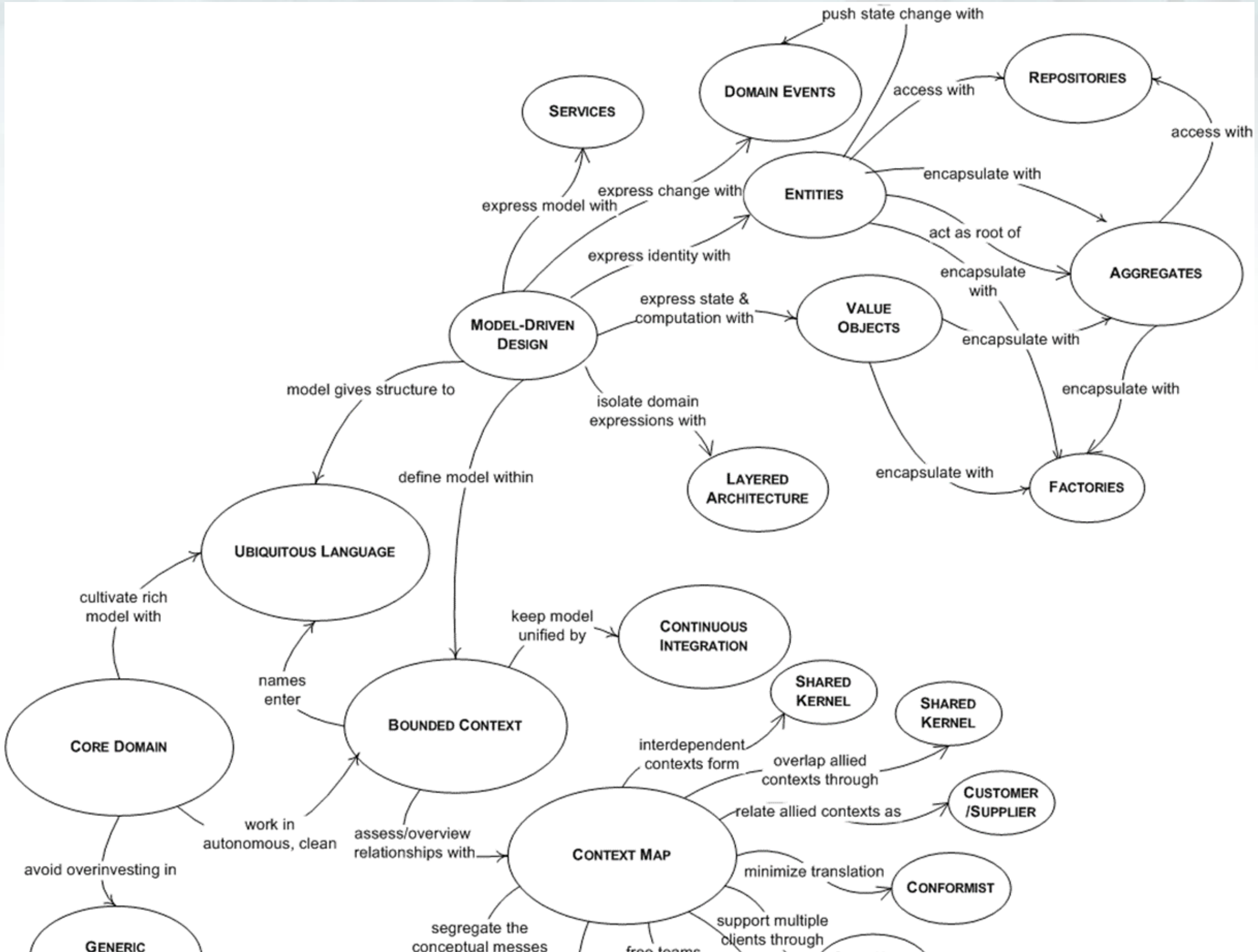
CON 2014

Domain-Driven Design
with ASP.NET MVC

Steve Smith
CTO, Falafel Software
@ardalis | steve.smith@falafel.com

What is Domain-Driven Design (DDD)?

C N 2014


push state change with
Domain Events
Services
access with
Repositories
access with
express change with
Entities
encapsulate with
express model with
act as root of
express identity with
Aggregates
encapsulate with
express state & computation with
Value Objects
Model-Driven Design
encapsulate with
encapsulate with
model gives structure to
isolate domain expressions with
define model within
encapsulate with
Layered Architecture
Factories
Ubiquitous Language
cultivate rich model with
keep model unified by
Continuous Integration
names enter
Shared Kernel
Shared Kernel
Core Domain
Bounded Context
interdependent contexts form
overlap allied contexts through
relate allied contexts as
Customer /Supplier
work in autonomous, clean
assess/overview relationships with
Context Map
minimize translation
Conformist
avoid overinvesting in
segregate the conceptual messes
free teams
support multiple clients through
Generic

