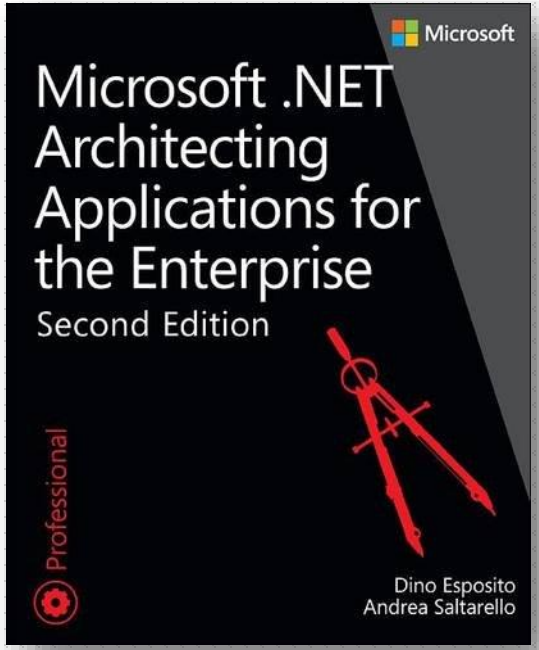
Architecting and Implementing Domain-driven Design Patterns in .NET

Dino Esposito
JetBrains

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



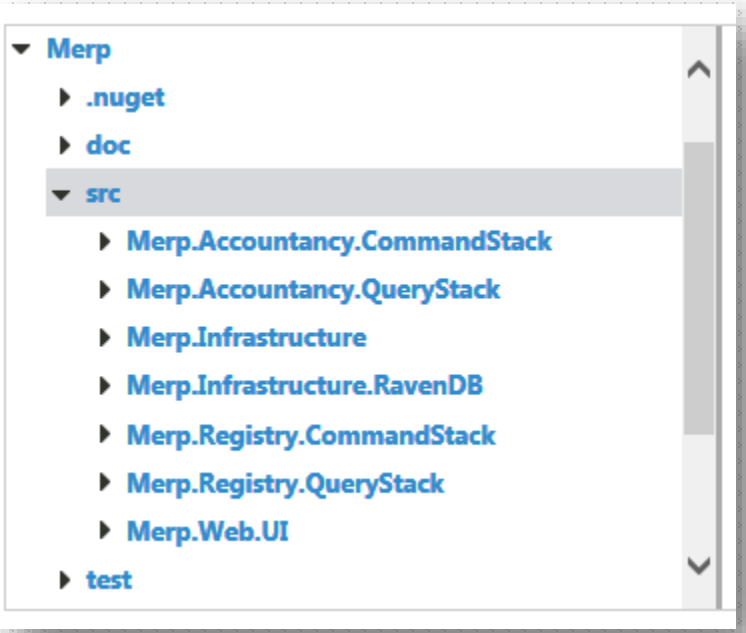
WARNING

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



http://www.laputan.org/pub/foote/mud.pdf

Big Ball of Mud (BBM)

A system that's largely **unstructured**, padded with **hidden dependencies** between parts, with a lot of data and code **duplication** and an **unclear** identification of layers and concerns—a spaghetti code jungle.

Why Is DDD So Intriguing?

