

Don't write the code

Don't write THAT code



Jeppe Cramon

TIGERTEAM®

LEAN THINKING

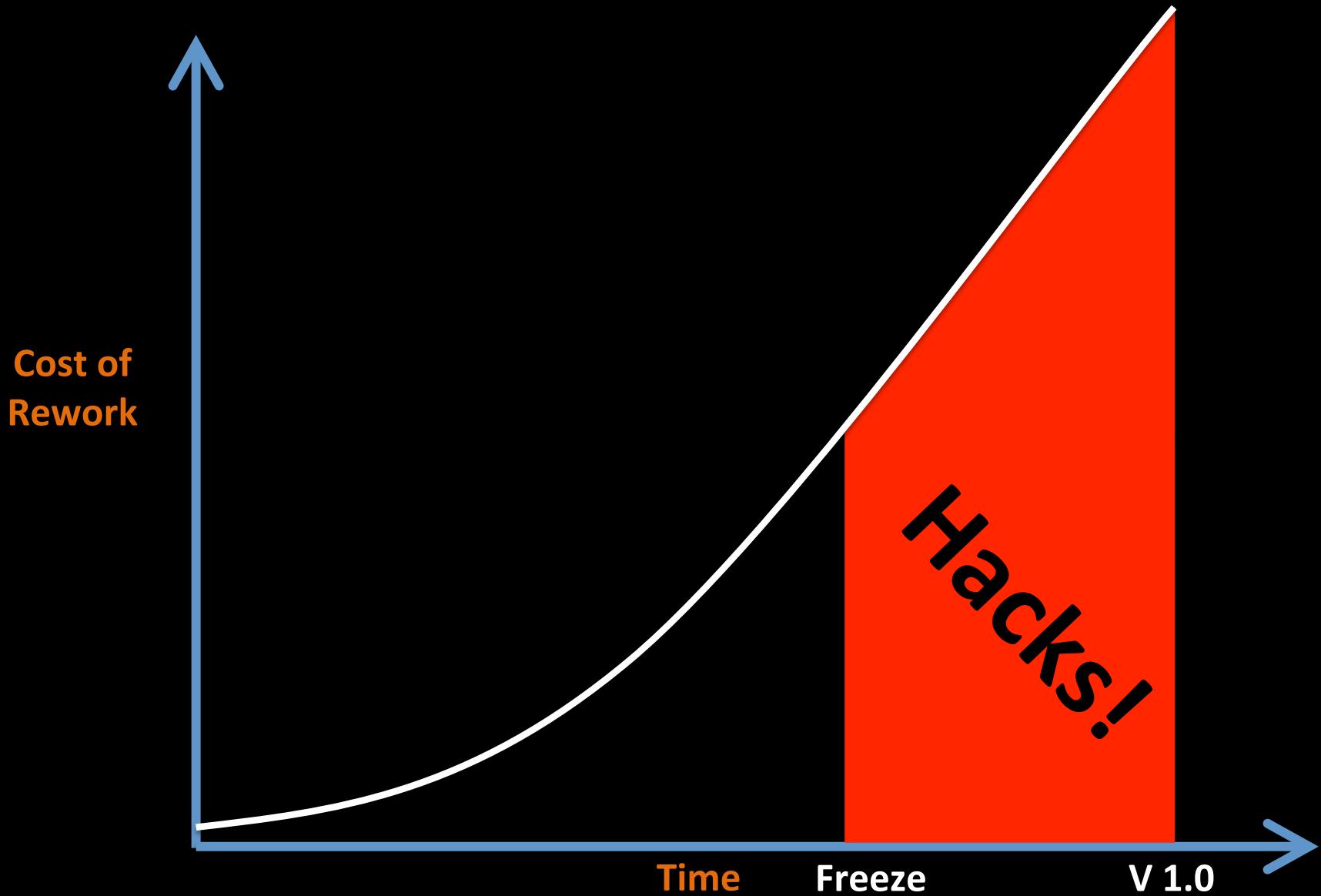
Corporate growth slowed due to lack of funding

Berlingske Finans

6 out of 10 public projects are going over time and budget

Version2

Hitting Point of No Return too Early?

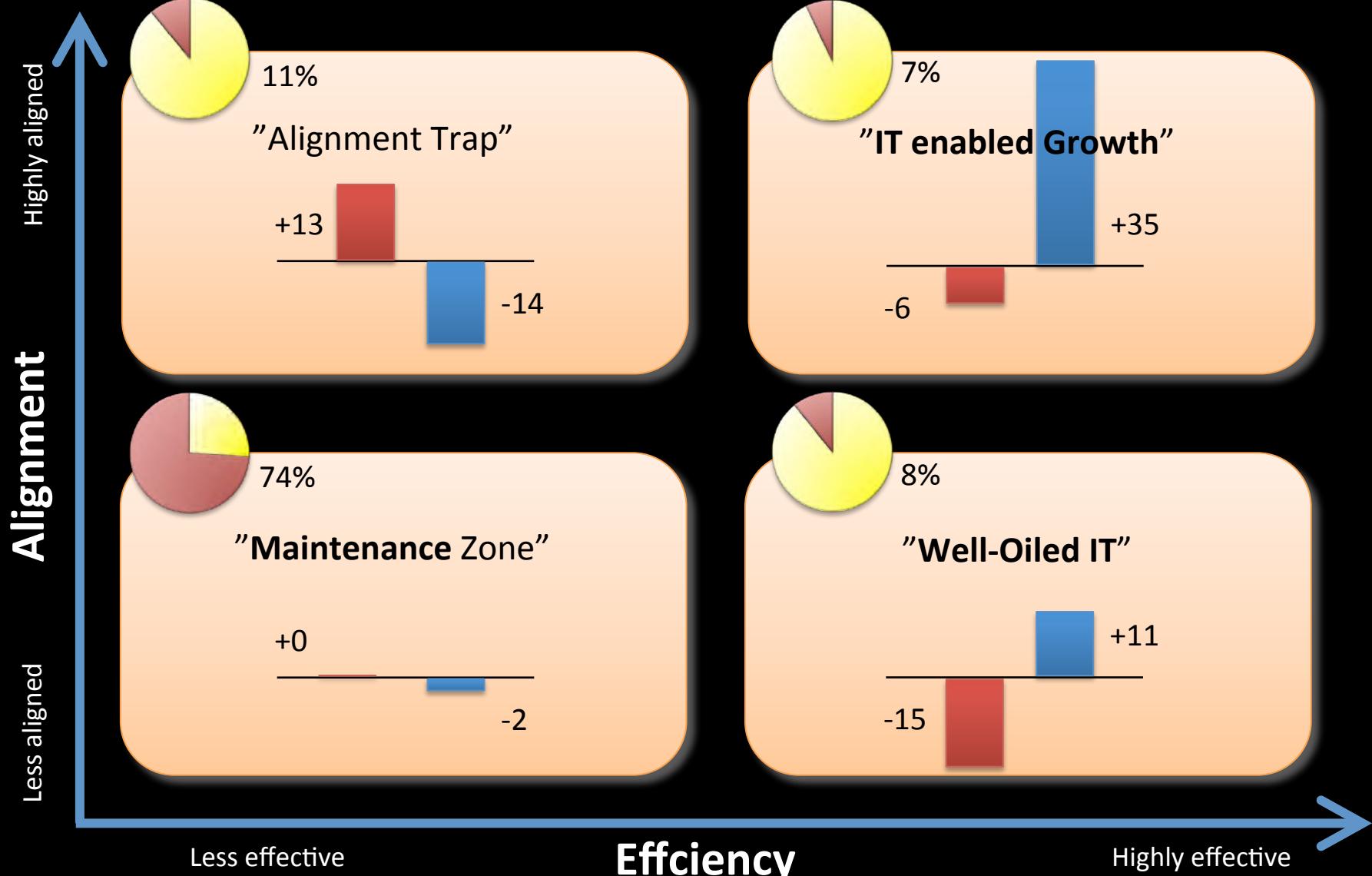


Same procedure as last year?



Same procedure as every year!!!

We need a **strong foundation** for development



% of the 504
respondents

% difference compared to the overall averages

IT spending

Combined yearly growth-
rate over a 3 year period

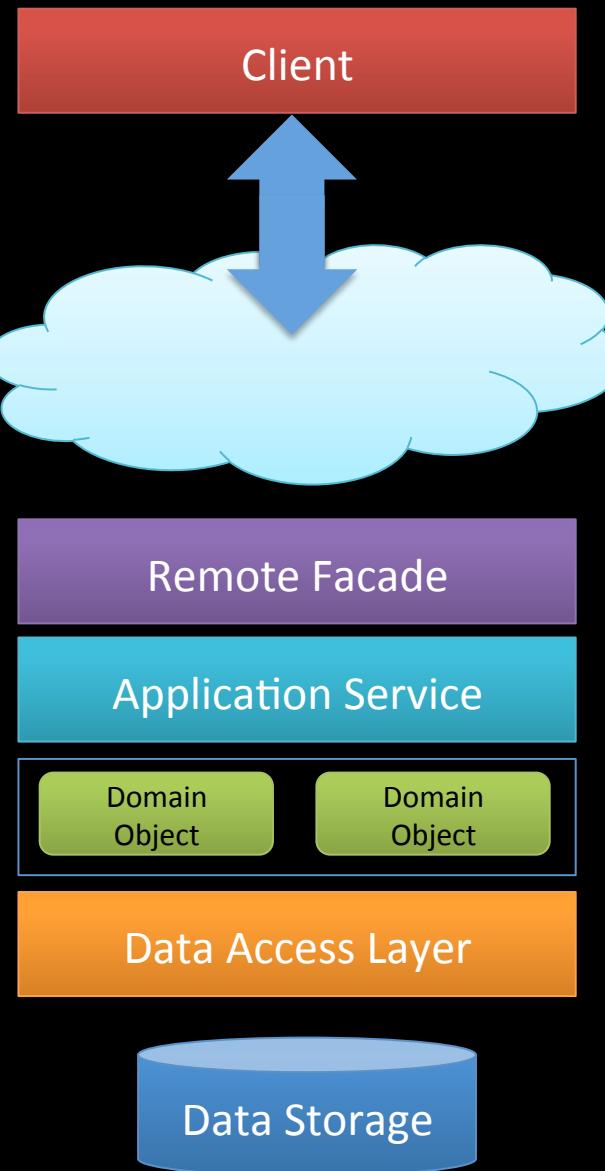
Open for new ideas?



What's for dinner?



One size fits all



Classic Java approach



Hibernate/JPA



Spring/EJB3/CDI



Developer meat

Classic Ruby on Rails approach



Developer meat

Ruby on Rails

But this is all TECHNOLOGY

When did we get so
caught up in technology,
that we forgot what we
were trying to solve?

WE'RE COMPLECTING THINGS

Rick Hickey, 2011

Git REST
Documentum

Cloud

Dart

MS SQL Server

NoSQL

Oracle DB

Policy Manager

Flash XML

JavaScript

Requirement Management tools

BlackBerry

Web Services

BPMN

Hg

Portal

SOA

iOS

Android

SharePoint

WebLogic

WebSphere

Layered Architecture

VB.NET C# F#

SubVersion

Scala

JAVA

.NET

Ruby on Rails

Team Foundation Server

Groovy

BizTalk

Oracle Service bus

The silverbullet syndrome



Let's start with the end

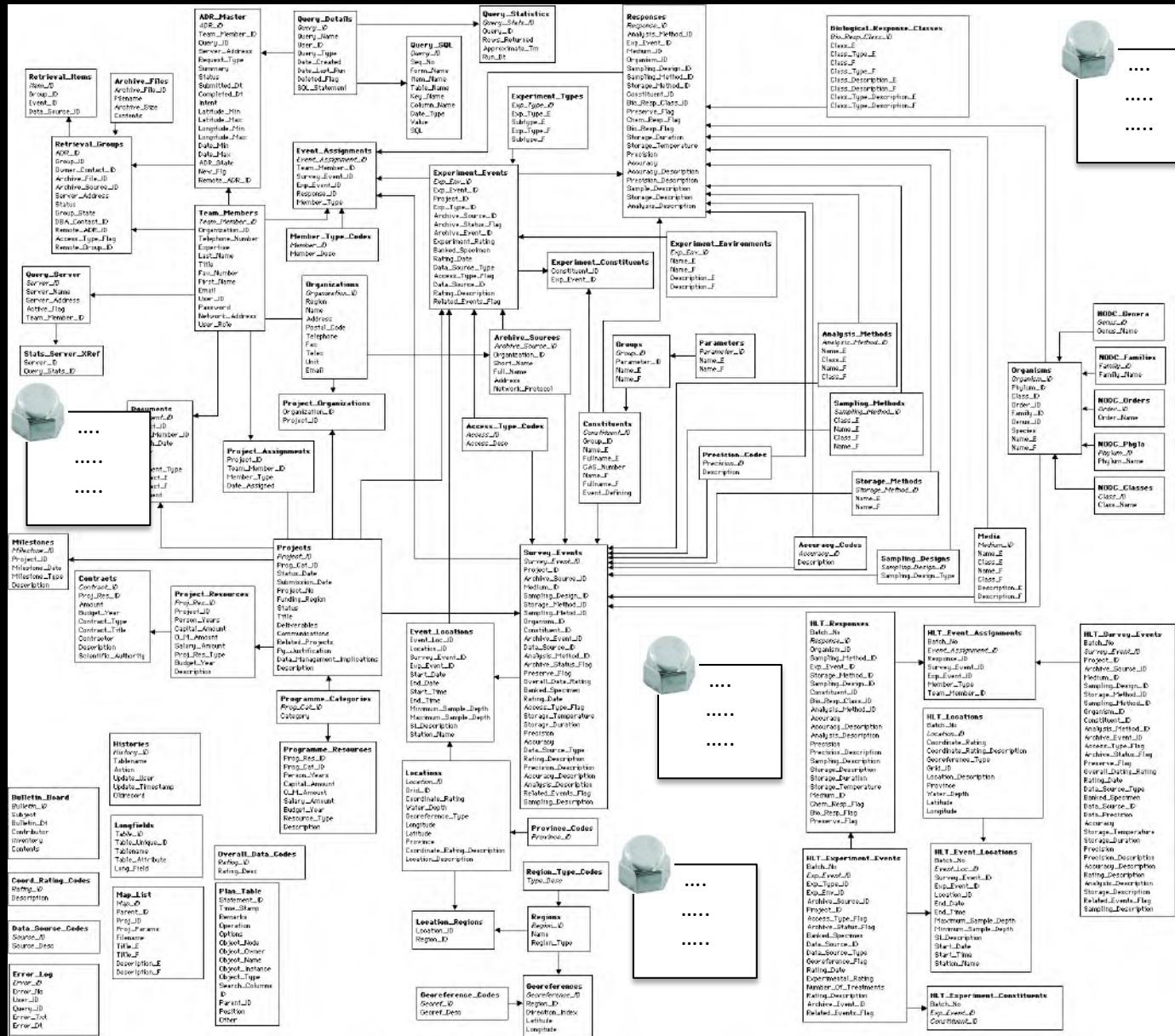


OUR HOLY DATABASE

Not all data is created equal

- Data survives your code, so treat it well
- But treat it according to its nature
- Who said a relational database was the best fit?
- Why do we create so big models?
- And why are we trying to solve all problems using the same approach?

The bigger the better – right?



A big unified model is every architects
pibe dream

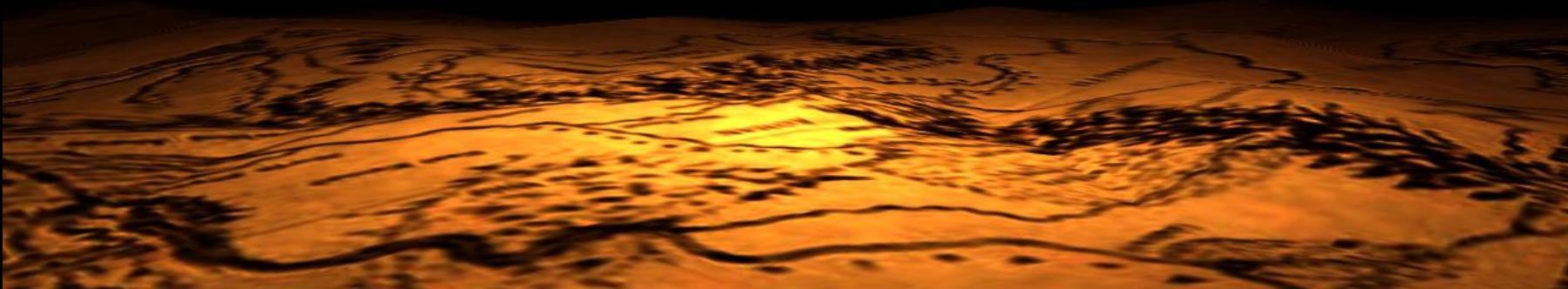
But as they say



ONE MODEL TO RULE THEM ALL

ONE MODEL TO FIND THEM

ONE MODEL TO BRING THEM ALL
AND IN THE DARKNESS BIND THEM



Result: Our queries get messy

The Query of Despair

Let's fix the complexity of SQL code

Let's use a big hammer – ORM

Introducing ORM brings other fun things such as

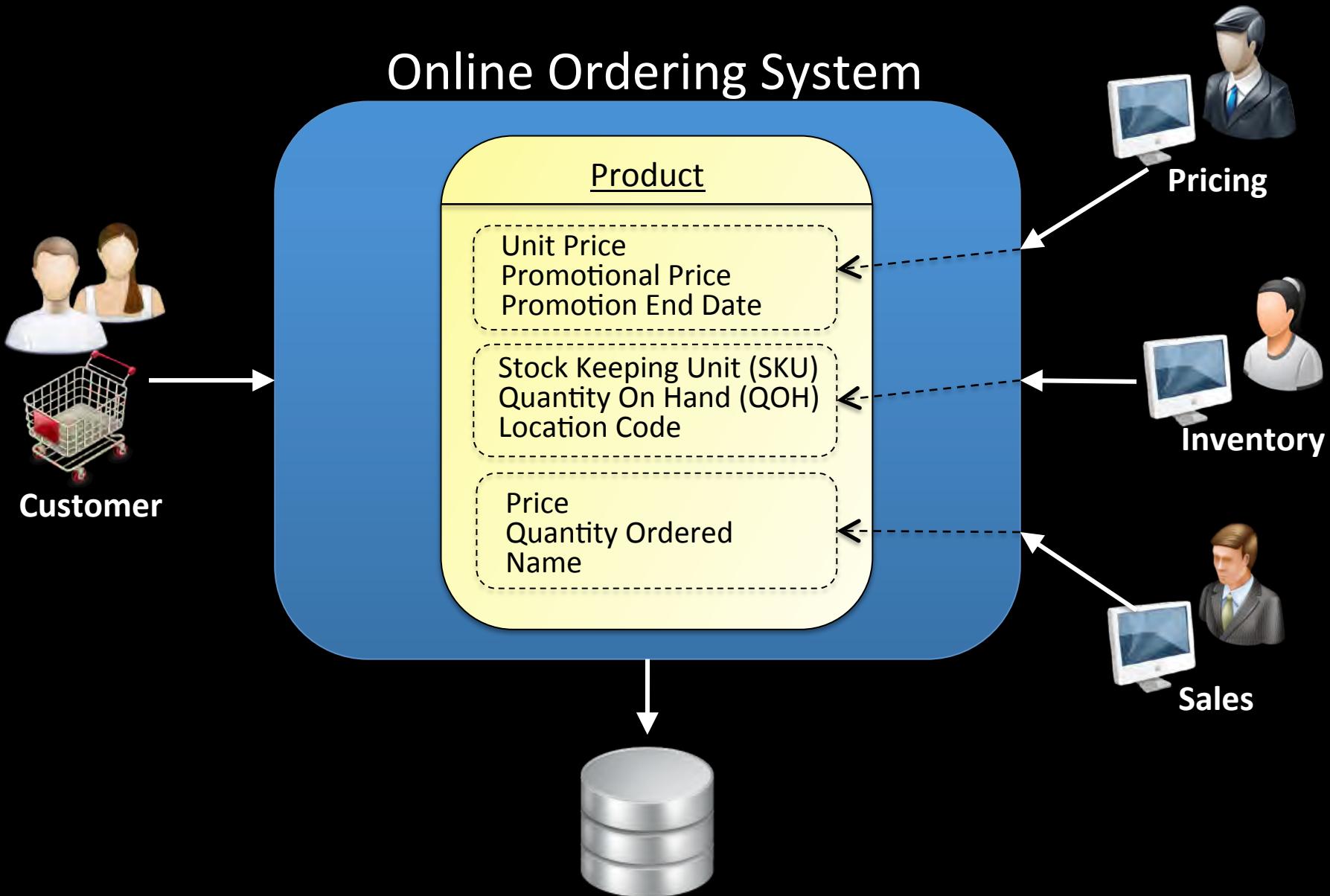
- When should we fetch associations **LAZY** and when **EAGER** ? It depends on the use case
- Batch processing becomes tricky
- By convention, names for Tables, Join columns, Join Tables Foreignkeys, etc. might not match corporate standard or be too long for **Oracle** ;)
- General OO vs. ER **impedance mismatch**
- Testing ORM code is tedious



