Enterprise-Softwarearchitektur und Domain Driven Design (DDD)

Manuel Meyer, Senior Consultant Trivadis AG

BASEL BERN BRUGG LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Über mich…

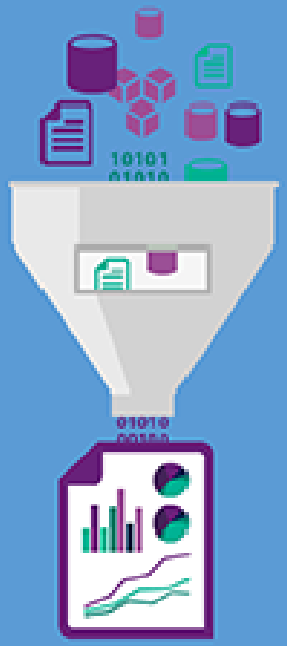
Microsoft CERTIFIED Trainer

CSM ScrumAlliance CERTIFIED ScrumMaster

- Senior Consultant/Trainer bei Trivadis AG
- .NET C#, WPF, Microsoft Azure
- .NET Performance Management & Troubleshooting

Motivation

.NET Software-Architektur
- S.O.L.I.D. Prinzipien
- Domain Driven Design



Enterprise-Lösungen
- Wartbarkeit
- Erweiterbarkeit

AGENDA

1. Einleitung
2. S.O.L.I.D Prinzipien
3. Domain-Driven Design
4. Von Data-Driven zu Domain-Driven in .NET
5. Zusammenfassung

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Einleitung

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Was ist Softwarearchitektur?

"Software architecture is the high level **structure** of a software system, the **discipline** of creating such a high level structure, and the **documentation** of this structure."

-wikipedia

An Architect's dream
is an Engineer's nightmare.

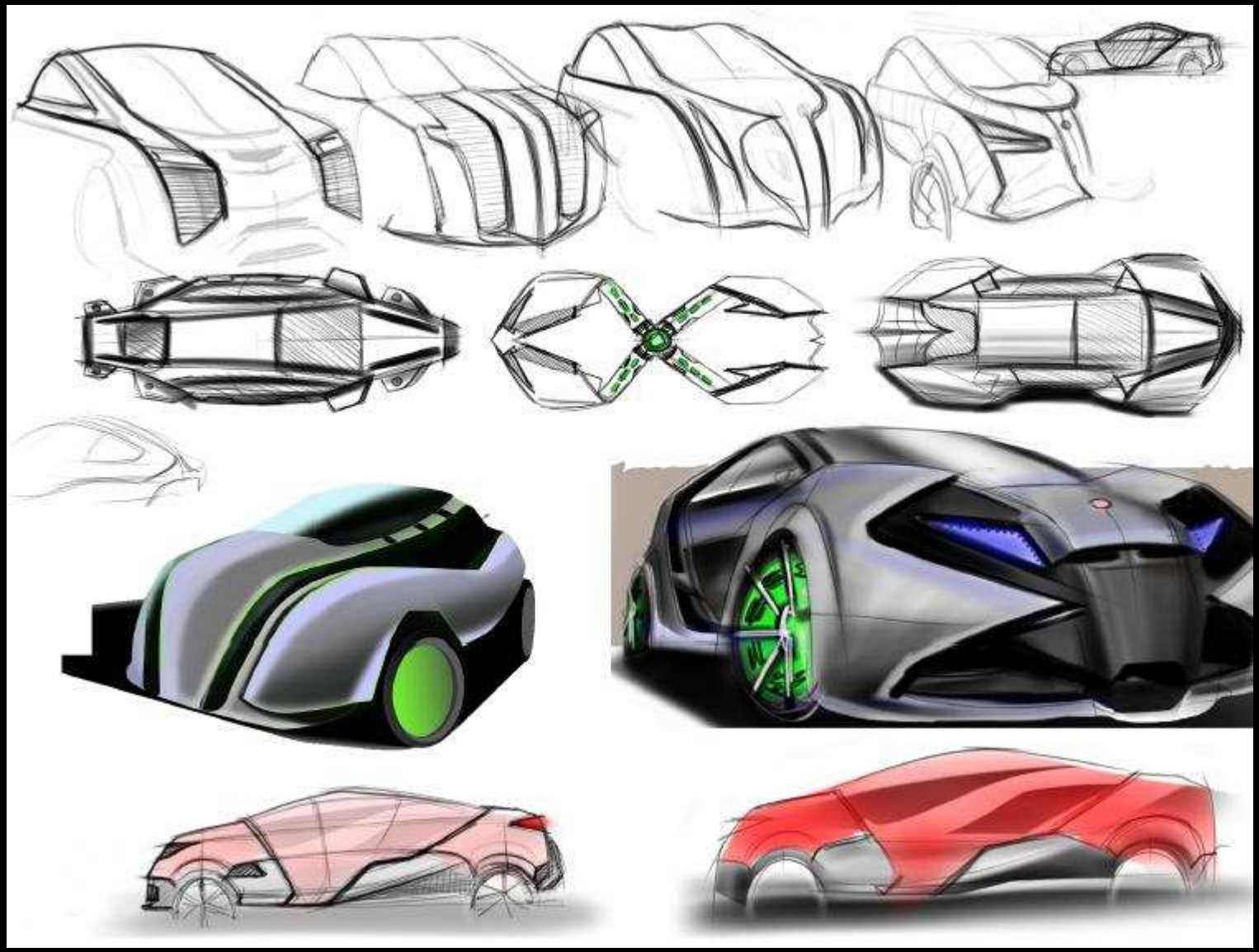
Was ist Softwaredesign?

"Software design is the process of **implementing software solutions** to one or more set of problems."

-wikipedia

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

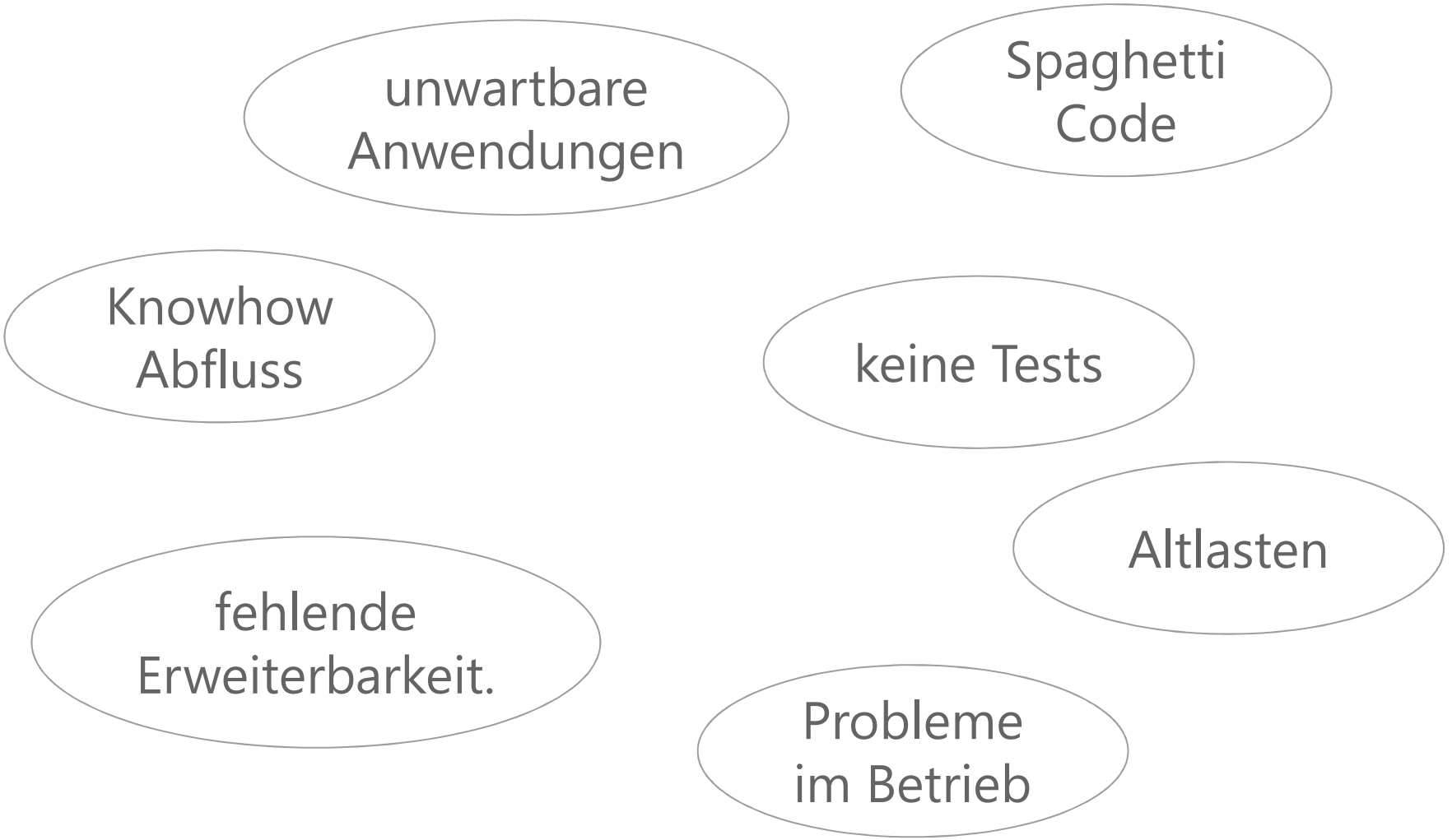


Was ist Softwaredesign?

"Industrial design is the use of both **applied art** and **applied science** to improve the aesthetics, design, ergonomics, functionality, and usability of a product."