Communication

- Ubiquitous Language
- Domain Expert Interaction

Modeling

- Core Domain
- Generic Subdomains
- Bounded Context
- Context Map
- Shared Kernel
- Anti-Corruption Layer

Implementation

- Model-Driven Design
- Layered Architecture
- Entities
- Value Objects
- Services
- Factories
- Aggregates
- Repositories
- Domain Events

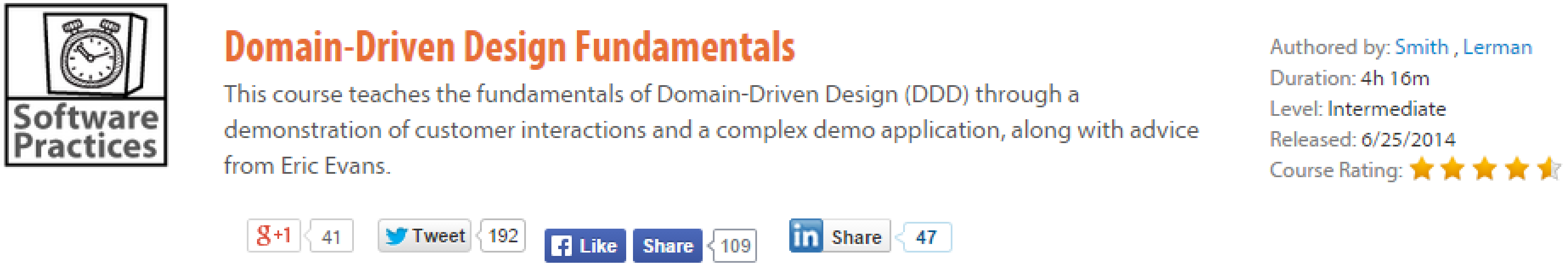
DDD is BIG

“The more you know, the more you realize you know nothing.”

Socrates

DDD Fundamentals Course

- Over 4 hours of content (demos using MVC + SignalR)
- http://bit.ly/PS-DDD



Software Practices

Domain-Driven Design Fundamentals

This course teaches the fundamentals of Domain-Driven Design (DDD) through a demonstration of customer interactions and a complex demo application, along with advice from Eric Evans.

Authored by: Smith , Lerman
Duration: 4h 16m
Level: Intermediate
Released: 6/25/2014
Course Rating: ★★★★☆

DDD Benefits

- Flexibility
- Software models customer's understanding of problem
- Breaks complexity into manageable pieces
- Well-organized; easily tested
- Business logic lives in one place