**LET'S TAKE A STEP BACK AND LOOK
AT THE NATURE OF DATA**

Many perspectives on data

Online Ordering System



There's always more than one
model at play in any big project

But when code based on
multiple models is combined

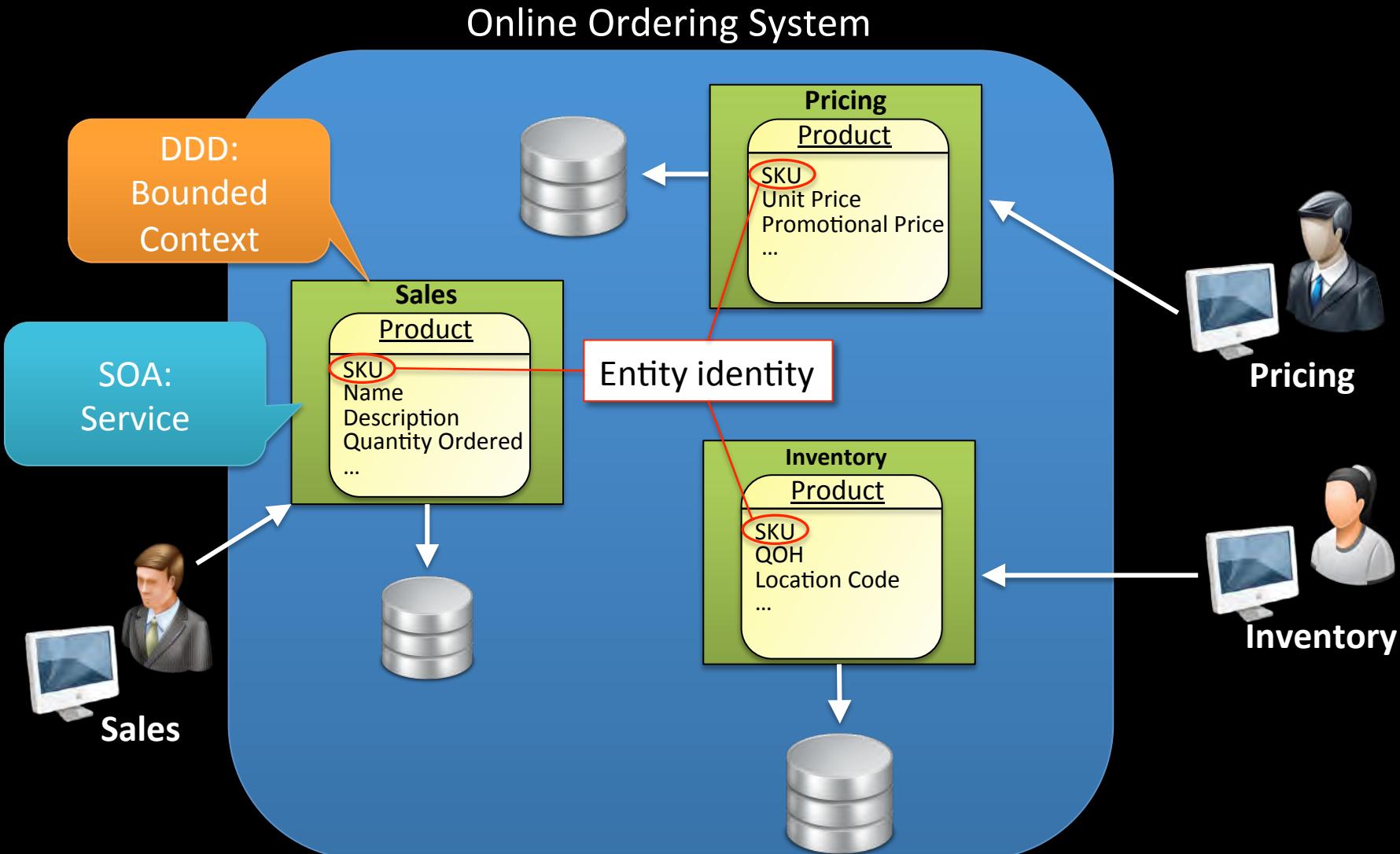
... then the code becomes filled with errors, unreliable and difficult to understand

Communication between team members becomes confusing.

It is often unclear in what context a model should NOT be used!

Context is lacking

Smaller models & clear data ownership



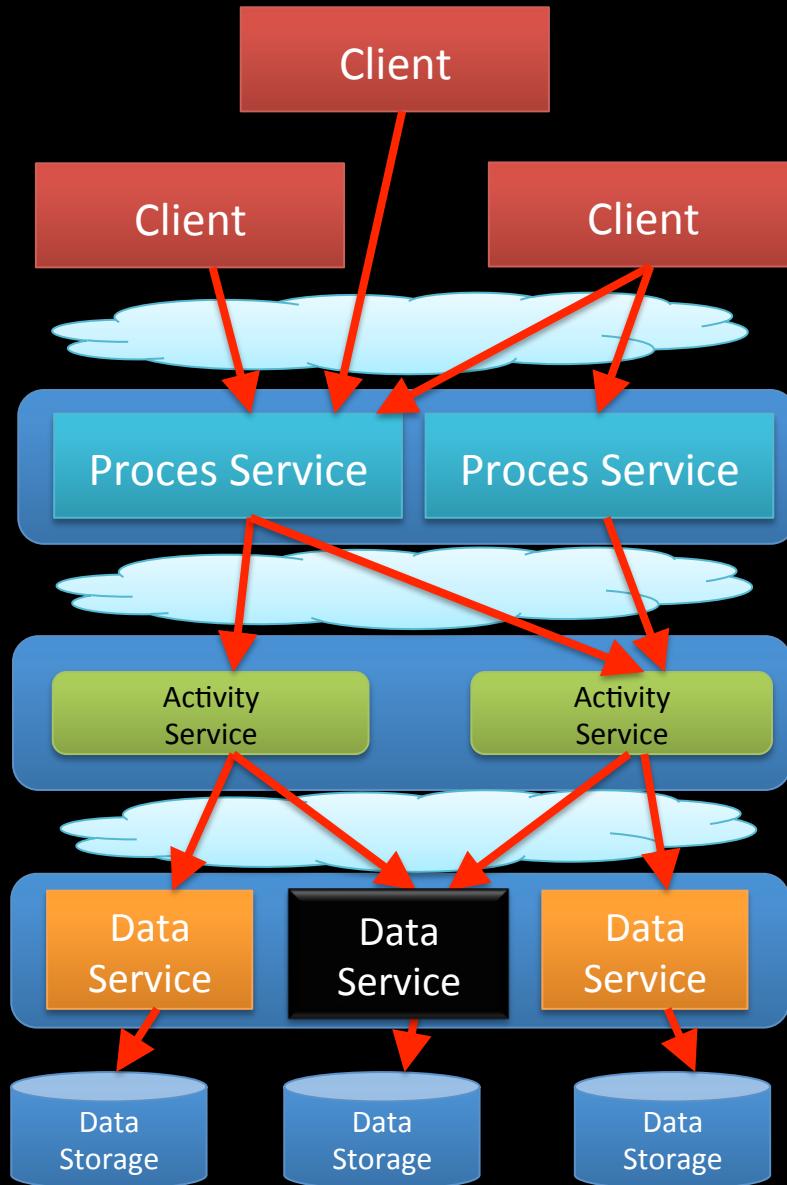
Somethings missing

How do we collaborate?

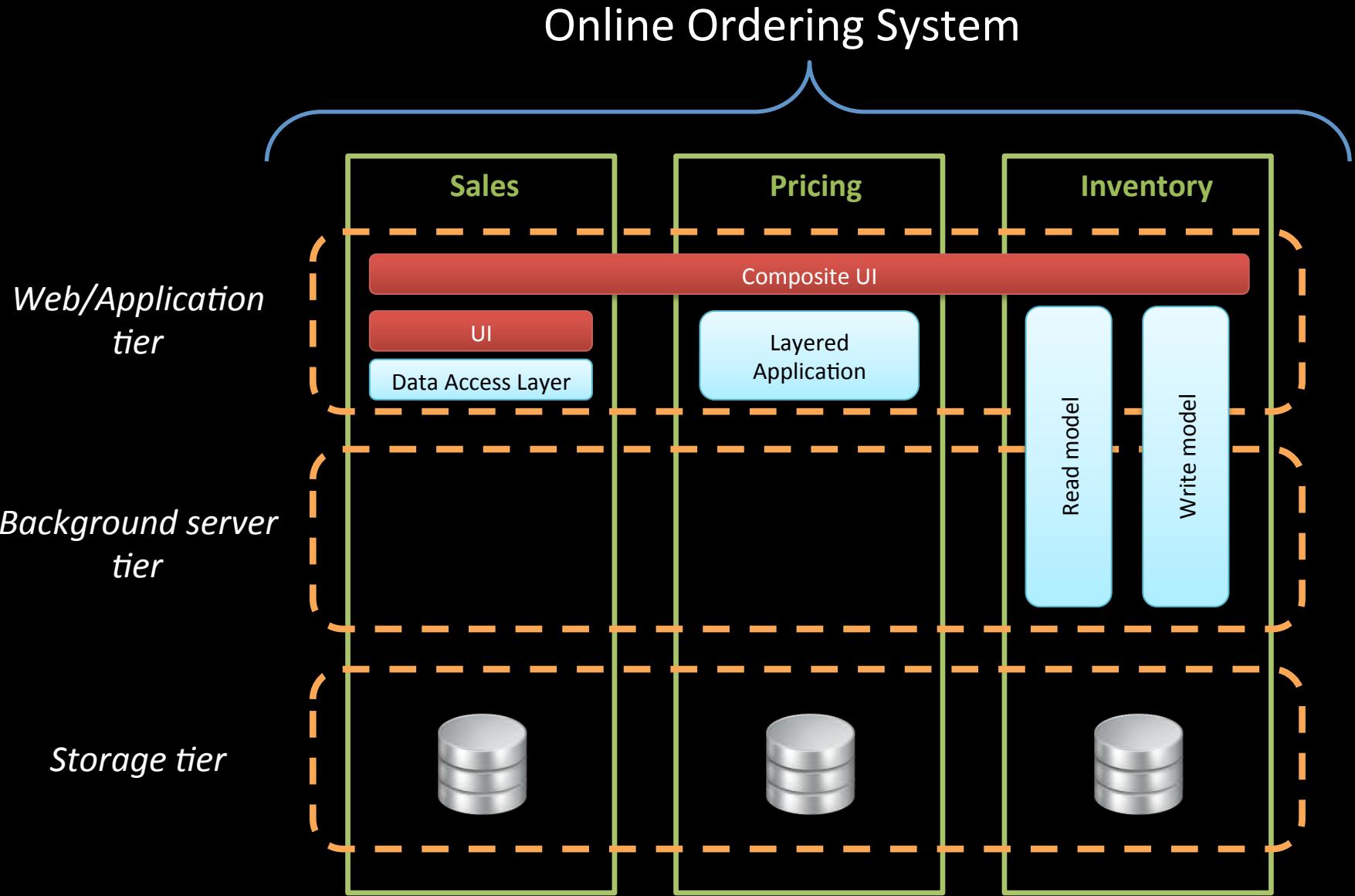
Status Quo



Traditional Layered (SOA) Solution



Composite UI's



Composite UI - example

amazon.co.uk Hello. Sign in to get [personalised recommendations](#). New Customer? [Start here.](#)

Your Amazon.co.uk | Today's Deals | Gift Cards | Gifts & Wish Lists

Shop All Departments Search Books Books Advanced Search Browse Genres Bestsellers New & Future Releases Paperback

[Domain-Driven Design](#) and over 900,000 other books are available for Amazon.co.uk

[Click to LOOK INSIDE!](#)

Domain-Driven DESIGN
Tackling Complexity in the Heart of Software

Domain-driven Design: Tackling Complexity in the Heart of Software [Hardcover]

Eric Evans (Author)

(12 customer reviews) (15)

RRP: £46.99
Price: **£30.09** & this item **Delivered FREE in the UK** with Super Saver Delivery. [See details and delivery options](#)
You Save: £16.90 (36%)

In stock.
Dispatched from and sold by **Amazon.co.uk**. Gift-wrap available.

Want guaranteed delivery by Friday, March 23? Order it in the next 21 hours and 9 minutes, and choose Express delivery.

Customers Who Bought This Item Also Bought

Patterns of Enterprise Application Architecture... by Martin Fowler

(14)
£33.59

Growing Object-Oriented Software, Guided by Test... by Steve Freeman

(15)
£18.35

Clean Code: A Handbook of Agile Software Craftsmanship by Robert C. Martin

(27)
£17.00

Enterprise Integration Patterns: Designing, Building, and Refactoring Message-Based Systems by Gregor Hohpe

(16)
£29.61

Continuous Delivery: Reliable Software Releases Through Feedback Loops by Jez Humble

(7)
£19.97

Refactoring: Improving the Design of Existing Code by Martin Fowler

(24)
£29.61

Product details

Hardcover: 560 pages

Publisher: Addison Wesley; 1 edition (20 Aug 2003)

That covers data presentation

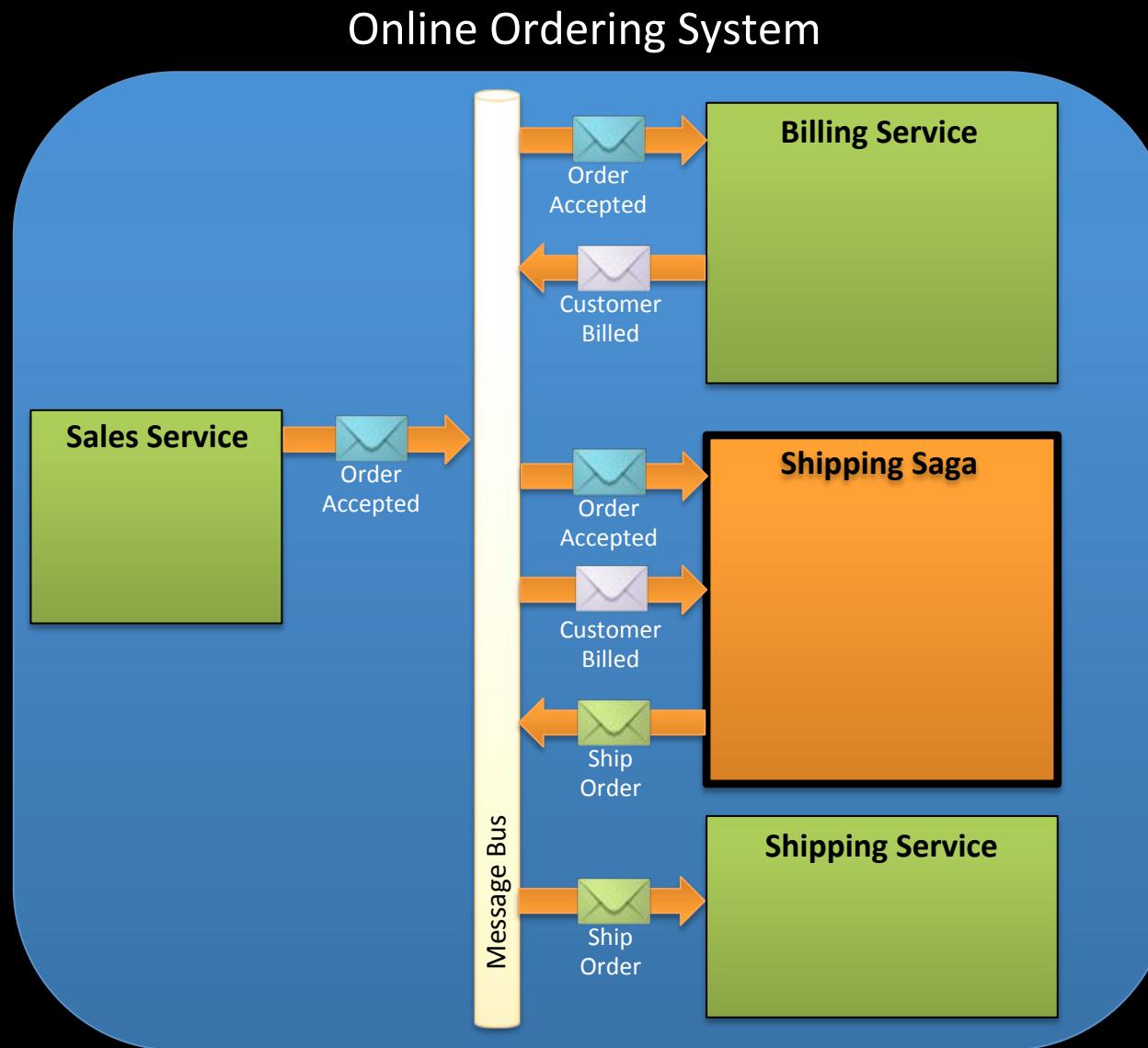
What about transactions?