-wikipedia

Probleme in der Softwareentwicklung (Auszug)

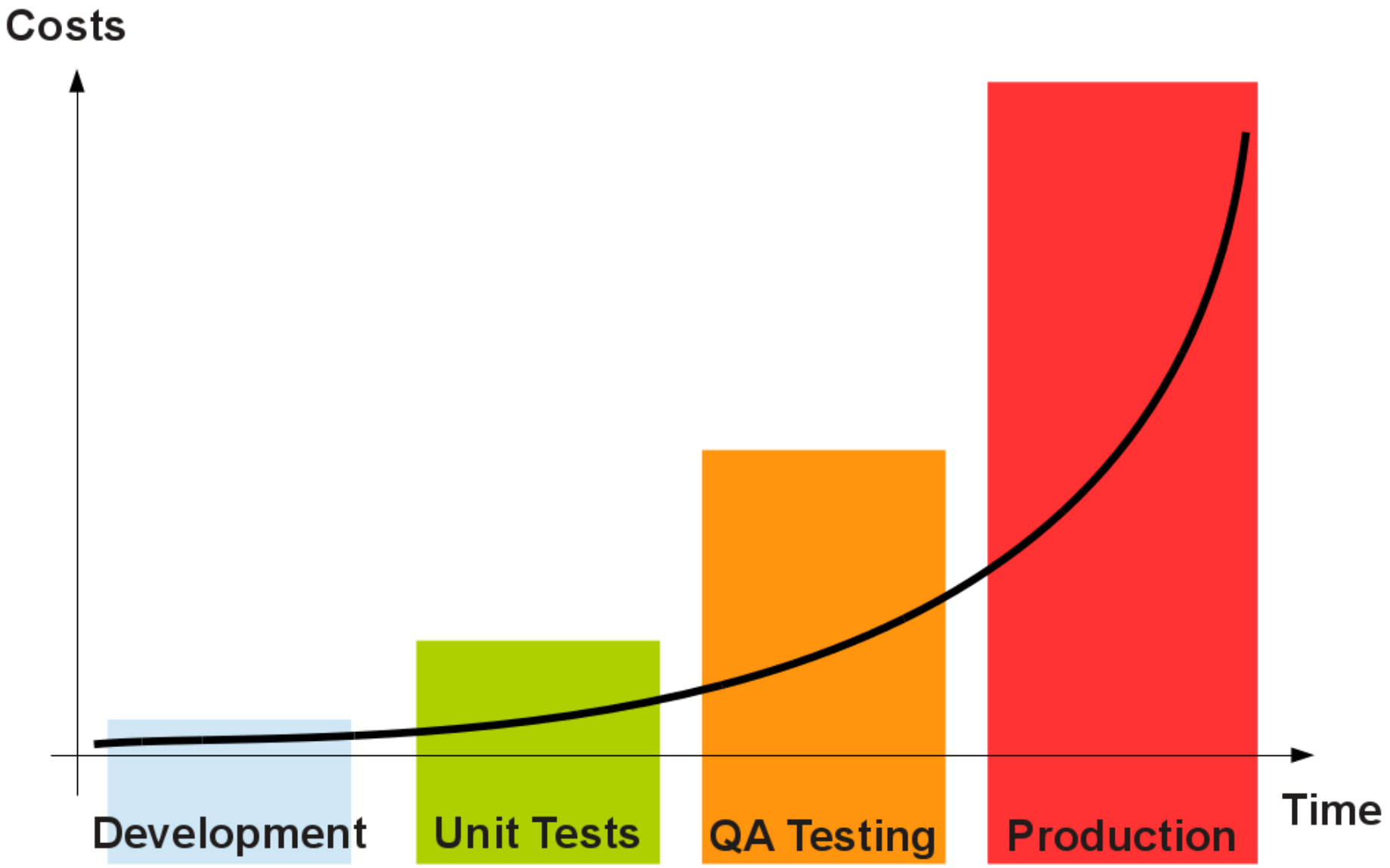


Probleme in der Softwareentwicklung

“Kosten & Legacy”

2014 © Trivadis

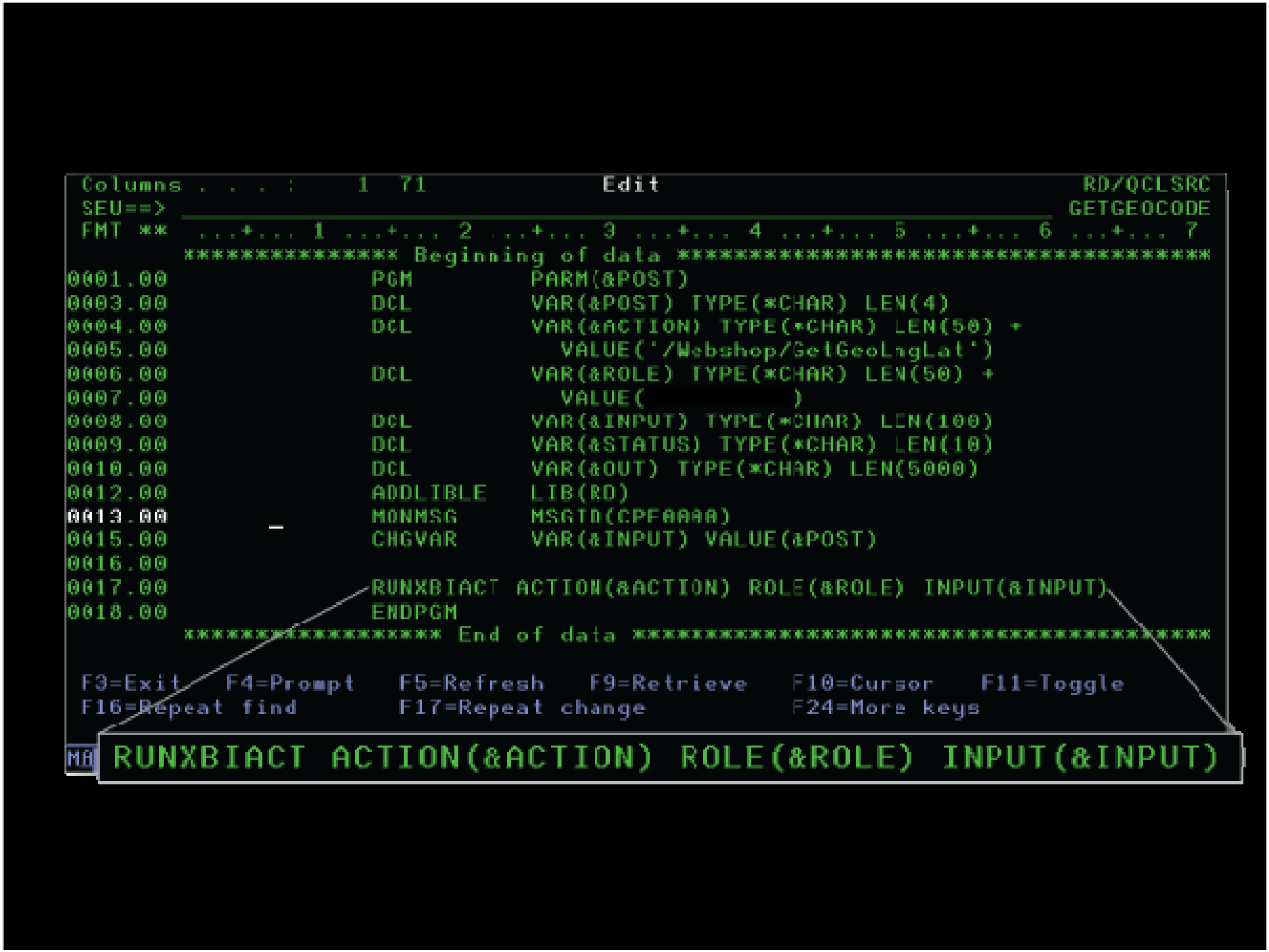
20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.


Costs
Development
Unit Tests
QA Testing
Production
Time

Probleme in der Softwareentwicklung

“Legacy”

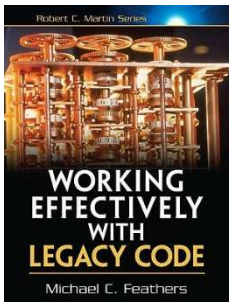
IBM
SYSTEM


Columns . . . : 1 71 Edit RD/QCLSRC
SEU==> GETGEOCODE
FMT ** ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
********** Beginning of data **********
0001.00 PGM PARM(&POST)
0003.00 DCL VAR(&POST) TYPE(*CHAR) LEN(4)
0004.00 DCL VAR(&ACTION) TYPE(*CHAR) LEN(50) +
0005.00 VALUE('/Webshop/GetGeoLngLat')
0006.00 DCL VAR(&ROLE) TYPE(*CHAR) LEN(50) +
0007.00 VALUE( )
0008.00 DCL VAR(&INPUT) TYPE(*CHAR) LEN(100)
0009.00 DCL VAR(&STATUS) TYPE(*CHAR) LEN(10)
0010.00 DCL VAR(&OUT) TYPE(*CHAR) LEN(5000)
0012.00 ADDLIBLE LIB(RD)
0013.00 MONMSG MSGID(CPF0000)
0015.00 CHGVAR VAR(&INPUT) VALUE(&POST)
0016.00
0017.00 RUNXBIACT ACTION(&ACTION) ROLE(&ROLE) INPUT(&INPUT)
0018.00 ENDPGM
********** End of data **********
F3=Exit F4=Prompt F5=Refresh F9=Retrieve F10=Cursor F11=Toggle
F16=Repeat find F17=Repeat change F24=More keys
MA
RUNXBIACT ACTION(&ACTION) ROLE(&ROLE) INPUT(&INPUT)

Probleme in der Softwareentwicklung

"…legacy code is simply code without tests…"

-Michael Feathers



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Wie können Architektur & Design helfen?

- Reduktion der Legacy
- Ermöglichen der NICHT-FUNKTIONALEN Anforderungen:
  - Extensibility
  - Maintainability
  - Reusability
  - Testability
  - ...

-> Unterstützung durch korrekten OOP Einsatz (S.O.L.I.D)

-> Unterstützung durch Domain Driven Design (DDD oder DDDD).