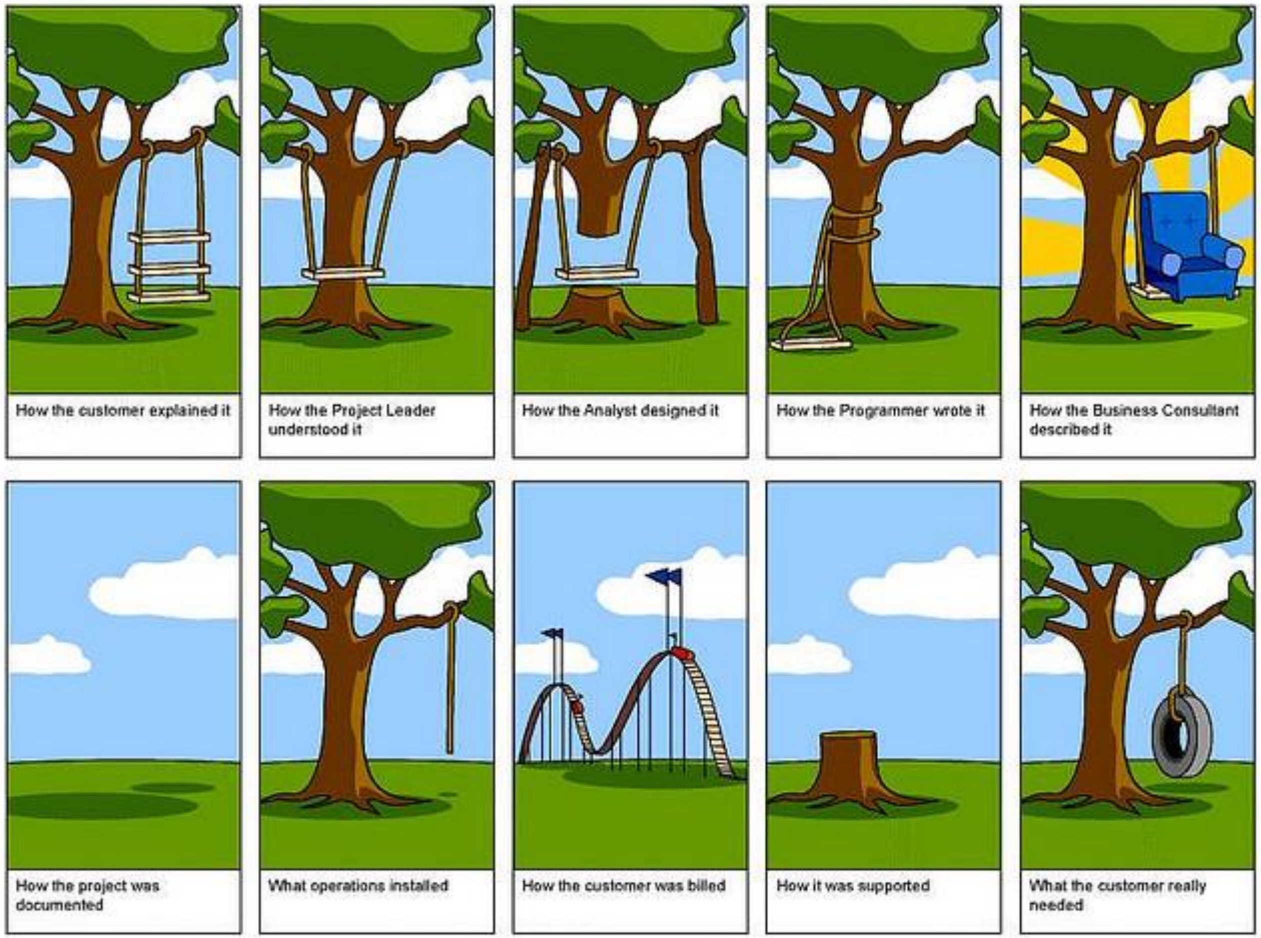
DDD Drawbacks

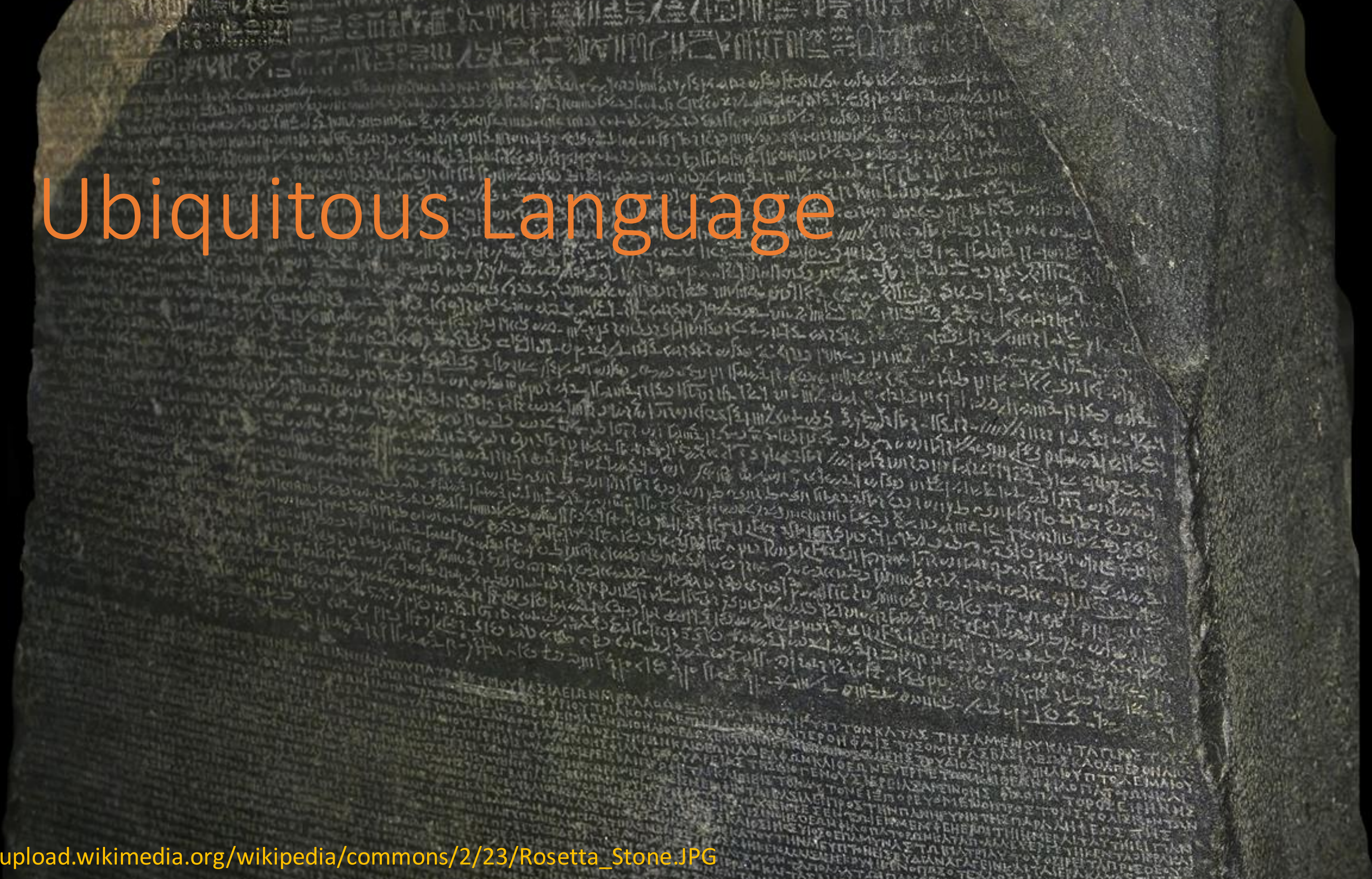
- Time and Effort
- Learning Curve
- Overkill without Complexity
  - "Anemic" domain model problem

Communication

"As software developers, we fail in two ways: we build the thing wrong, or we build the wrong thing."