Next Step – Event Driven Architecture

Online Ordering System



Introducing Sagas

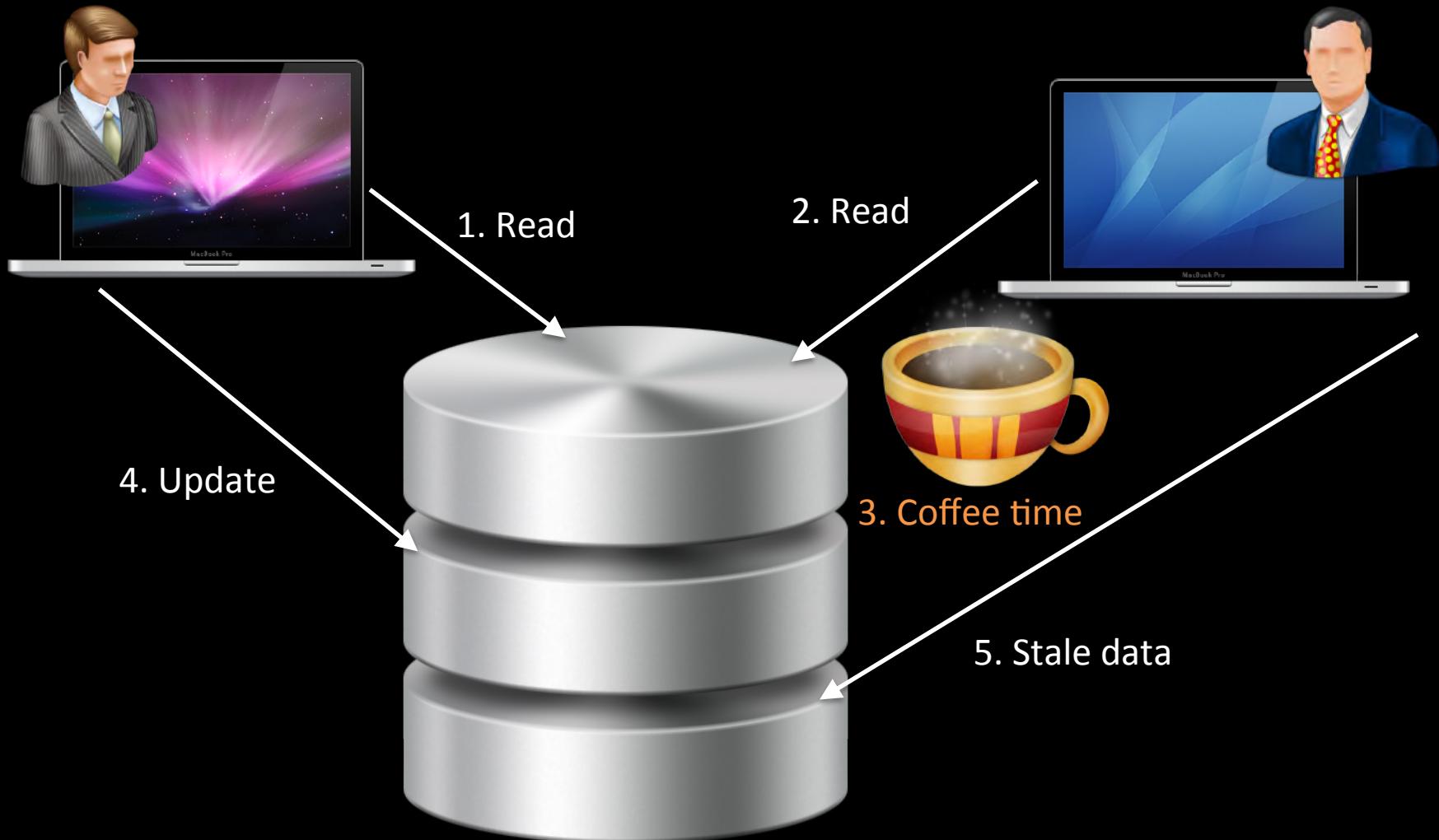


How do we scale?

Both in terms of:
developer productivity & runtime performance

Let's first check the premise

Collaborative domains



We're working with stale data

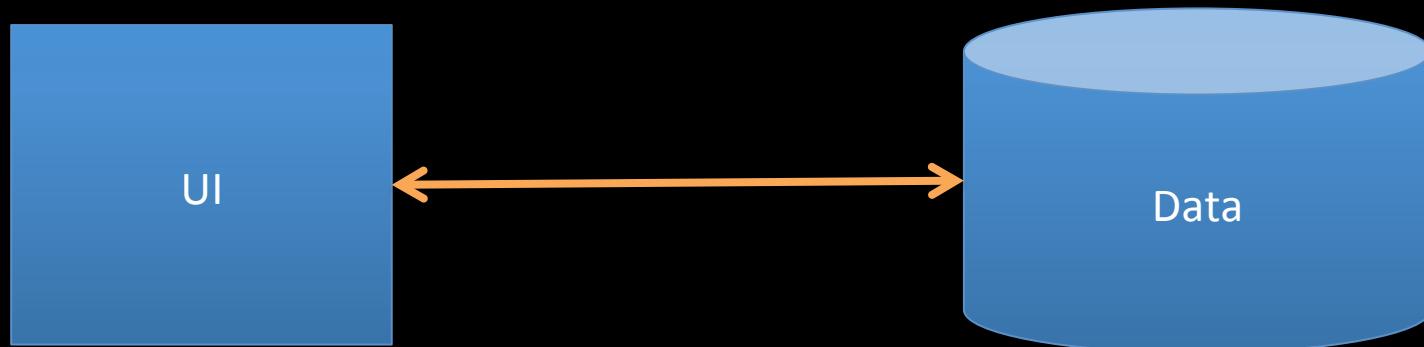
Optimistic Locking Exception



Our architectures often have
problems handling this type of
problem

Let's look at how we've evolved

Thin Layers of complexity

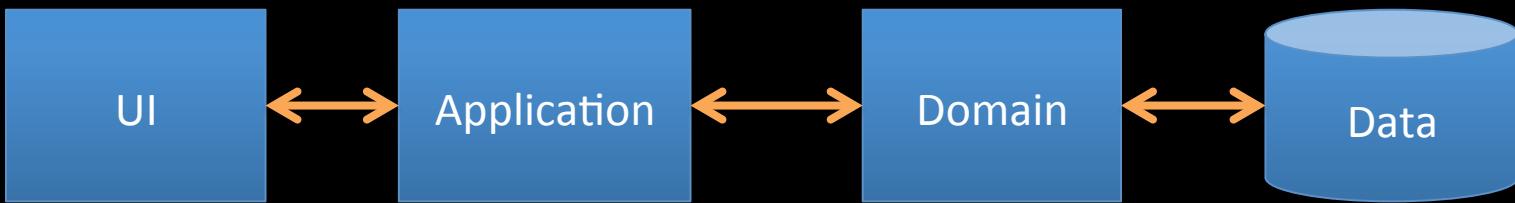


Pro: Easy and simple

Con: Scales poorly for complex domains

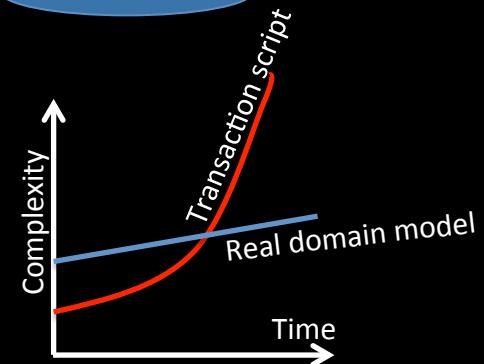


Next step – more layers



Pro: Handles complex domains better by Separating usecase and domain logic

Con: Increasing complexity. Risk of Anemic Domain model combined with transaction-script.



Queries are just data – no logic

- Give me the customer with his last 10 orders
- Give me the customer with total sum of orders for the last year
- Give me the complete order
- Give me the shipping status for the order

So why go through all those layers?

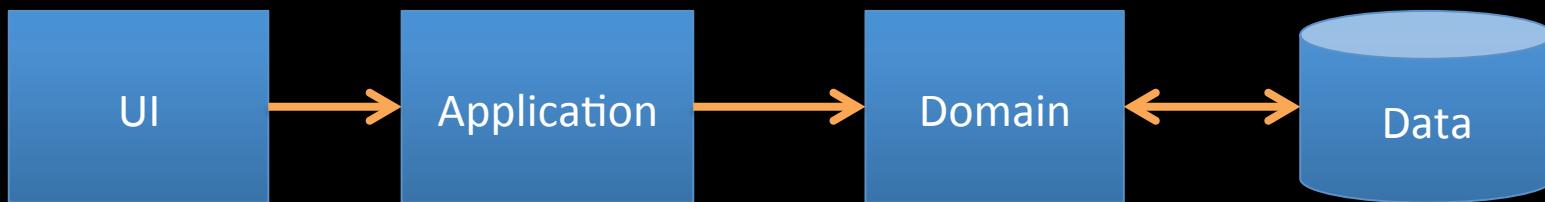
A single model cannot be appropriate for reporting, searching and transactional behavior

Greg Young, 2008

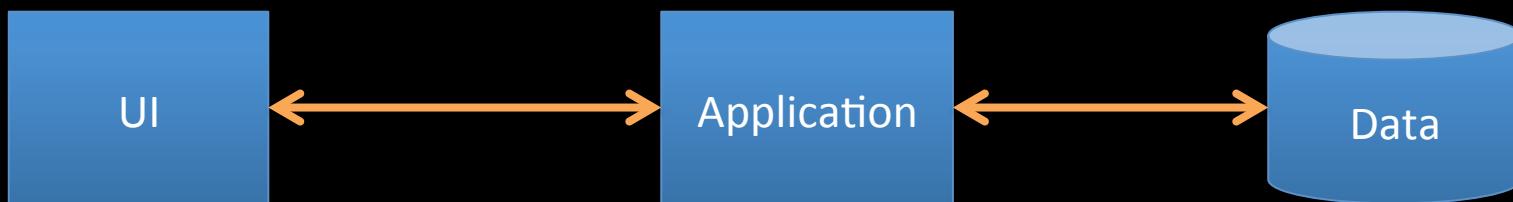
CQS

Separation of
functions that **write**
&
functions that **read**

Commands – Change data



Queries – Ask for data (no side effects)



CQS from a code perspective

```
public class CustomerService
{
    // Commands
    void MakeCustomerPreferred(CustomerId)
    void ChangeCustomerLocale(CustomerId, NewLocale)
    void CreateCustomer(Customer)
    void EditCustomerDetails(CustomerDetails)

    // Queries
    Customer GetCustomer(CustomerId)
    CustomerSet GetCustomersWithName(Name)
    CustomerSet GetPreferredCustomers()
}
```

NEXT STEP - CQRS

*CQRS is simply the creation of two objects
where there was previously only one*

Greg Young

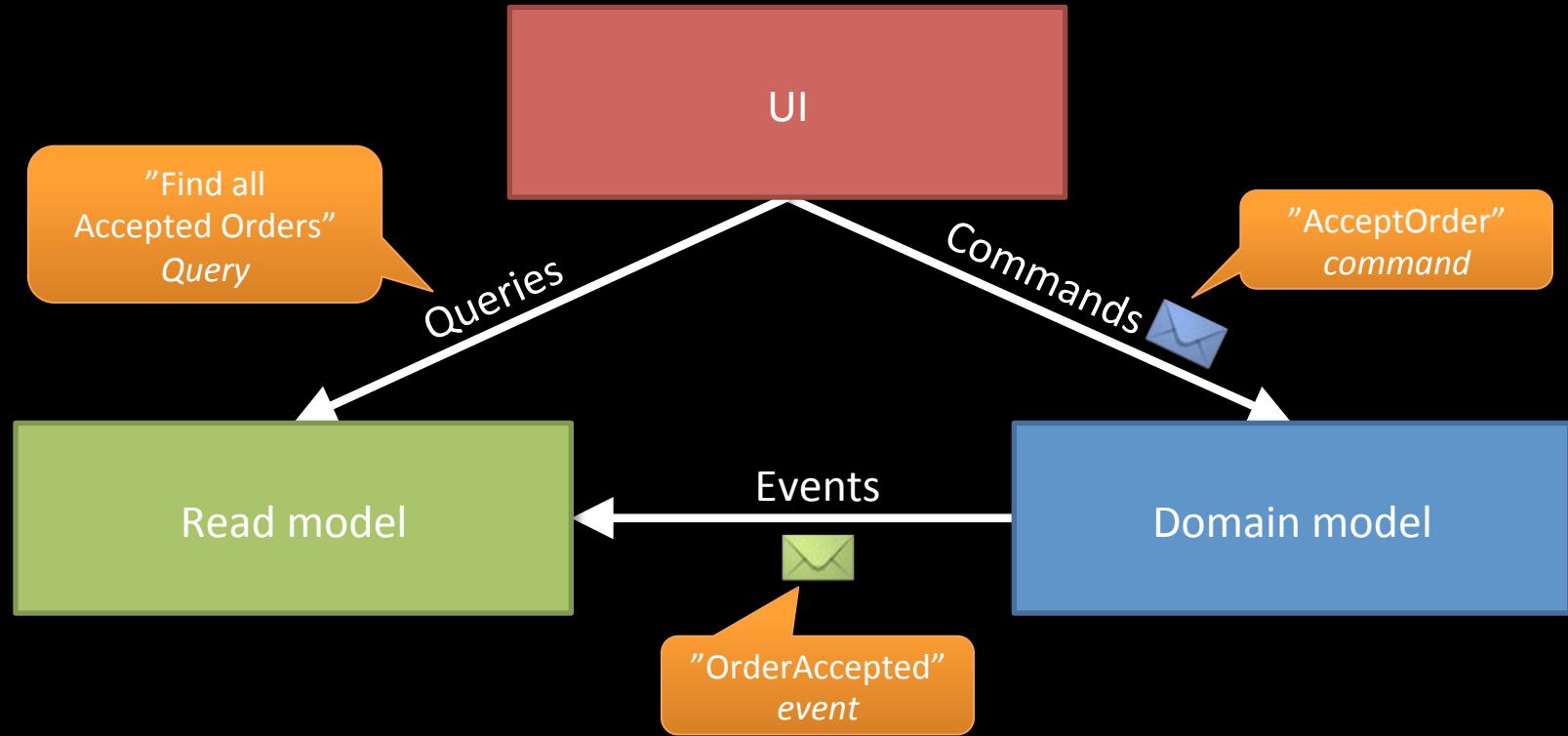
CQRS

In its simplest form

```
public class CustomerWriteService
{
    // Commands
    void MakeCustomerPreferred(CustomerId)
    void ChangeCustomerLocale(CustomerId, NewLocale)
    void CreateCustomer(Customer)
    void EditCustomerDetails(CustomerDetails)
}

public class CustomerReadService
{
    // Queries
    Customer GetCustomer(CustomerId)
    CustomerSet GetCustomersWithName(Name)
    CustomerSet GetPreferredCustomers()
}
```