S.O.L.I.D

Principles

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles

- **S**RP: Single Responsibility Principle
- **O**CP: Open Closed Principle
- **L**SP: Liskov Substitution Principle
- **I**SP: Interface Segregation Principle
- **D**IP: Dependency Inversion Principle

SOLID Principles

Robert C. Martin (a.k.a. Uncle Bob)

- Software Consultant
- Writer
  - Principles of OOD
  - Agile Software Development: Principles, Patterns and Practices
  - Clean Code
- First Chairman of the Agile Alliance

Michael Feathers

SOLID Principles: SRP

SRP: Single Responsibility Principle

"There should never be more than one reason for a class to change"

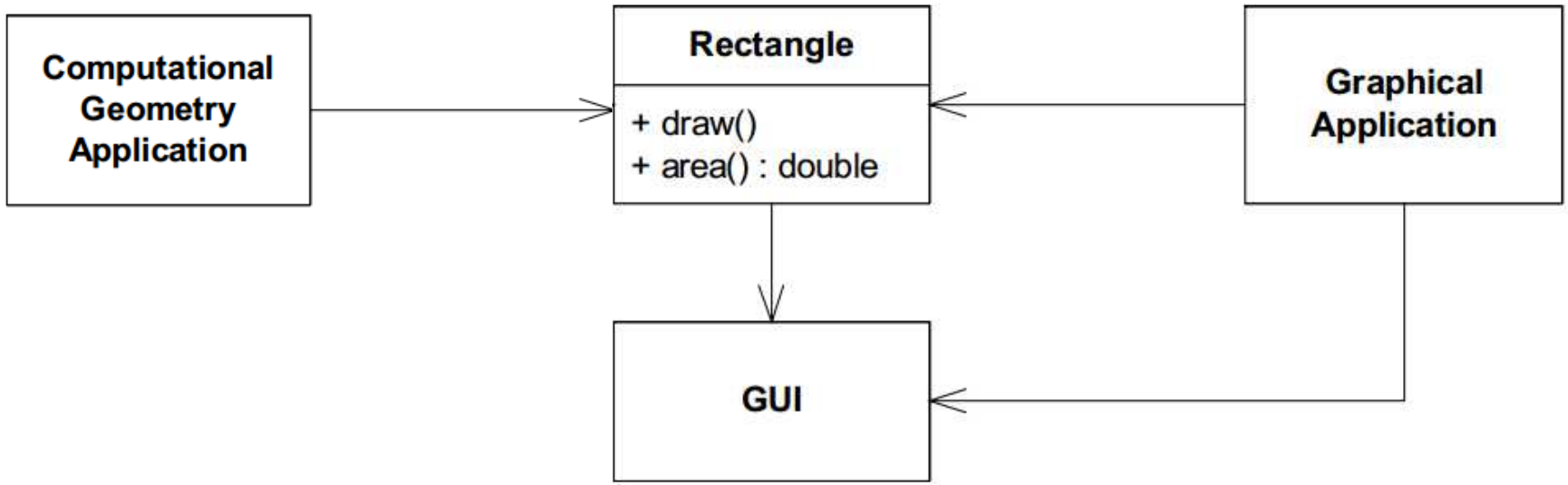
SOLID Principles: SRP

SRP: Single Responsibility Principle

- Verantwortlichkeit = Axis of change
- Gilt für alle Objekte: Assemblies, Klassen, Methoden.

SOLID Principles: SRP

Beispiel 1: Rectangle

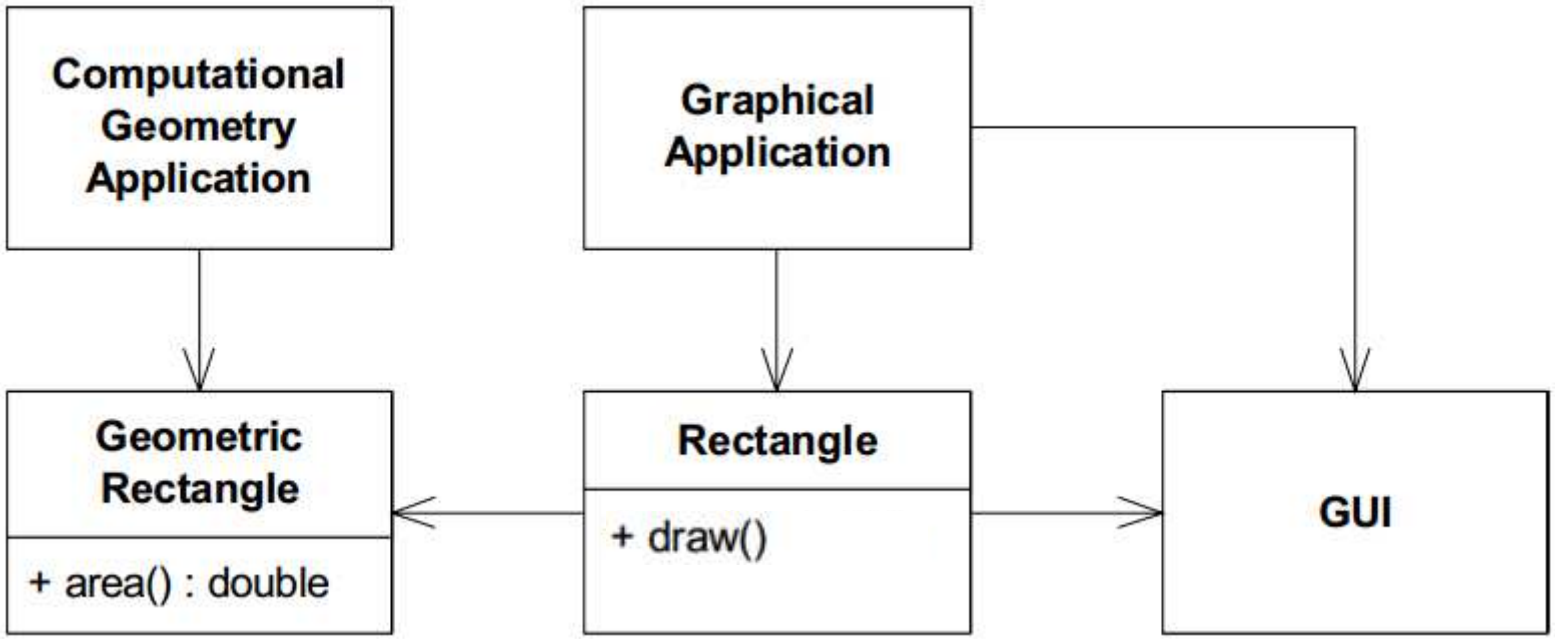


2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles: SRP

Beispiel 1: Rectangle

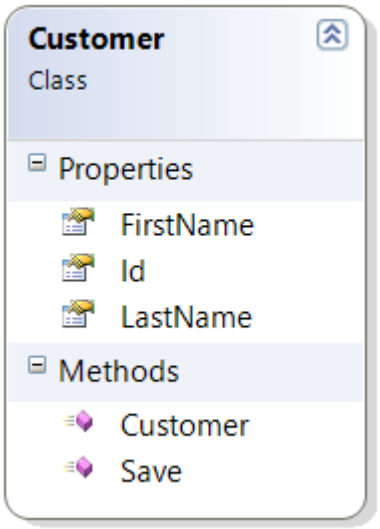


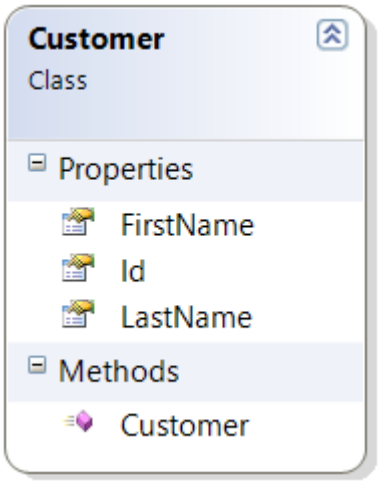
2014 © Trivadis

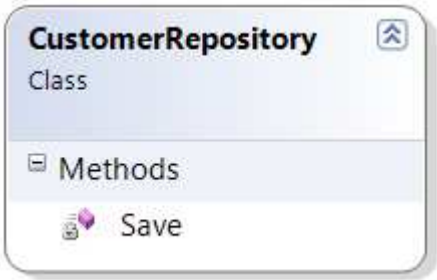
20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles: SRP

Beispiel 2 Customer Persistierung:







2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles

OCP: Open/Closed Principle

"Software entities (classes, modules, functions, etc.), should be **open for extension**, but **closed for modification**"

Bertrand Meyer, 1988

SOLID Principles

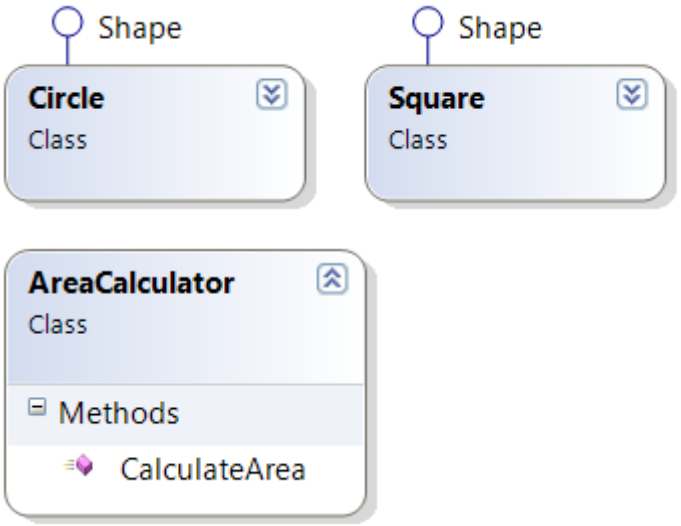
OCP: Open/Closed Principle

- Open for Extension = Das Objekt kann um neue Anforderungen erweitert werden
- Closed for Modification = Bestehender Code wird nicht verändert

- -> Änderungen werden durch hinzufügen von neuem Code, NICHT durch Änderung von altem Code gemacht.
- -> Der Schlüssel liegt in der Abstraktion.

