Refactoring from Anemic Domain Model Towards a Rich One

INTRODUCTION

Vladimir Khorikov

@vkhorikov www.enterprisecraftsmanship.com


push state change with
DOMAIN EVENTS
access with
REPOSITORIES
access with
SERVICES
express model with
express change with
ENTITIES
encapsulate with
act as root of
express identity with
encapsulate with
AGGREGATES
express state &
computation with
MODEL-DRIVEN
DESIGN
VALUE
OBJECTS
encapsulate with
model gives structure to
isolate domain
expressions with
encapsulate with
define model within
LAYERED
ARCHITECTURE
encapsulate with
FACTORIES
UBIQUITOUS LANGUAGE
cultivate rich
model with
keep model
unified by
CONTINUOUS
INTEGRATION
names
enter
SHARED
KERNEL
SHARED
KERNEL
BOUNDED CONTEXT
interdependent
contexts form
overlap allied
contexts through
CORE DOMAIN
relate allied contexts as
CUSTOMER
/SUPPLIER
work in
autonomous, clean
assess/overview
relationships with
CONTEXT MAP
minimize translation
CONFORMIST
avoid overinvesting in
support multiple
clients through
segregate the
conceptual messes
free teams
to go
OPEN HOST
SERVICE
GENERIC
SUBDOMAINS
translate and insulate
unilaterally with
loosely couple
contexts
through
formalize
BIG BALL OF MUD
ANTI-
CORRUPTION
LAYER
SEPARATE
WAYS
PUBLISHED
LANGUAGE