Commands & Events are Messages

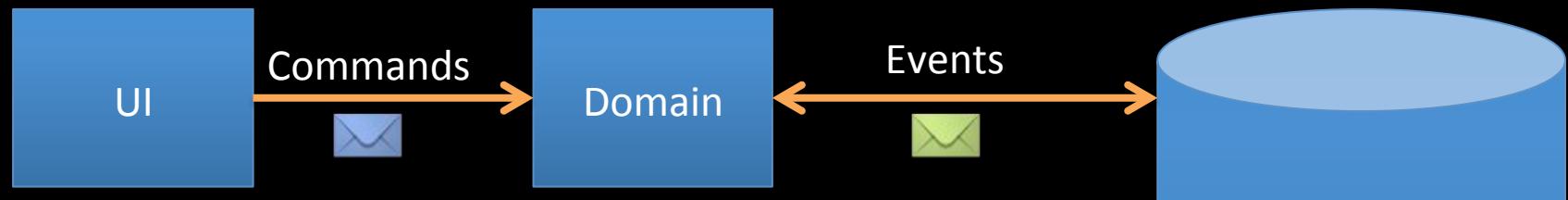


Commands are Imperative: DoStuff
Events are Past tense: StuffDone

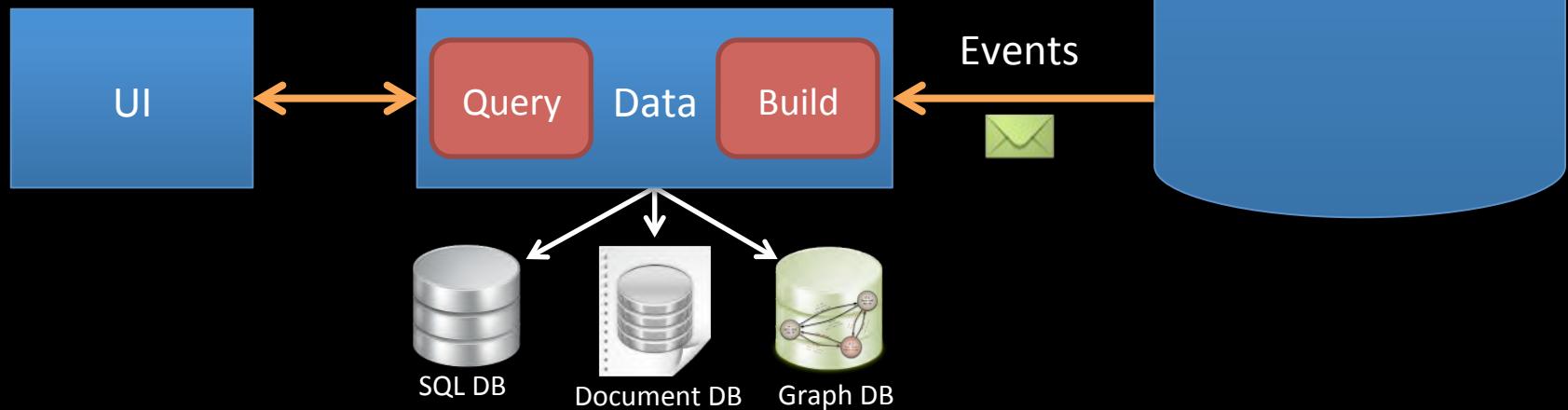
CQRS

As its often practiced

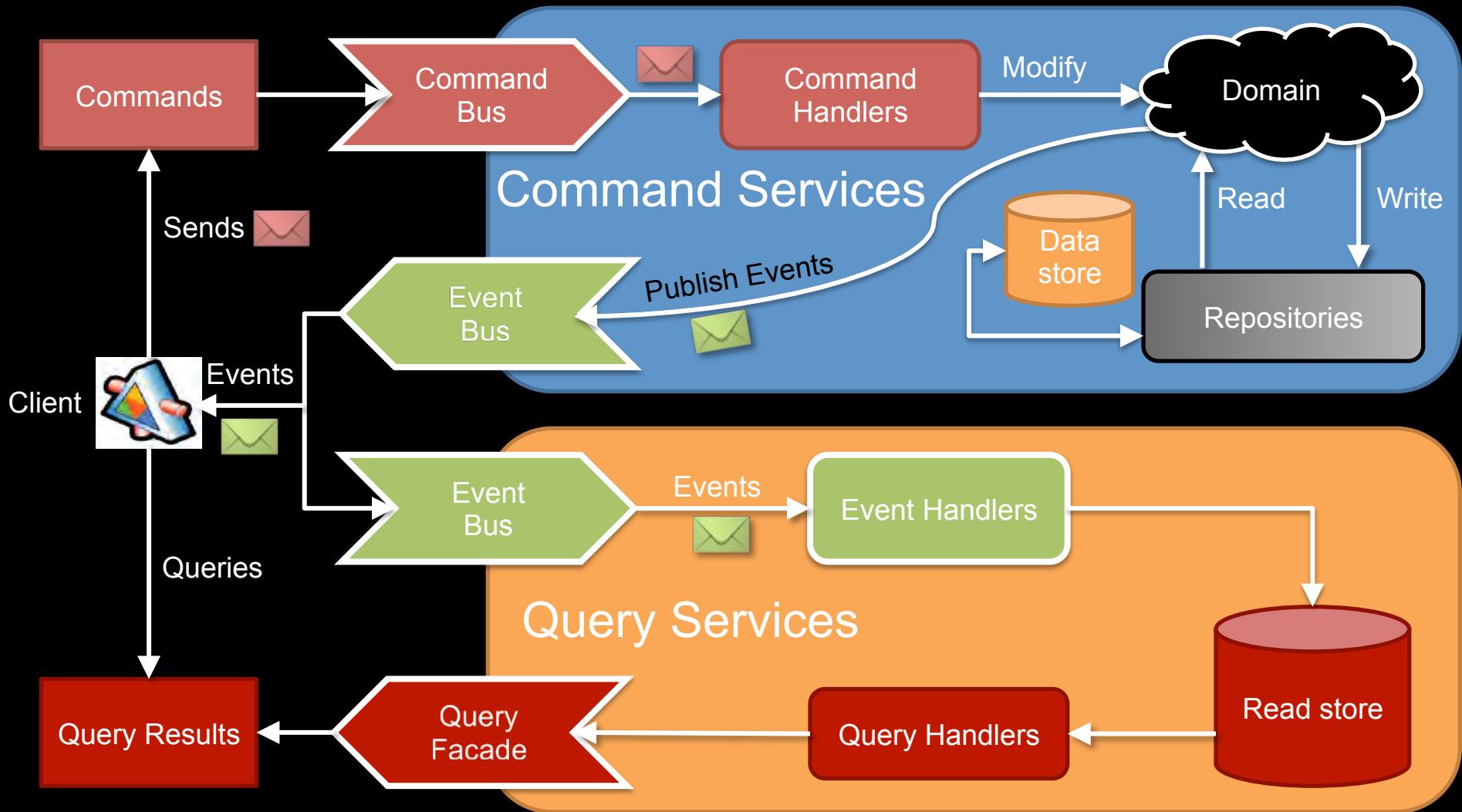
Commands – Change data



Queries – Ask for data



CQRS Building blocks



New UI paradigm

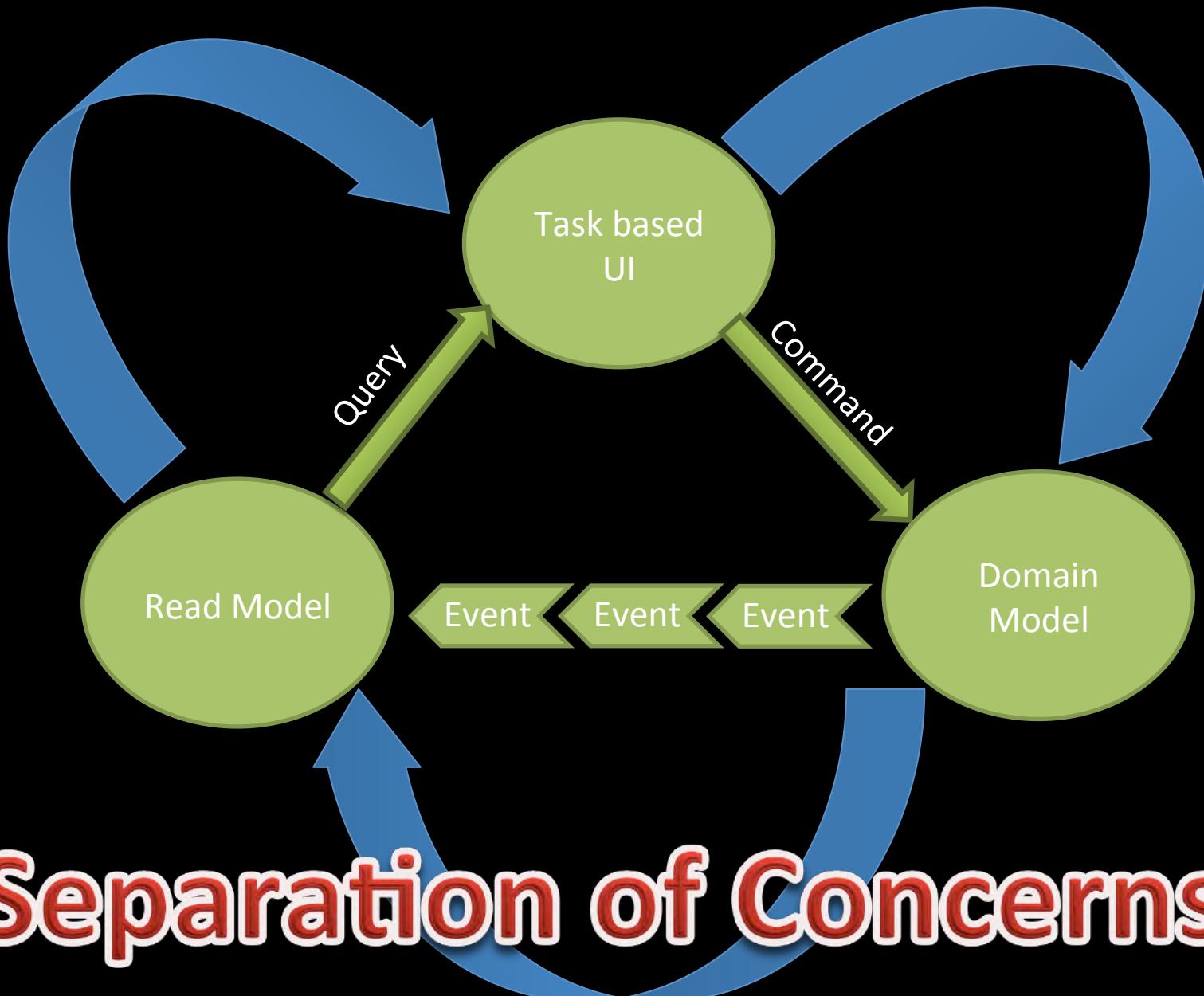
Task-based/Inductive UI

- A departure from the traditional **WHAT** UI (CRUD)
- Task based UI's focuses on **HOW** the user wants to use the application
- **Guides users** through the work process
- The UI becomes an **intrinsic part** of the design
- The UI design directly affects our **commands** and thereby our **transactional boundaries**
- Should be **preferably** use **asynchronous Commands**

What about Validation

- Happens in the UI before Commands are sent
- Typically validated using Read models
- Helps to ensure that commands don't often fail
- Validates simple things (values ranges, unique keys/values)
- Doesn't enforce business logic rules

Circular architecture



Separation of Concerns

Separation of Concerns

- Different developer skills and roles for command and query parts
- New Business Requirements ripples controllably between areas
- No need for Getters & Setters in your domain model
- Testability – Event based testing is very easy

```
public class CustomerCommandHandler {
    private Repository<Customer> customerRepository;

    public CustomerCommandHandler(Repository<Customer> customerRepository) {
        this.customerRepository = customerRepository;
    }

    @CommandHandler
    public void handle(UnsignCustomer cmd) {
        Customer customer = repository.load(cmd.getCustomerId());
        customer.unsign();
    }
}
```

```
public class Customer {
    private boolean signedUp;

    public void unsign() {
        if (signedUp) {
            apply(new CustomerUnsignedEvent());
        }
    }

    @EventHandler
    private void handle(CustomerUnsignedEvent event) {
        signedUp = false;
    }
}
```

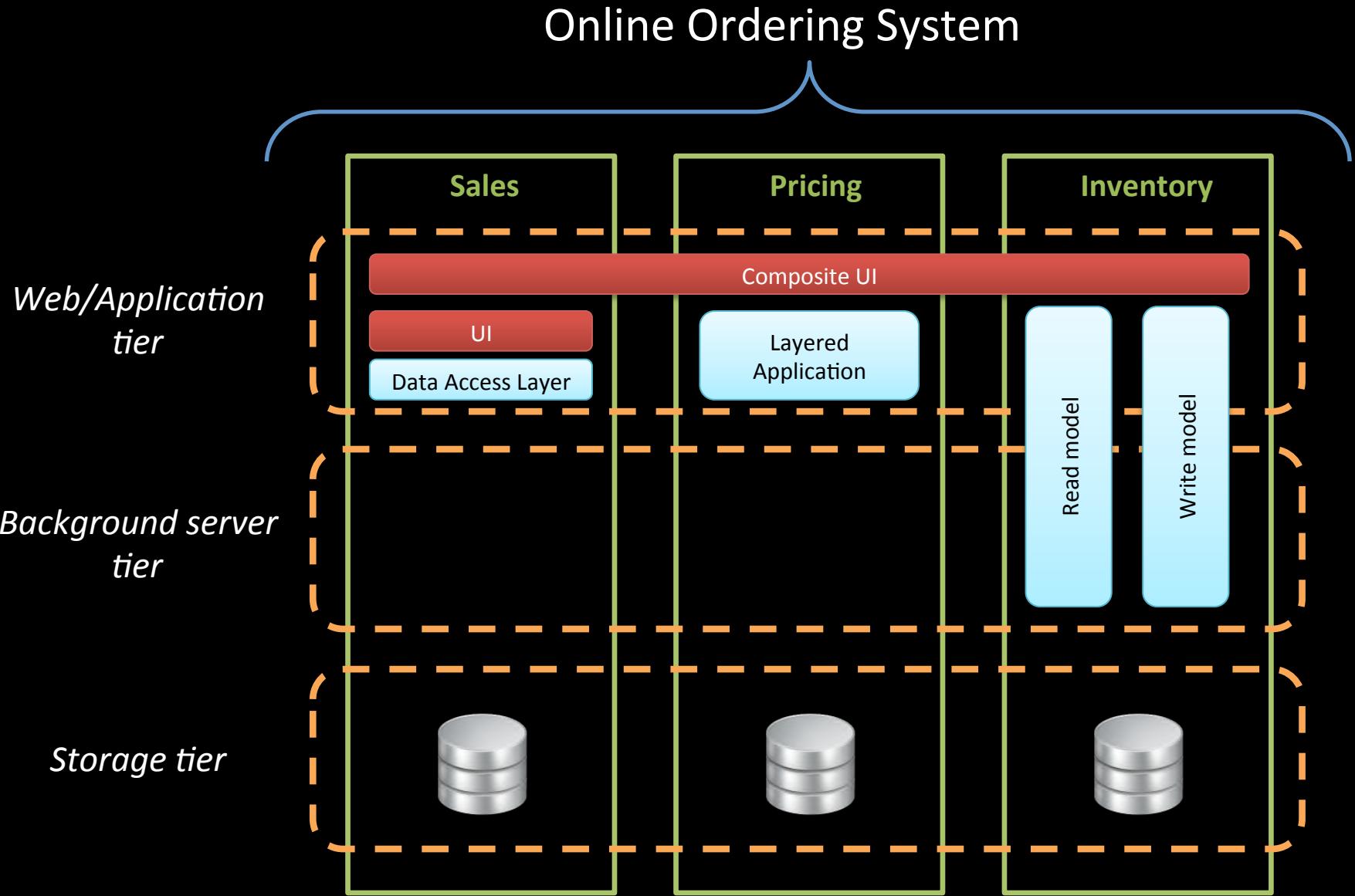
Testing CQRS

```
@Test
public void a_signup_customer_can_unsign() {
    UUID customerId = UUID.randomUUID().toString();

    FixtureConfiguration fixture = Fixtures.newGivenWhenThenFixture();
    fixture.registerAnnotatedCommandHandler(
        new CustomerCommandHandler(fixture.createGenericRepository(Customer.class))
    );
    fixture.setAggregateIdentifier(customerId);

    fixture
        .given(
            TestFactory.customerCreatedEvent(customerId),
            TestFactory.customerSignedUpEvent(customerId)
        )
        .when(
            TestFactory.unsignCustomerCommand(customerId)
        )
        .expectEvents(
            TestFactory.customerUnsignedEvent(customerId)
        );
}
```

CQRS is not top level architecture



Don't WRITE the code



TIGERTEAM®
LEAN THINKING



Electronic Landregistration

Case study



Context

- Extremely complex domain area
- The rules are determined by Law (not logic)
- Covering registrations there are several hundred years old
- Complex logic required to automate law
- Very short deadline for the third attempt to build ETL

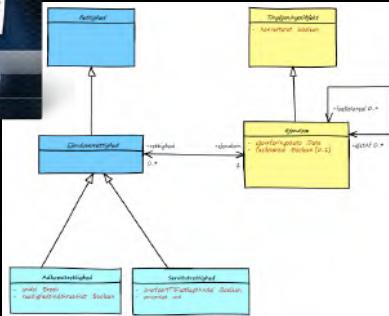


Electronic Landregistration



- Technical choices
 - Programming language: Java
 - Database: Oracle
- First challenge
 - How do we integrate Java and Oracle database?

How it's usually done



```

create table Tinglysningsobjekt(DTYPE varchar(255) char) not null, id number(19,0) not null,
oprettet timestamp, optlockversion number(19,0) not null,
sistOpdateret timestamp, uid char(36) unique,
konverteret number(1,0) not null, ajourfør-
fællesareal number(1,0), primær key (id),
andel_nævner number(10,0), andel_tæller number(10,
raadighedsindskrænket number(1,0), ejtaf number(19
  
```



SQL

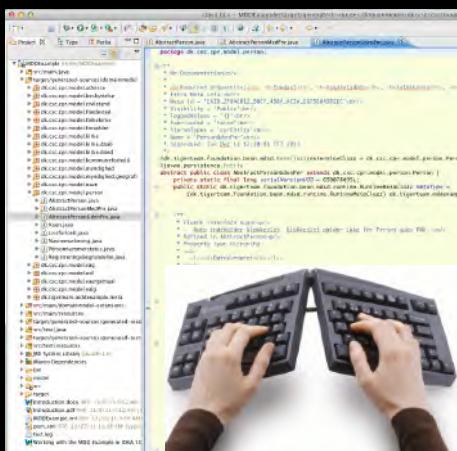
RETTIGHED	
DTYPE	VARCHAR(255)
ID	DECIMAL(19,0)
OPPRETTET	TIMESTAMP
OPTLOCKVERSION	DECIMAL(19,0)
SISTOPDATERET	TIMESTAMP
UUID	CHAR(36)
OVERFØRTILFASTLAGTÅNDDEL	DECIMAL(1,0)
PRIORITET	DECIMAL(10,0)
ANDEL_NÆVNER	DECIMAL(10,0)
ANDEL_TÆLLER	DECIMAL(10,0)
RAADIGHEDSINDSKRÆNKT	DECIMAL(1,0)
EJIDID	DECIMAL(19,0)

EJENDOM	
DTYPE	VARCHAR(255)
ID	DECIMAL(19,0)
OPPRETTET	TIMESTAMP
OPTLOCKVERSION	DECIMAL(19,0)
SISTOPDATERET	TIMESTAMP
UUID	CHAR(36)
KONVERTERET	DECIMAL(1,0)
AJOURFØRINGSdato	DATE
FÆLLESAREAL	DECIMAL(1,0)

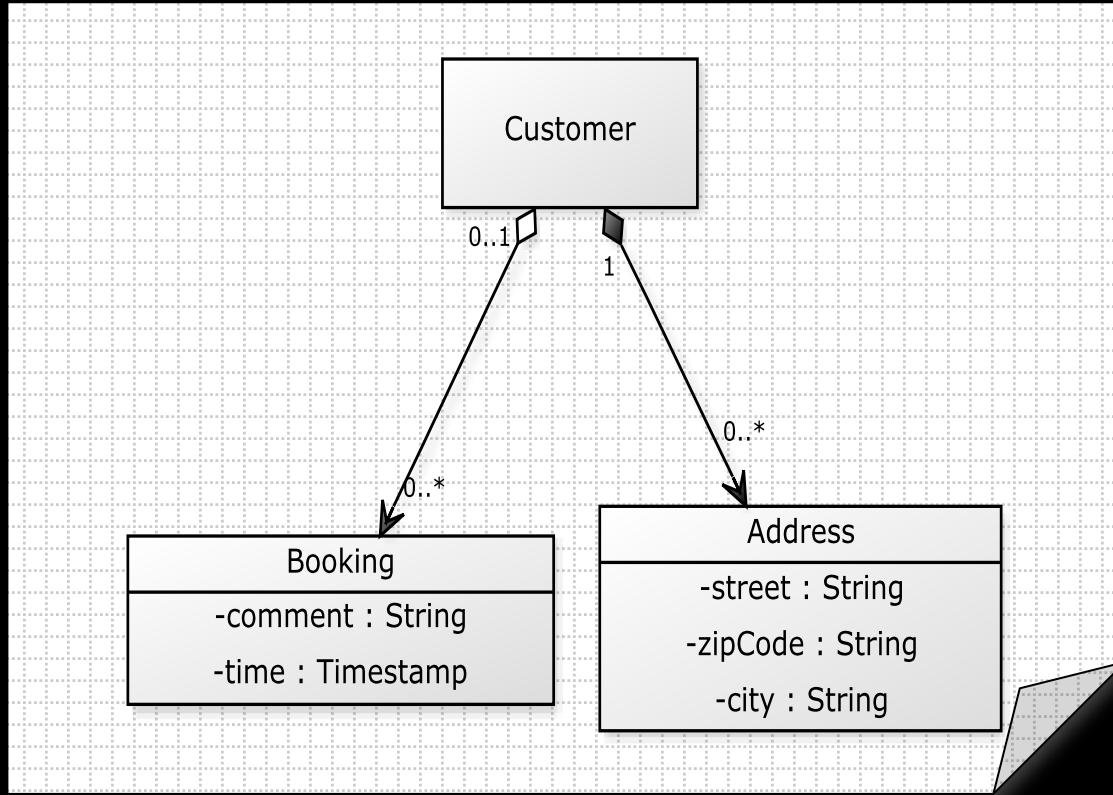
TINGLYSNINGSOBJEKT	
DTYPE	VARCHAR(255)
ID	DECIMAL(19,0)
OPPRETTET	TIMESTAMP
OPTLOCKVERSION	DECIMAL(19,0)
SISTOPDATERET	TIMESTAMP
UUID	CHAR(36)
KONVERTERET	DECIMAL(1,0)
AJOURFØRINGSdato	DATE
FÆLLESAREAL	DECIMAL(1,0)



Java code



Means we go from this...



```
package dk.tigerteam.mdsd.demo.model.internal;

@Entity
@Table(name = "Customer")
public class Customer extends AbstractEntity {
    private static final long serialVersionUID = 2098912667L;

    @Basic
    @Column(name = "name", nullable = false)
    private String name;

    @OneToOne(cascade = {
        CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH},
        fetch = FetchType.LAZY)
    @JoinColumn(name = "addressId")
    @NotNull
    private dk.tigerteam.mdsd.demo.model.internal.Address address;

    @OneToMany(cascade = {
        CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH},
        targetEntity = Booking.class, mappedBy = "customer",
        fetch = FetchType.LAZY)
    private Set<Booking> bookingCollection =
        new java.util.HashSet<Booking>();

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
}
```

```
package dk.tigerteam.mdsd.demo.mode.internal;

@Entity
@Table(name = "Booking")
public class Booking extends AbstractEntity {
    private static final long serialVersionUID = 170080605L;

    @Basic
    @Column(name = "comment", nullable = false)
    private String comment;

    @Basic
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "time", nullable = false)
    private java.util.Date time;

    @Basic
    @Column(name = "timeslot", nullable = false)
    private int timeslot;

    @ManyToOne(cascade = {
        CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH},
        fetch = FetchType.LAZY)
    @JoinColumn(nullable = false, name = "customerId")
    private Customer customer;

    public String getComment() {
        return comment;
    }

    public void setComment(String parameter) {
        this.comment = parameter;
    }

    public java.util.Date getTime() {
        return time;
    }

    ...
    ...
    ...
    ...
}
```

To this...

```
@Entity
@Table(name = "Address")
public class Address extends AbstractEntity {
    private static final long serialVersionUID = 1697028161L;

    @Basic
    @Column(name = "street", nullable = false)
    private String street;

    @Basic
    @Column(name = "zipCode", nullable = false)
    private String zipCode;

    @Basic
    @Column(name = "city", nullable = false)
    private String city;

    public String getStreet() {
        return street;
    }

    public void setStreet(String parameter) {
        this.street = parameter;
    }

    ...
    ...
}
```

This works fairly well, until...