Captures known elements of the design process

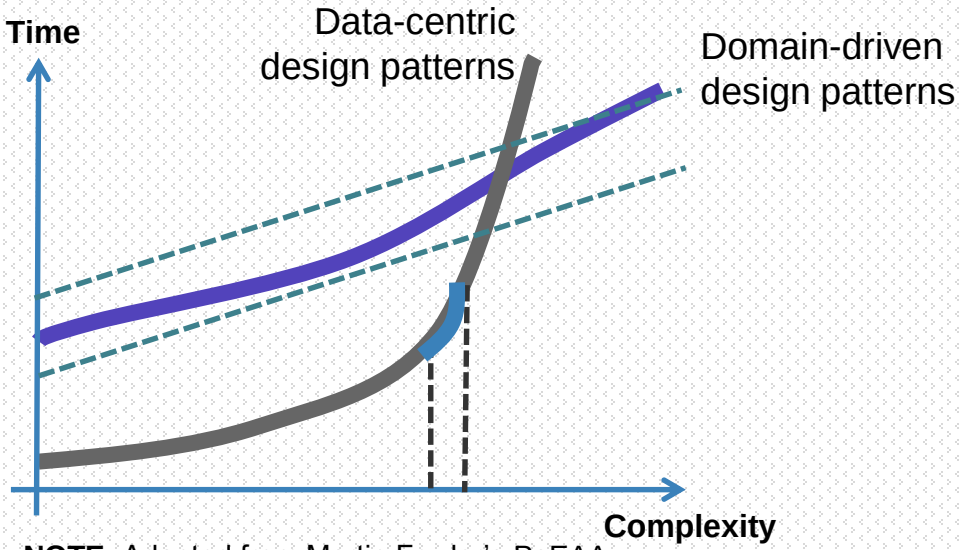
Organizes them into a set of principles

Domain modeling is the focus of development

Different way of building business logic

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.

Supreme Goal

Tackling **Complexity** in the Heart of Software

- Wonderful idea
- Not a mere promise
- Not really hard to do right
- But just easier to do wrong

DDD Is Still About Business Logic

1. Crunch knowledge about the domain
2. Recognize subdomains
3. Design a rich domain model
4. Code by telling objects in the domain model what do to

DDD Key Misconception

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

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

Book a court

New Booking object created

How do you get the ID of the booking?

DESIGN

DOMAIN

PERSISTENCE

DESIGN
DRIVEN
BY THE DOMAIN

Start from User Requirements

Noun

Verb

As a registered customer of the I-Buy-Stuff online store, I can redeem a voucher for an order I place so that I don't actually pay for the ordered items myself.

Registered Customer

Redeem

Voucher

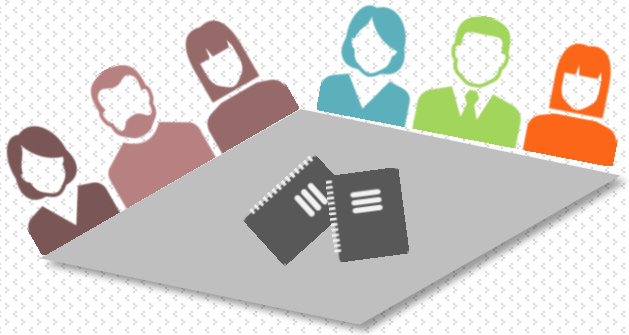
Order

Place

Pay

Ordered Items

- **Official name is voucher**
- **Synonyms like coupon or gift card are not allowed.**