Overview

Introduction

Introducing an anemic domain model

Decoupling the domain model from data contracts

Using value objects as domain model building blocks

Pushing logic down from services to domain classes

Organizing the application services layer

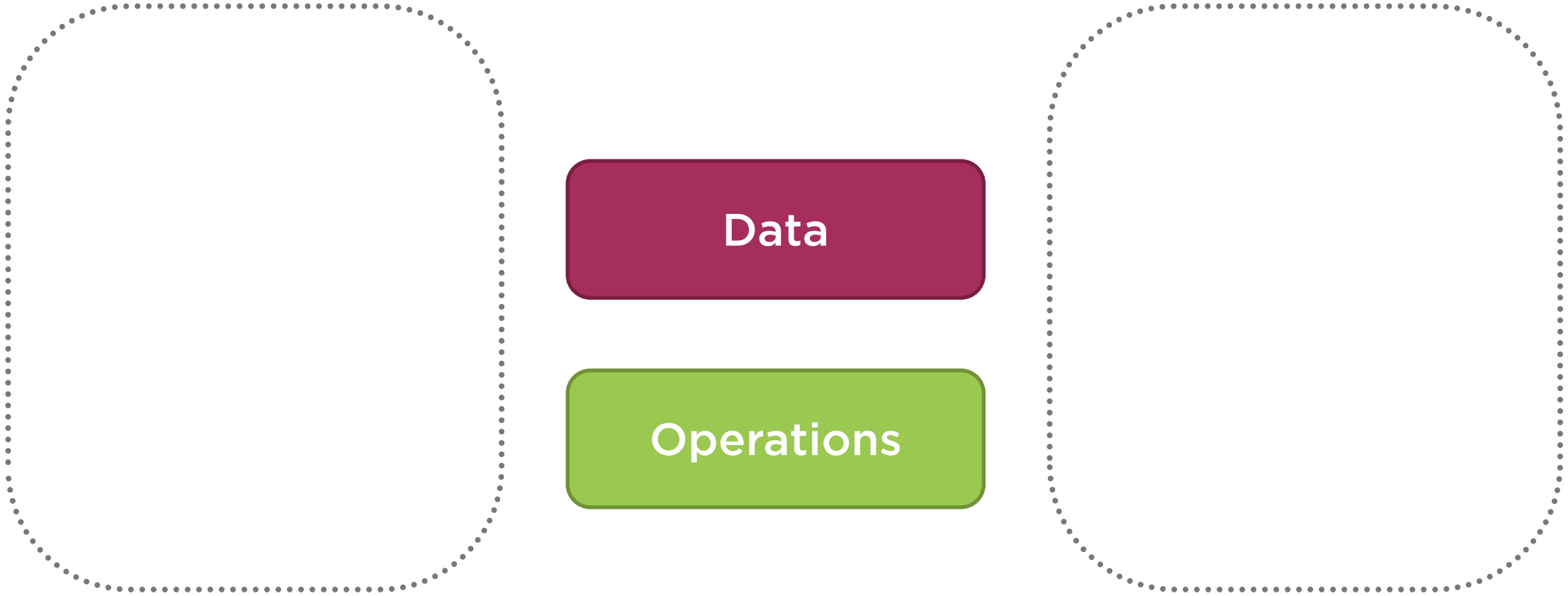
Domain modeling best practices

Prerequisites

“Domain-Driven Design Fundamentals” by Julie Lerman and Steve Smith