- We get tired of writing the same tedious code by hand
- We suddenly realize that:
 - Our assumptions didn't hold up and we need to change many of our mappings
 - We need to write a lot of test code to ensure that our mappings are correct
 - We're writing a lot of technical code and very little business code

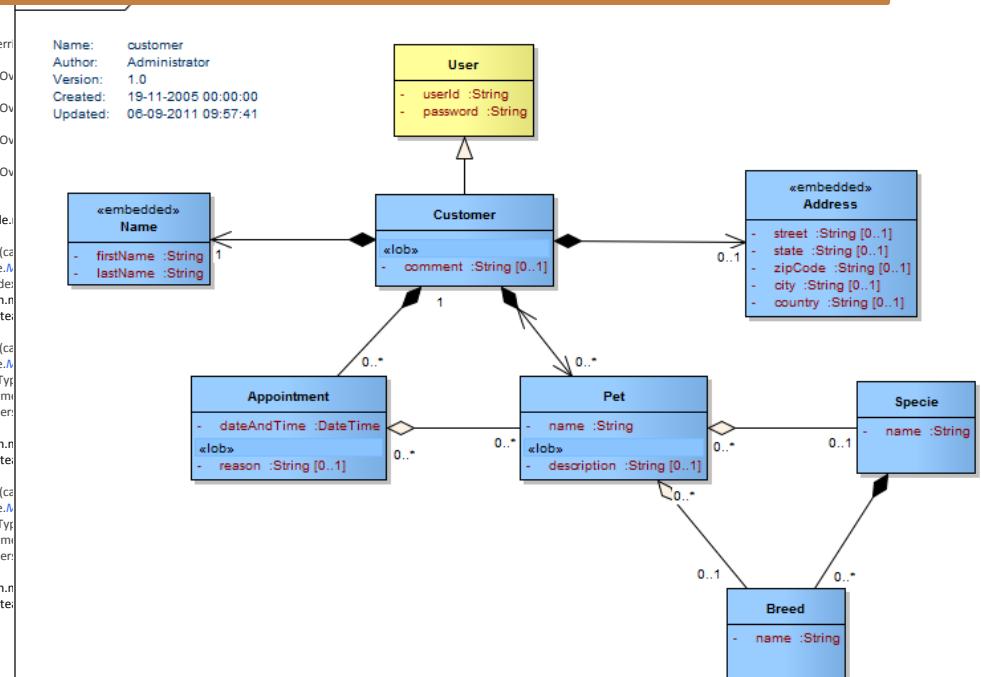
The line between **modeling** og
muddling in hand written code is
very thin



Can't see the forest for the trees?

```
dk.tigerteam.mddexample.model.customer
  ► Address.java
  ► Appointment.java
  ► Breed.java
  ► Customer.java
    ▼ Customer
      ◉ metaType
      ◉ serialVersionUID
      □ address
      □ appointmentCollection
      □ comment
      □ invoiceCollection
      □ name
      □ petCollection
      ● addToAppointmentCollection(Appointment...)
      ● addToInvoiceCollection(Invoice...) : Customer
      ● addToPetCollection(Pet...) : Customer
      ● addToRoleCollection(Role...) : Customer
      ● getAddress() : Address
      ● getAppointmentCollection() : Set<Appointment>
      ● getComment() : String
      ● getInvoiceCollection() : Set<Invoice>
      ● getMetaType() : RuntimeMetaClazz
      ● getName() : Name
    ► getPetCollection() : Set<Pet>
      ● setAddress(Address) : void
      ● setAppointmentCollection(Set<Appointment>)
      ● setComment(String) : void
      ● setInvoiceCollection(Set<Invoice>) : void
      ● setName(Name) : void
      ● withAddress(Address) : Customer
      ● withComment(String) : Customer
      ● withName(Name) : Customer
      ● withPassword(String) : Customer
      ● withUserId(String) : Customer
      ● withVeterinarian(Veterinarian) : Customer
```

When all you wanted to
convey was this



Hand held consistency...



```

@Entity
public class Customer {
    @OneToMany(
        targetEntity = Pet.class,
        mappedBy = "customer")
    private Set<Pet> petCollection = new HashSet<Pet>();

    public Set<Pet> getPetCollection() {
        return new OneToManySetWrapper<Customer, Pet>(this, petCollection) {
            @Override
            protected Customer getOneSideObjectInManySideObject(Pet manySideObject) {
                return manySideObject.getCustomer();
            }

            @Override
            protected void setOneSideObjectInManySideObject(Pet manySideObject,
                Customer oneSideObject) {
                manySideObject.setCustomer(oneSideObject);
            }
        };
    }
}
  
```

```

@Entity
public class Pet {
    @ManyToOne
    private Customer customer;

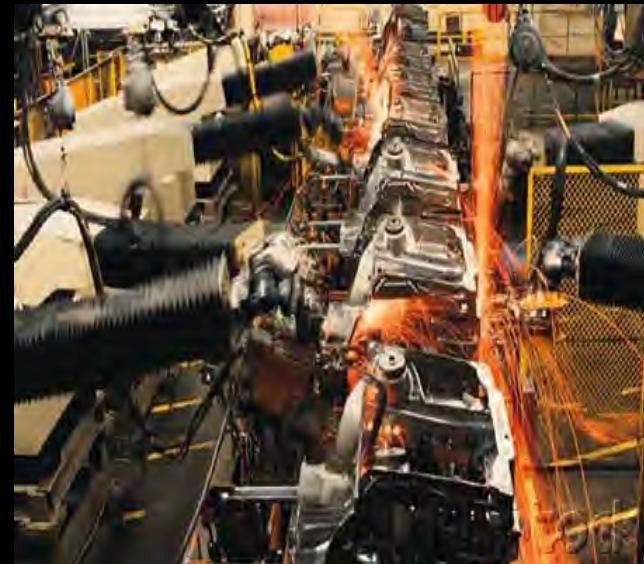
    public void setCustomer(Customer parameter) {
        new ManyToOneWrapper<Customer, Pet>(this) {
            @Override
            protected void addManySideObjectToOneSideCollection(Customer oneSide, Pet manySide) {
                ((WrappedSet<Pet>) oneSide.getPetCollection()).getWrappedCollection().add(manySide);
            }
            @Override
            protected void removeManySideObjectFromOneSideCollection(Customer oneSide, Pet manySide) {
                ((WrappedSet<Pet>) oneSide.getPetCollection()).getWrappedCollection().remove(manySide);
            }

            @Override
            protected Customer getOneSideObjectInManySideObject(Pet manySide) {
                return manySide.customer;
            }

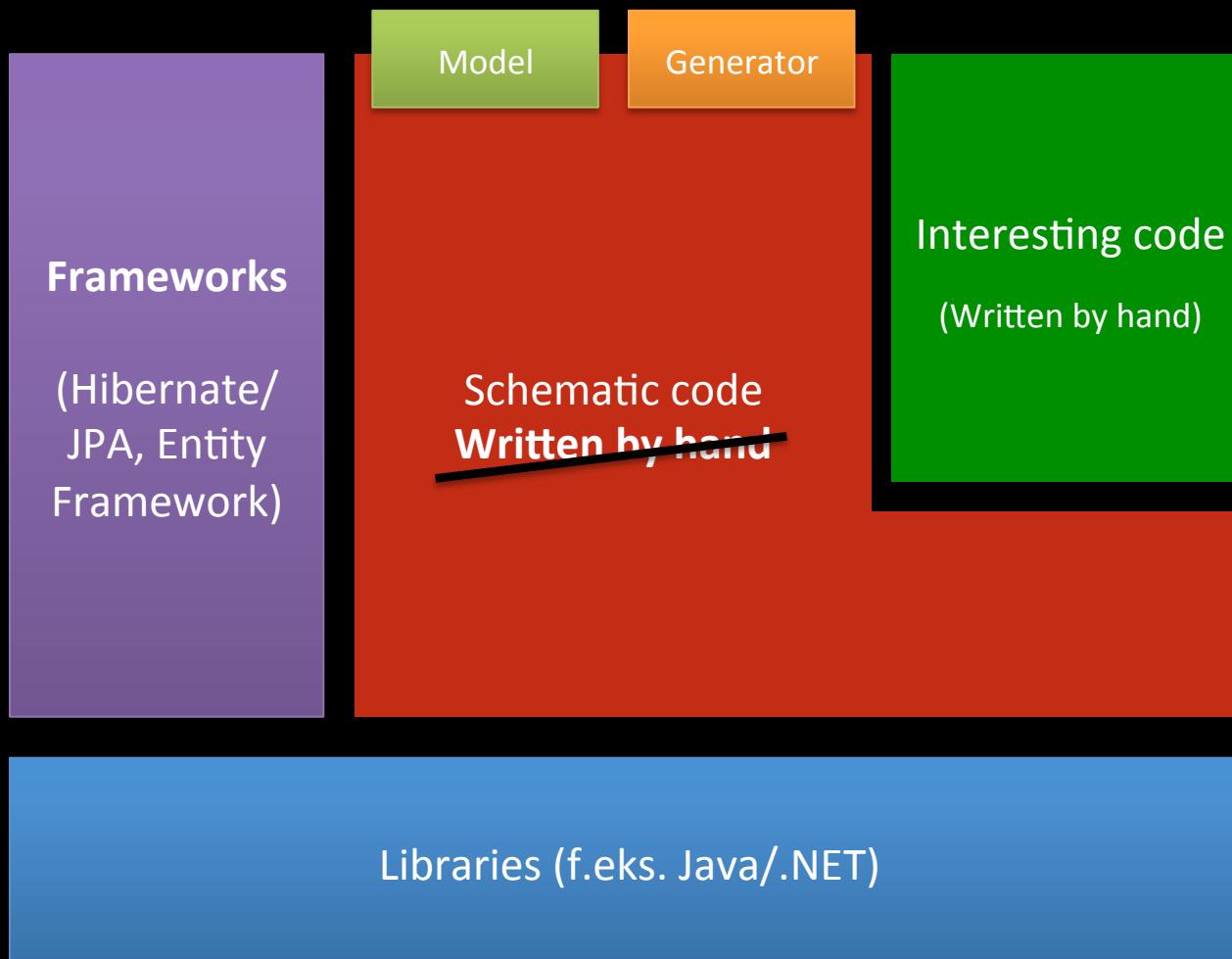
            @Override
            protected void setOneSideObjectInManySideObject(Pet manySide, Customer oneSide) {
                manySide.customer = oneSide;
            }
        }.updateOneSideObject(parameter);
    }
}
  
```

Automation

From sweatshops to automation



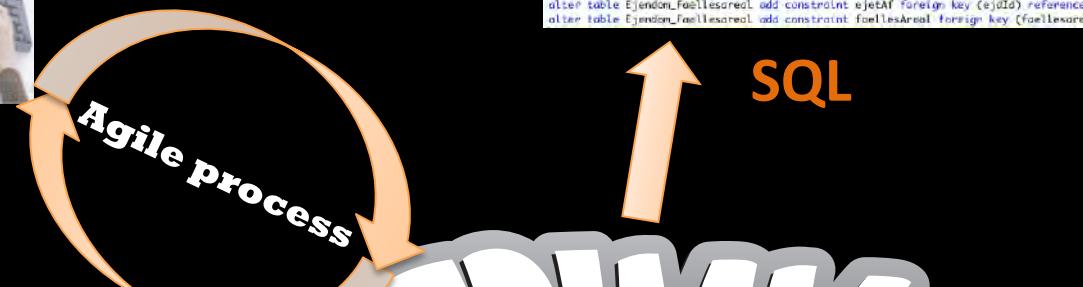
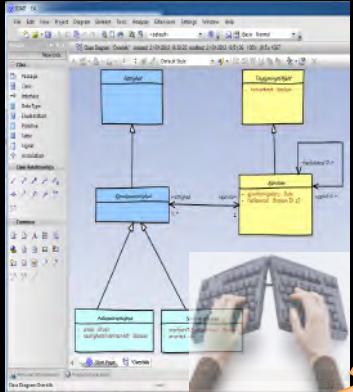
The end of writing tedious code by hand



At ETL we introduced this process

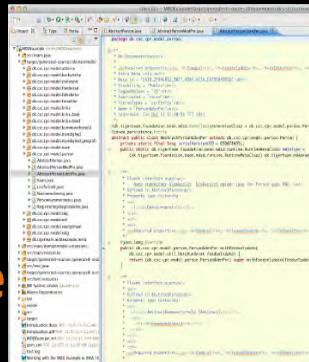


UML class diagrams

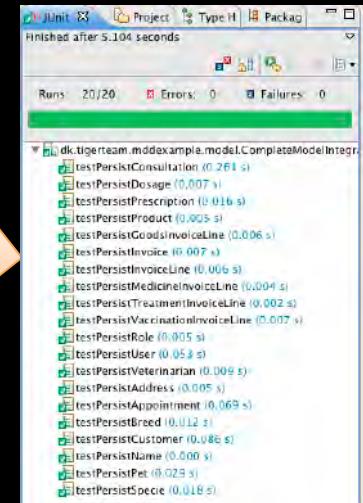
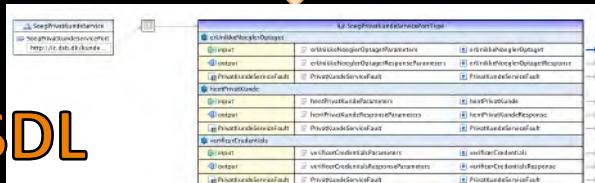


```
create table TinglysningsObjekt (objektId varchar(255) char not null, id number(19,0) not null,
oprettet timestamp, oplockVersion number(19,0) not null,
sidsOpdateret timestamp, vindtId char(36) unique,
konverteret number(1,0) not null, ejaforslagsdata date,
Faellesareal number(1,0), primary key (id);
create table Rettighed (objektId varchar(255) char not null, id number(19,0) not null,
oprettet timestamp, oplockVersion number(19,0) not null,
sidsOpdateret timestamp, vindtId char(36) unique,
overforstTilFaerlagtAndel number(1,0), prioritet number(10,0),
andel_nuevne number(19,0), andel_toeller number(10,0),
raadighedsindskramket number(1,0), ejId number(19,0), primary key (id));
create table Ejendom_Foellesareal (ejIdId number(19,0) not null, FaellesarealId number(19,0) not null);
alter table Rettighed add constraint ejendom_Foellesareal foreign key (ejIdId) references TinglysningsObjekt;
alter table Ejendom_Foellesareal add constraint ejefat Foreign key (ejIdId) references TinglysningsObjekt;
alter table Ejendom_Foellesareal add constraint fællesareal foreign key (foellesarealId) references TinglysningsObjekt;
```

SQL



WSDL

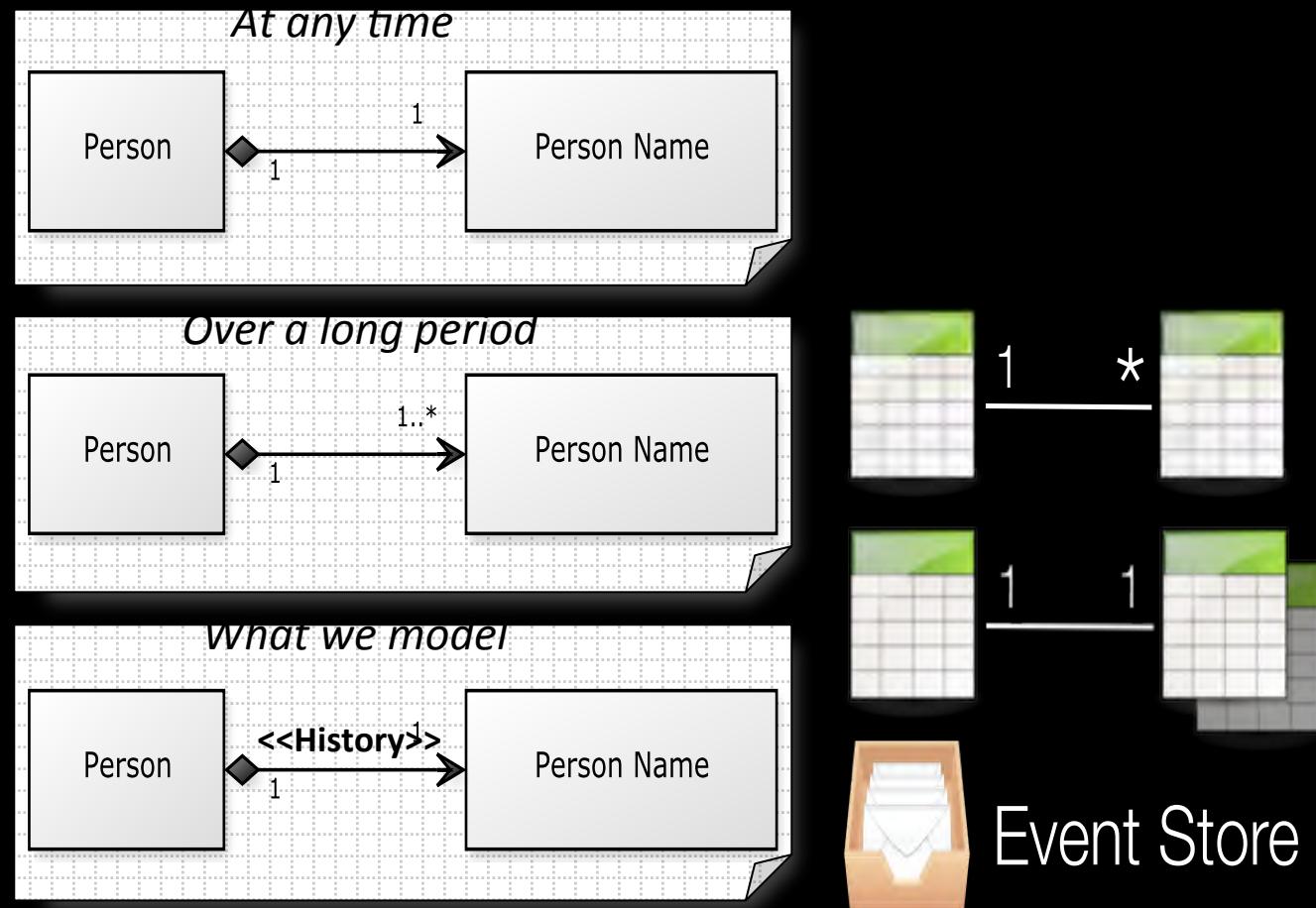


Tests

Java code
Hibernate

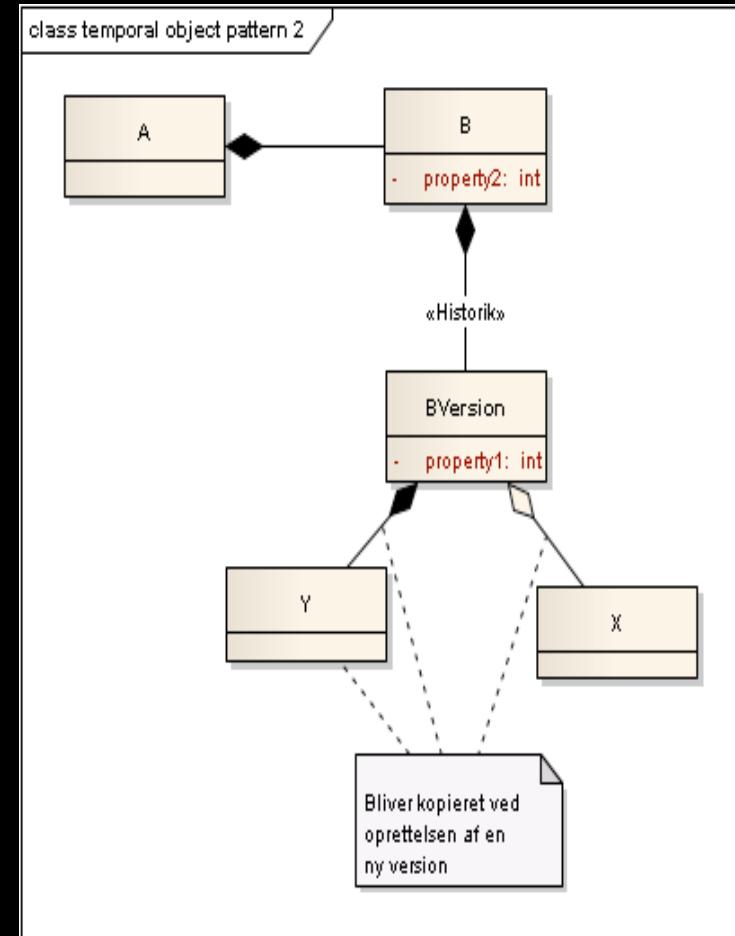
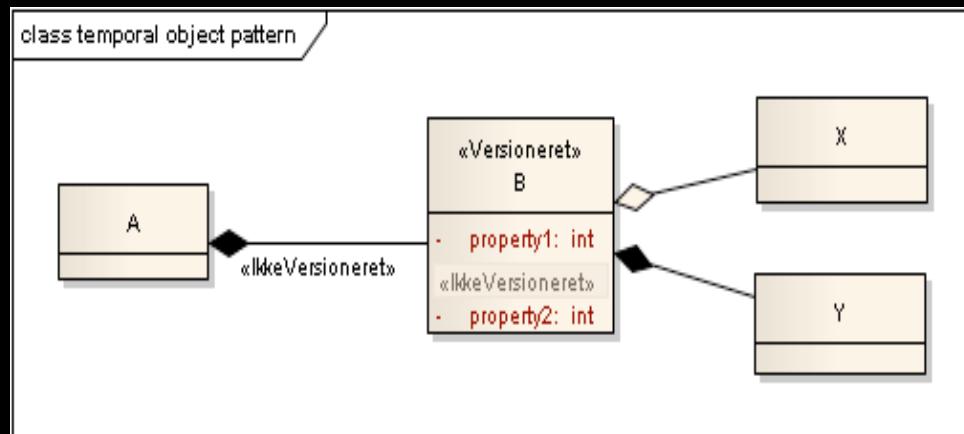
Higher abstraction level