Me


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed



Ubiquitous Language

http://upload.wikimedia.org/wikipedia/commons/2/23/Rosetta_Stone.JPG

Language

“A project faces serious problems when its language is fractured.”

Eric Evans

Ubiquitous Language

- **Ubiquitous** – *adjective*. Present, appearing, or found everywhere.
    - Synonyms: pervasive, universal
- Used within a given Bounded Context
- Used in code, design documents, and conversations

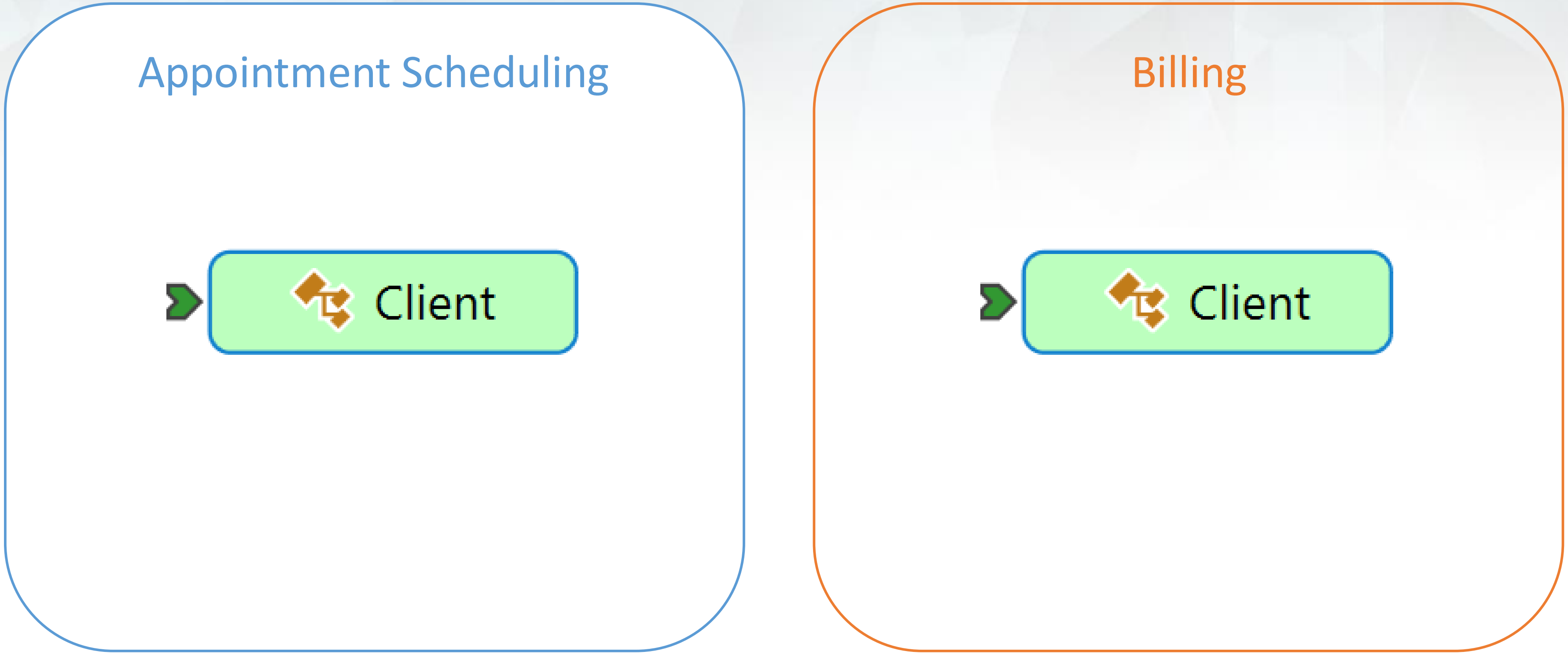
  -- *Everywhere*

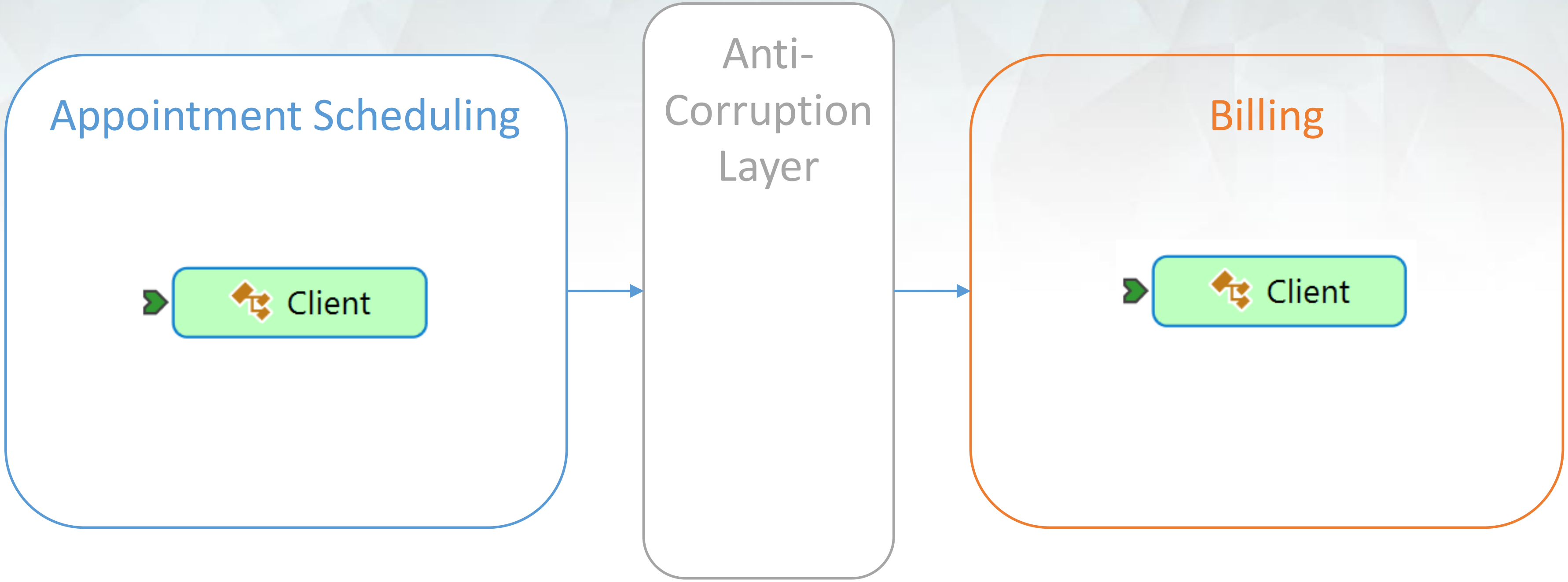
Domain Terms

- Domain Experts
- Problem Domain
- Core Domain
- Sub-Domains

Bounded Contexts

- Provide Separation of Concerns
- Limit complexity
- Should be clearly bounded and separate