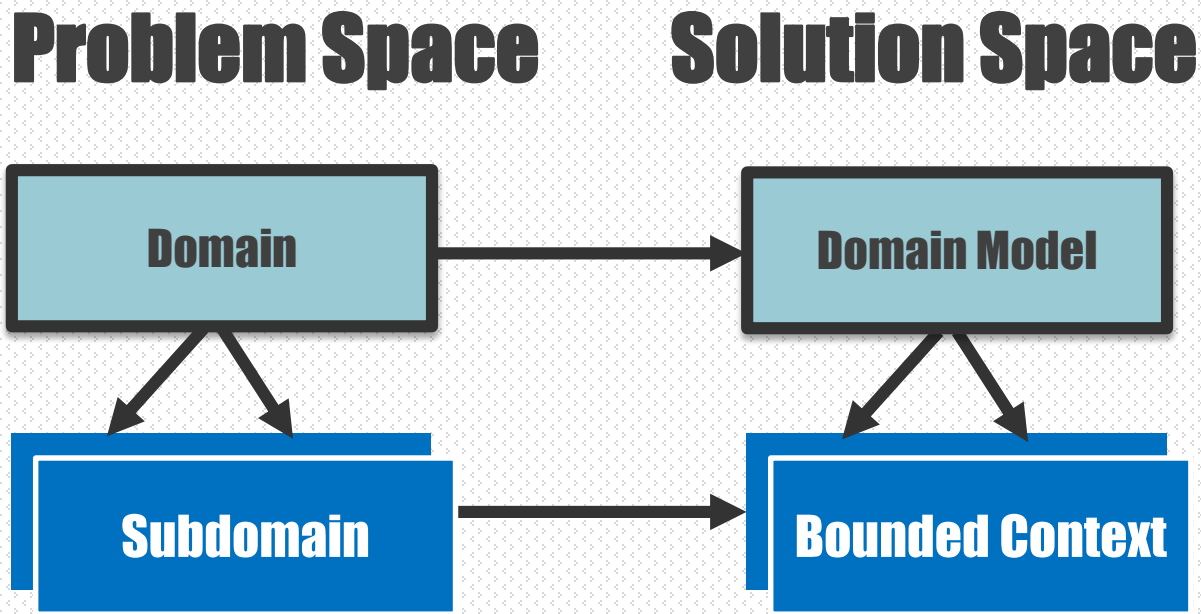