Bi-temporal history

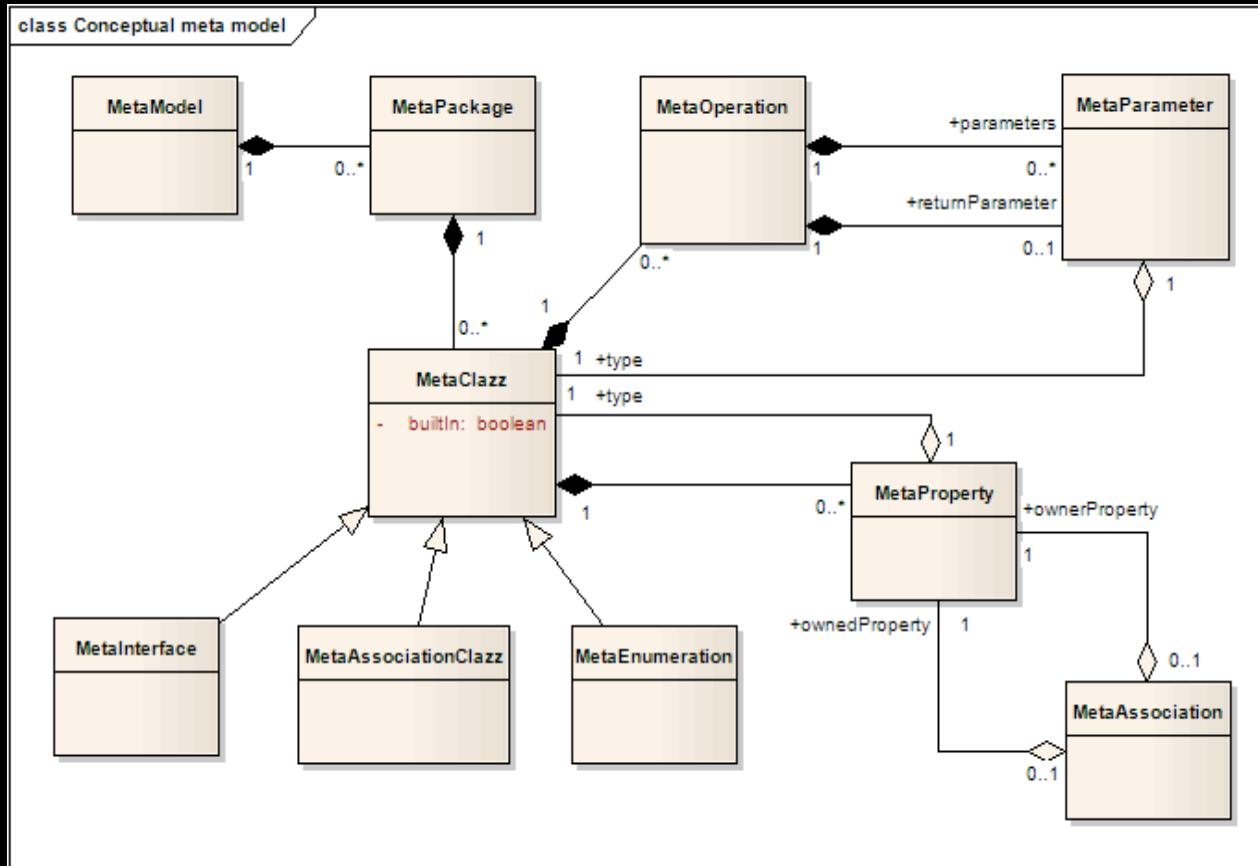


Gives us the freedom
to chose the right
implementation
without revealing it in the
model

Versioning (Temporal Object Pattern)



The Core of Flexible Code Generator



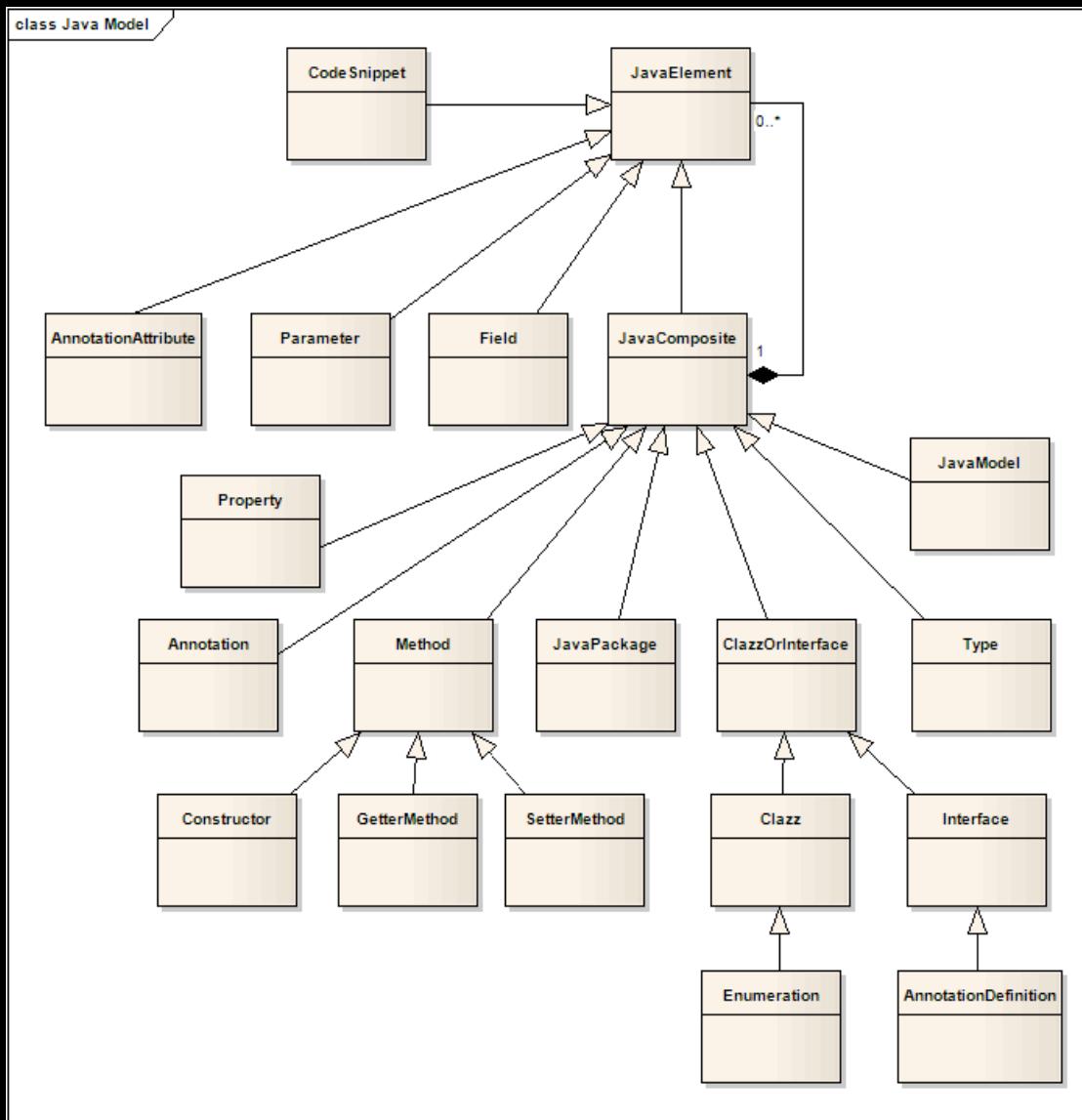
```
XmiReader reader = new EAXmiReader();
XmiReader reader = new MagicDrawXmiReader();
MetaModel metaModel = reader.read("model.xml");
```

Model Transformation

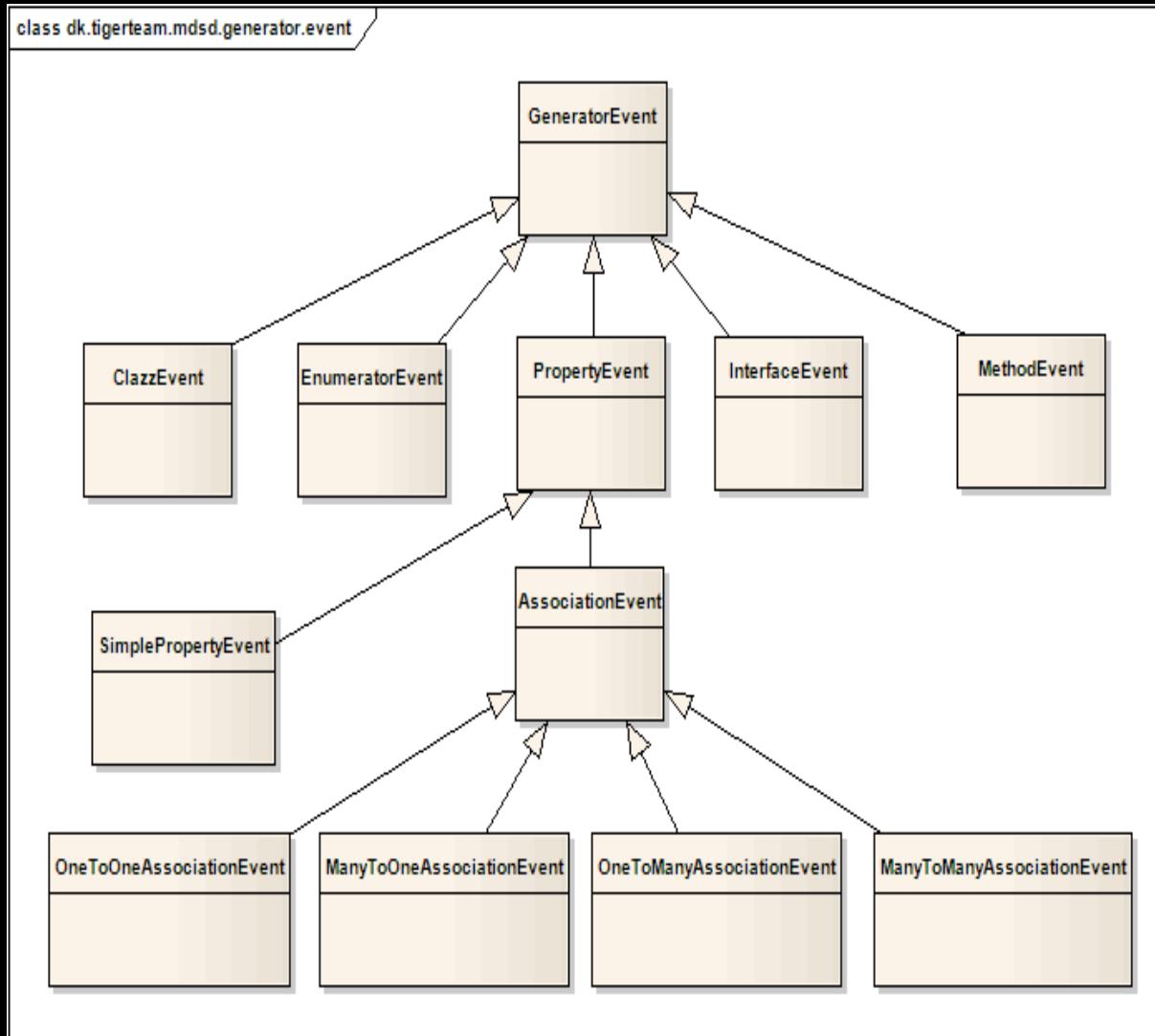
```
JavaGenerator javaGenerator = new JavaGenerator();  
List<ClazzOrInterface> allGeneratedClazzes = javaGenerator.execute(metaModel);
```

Meta Type	Java Model
MetaPackage	JavaPackage
MetaClazz	Clazz
MetaAssociationClazz	Clazz
MetaEnumeration	Enumeration
MetaInterface	Interface
MetaProperty	Property (Består af Field, GetterMethod og SetterMethod)
MetaOperation	Method

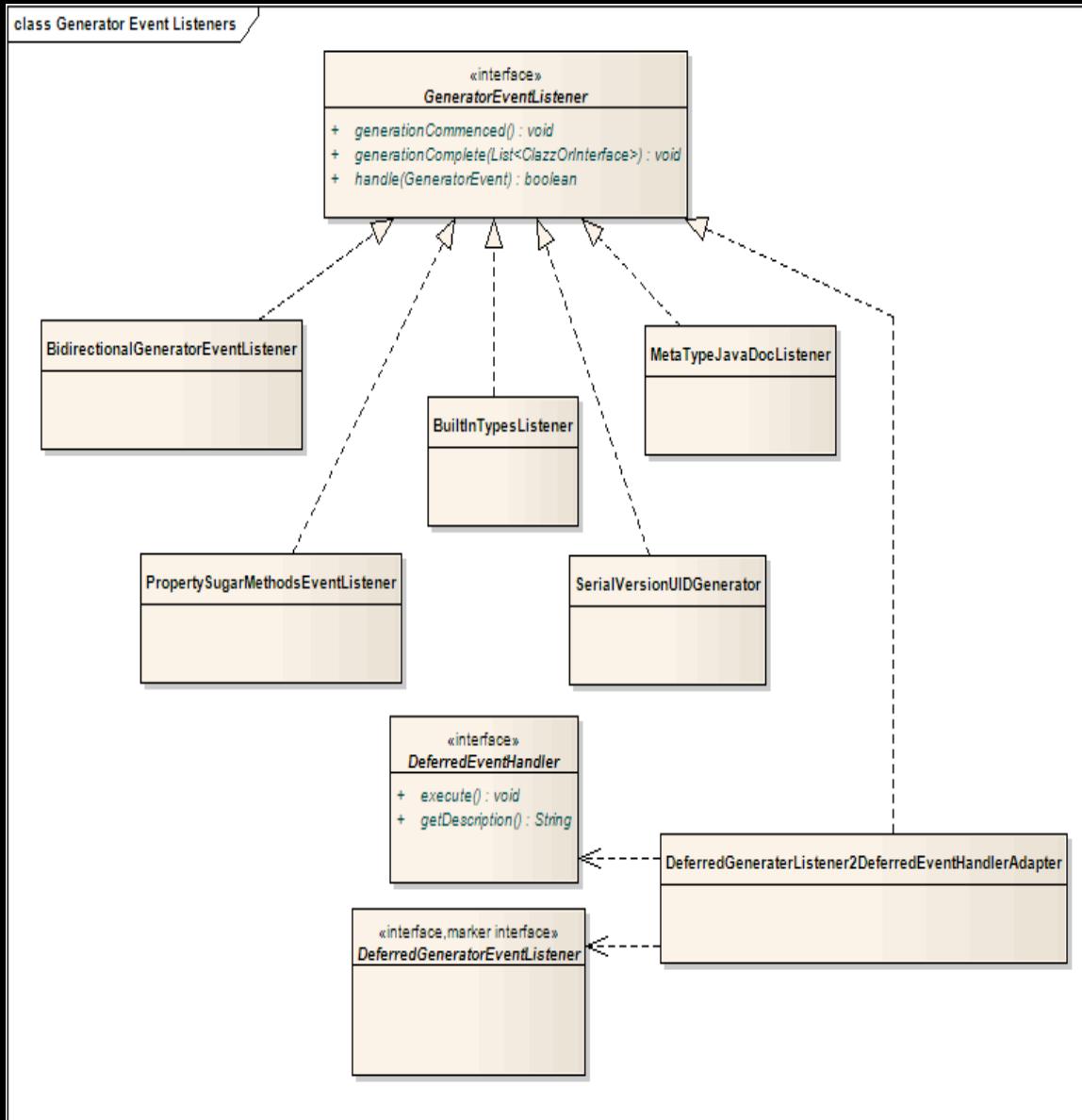
Code DOM



An Event Based Extension Model



Extensions



Example Extensions

- Built-in Types
- Bidirectional associations
- Property Sugar methods
- Get Or New Property methods
- Constructor (immutable properties)
- Class Hierarchy Java doc generator
- Serial Version UID generator
- MetaType Java doc generator
- Serializable Pojo's
- ToString/Equals/HashCode
- JPA Field based persistence
- JPA Named Tables and Columns
- JPA OptimisticLocking exceptions
- Hibernate Foreignkey Constraints
- Hibernate Foreignkey Index
- Hibernate Fetch Optimization
- Hibernate Association Unproxying
- Hibernate Table Comments
- Hibernate HH-3544 bug fix
- Temporal logic (Audit/History)
- Groovy
- GORM
- Envers
- Mixins

Example Extension

```
@OneToMany  
@Cascade(org.hibernate.annotations.CascadeType.DELETE_ORPHAN)  
private Set<Tire> tires;  
  
public class HibernateDeleteOrphanListener extends BaseJpaGeneratorEventListener {  
    @Override  
    protected boolean handleOneToManyOwnerOfAssociation(OneToManyAssociationEvent event) {  
        if (isDeleteOrphanCandidate(event)) {  
            event.getProperty().getField().addAnnotations(  
                new Annotation(Cascade.class).addAnnotationAttribute("value", CascadeType.DELETE_ORPHAN)  
            );  
            event.getProperty().removeSetterMethod();  
        }  
        return true;  
    }  
  
    protected boolean isDeleteOrphanCandidate(OneToManyAssociationEvent event) {  
        ...  
    }  
}
```

Example Extension - continued

```
protected boolean isDeleteOrphanCandidate(OneToManyAssociationEvent event) {  
    if (event.getMetaProperty().isOwnerOfAssociation() &&  
        !event.getMetaProperty().getAssociation().isBidirectional() &&  
        !event.getMetaProperty().getAssociation().isSelfReferencing()) {  
        // Check the clazz of the opposite property to see what kind of associations it has  
        for (MetaProperty subMetaProperty : event.getMetaProperty().getType().getProperties()) {  
            if (subMetaProperty.isPartInAnAssociation()) {  
                if (subMetaProperty.isOwnerOfAssociation()) {  
                    if (subMetaProperty.getAssociationType() == AssociationType.ManyToMany ||  
                        subMetaProperty.getAssociationType() == AssociationType.OneToOne) {  
                        return false;  
                    }  
                } else if (subMetaProperty.getAssociation().isBidirectional()) {  
                    // The type of the our sub property is not an owning association and we have  
                    // a java association in both directions (bidirectional), which hibernate doesn't handle  
                    return false;  
                }  
            }  
        }  
        return true;  
    }  
    return false;  
}
```

What about Domain Logic?