“Domain-Driven Design in Practice” by Vladimir Khorikov

Anemic Domain Model



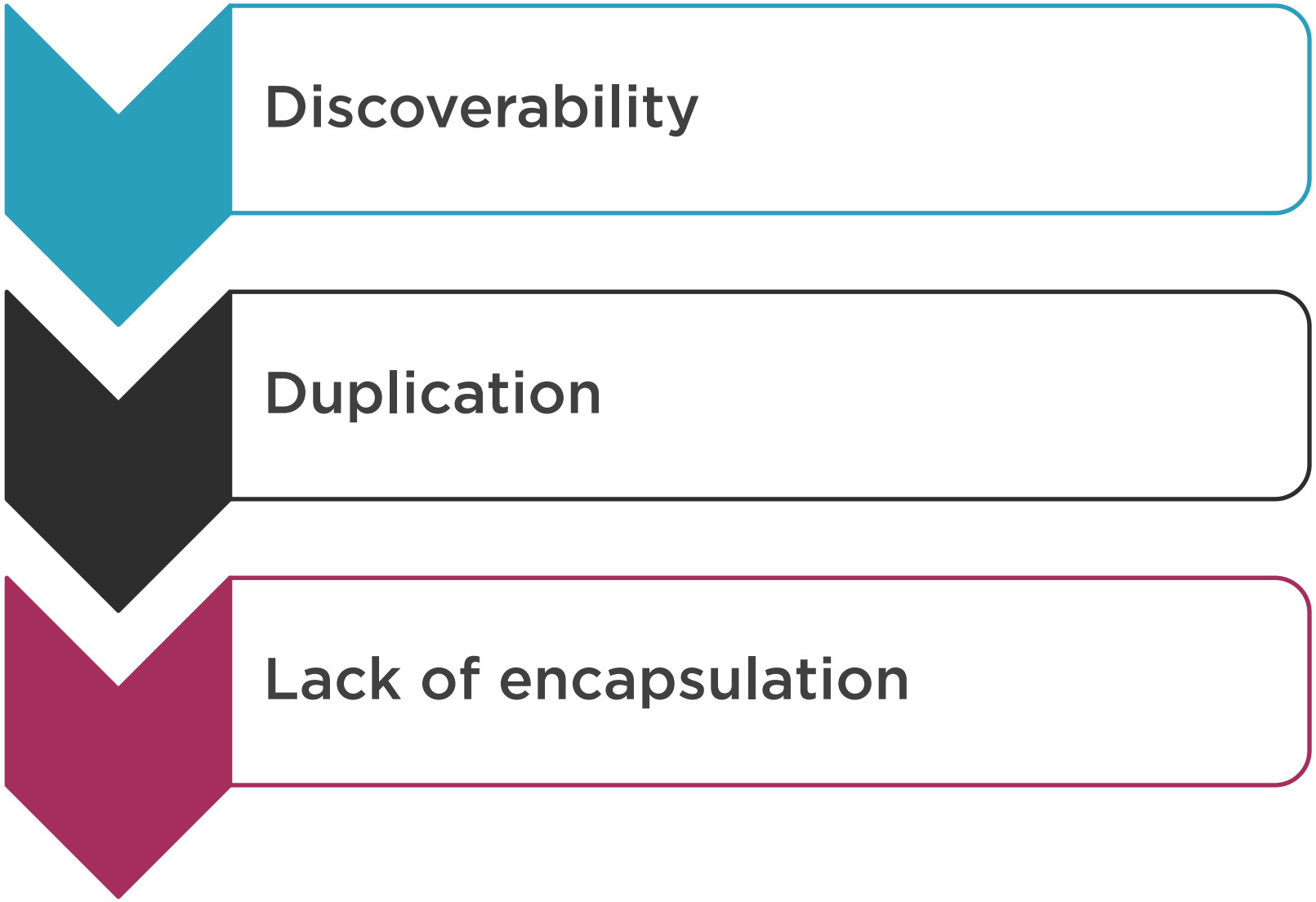
Public getters/setters

Stateless services

Anemic Domain Model

Doesn’t comply with ideas of object-oriented design

Anemic Domain Model



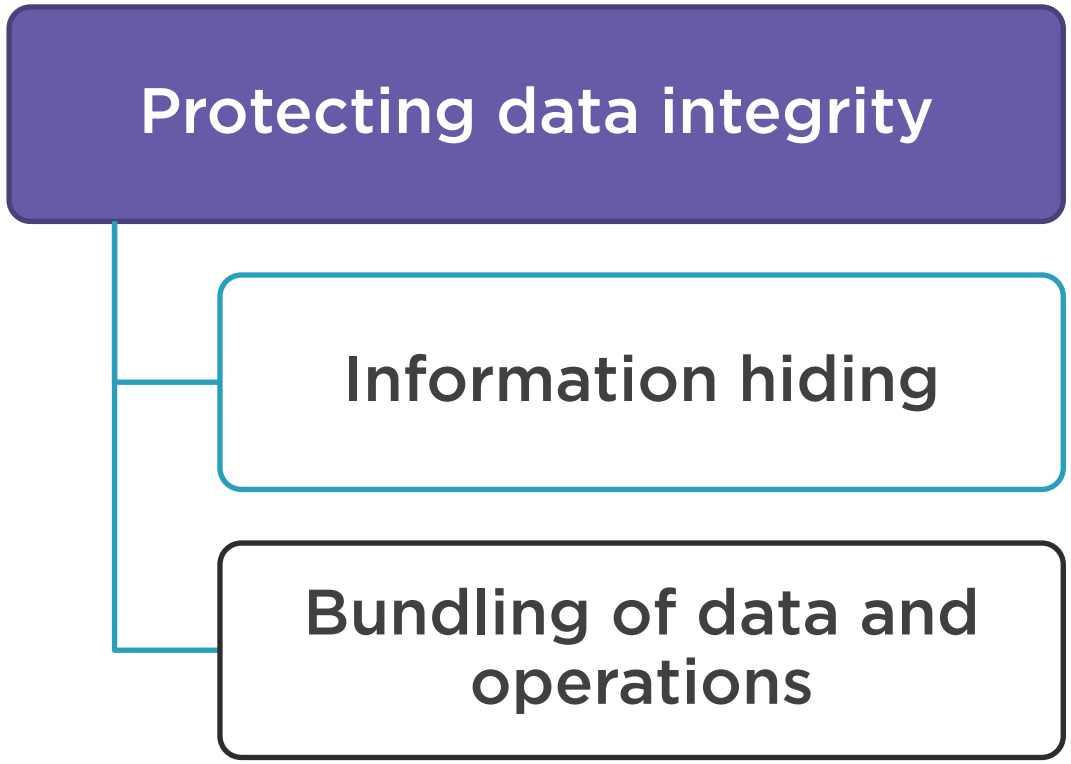
Encapsulation

Information hiding 

Bundling data and operations together 

Encapsulation is an act of protecting the data integrity.