SOLID Principles OCP

OCP: Open/Closed Principle



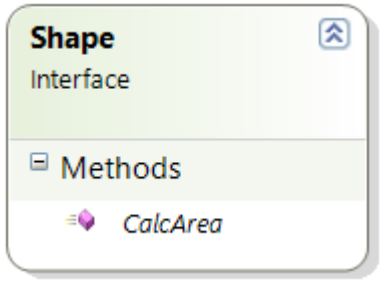

```
public int CalculateArea(List<Shape> shapes)
{
    int totalArea = 0;

    foreach (var shape in shapes)
    {
        if (shape is Circle)
        {
            //calculate Circle Area
        }
        else if (shape is Square)
        {
            //calculate Square Area
        }
        else if (shape is Rectangle)
        {
            //calculate Rectangle Area
        }
    }
    return totalArea;
}
```


2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles OCP

OCP: Open/Closed Principle




```
public class AreaCalculator
{
    public int CalculateArea(List<Shape> shapes)
    {
        int totalArea = 0;

        foreach (var shape in shapes)
        {
            totalArea += shape.CalcArea();
        }
        return totalArea;
    }
}
```

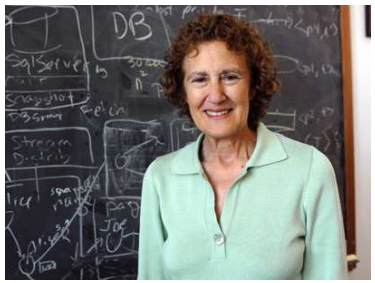

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles: LSP

LSP: Liskov Substitution Principle

"Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program"



Barbara Liskov, MIT 1987

SOLID Principles: LSP

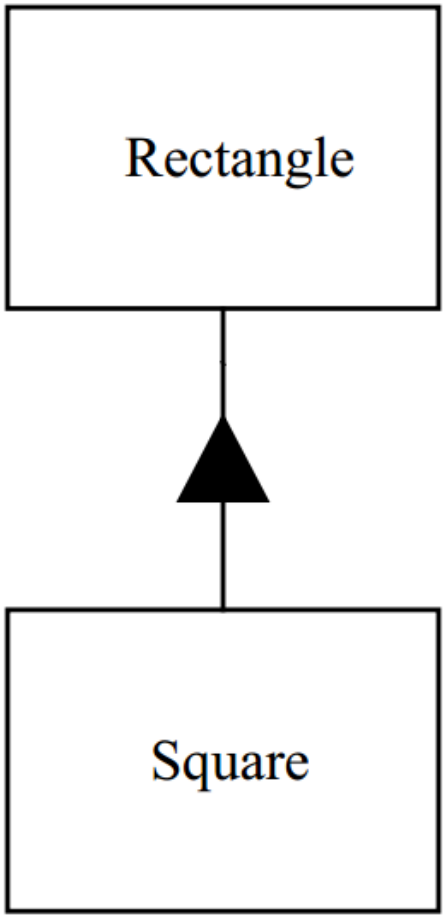
LSP: Liskov Substitution Principle

- Eigentlich eine Erweiterung von OCP
- Wir müssen sicherstellen, dass abgeleitete Klassen das Verhalten der Basisklasse (inklusive möglicher ANNAHMEN!) NICHT verändern.

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles: LSP



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles: LSP

LSP: Liskov Substitution Principle

- Unsere Ableitung war syntaktisch ok, hat aber zu einem semantischen Fehler geführt.
- Der Fehler kam erst beim Testing raus
- Wir haben die IS-A Beziehung missbraucht
- Mit Breite und Höhe hatten wir einen Hinweis
- Wir müssen sicherstellen, dass abgeleitete Klassen das Verhalten der Basisklasse (inklusive möglicher ANNAHMEN!) NICHT verändern.

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles: ISP

ISP: Interface Segregation Principle

"Clients should not be forced to depend upon interfaces that they do not use"

Robert C. Martin, Xerox

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles: ISP

ISP: Interface Segregation Principle

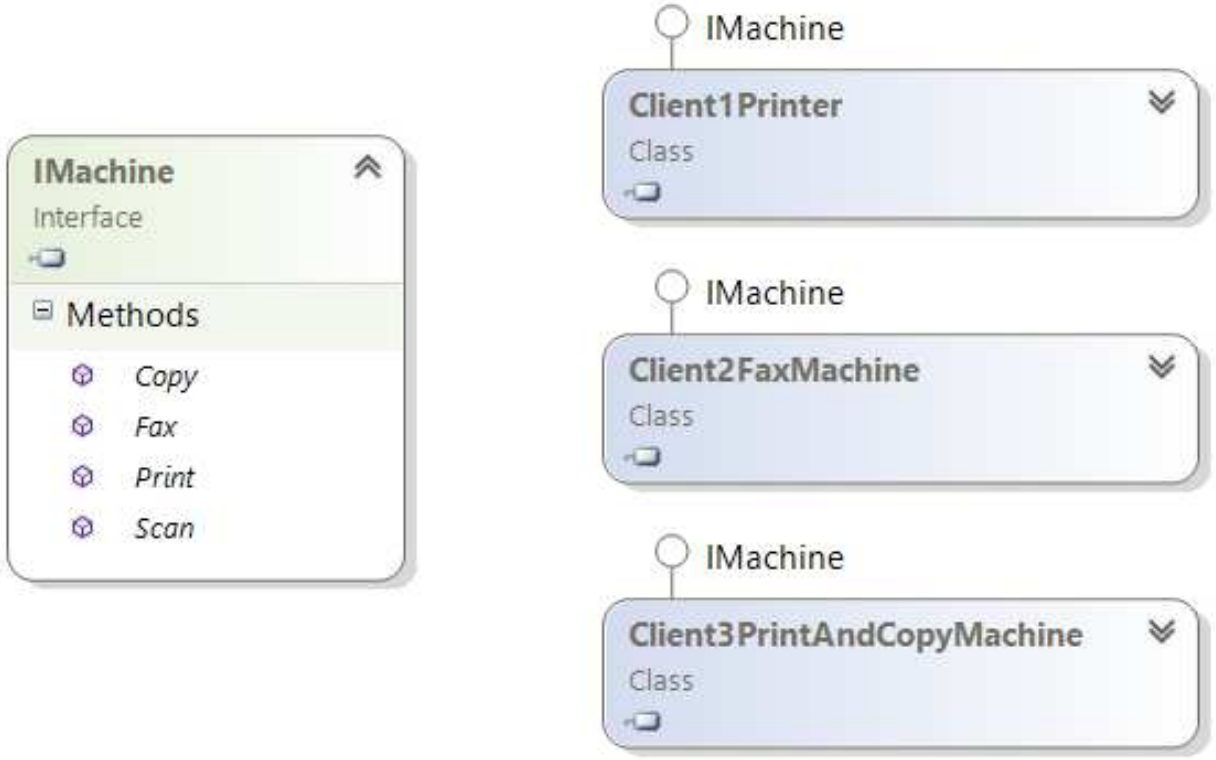
- Interface Pollution (Clients müssen Interfaces implementieren, welche sie nicht brauchen.)
- Keine„fat“ Interfaces
- Viele kleine Interfaces
- Grosse Interfaces = mehr Abhängigkeiten.

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles: ISP

Beispiel 1: Xerox



2014 © Trivadis

SOLID Principles: ISP

Beispiel 2: ASP.NET MembershipProvider

SOLID Principles

DIP: Dependency Inversion Principle

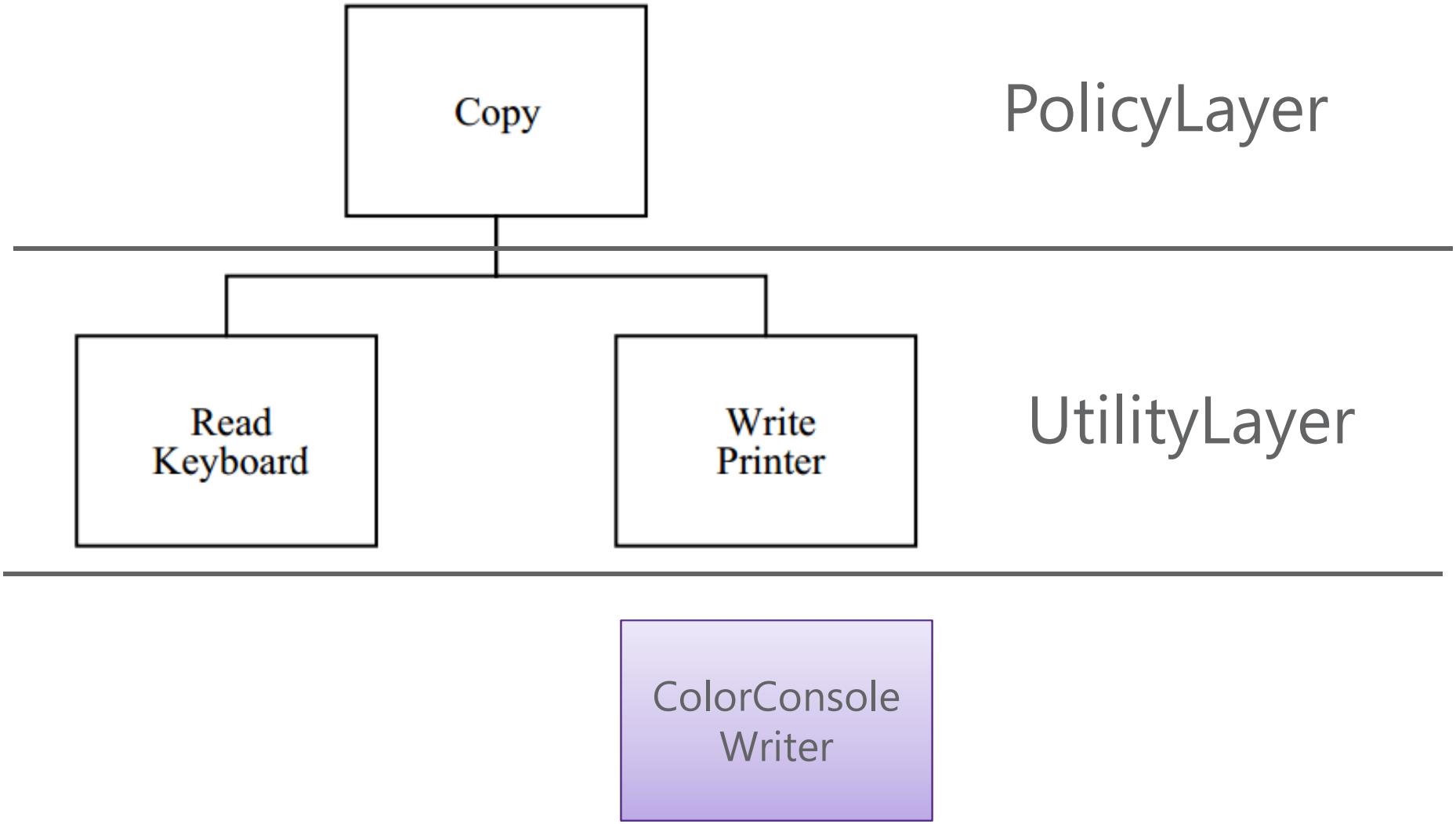
"A. High-level modules should not depend on low-level modules. Both should depend on abstractions."

