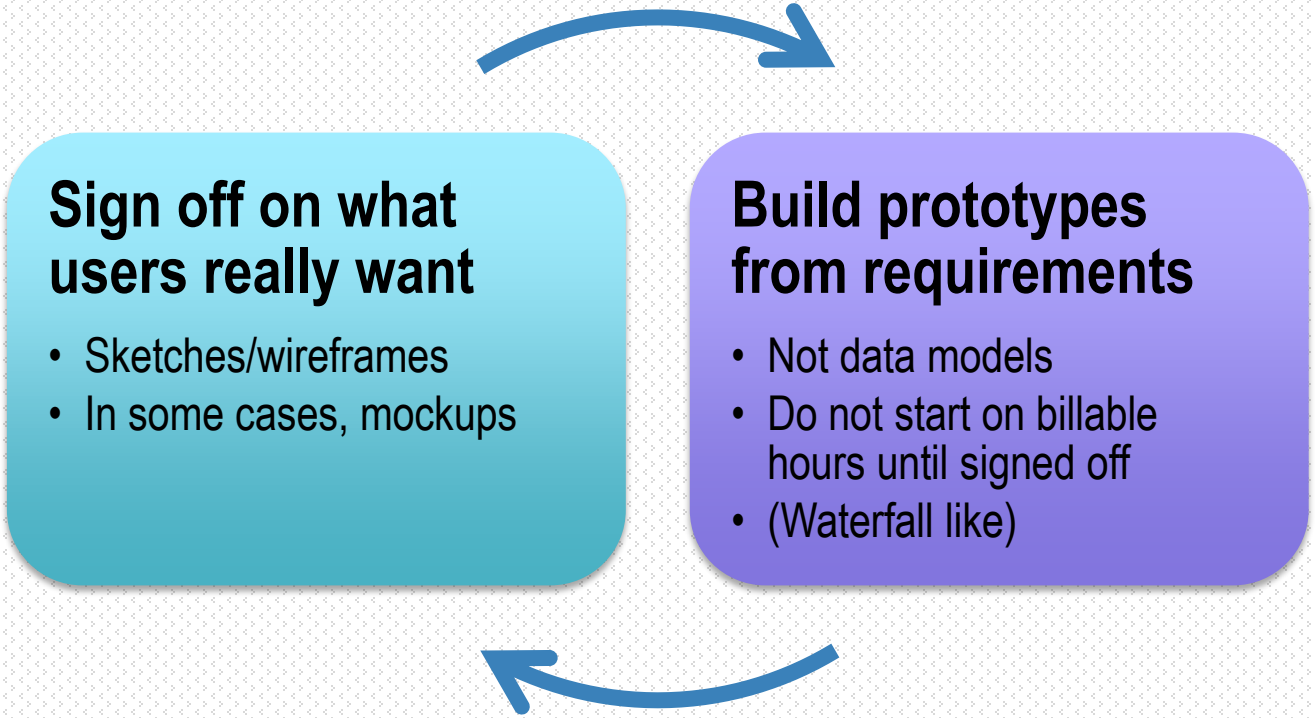
Responsibilities of a UX Expert

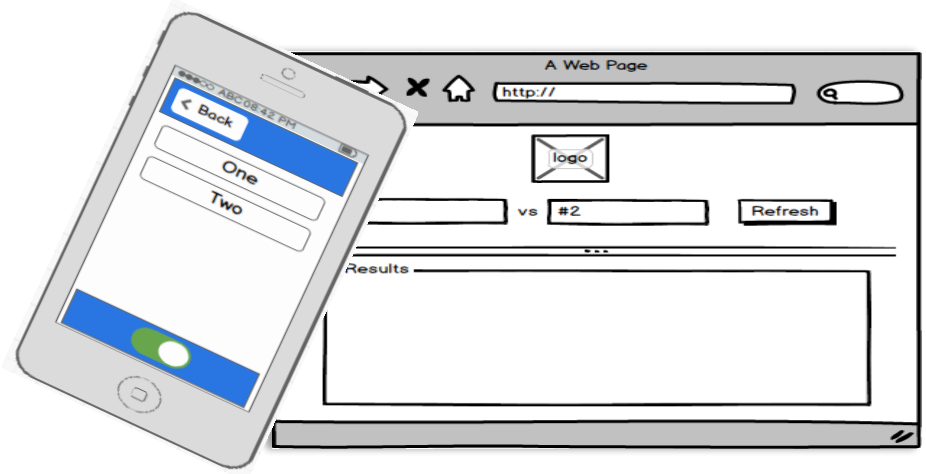
- **Information architecture**
  - UI and UX
  - Interaction and visual design
- **Usability reviews**
  - Observing users live in action (even filming users)
  - Listening to their feedback
- **Catch design/process bottlenecks soon**
  - Focus is data flow, NOT graphics