Appointment Scheduling
Client
Billing
Client

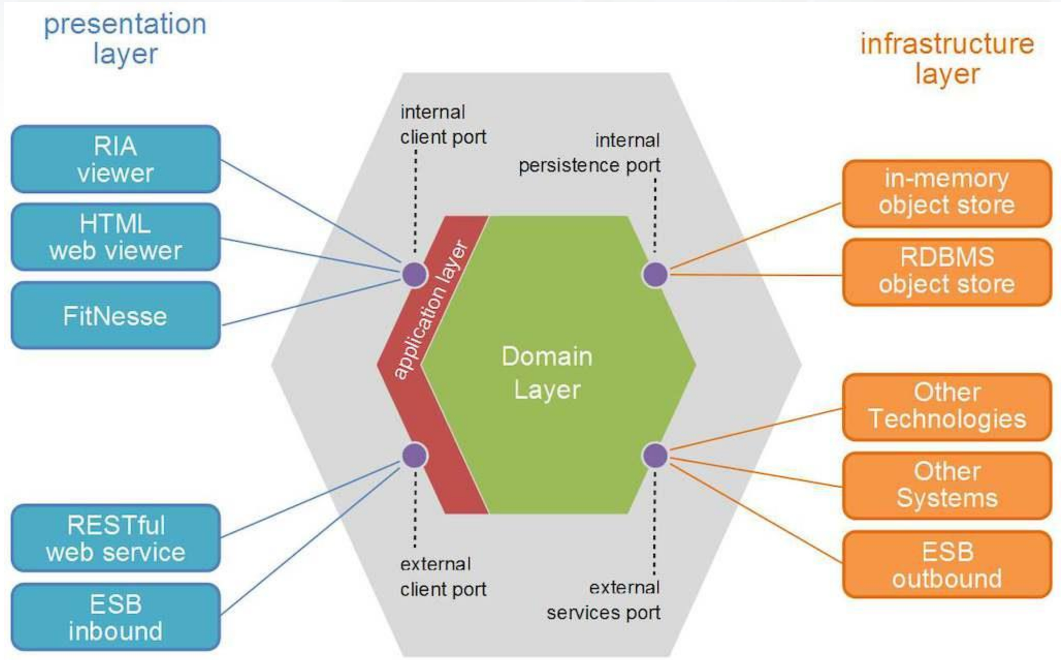

Appointment Scheduling
Client
Anti-Corruption Layer
Billing
Client

Model Driven Design

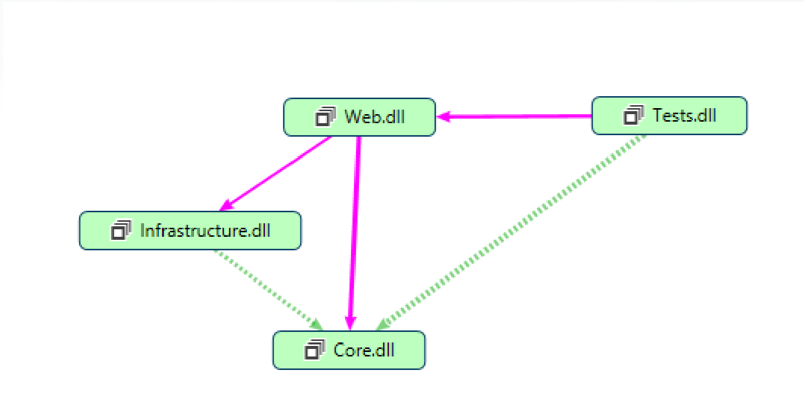
Not Data-Driven

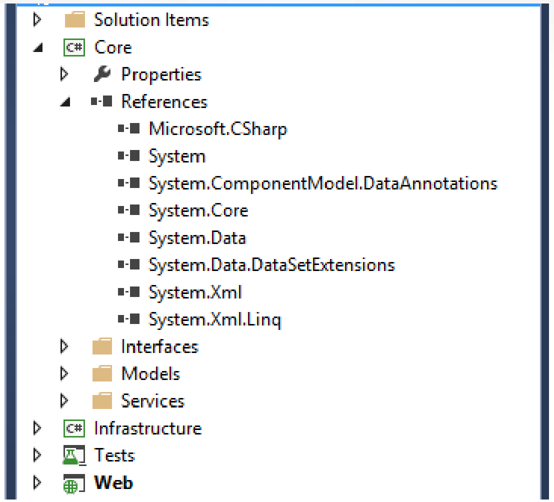
Layered Architecture

- Ports and Adapters
- Hexagonal
- Onion



Organizing in a Solution





```
▷ Solution Items
◢ Core
    ▷ Properties
    ◢ References
        Microsoft.CSharp
        System
        System.ComponentModel.DataAnnotations
        System.Core
        System.Data
        System.Data.DataSetExtensions
        System.Xml
        System.Xml.Linq
    ▷ Interfaces
    ▷ Models
    ▷ Services
▷ Infrastructure
▷ Tests
▷ Web
```

Entities

“Many objects are not fundamentally defined by their attributes, but rather by a thread of continuity and identity.”

Eric Evans

Changing Attributes Doesn't Change Which One We're Talking About

ID:1

- Name: Steve Smith
- Twitter: @ardalis
- Favorite Color: Blue

→

ID: 1

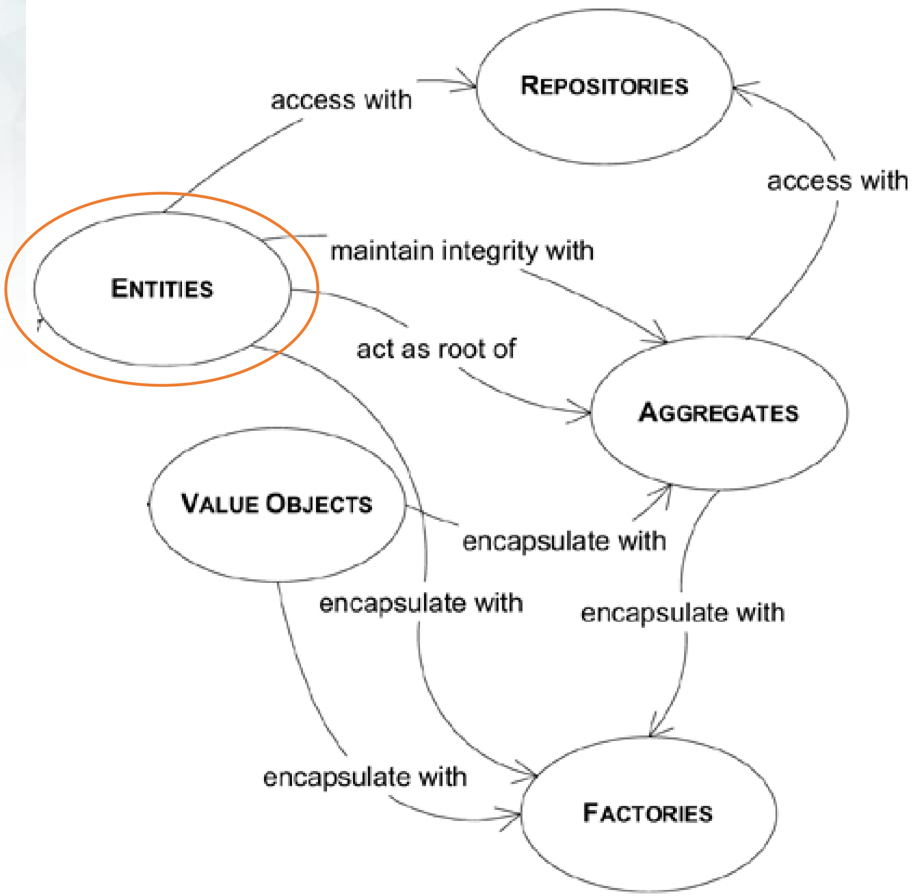
- Name: Steve**n** Smith
- Twitter: @ardalis
- Favorite Color: Blue