B. Abstractions should not depend on details. Details should depend on abstractions.

Robert C. Martin

2014 © Trivadis

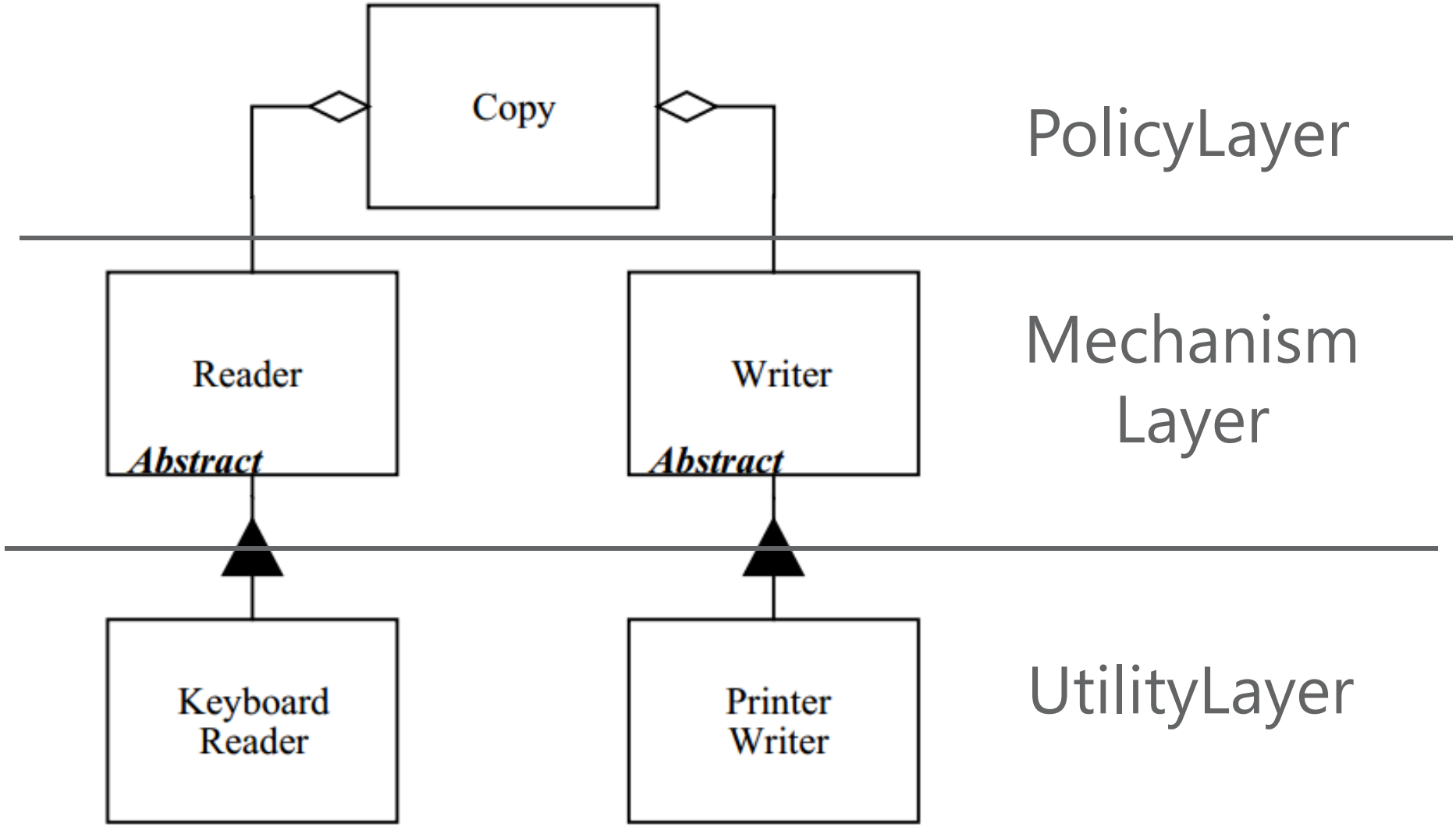
SOLID Principles: DIP



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles: DIP



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

S.O.L.I.D Bad Smells

- If-Statement, welches Typen unterscheidet -> OCP
- Interface-Methoden, welche in vielen Fällen nicht implementiert werden müssen -> ISP
- Modelierung von IS-A Beziehungen, welche nicht ganz passen -> LSP
- Interface Pollution -> ISP
- FAT Interfaces -> ISP
- Methodennamen wie: UpdateAndSaveCustomer(), VerifyAndSaveData() -> SRP
- Klassennamen wie CustomerManager -> SRP.

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

S.O.L.I.D Resources

- SOLID Wikipedia
  - http://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
- Hanselminutes Podcast 145: SOLID Principles with Uncle Bob
  - http://www.hanselman.com/blog/HanselminutesPodcast145SOLIDPrinciplesWithUncleBobRobertCMartin.aspx
- Pablos Solid Ebook
  - http://lostechies.com/wp-content/uploads/2011/03/pablos_solid_ebook.pdf
- Objectmentor.com (Uncle Bob)
  - http://www.objectmentor.com/resources/articles/srp.pdf
  - http://www.objectmentor.com/resources/articles/ocp.pdf
  - http://www.objectmentor.com/resources/articles/lsp.pdf
  - http://www.objectmentor.com/resources/articles/isp.pdf
  - http://www.objectmentor.com/resources/articles/dip.pdf

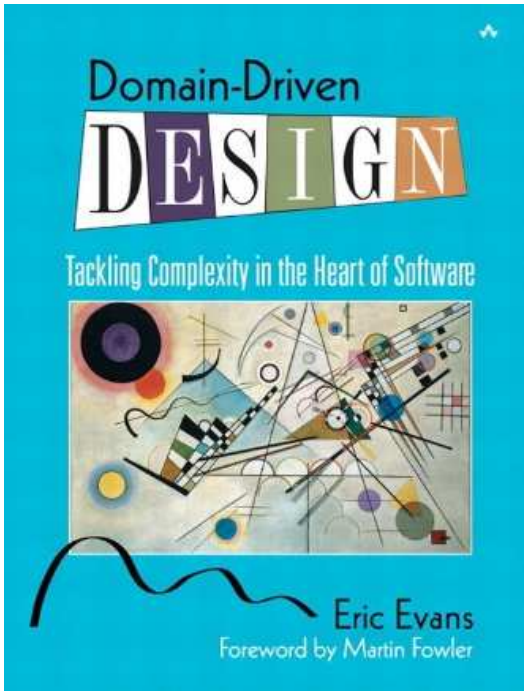
Domain Driven Design

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Domain Driven Design

- 2003: Domain-Driven Design by Eric Evans
- „Tackling Complexity in the Heart of Software“



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Was ist Domain Driven Design?

“a collection of principles and patterns that help developers craft elegant object system.”

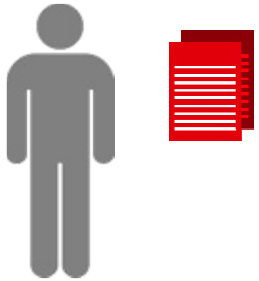
“Ein philosophischer Ansatz, wie Problemdomänen in Software implementiert werden sollten”

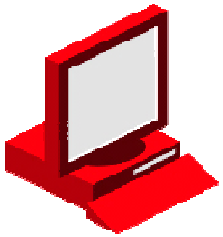
“DDD is just OO software done right.”

Domain Driven Design

- Der Hauptfokus liegt auf der Domäne und deren Logik
- Komplexität wird durch ein Model sichtbar gemacht
- Domain Experts und Entwickler verfeinern das Model iterativ
- Aus dem Model entsteht der Code.

Der Klassische Ansatz

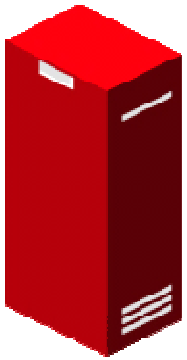




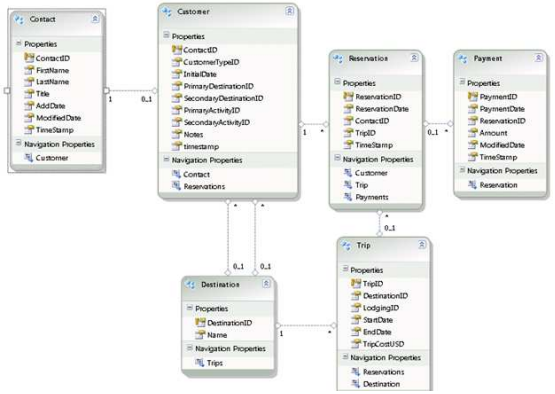
Microsoft .NET

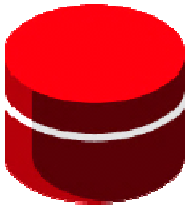
& XAML

ASP.NET WebAPI



Microsoft .NET



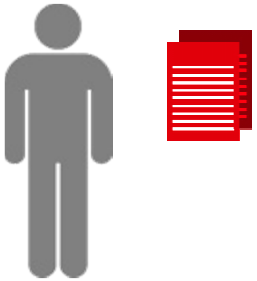


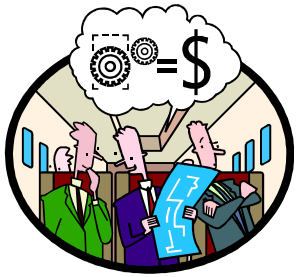
ORM

2014 © Trivadis

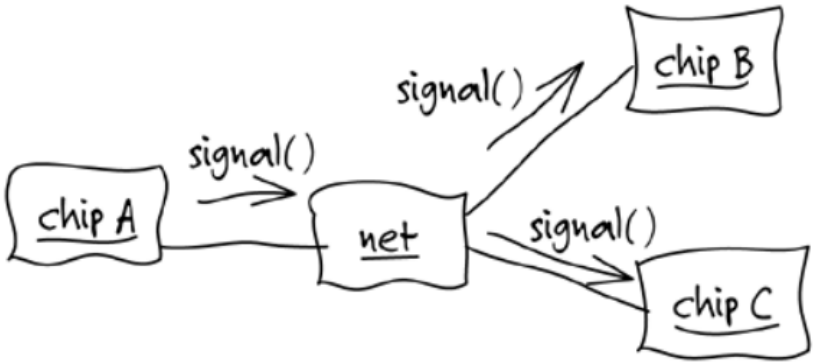
20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Domain Driven Design