UX-first in 3 Steps



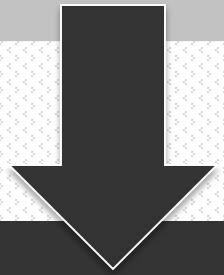
Top-down

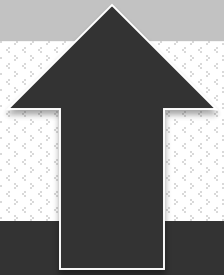

Sign off on what users really want
• Sketches/wireframes
• In some cases, mockups
Build prototypes from requirements
• Not data models
• Do not start on billable hours until signed off
• (Waterfall like)



Top down

Presentation





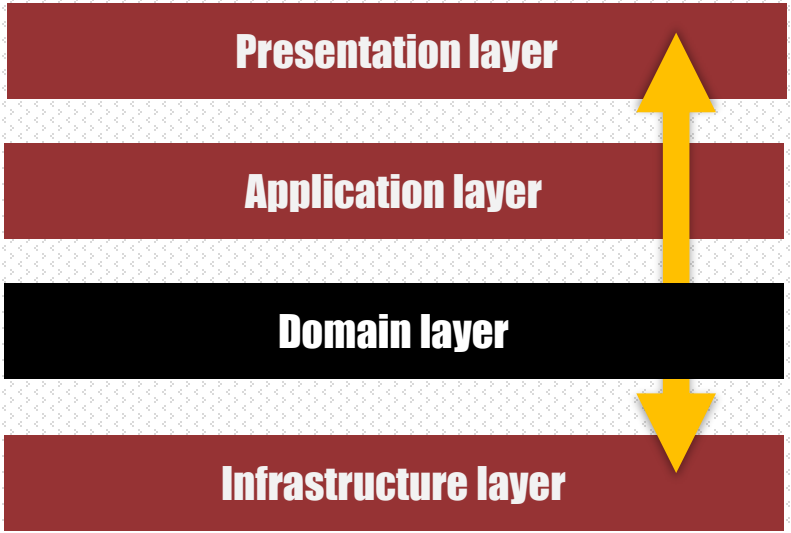
Black box

Tools Do It Better

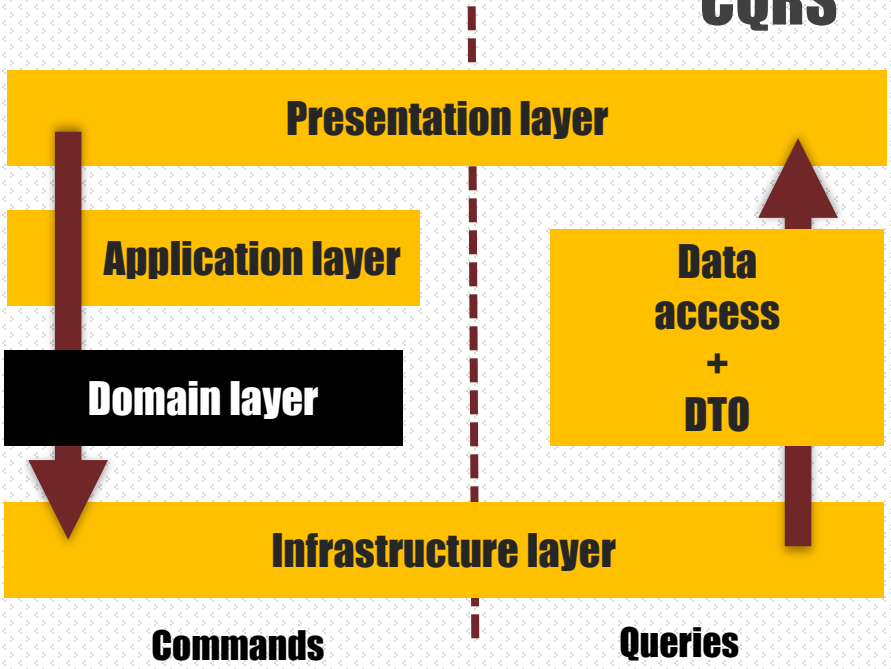
- Renaming
- Adjusting namespaces
- Refactoring