→

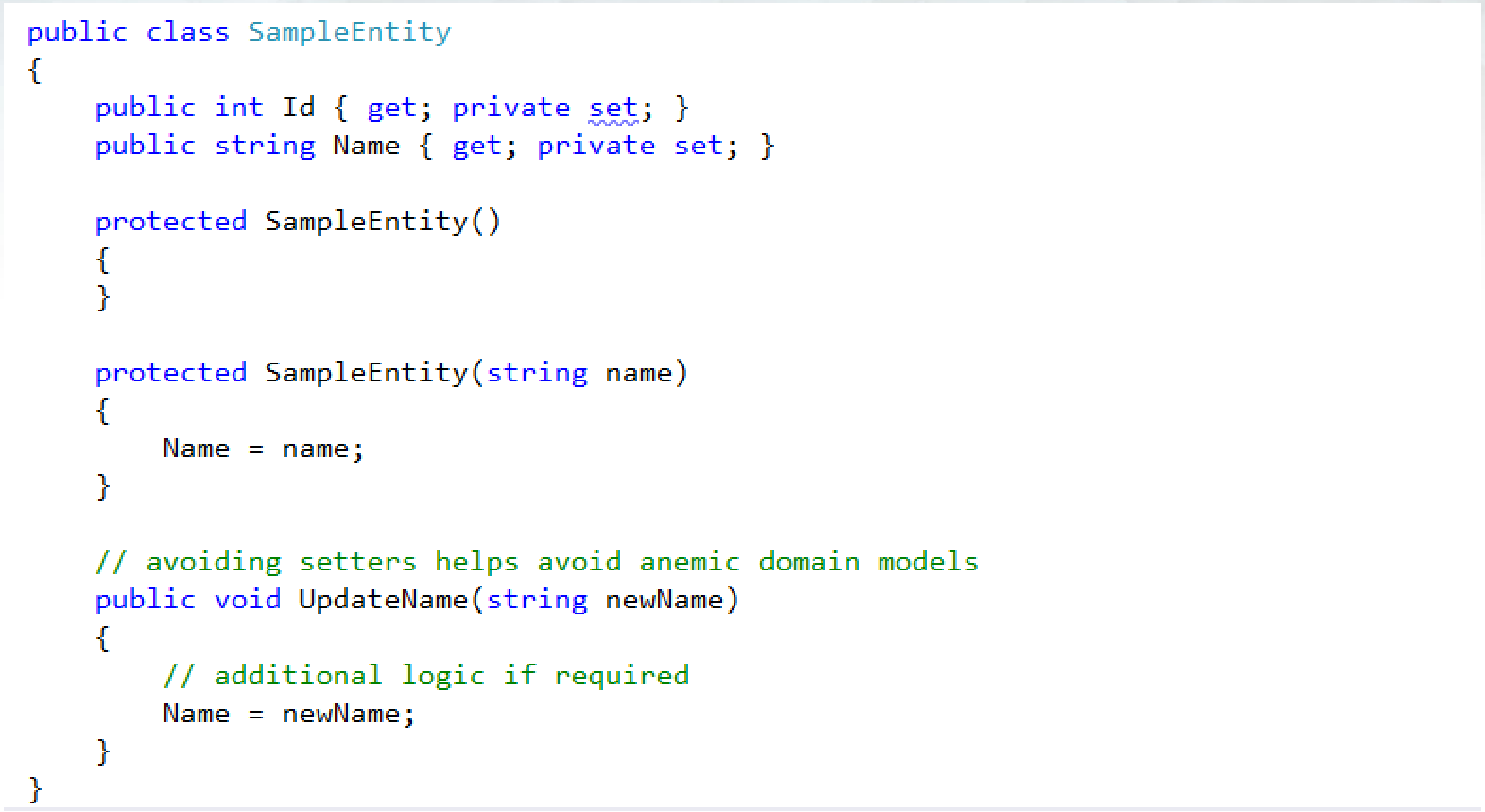
ID: 1

- Name: Steven Smith
- Twitter: @ardalis
- Favorite Color: **Orange**

CON 2014



REPOSITORIES
access with
access with
ENTITIES
maintain integrity with
act as root of
AGGREGATES
VALUE OBJECTS
encapsulate with
encapsulate with
encapsulate with
encapsulate with
FACTORIES



```
public class SampleEntity
{
    public int Id { get; private set; }
    public string Name { get; private set; }

    protected SampleEntity()
    {
    }

    protected SampleEntity(string name)
    {
        Name = name;
    }

    // avoiding setters helps avoid anemic domain models
    public void UpdateName(string newName)
    {
        // additional logic if required
        Name = newName;
    }
}
```