Ubiquitous Language

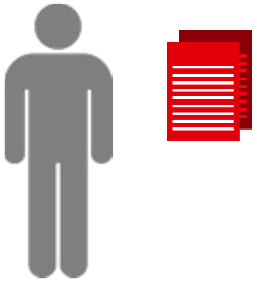


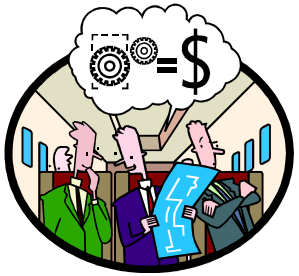
Model

2014 © Trivadis

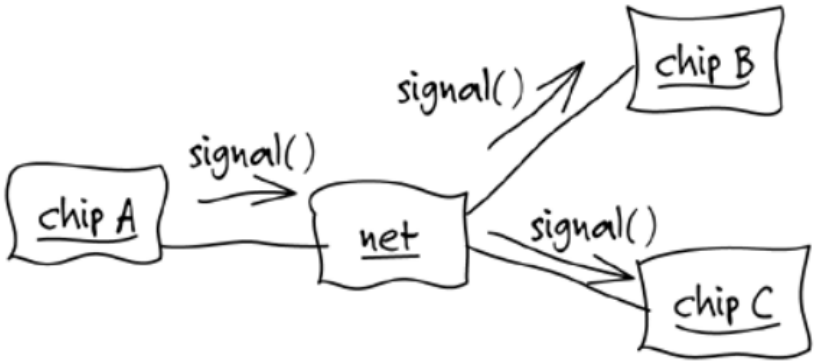
20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Domain Driven Design





Ubiquitous Language

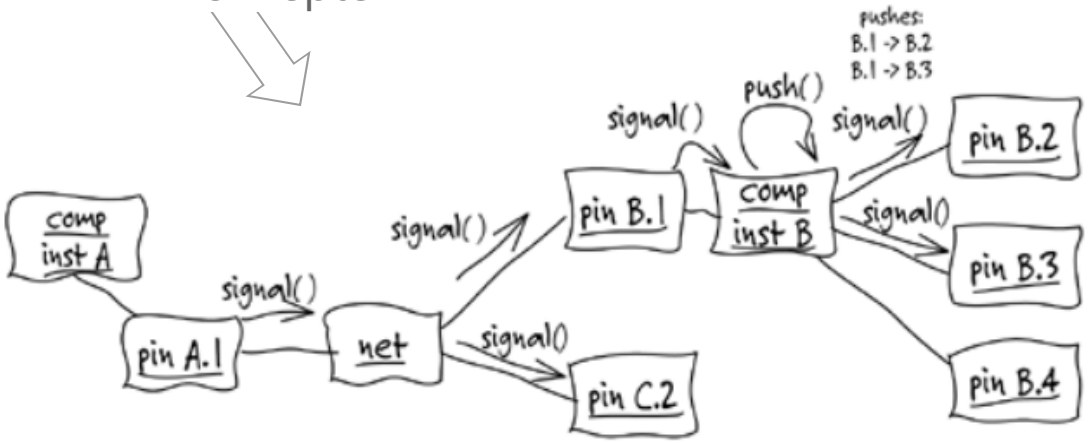


"Make implicit concepts explicit»

Supple Design

Persistence Ignorance.

DDD Konzepte

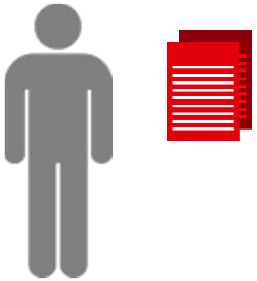


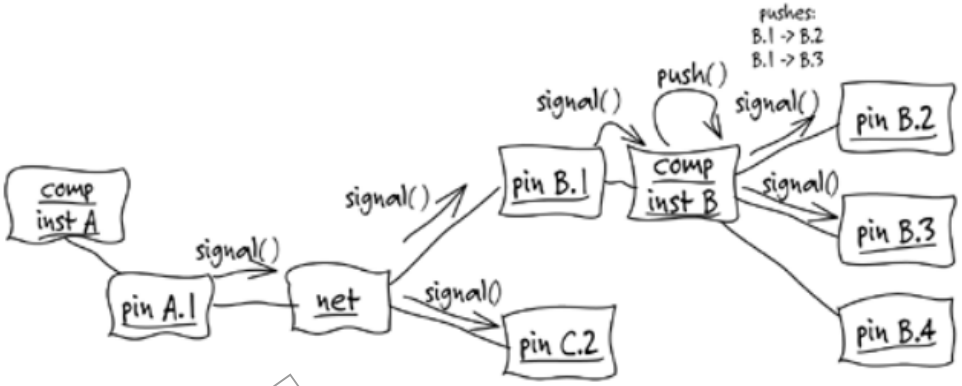
Knowledge-Rich Model

2014 © Trivadis

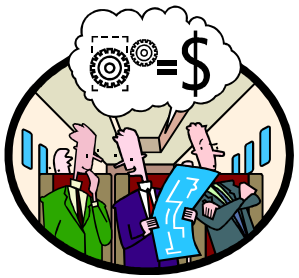
20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Domain Driven Design



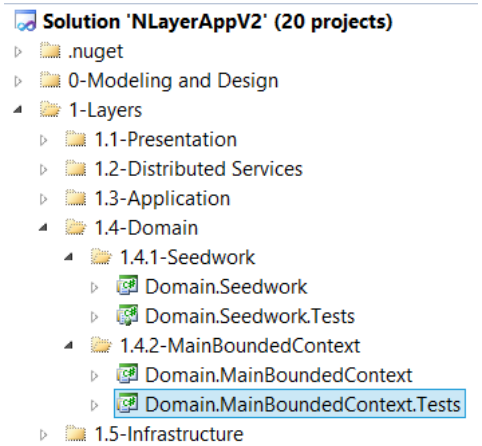


- Entities
- Aggregates
- Value Objects
- Strategies
- Repositories
- Modules.



Ubiquitous Language

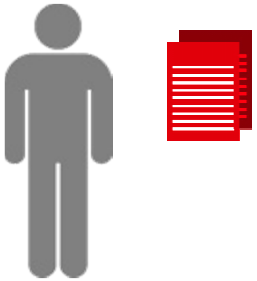
Mit DDD Building Blocks und Techniken wird das Model zu Code

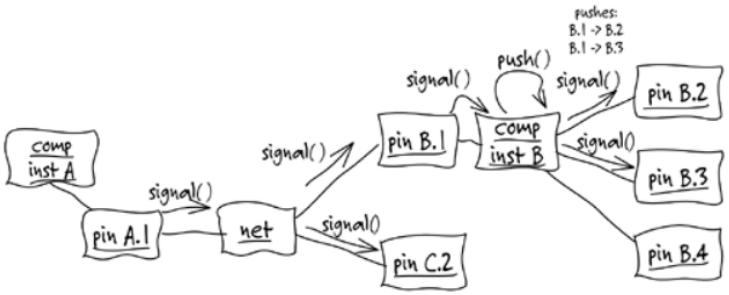


2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Domain Driven Design

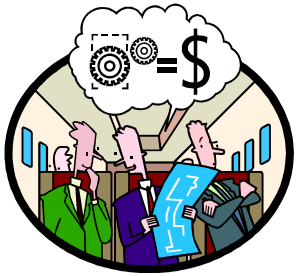




Bounded Context

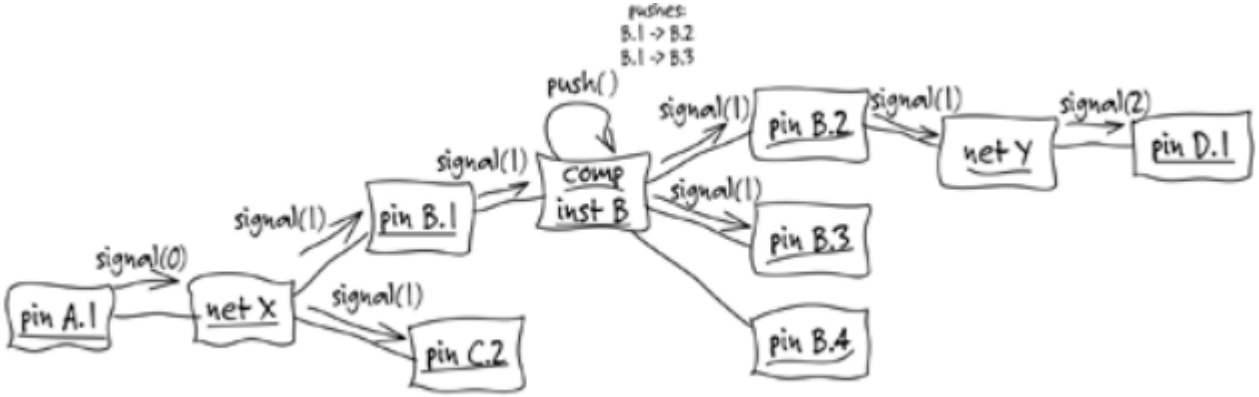
Context Map

Distillation



Ubiquitous Language

Das Model und die UL entwickeln sich weiter…

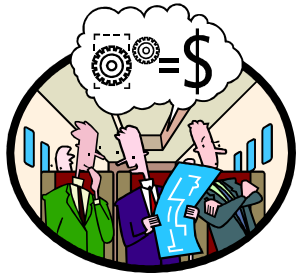


2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Ubiquitous Language

- Vereinbarte Sprache zwischen Domain Experts und Technik
- Wird im Lauf des Projekts iterativ optimiert und erweitert
- Was im Model benannt wird fliesst in die UL ein.