Development

Canonical layered architecture

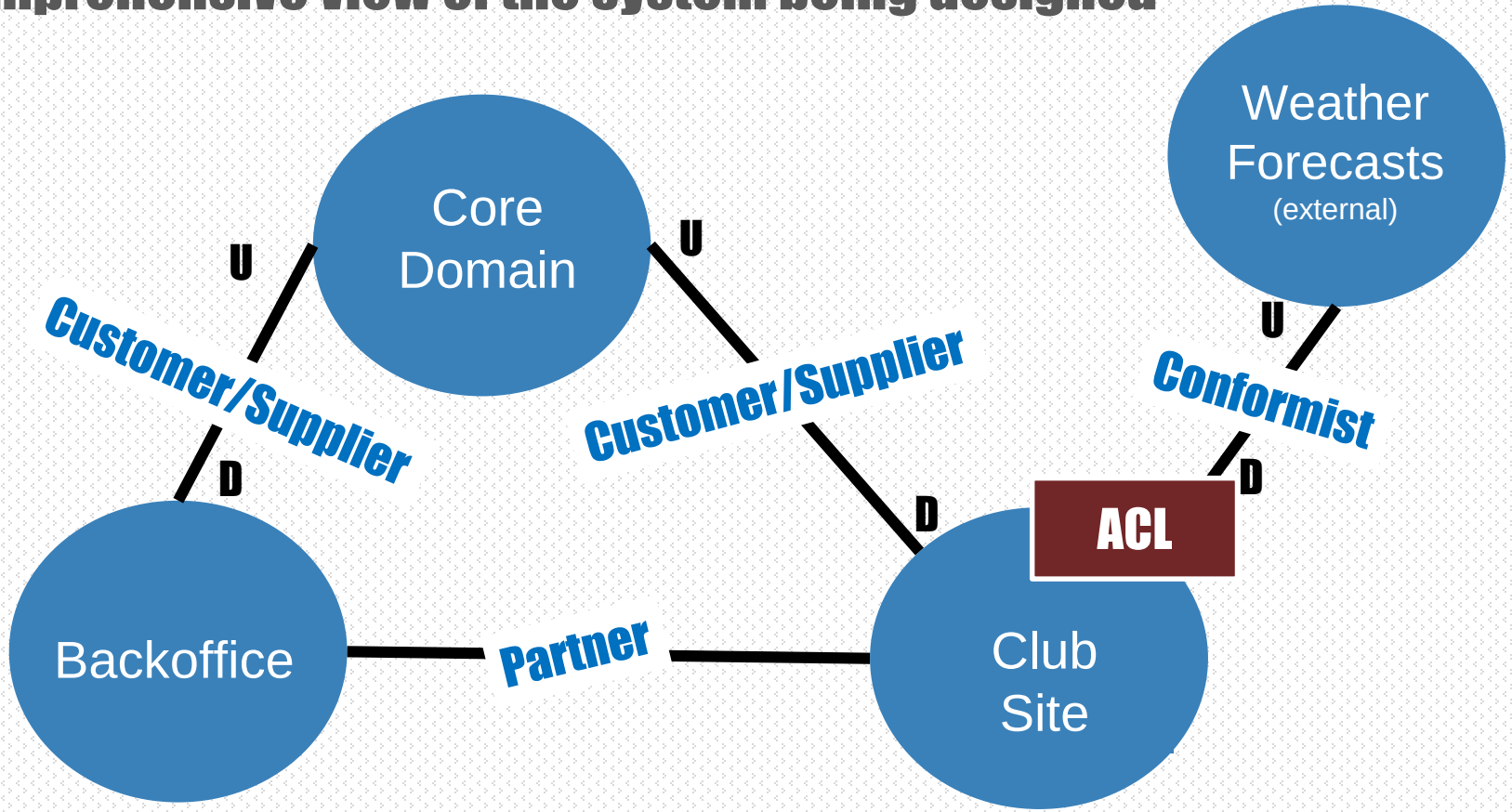


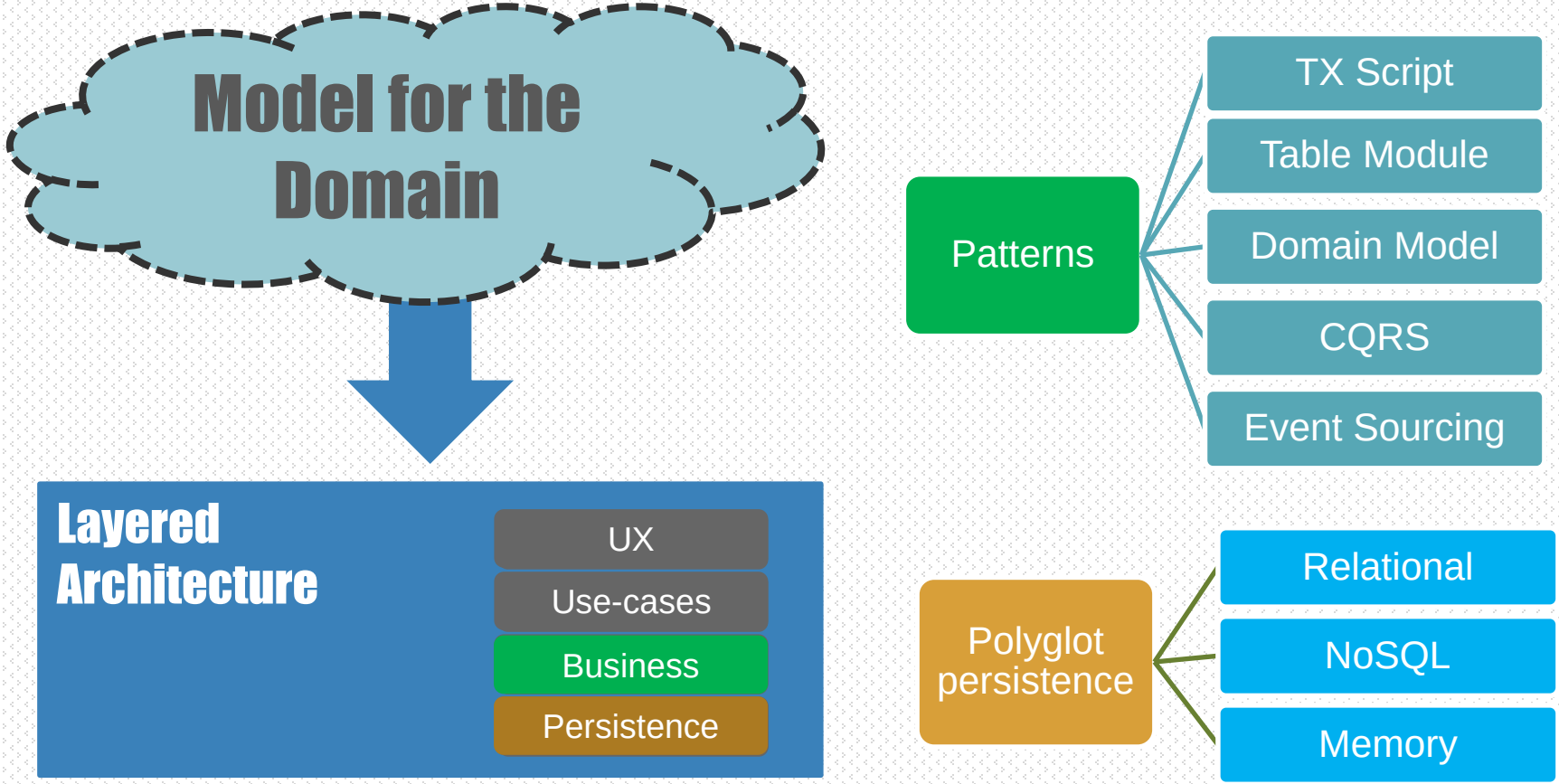
CQRS



DEMO

Context map is the diagram that provides a comprehensive view of the system being designed




Model for the Domain
Patterns
TX Script
Table Module
Domain Model
CQRS
Event Sourcing
Layered Architecture
UX
Use-cases
Business
Persistence
Polyglot persistence
Relational
NoSQL
Memory

Logic invariant to use-cases

- Domain model
- Domain services

Not necessarily
an implementation of the **Domain Model** pattern

Takes care
of persistence tasks

DEMOs

At the end of the day ...

The key lesson today is being aware of **emerging new ways** of doing old things.

Not because you can no longer do the same old things in the same known way, but because newer implementation may let the system evolve in a much smoother way saving you a BBM and some maintenance costs.

{SDD}

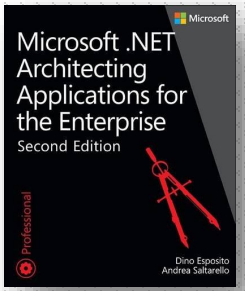
Thank You!

FOLLOW @despos

facebook.com/naa4e

dino.esposito@jetbrains.com

software2cents.wordpress.com



http://naa4e.codeplex.com/ Project MERP