- It's important to separate **Entity Logic** and **Use Case Logic**
- **Use Case Logic** doesn't belong in Entities (violates coherence)

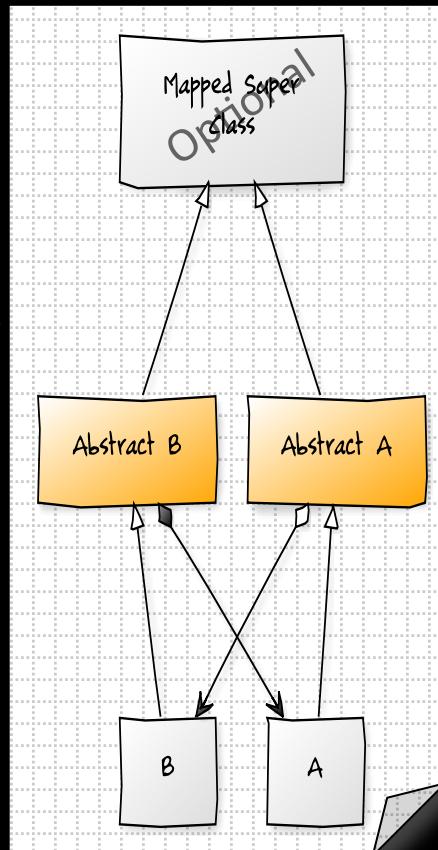
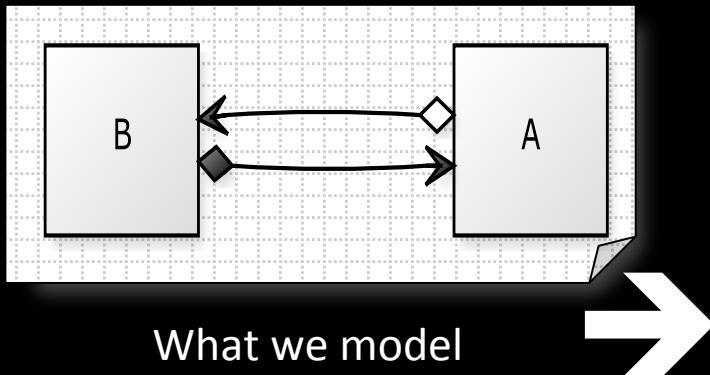
Domain Modeling

**Domain Object design should focus
on WHAT the SYSTEM IS
NOT on what the system DOES**

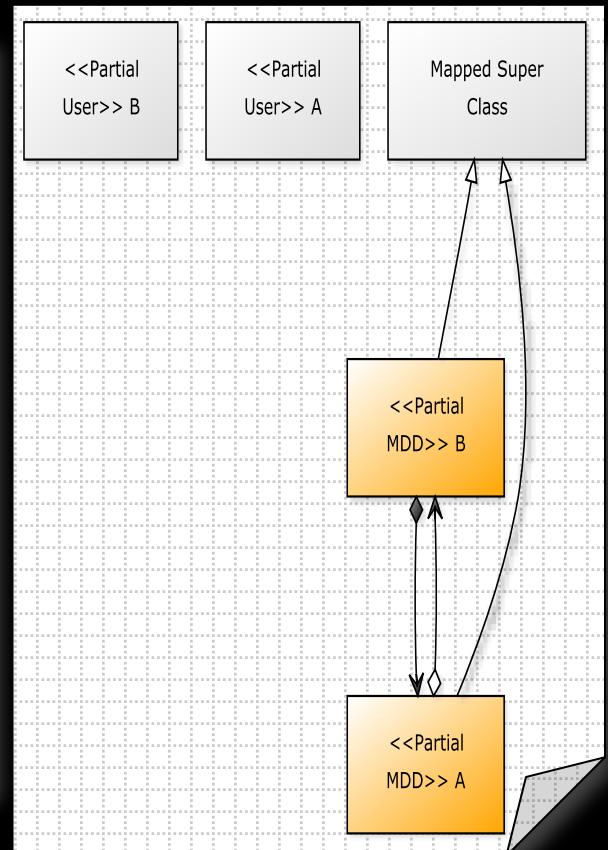
James Coplien – DCI Talk at Öredev 2009

Domain Logic in Entities

We have several options with regards to Entity Logic



3 level inheritance



Partial classes

Alternatives:

- Mixins / Traits
- Extension Methods
- Privileged Aspects
- Protected Regions

DCI – A different approach to handling logic

- Allows us to separate **Form**
 - What the System IS – AKA. The domain mode
- from **Structure**
 - What the System DOES
- Identifying that Form changes much slower than the Structure

Use Case Logic

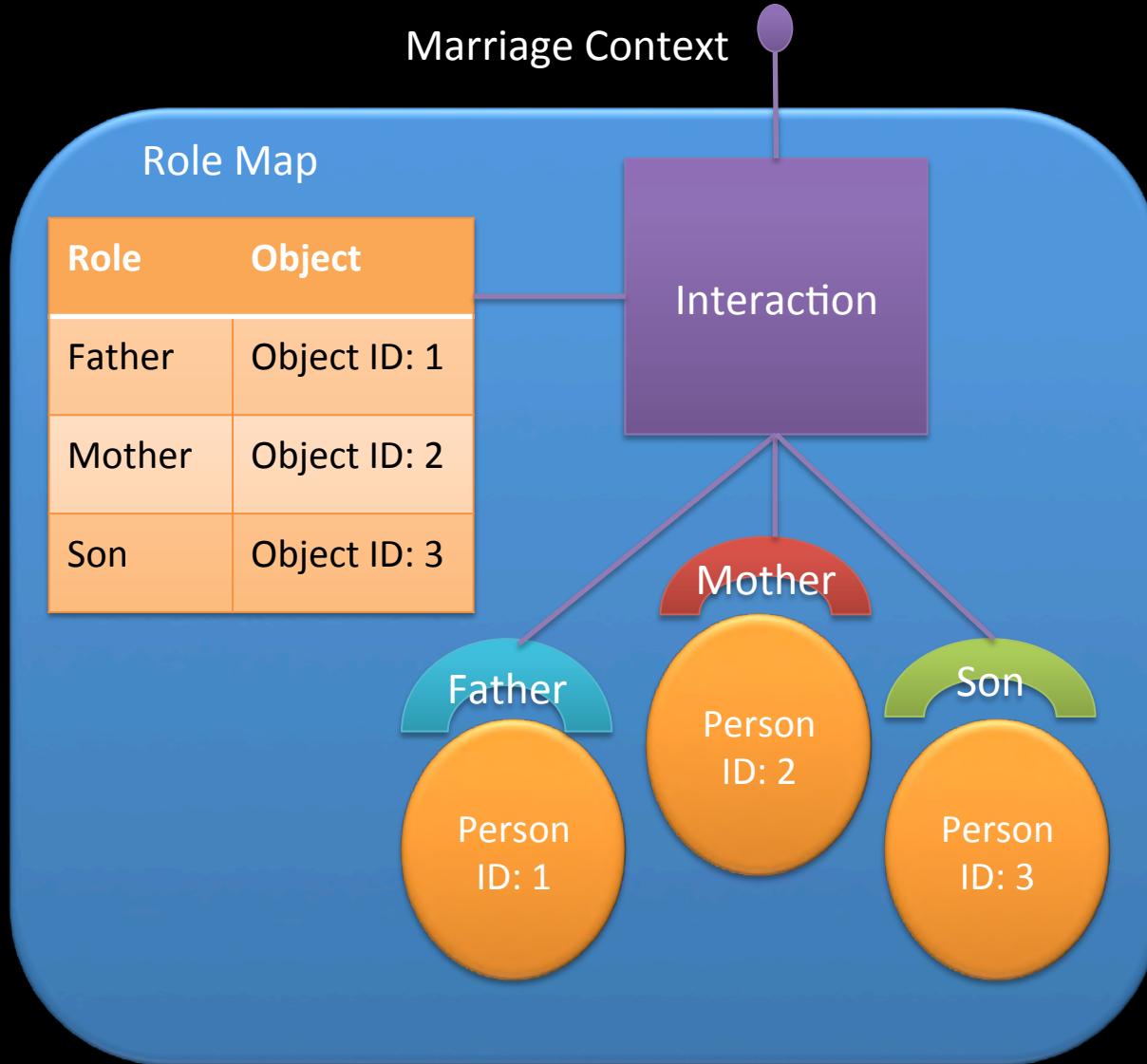


- This type of logic spans Domain Objects.
- Data Context Interaction (DCI) offers a very flexible way of handling this complexity, by allowing Domain Objects to **play** different **Roles** within different **Contexts** (Use cases)

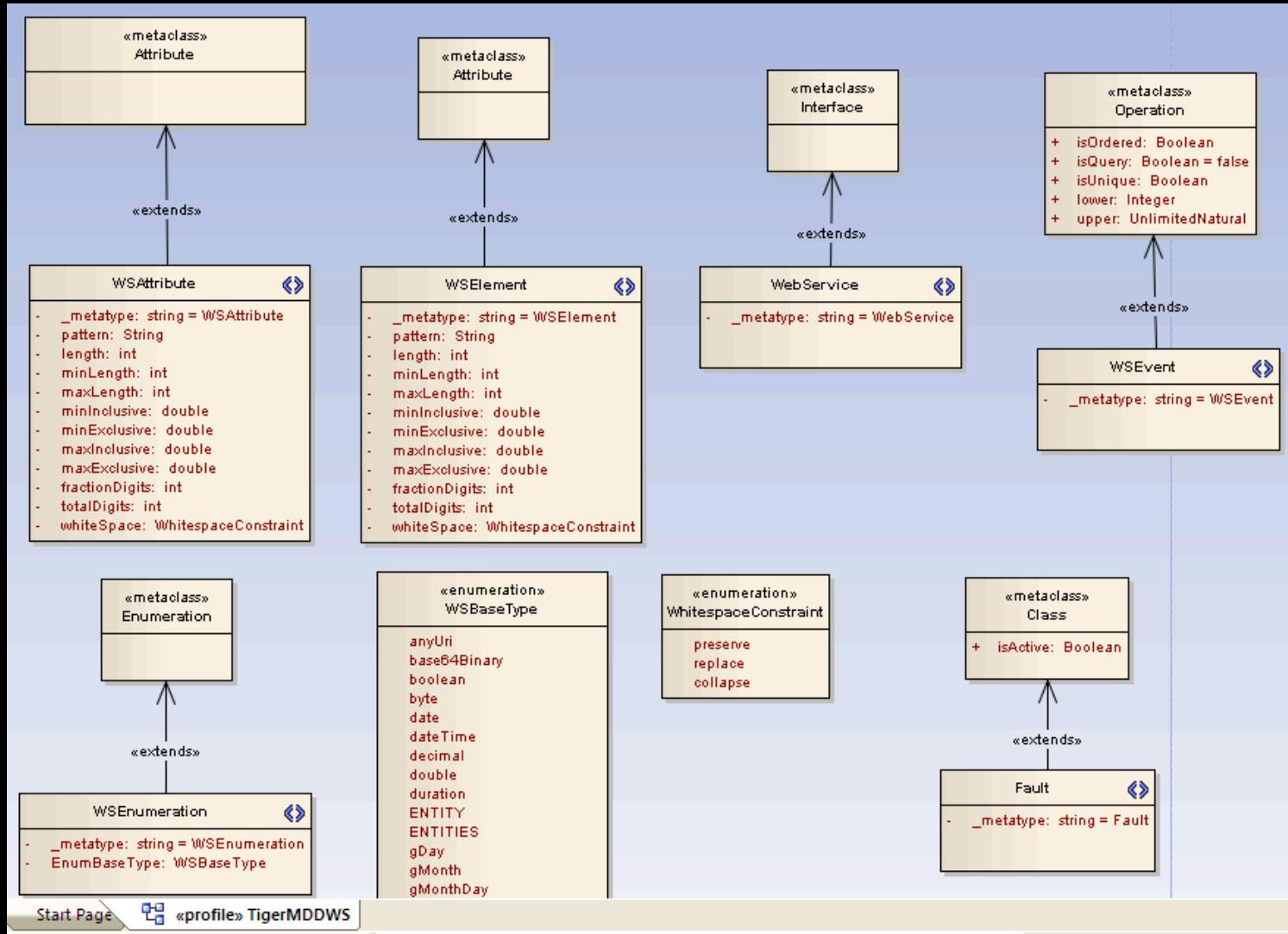
DCI

- Separating Structure from Form is done using Roles, which are (typically) played by Domain Objects
- Mixins/Traits/Partial Classes/Extension Methods are used to enhance Domain Objects with extra functionality (Role Methods) within a Context