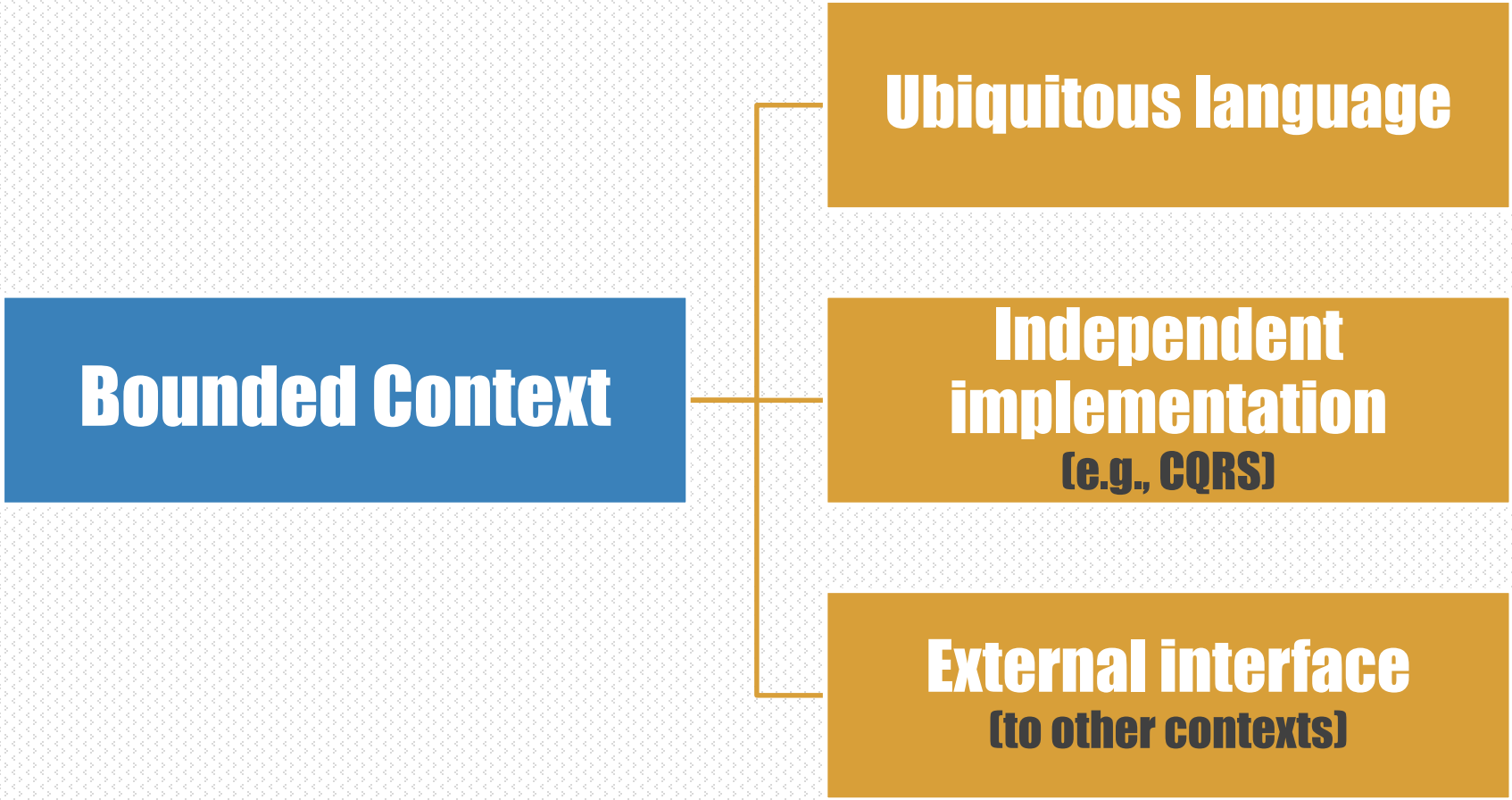
At Work Defining the Ubiquitous Language