Value Objects

- Defined by their attributes
- Immutable
- Should have no side effects
- Examples: strings, addresses, currency

```
// this base class comes from Jimmy Bogard
// http://grabbagoft.blogspot.com/2007/06/generic-value-object-equality.html
public abstract class ValueObject<T> : IEquatable<T>
    where T : ValueObject<T>
{
    public override bool Equals(object obj)
    {
        if (obj == null)
            return false;

        T other = obj as T;

        return Equals(other);
    }

    public override int GetHashCode()...

    public virtual bool Equals(T other)...

    private IEnumerable<FieldInfo> GetFields()...

    public static bool operator ==(ValueObject<T> x, ValueObject<T> y)
    {
        return x.Equals(y);
    }

    public static bool operator !=(ValueObject<T> x, ValueObject<T> y)
    {
        return !(x == y);
    }
}
```

```
public class DateTimeRange : ValueObject<DateTimeRange>
{
    public DateTime Start { get; private set; }
    public DateTime End { get; private set; }

    public DateTimeRange(DateTime start, DateTime end)
    {
        Guard.ForPrecedesDate(start, end, "start");
        Start = start;
        End = end;
    }

    public DateTimeRange(DateTime start, TimeSpan duration)
        : this(start, start.Add(duration))...
    protected DateTimeRange() { }

    public DateTimeRange NewEnd(DateTime newEnd)
    {
        return new DateTimeRange(this.Start, newEnd);
    }
    public DateTimeRange NewDuration(TimeSpan newDuration)...
    public DateTimeRange NewStart(DateTime newStart)...

    public int DurationInMinutes()...
    public bool Overlaps(DateTimeRange dateTimeRange)...

    public static DateTimeRange CreateOneDayRange(DateTime day)
    {
        return new DateTimeRange(day, day.AddDays(1));
    }
    public static DateTimeRange CreateOneWeekRange(DateTime startDay)...
}
```

Immutable!

Domain Services

- Not a natural part of an Entity or Value Object
- Interface defined in terms of other model elements
- Should be stateless (but may have side effects)

Services in Different Layers

UI Layer & Application Layer	Domain ("Application Core")	Infrastructure
Message Sending	Transfer Between Accounts	Send Email
Message Processing	Process Order	Log to a File
XML Parsing		
UI Services		

Domain Events

"Use a Domain Event to capture an occurrence of something that happened in the domain."

Vaughn Vernon

Implementing Domain-Driven Design

Domain Event Tips

- Consider for cases of “when this happens, then…”
    - Or “Notify someone when…”
- Domain events represent the past
    - They already happened
- Thus, they should be immutable

Examples of Domain Events

User Authenticated

Appointment Confirmed

$¢£¥ Payment Received

Designing Domain Events

- Each Event is a Class
- Use a common interface (e.g. IDomainEvent)
  - Capture when the event took place
- Include details
  - What would you need to know to trigger this event again?
  - Include identities of any entities involved
- Initialize all state in constructor
- No behavior or side effects – just state

More DDD Topics

- Aggregates
- Repositories
- Factories

DDD Fundamentals on Pluralsight
Eric Evans' DDD Book
steve.smith@falafel.com

Domain Models and MVC Models

- UI interacts directly with Domain Model
    - Entities, Value Objects
    - Interfaces, Services
- Views may work with custom ViewModels
- Client (HTML/JS) code may use another ViewModel as well

Controllers

- Keep as small as possible
- Eliminate business logic
- Inject all dependencies

Views

- No logic unless encapsulated in tested helpers
- No business logic if it can instead be modeled in the domain

SignalR

- Awesome addition to ASP.NET
- Great for notifications to multiple users
- Ties in easily with Domain Events

Solution Structure

- Core
  - Interfaces
  - Model (Entities, Value Objects)
  - Domain Services
- Infrastructure
  - DbContext
  - File Access
  - System Clock Access
  - Email services
- Web
  - MVC Project
  - No direct use of Infrastructure

Demo

Putting DDD into ASP.NET MVC

FU FALAFEL UNIVERSITY

Unlimited Access to Training on

Telerik
Sitefinity
Kendo UI
Xamarin
SmartBear
Microsoft
& More

get 20% off

coupon code
FalafelCON2014

falafel.com/fu

Live Online Classes

Expert Office Hours

Learn From the Best

Flexible Class Schedule

Unlimited Annual Access

Affordable Tuition