Encapsulation



Encapsulation

Encapsulation

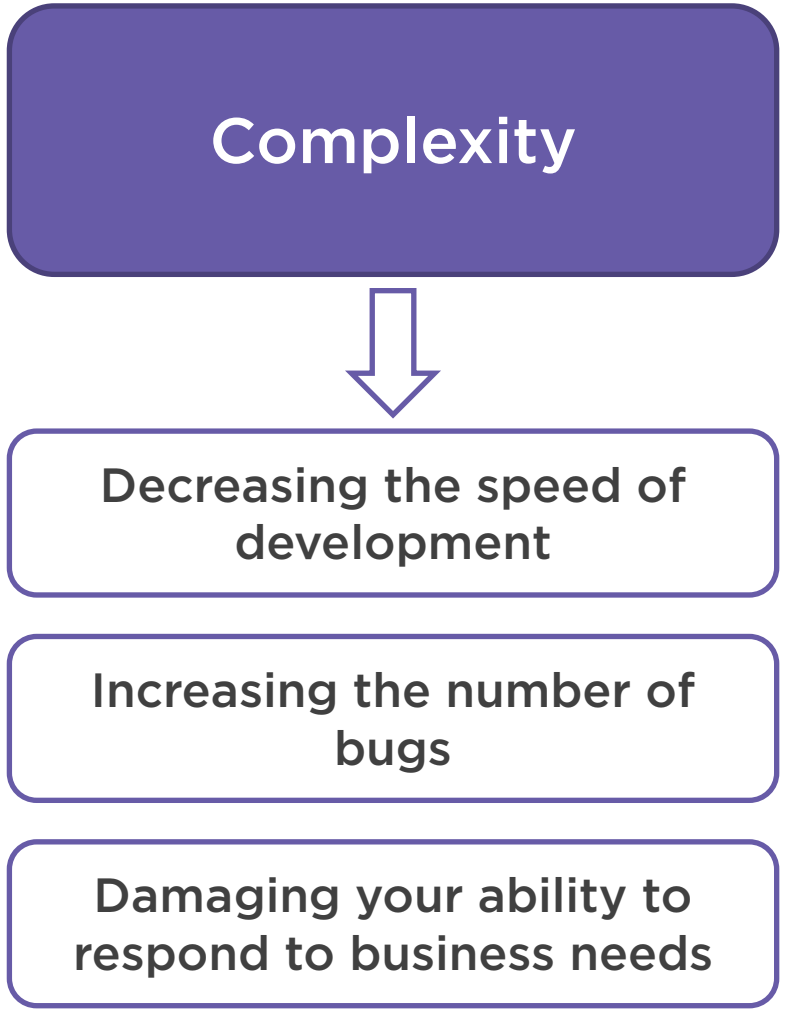
Square

- 4 sides
- Sides' lengths are equal
- Angles are of 90 degrees

No operation should be able to violate the invariants

Why encapsulation is so important?