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

DEMO

How would you define a model for a sport match?


Problem Space
Solution Space
Domain
Domain Model
Subdomain
Bounded Context


Bounded Context
Ubiquitous language
Independent implementation
(e.g., CQRS)
External interface
(to other contexts)

Key facts for
Domain-driven Design Patterns

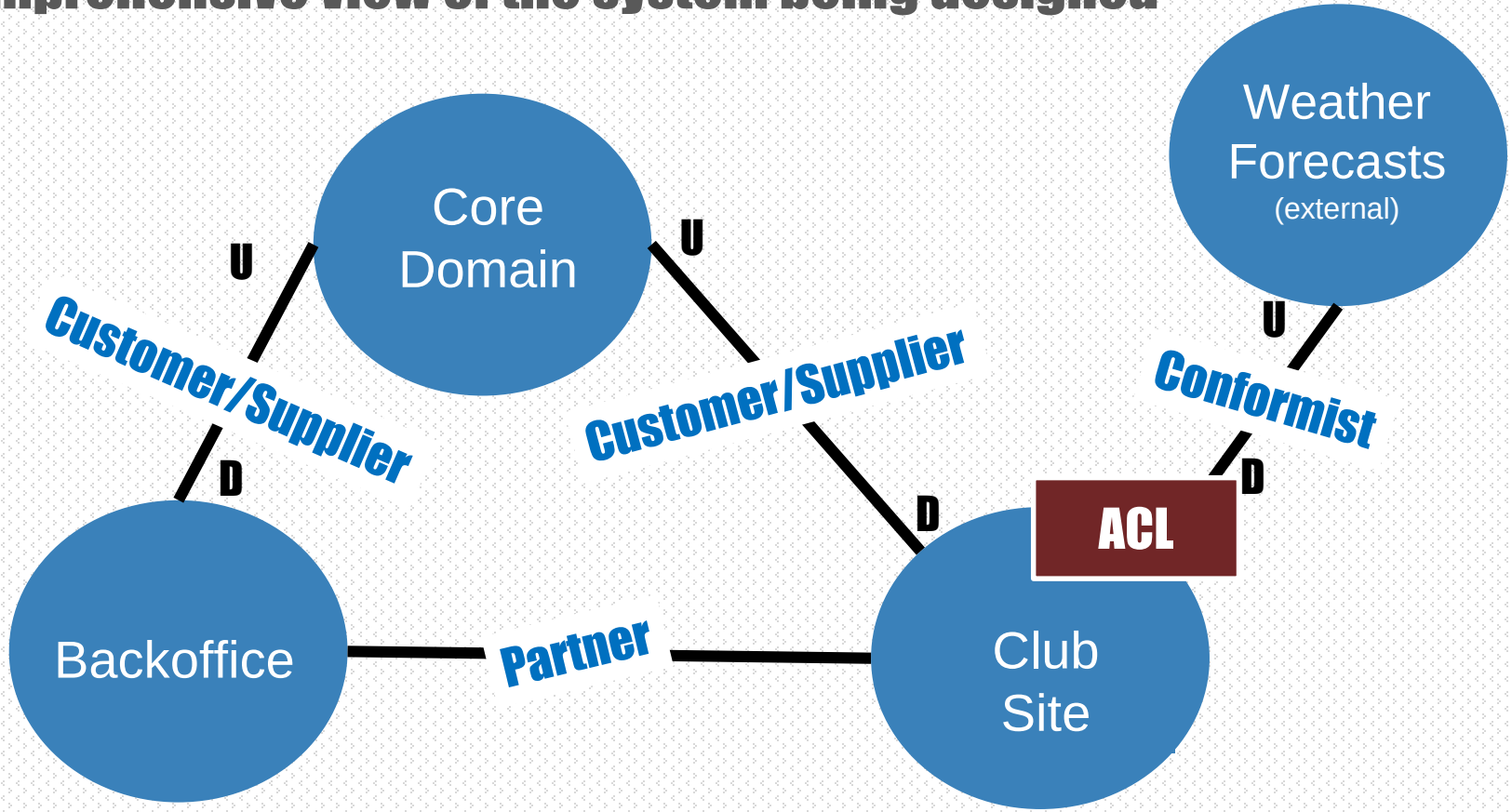
Mirroring
vs.
Modeling

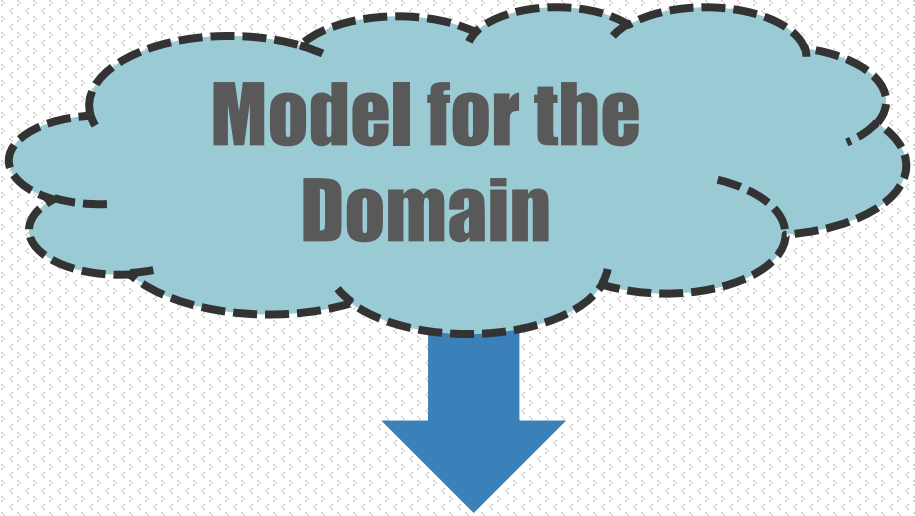
Tasks
vs.
Objects

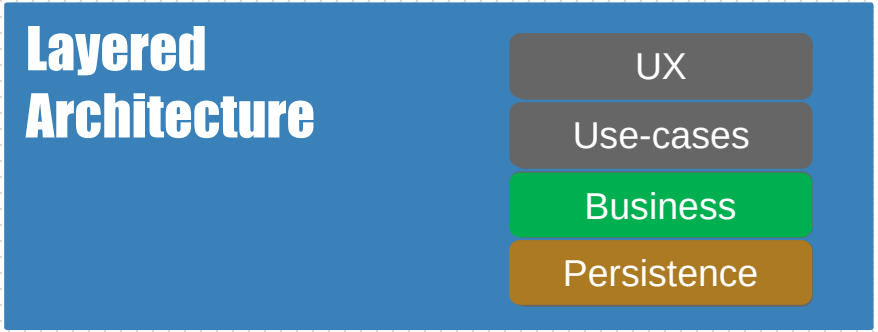
Events
vs.
Models

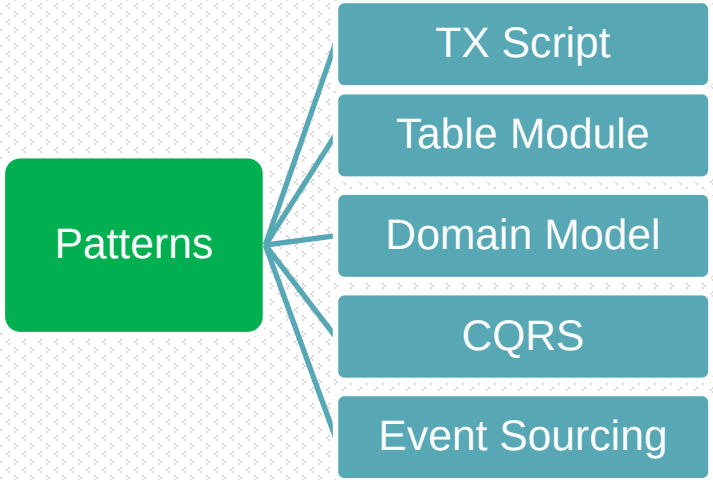
Pragmatism
vs.
Ideology

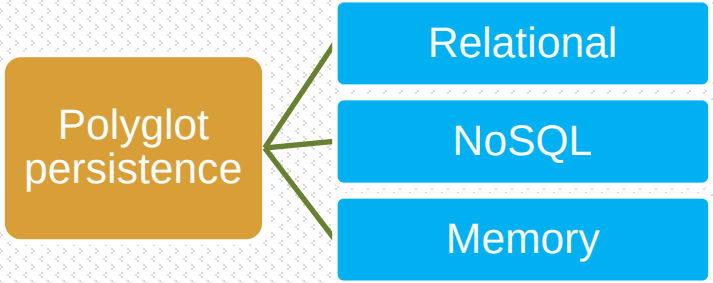
Context map is the diagram that provides a comprehensive view of the system being designed