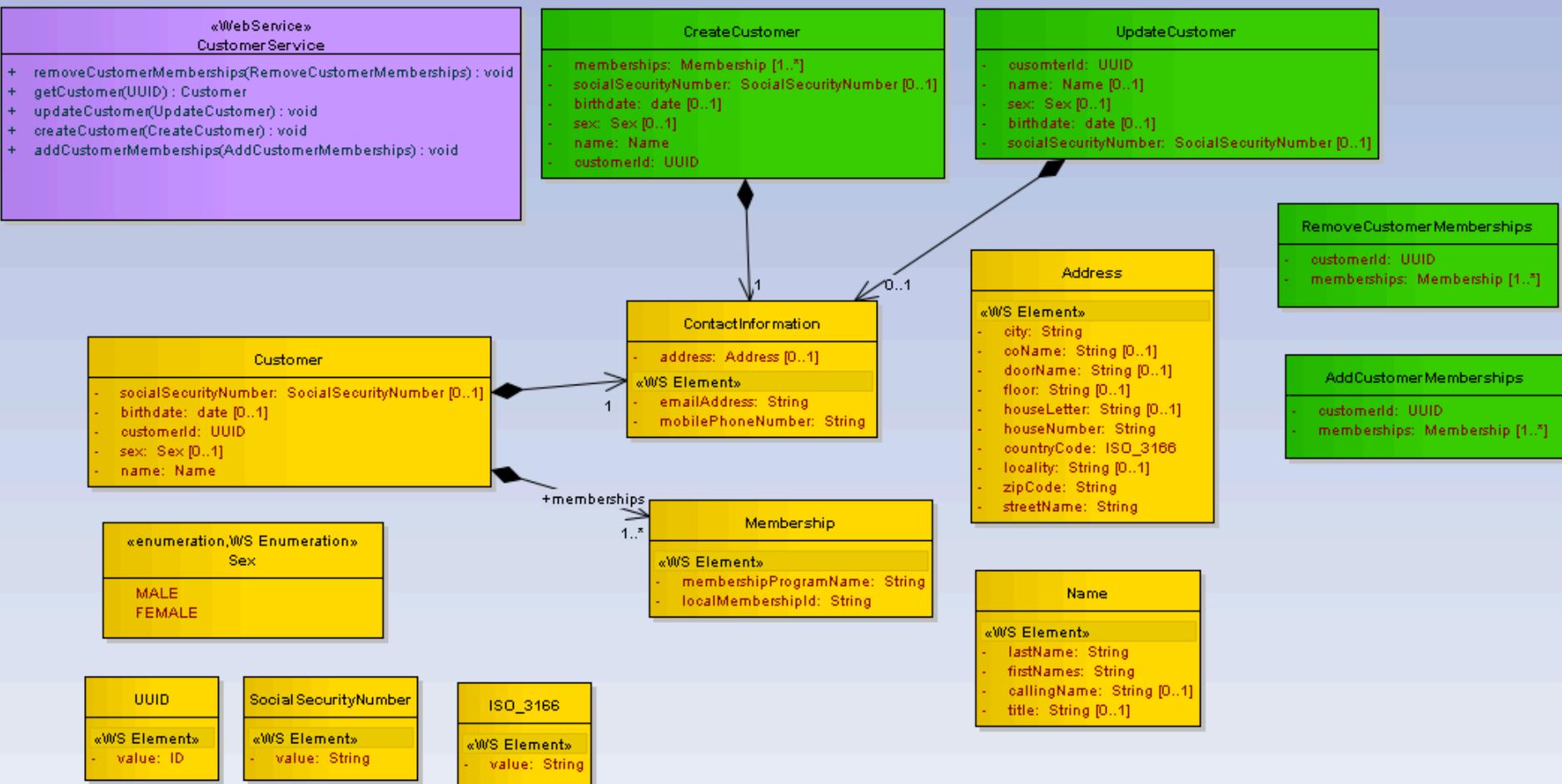
DCI



Create your own UML language

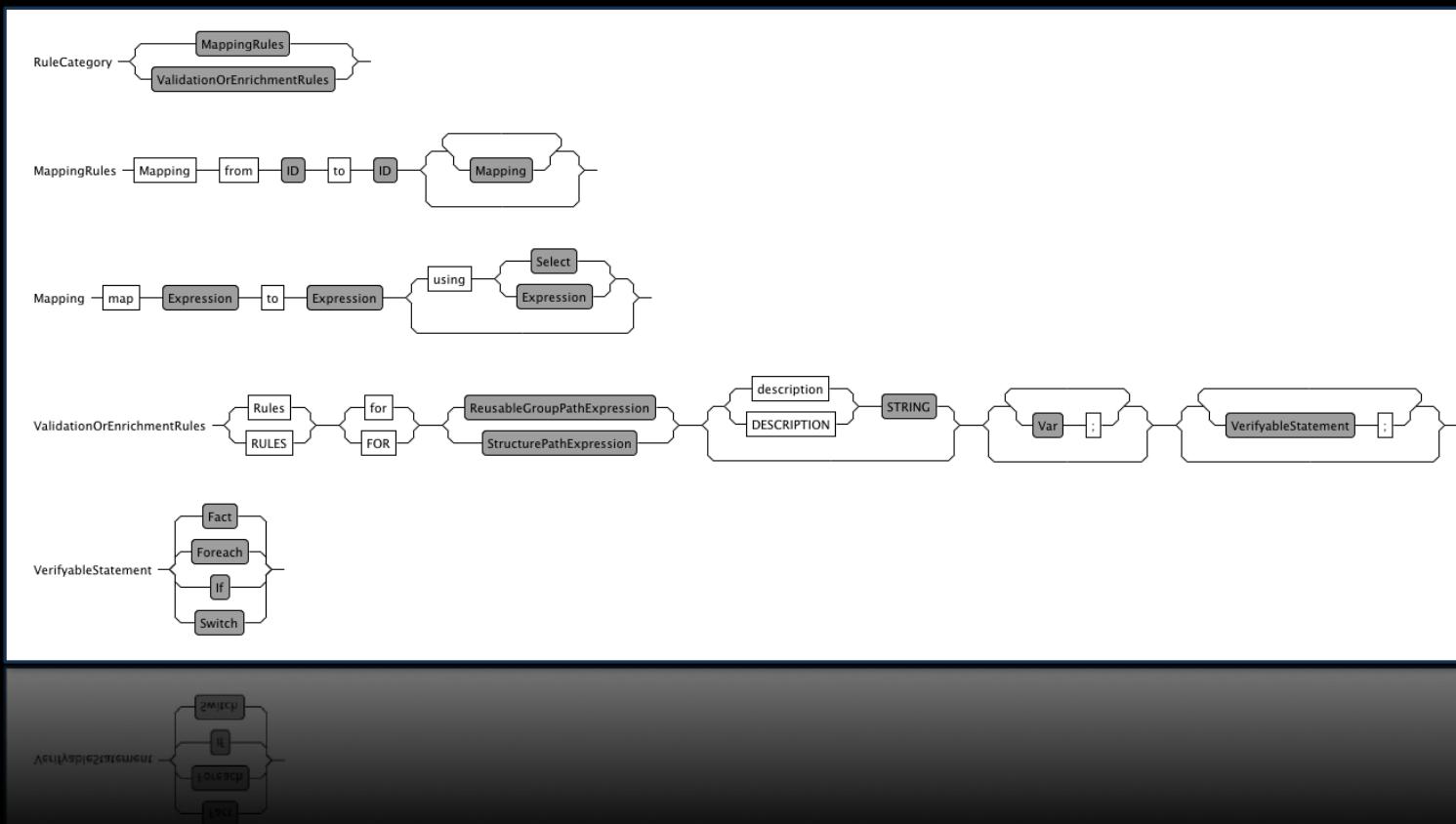


WebService model



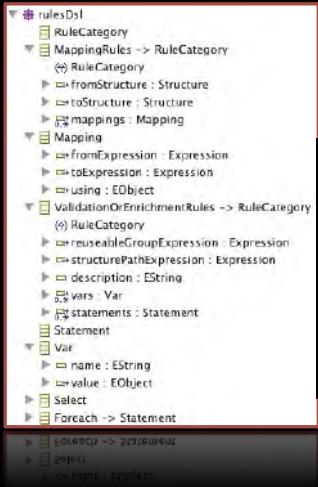
Another possibility

Build your own standalone language

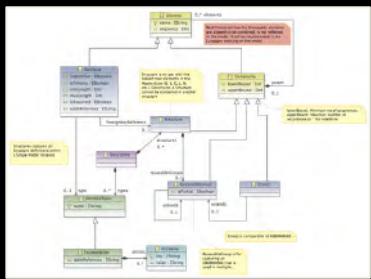


Sounds hard, but it's quite easy 😊

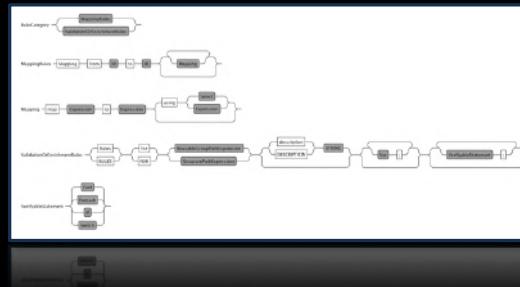
Rule Language Meta Model



Data Language Meta Model



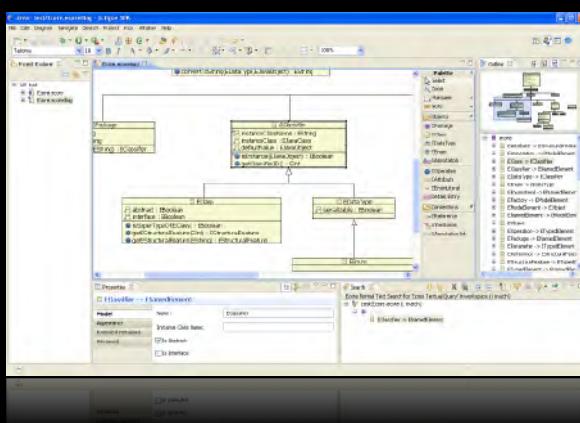
Rule Grammar (using Xtext)



A screenshot of a text editor showing mapping rules in a plain text format. The code includes mappings from X to Y, such as `map #X.COMPANY_ID to #N0_Y.CORPORATE_ID.G_KEY`, and various map statements for different fields like Address, Corporate_ID, and Corporate_Info.

Text Editor

Editor & IDE



On the fly code generation with Xtend2

```
class StructureDslGenerator implements IGenerator {
    @Inject extension IQualifiedNameProvider nameProvider

    override void doGenerate(Resource resource, IFileSystemAccess fsa) {
        for(e: resource.allContentsIterable.filter(typeof(Composite))) {
            fsa.generateFile(
                e.fileName,
                e.generate)
        }
    }

    def dispatch generate(Group e) ...
    «IF !e.packageName.isNullOrEmpty»package «e.packageName»;«ENDIF»

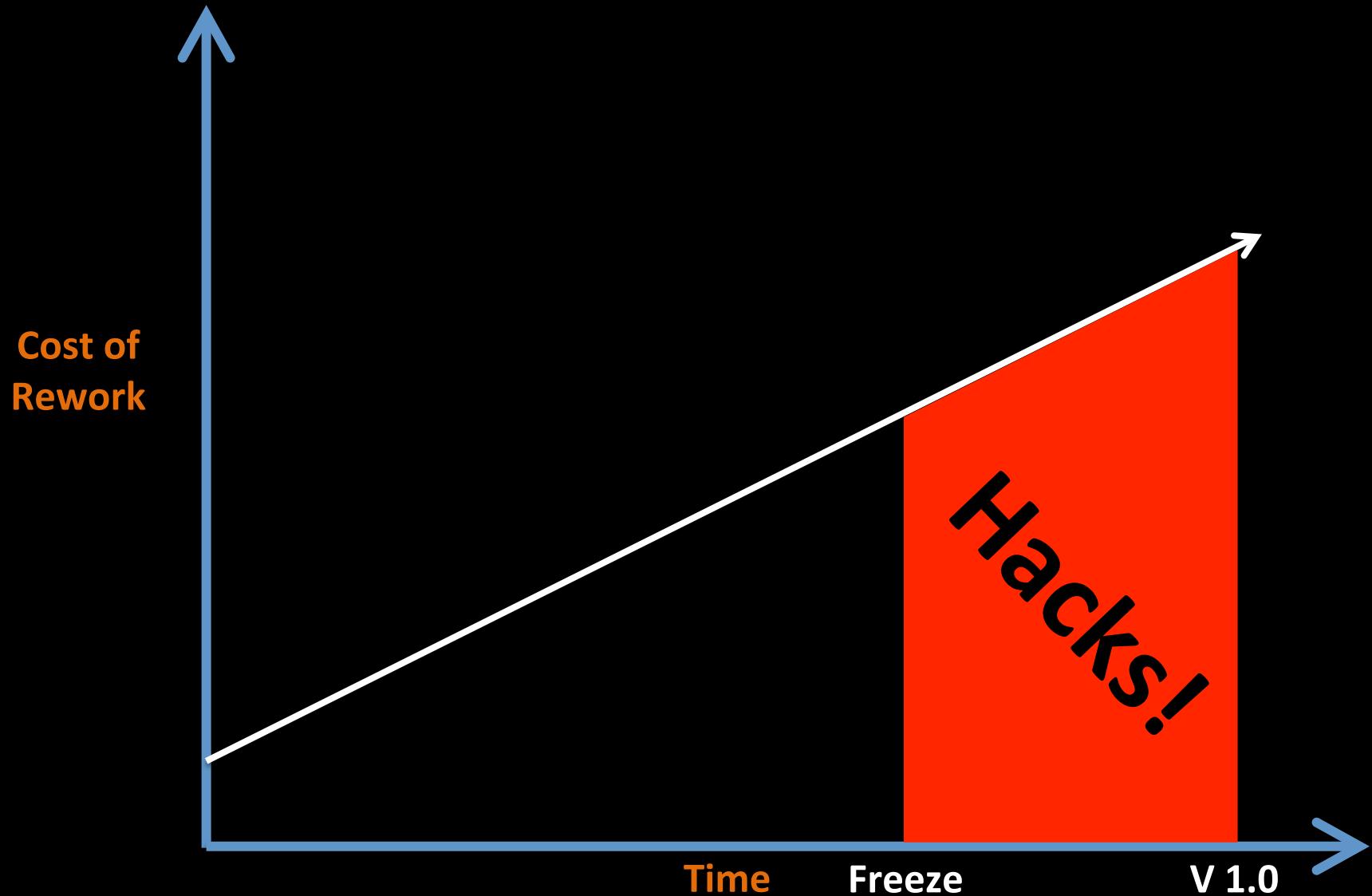
    public class «e.name» «IF e.^extends != null»extends «e.^extends.name» «ENDIF»{
        «FOR elem:e.elements»
            «generateAttributeFor(elem)»
        «ENDFOR»
    }
    ...
}

def dispatch generateAttributeFor(Attribute attr) ...
    «val attrType = resolveAttributeType(attr)»
        private «attrType» «attr.name»;

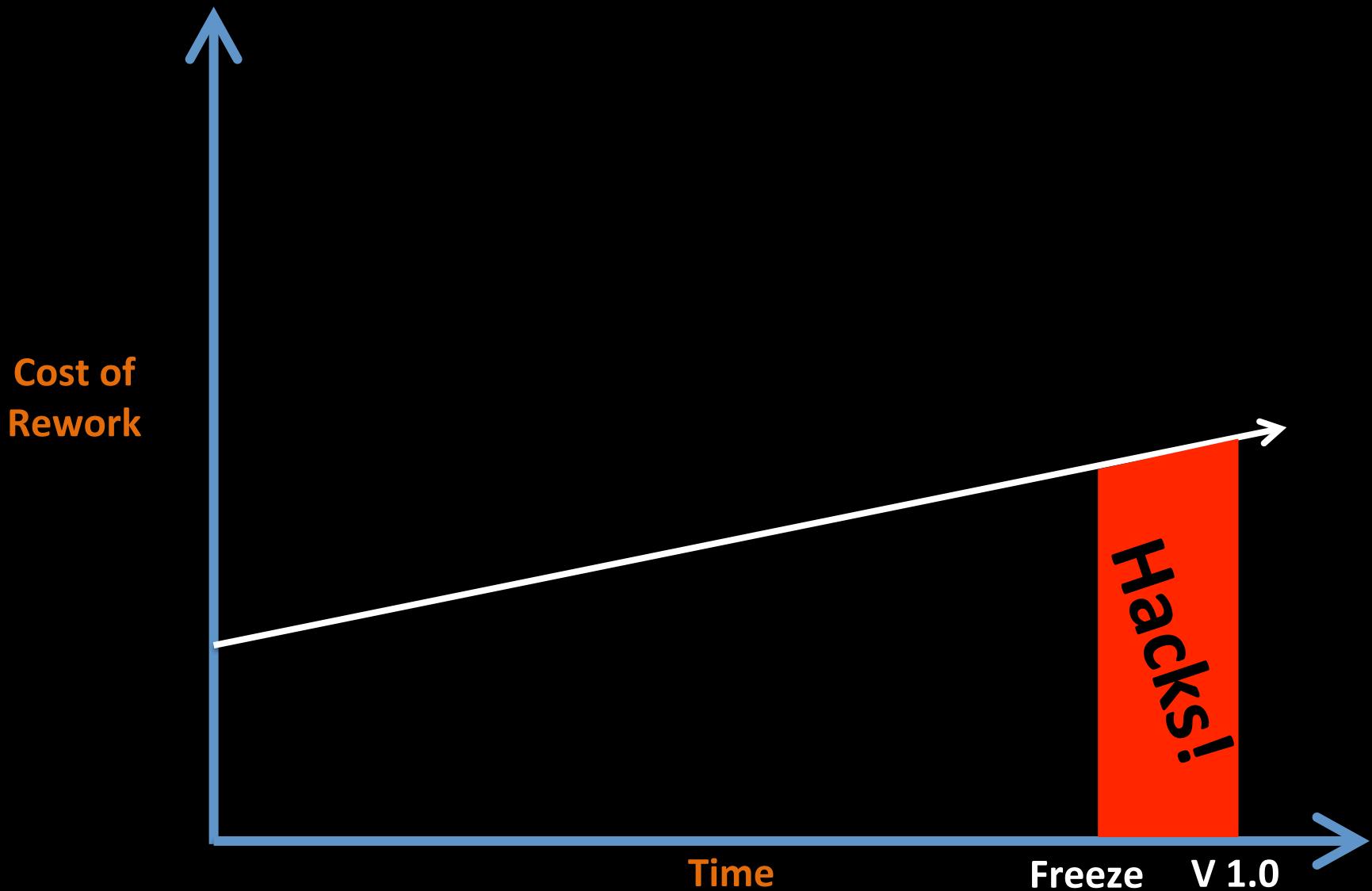
        public «attrType» get«attr.name.toFirstUpper»() {
            return «attr.name»;
        }

        public void set«attr.name.toFirstUpper»(«attrType» «attr.name») {
            this.«attr.name» = «attr.name»;
        }
    ...
}
```

Point of No Return, traditionally



Point of No Return, Model Driven



Positive Side-effects of MDD

- Better and more coherent quality
- Model Documentation is always up-to-date
- Closer to the customer – speak the same language
- Fewer resources – better economy
- Removes focus from technical mumbo-jumbo
- Automation of tests
- Easier to follow progress

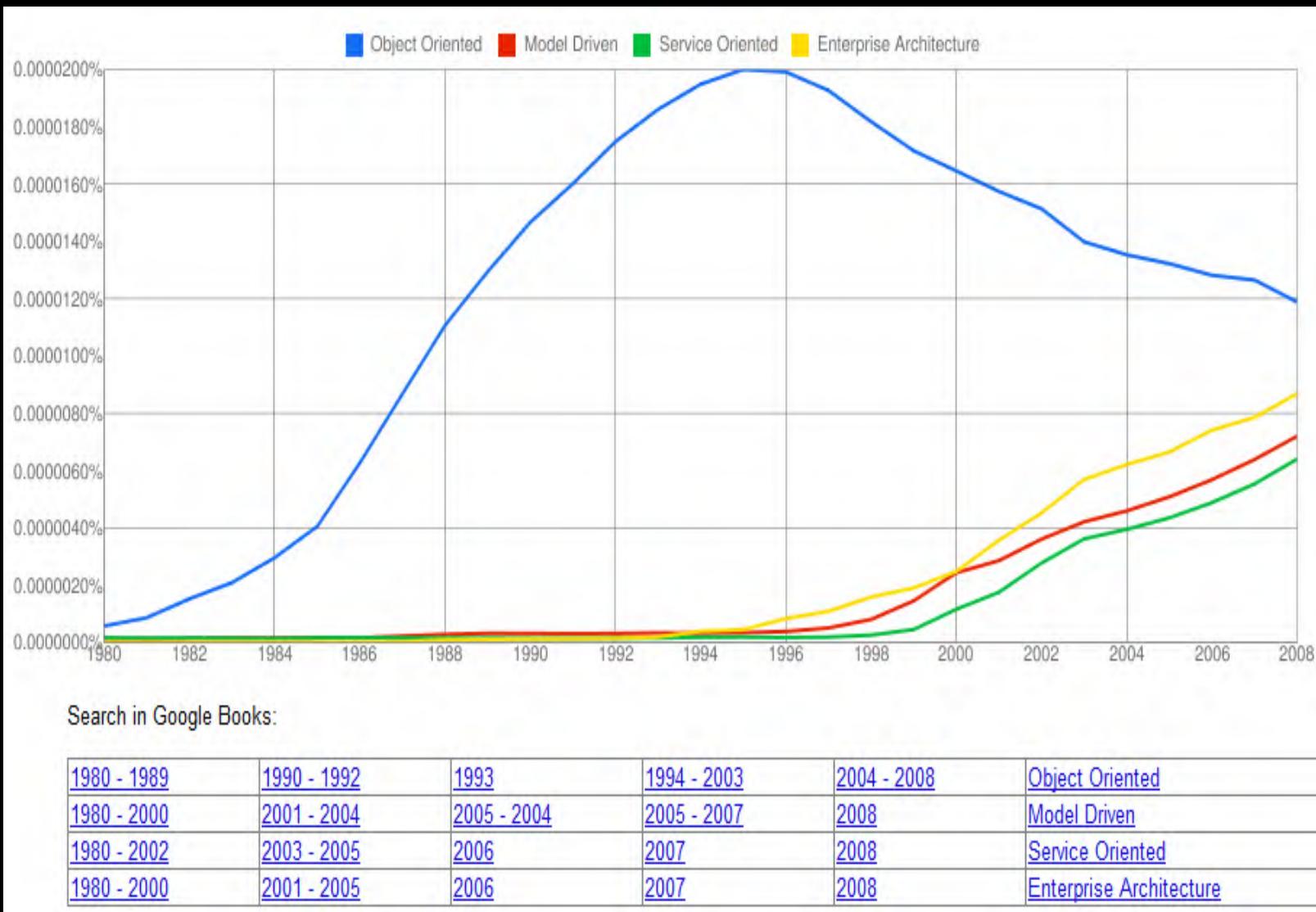
Challenges with MDD

- New way of thinking and working
- This is NOT Case Tools re-invented!
- Parallel development (Branching & Merging requires proper tool support)
- Handling new production releases requires tool support for updating databases, etc.
- The Ketchup effect – Don't forget it's iterative
- Model is KING

References

- BEC Pension
- Letpension
- Elektronisk Tinglysning (CSC)
- Miracle A/S
- PFA Pension
- DSB
- ...

A little statistics



CSC

Electronic Land registration



TIGERTEAM®

LEAN THINKING

Questions?



<http://tigerteam.dk>

@TigerTeamDK on Twitter

Useful links

CQRS/DDD:

- Bounded Contexts and Context Maps: <http://www.infoq.com/articles/ddd-contextmapping>
- Rinat Abdullings Getting Started with CQRS: <http://abdullin.com/cqrs/>
- CQRS journey: <https://github.com/msponp/cqrs-journey-doc/>
- Greg Youngs old blog: <http://codebetter.com/gregyoung/>
- Greg Youngs new blog: <http://goodenoughsoftware.net>
- Greg Youngs CQRS info site: <http://cqrss.wordpress.com>
- Sagas: <http://www.udidahan.com/2009/04/20/saga-persistence-and-event-driven-architectures/>
- Clarified CQRS: <http://www.udidahan.com/2009/12/09/clarified-cqrs/>
- When to avoid CQRS: <http://www.udidahan.com/2011/04/22/when-to-avoid-cqrs/>
- Why you should be using CQRS almost everywhere:
<http://www.udidahan.com/2011/10/02/why-you-should-be-using-cqrs-almost-everywhere.../>
- Videos from DDD exchange 2011: <http://skillsmatter.com/event/design-architecture/ddd-exchange-2011>
- Fohjin DDD example: <https://github.com/MarkNijhof/Fohjin/tree/master/Fohjin.DDD.Example>
- Axon Framework: <http://www.axonframework.org/>

MDD:

- TigerTeam MDSD/Trimm videos: & presentations: <http://www.tigerteam.dk/resources/>
- xText: <http://www.eclipse.org/Xtext/>
- Xtend: <http://www.eclipse.org/xtend/>
- EMF/eCore: <http://www.eclipse.org/modeling/emf/>
- EMF tutorial: <http://www.vogella.de/articles/EclipseEMF/article.html>