Encapsulation

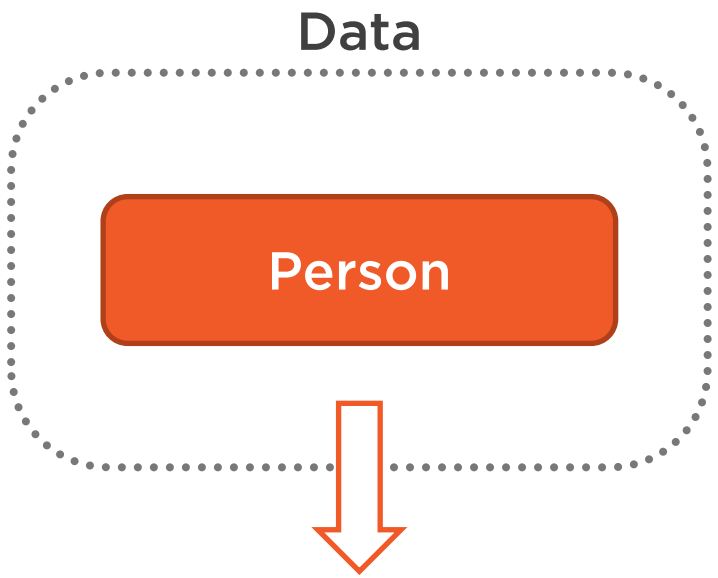


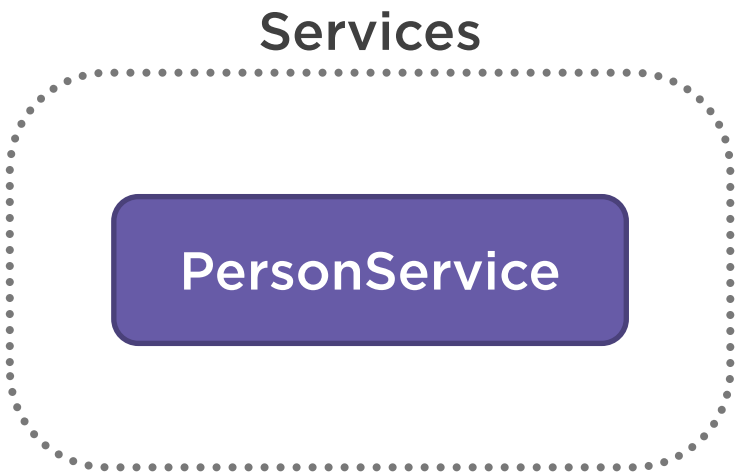
You cannot trust yourself to do the right thing all the time.

Anemic Domain Model and Encapsulation

Does an anemic domain model automatically entail lack of encapsulation?

Anemic Domain Model and Encapsulation






```
public class Person
{
    public string Name { get; set; }
    public string Address { get; set; }
    public string Email { get; set; }
}
```

- No restrictions
- Potential duplication

Attribute the task of maintaining data integrity to the classes containing that data.