Model for the Domain


Layered Architecture
UX
Use-cases
Business
Persistence


Patterns
TX Script
Table Module
Domain Model
CQRS
Event Sourcing


Polyglot persistence
Relational
NoSQL
Memory

Business Logic—An Abstract Definition

Application Logic

Dependent on use-cases

- Application entities
- Application workflow components

Domain Logic

Invariant to use-cases

- Business entities
- Business workflow components

Business Logic—DDD Definition

Application Logic

Dependent on use-cases

- Data transfer objects
- Application services

Domain Logic

Invariant to use-cases

- Domain model
- Domain services

Transaction Script Pattern

System actions

- Each procedure handles a single task

Logical transaction

- end-to-end from presentation to data

Common subtasks

- split into bounded sub-procedures for reuse

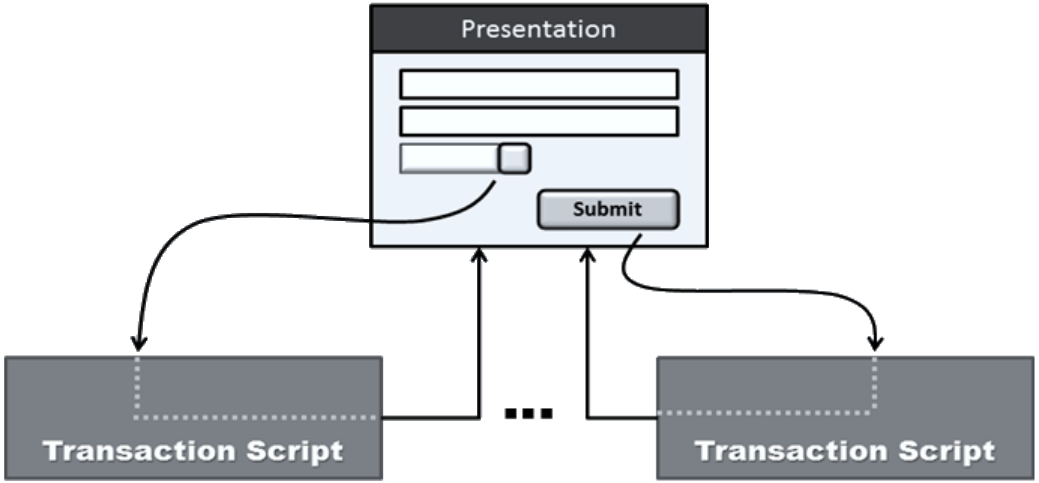


Table Module Pattern

One module per table in the database

Module contains all methods that will process the data

- Both queries and commands

May limit modules to “significant” tables

- Tables with only outbound foreign-key relationships

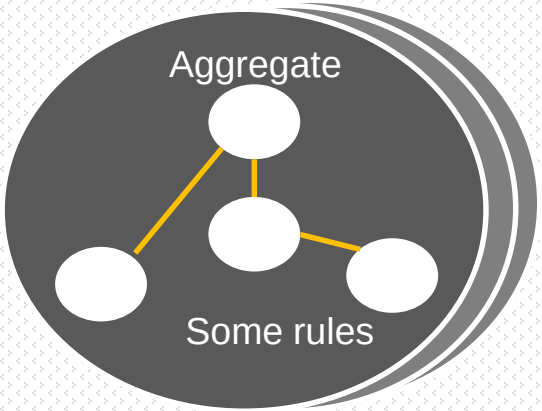
Domain Model Pattern

Aggregated objects

- Data and behavior

Persistence agnostic

Paired with domain services



Workflows & rules

Domain Layer

Logic invariant to use-cases

- Domain model
- Domain services

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

Takes care
of persistence tasks

Domain Model

Models for the business domain	**Object-oriented entity model**
	Functional model
Guidelines for classes in an entity model	**DDD conventions** (factories, value types, private setters)
	Data and behavior
Anemic model	**Plain data containers**
	Behavior and rules **moved** to domain services

Domain Services

- Pieces of domain logic that don't fit into any of the existing entities
- Classes that group logically related behaviors
  - Typically operating on multiple domain entities
- Implementation of processes that
  - Require access to the persistence layer for reads and writes
  - Require access to external services

DEMO

Aggregates

- **Ensure business consistency**
  - Transactional consistency only, within the domain
- **Work with fewer and coarse-grained objects**
  - Aggregate root encapsulates child entities
  - Fewer entity-to-entity relationships to care about

"An aggregate is a cluster of associated objects that we treat as a single unit for the purpose of data changes." –**E. Evans**

Domain Events

- Something noteworthy within the domain
  - Simplest is notification of CRUD operations
- Relevant as it helps scheduling complex operations in a more natural way
  - Requires ad hoc infrastructure in domain model classes

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.

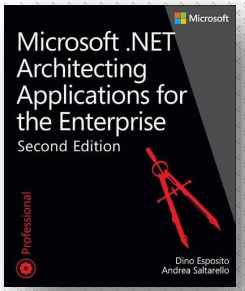
Thank You!

FOLLOW @despos

facebook.com/naa4e

dino.esposito@jetbrains.com

software2cents.wordpress.com



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