Ubiquitous Language

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

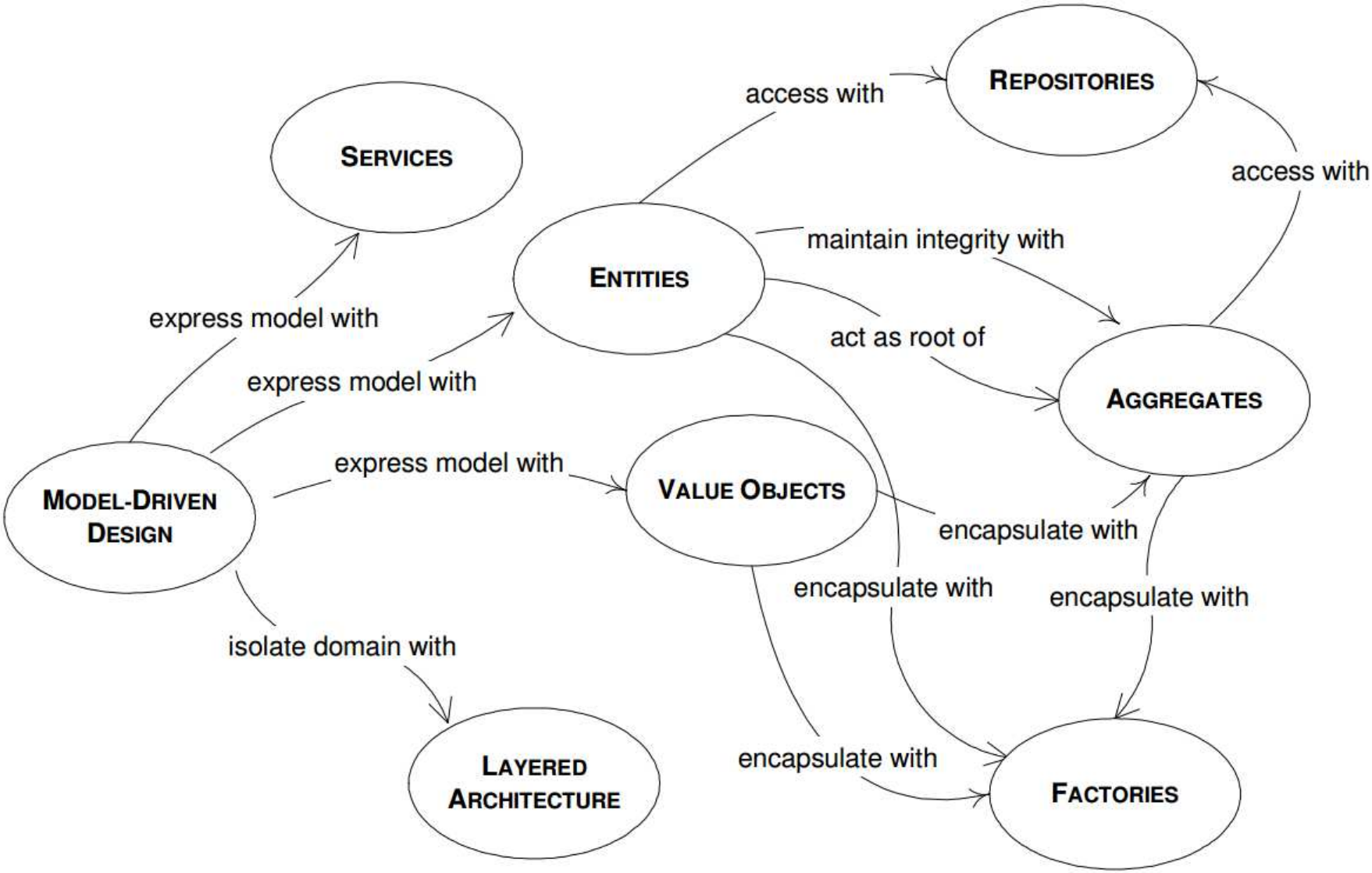
PI: Persistence Ignorance

- „Es gibt keine Datenbank!“
- Der Fokus liegt auf der Domäne
- Entkopplung von Applikation und Infrastruktur.

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

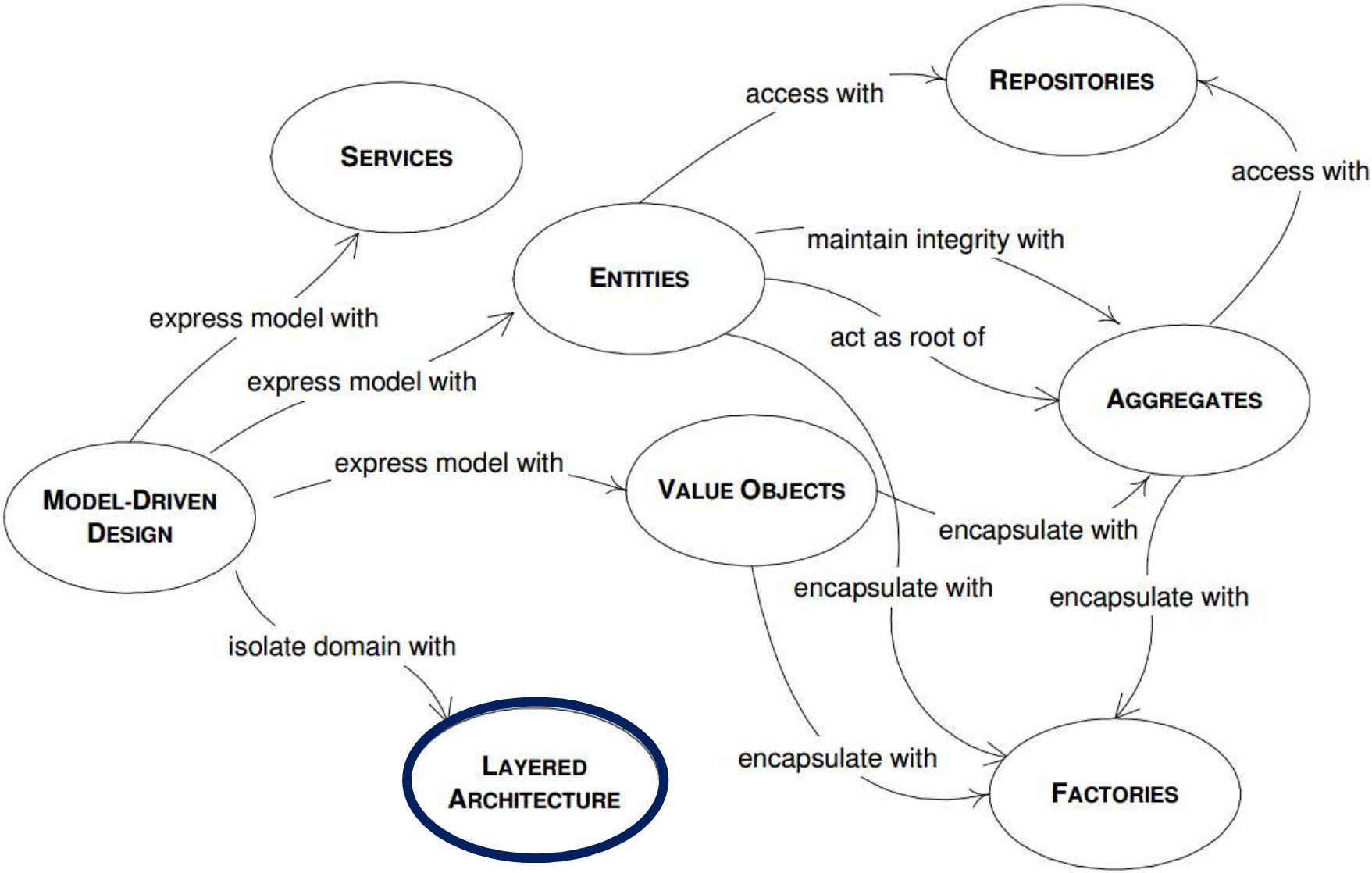
DDD Building Blocks



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

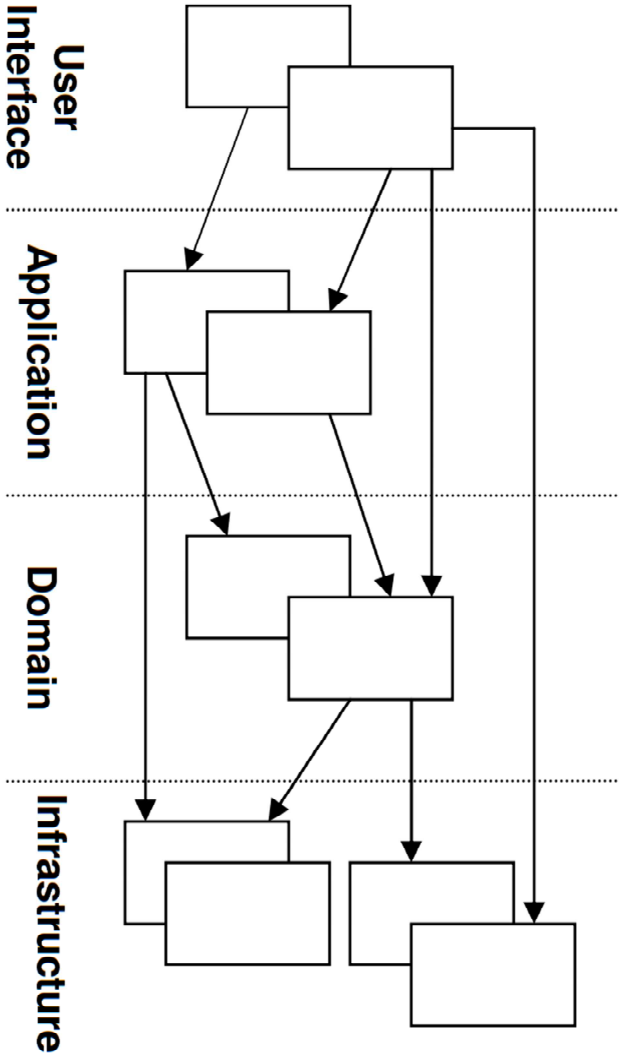
DDD Building Blocks



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Layered Architecture gemäss Eric Evans