Advantages of Anemic Domain Model

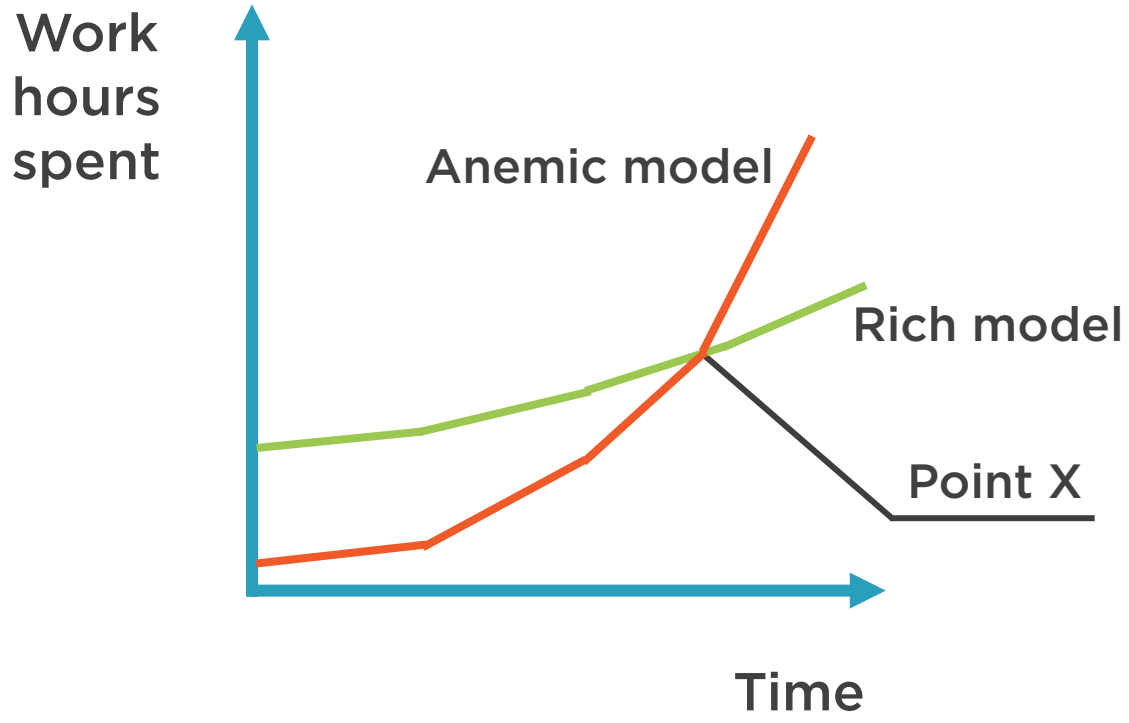
Intuitive

Developers pick it up naturally

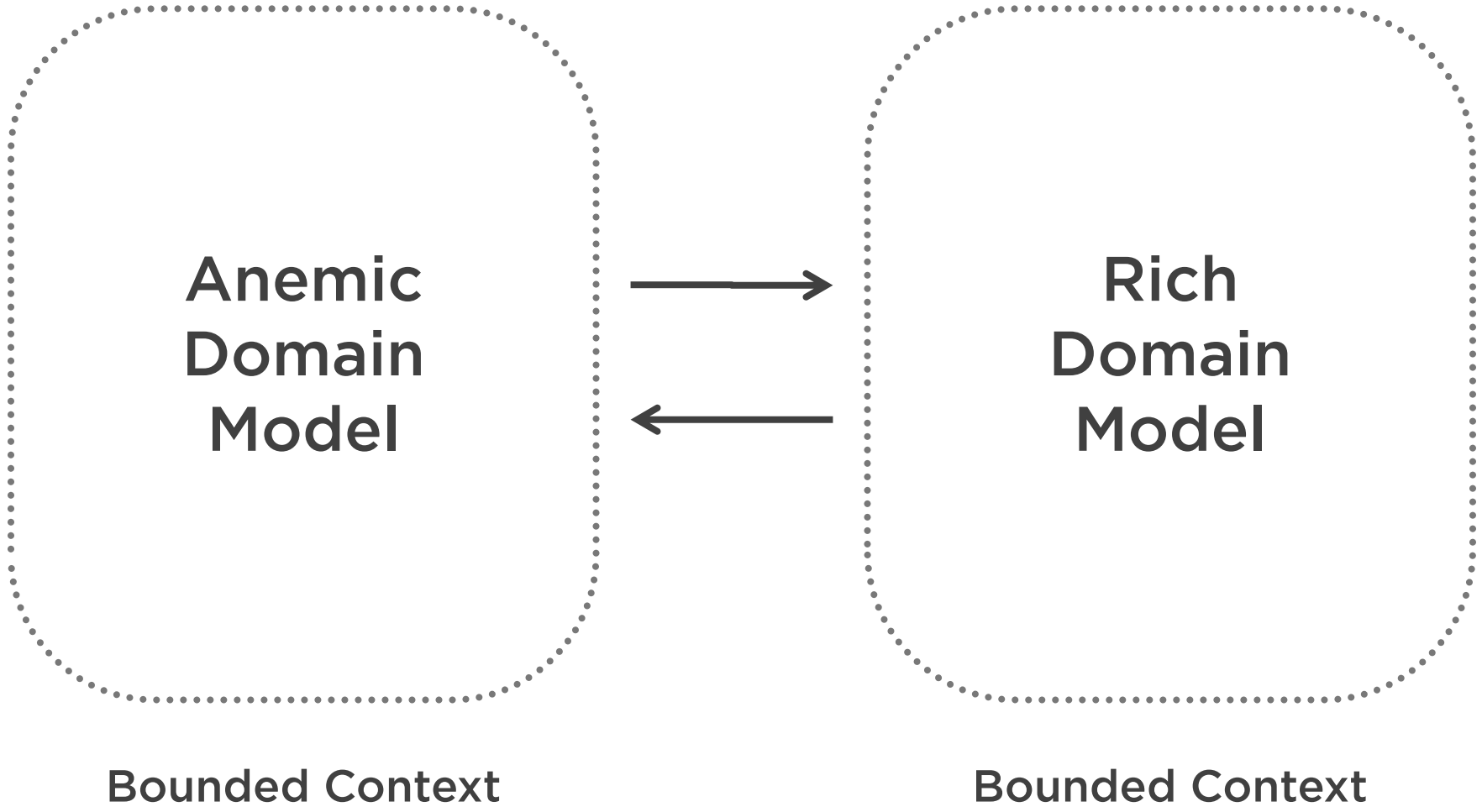
Easy to implement

Fast development, at least at the beginning

Anemic Domain Model Applicability



Anemic Domain Model Applicability



Anemic Domain Model and Functional Programming

Immutable data structures

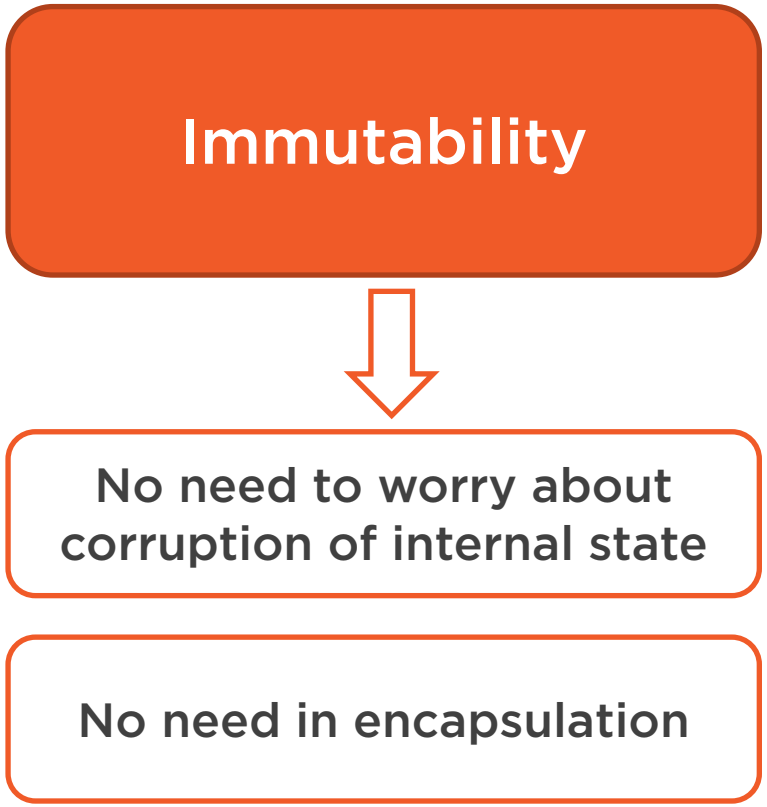
Operations upon data are kept separately

Anemic Domain Model and Functional Programming

```
public class Square {
    public readonly double SideLength;

    public Square(double sideLength) {
        if (sideLength <= 0)
            throw new Exception("Invalid square side length: " + sideLength);

        SideLength = sideLength;
    }
}

public class Services {
    public static double CalculateArea(Square square) {
        return square.SideLength * square.SideLength;
    }
}
```

Do we have a problem here?

Anemic Domain Model and Functional Programming



"Object-oriented programming makes code understandable by encapsulating moving parts. Functional programming makes code understandable by minimizing moving parts."

Michael Feathers

Anemic Domain Model and Functional Programming

```
public class Square {
    public readonly double SideLength;

    public Square(double sideLength) {
        if (sideLength <= 0)
            throw new Exception("Invalid square side length: " + sideLength);

        SideLength = sideLength;
    }
}

public class Services {
    public static double CalculateArea(Square square) {
        return square.SideLength * square.SideLength;
    }
}
```

Anemic Domain Model and Functional Programming

How about the discoverability?

Can be mitigated by using conventions

Applying Functional Principles in C#

by Vladimir Khorikov

Functional programming in C# can give you insight into how your programs will behave. You'll learn the fundamental principles that lie at the foundation of functional programming, why they're important, and how to apply them.

▶ Resume Course

Table of contents | Description | Transcript | Exercise files | Discussion | Recommended

Ex

Summary

Anemic domain model: keeping data and operations separately from each other

- Classes with publicly accessible properties
- Stateless services

Problems with the anemic domain model:

- Discoverability of operations
- Potential duplication
- Encapsulation

Encapsulation is an act of protecting the data integrity

- Without encapsulation, it's hard to maintain code correctness

Anemic domain models always entail the lack of encapsulation

- The only exception is immutable classes

Summary

Anemic domain model is applicable when:

- Your project is simple, or
- Will not be developed for a long period of time

In all other cases, consider implementing a rich domain model

In the Next Module

Introducing an Anemic Domain Model