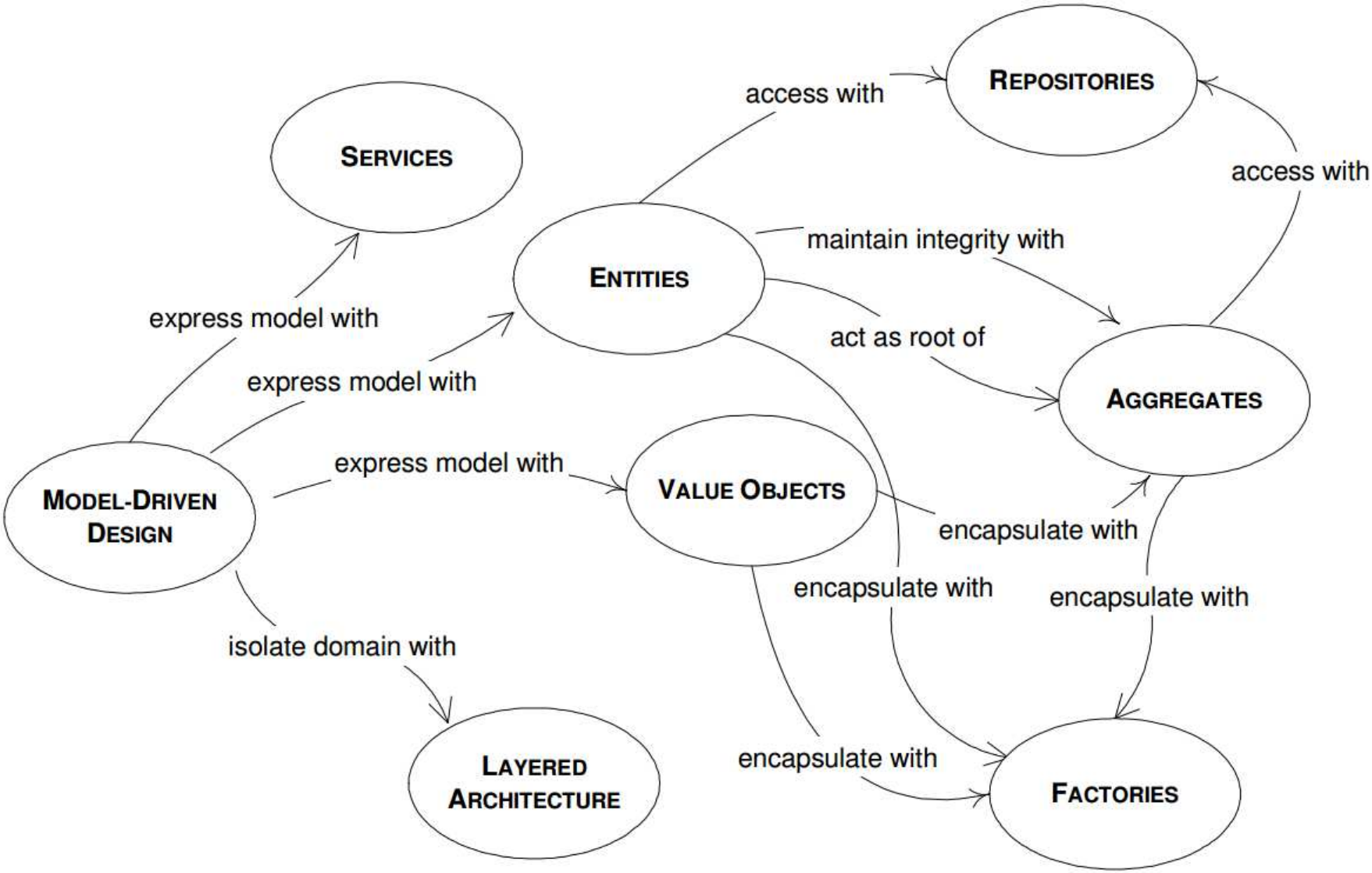


- Wir isolieren den Domain Layer
- Der Fokus liegt auf dem Domain Layer
- Bei der Entwicklung und Diskussion des Models blenden wir die anderen Layers aus.

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

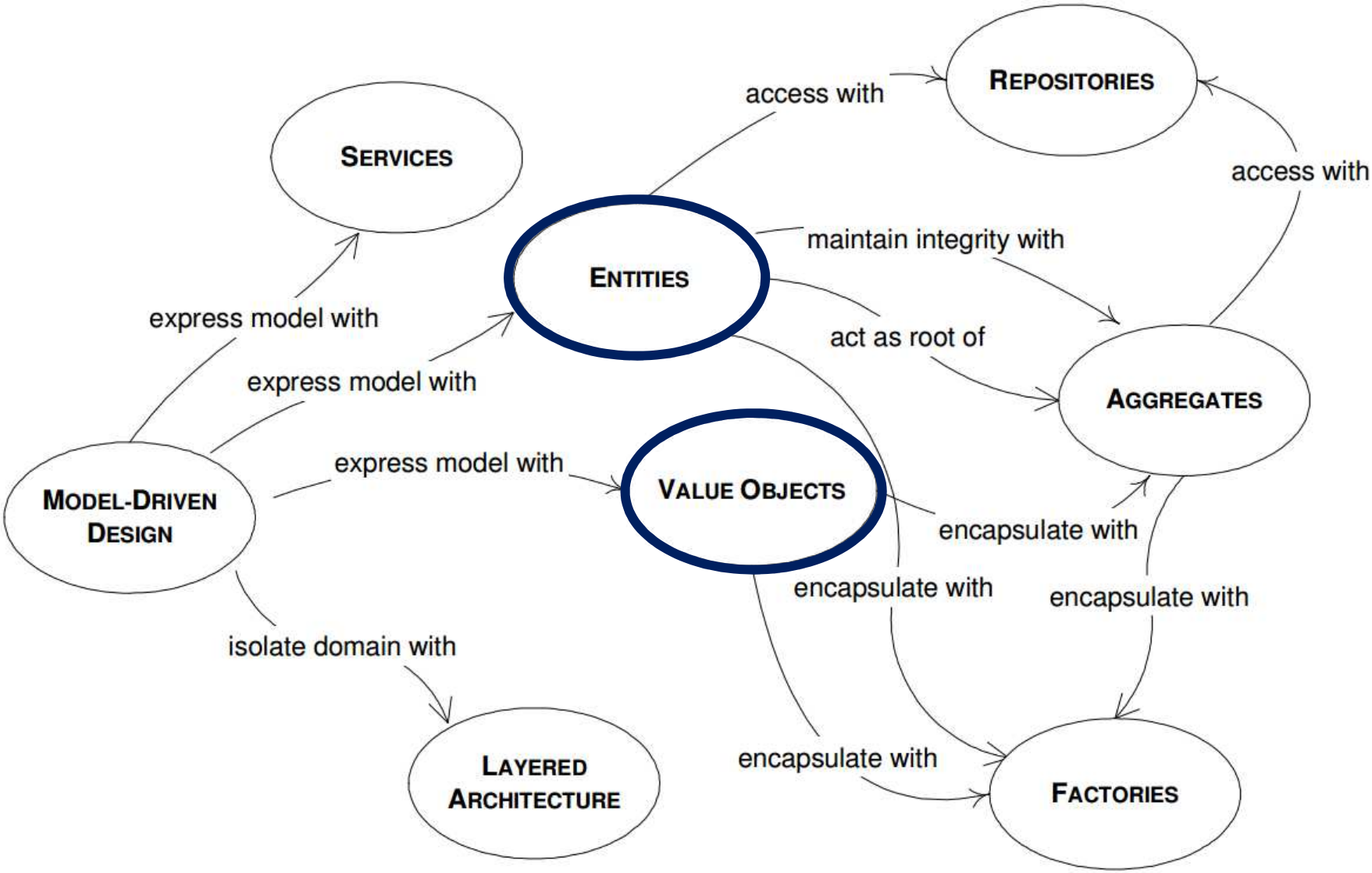
DDD Building Blocks



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

DDD Building Blocks

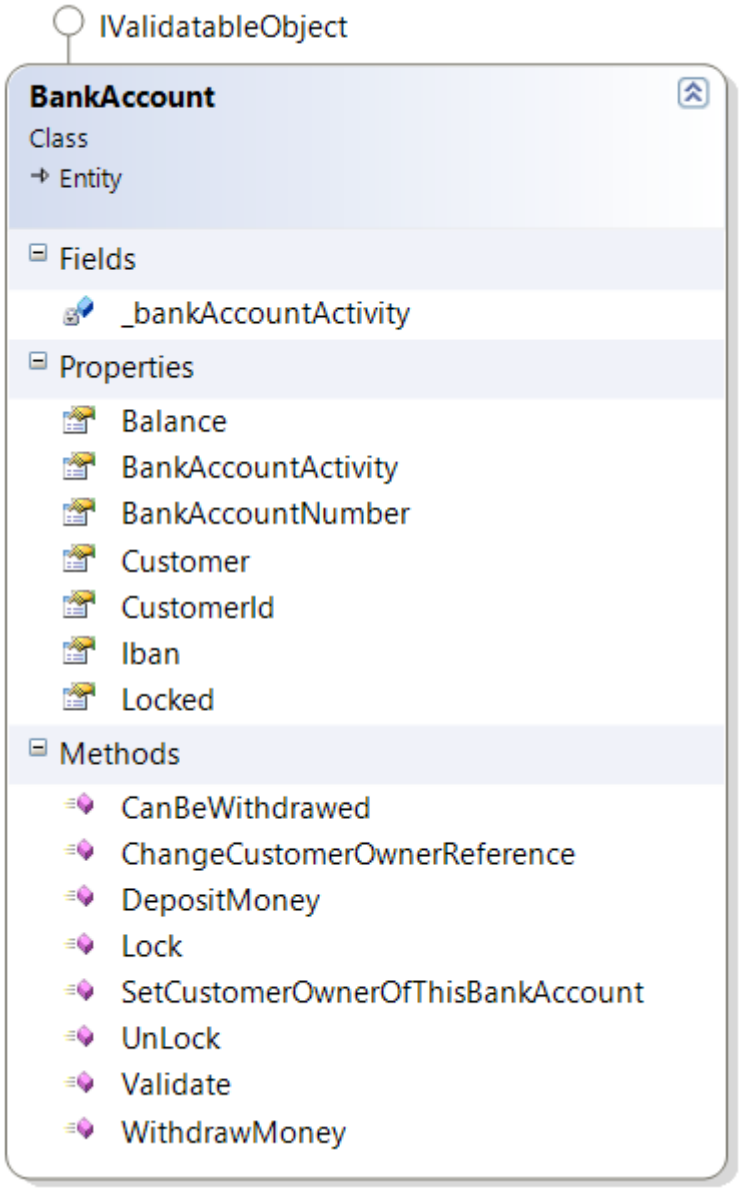


2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Entities

- Entities stellen Business Objekte dar
- Entities haben eine Identität und einen Life Cycle
- Entities enthalten die Business Logik
- Entities sind Knowledge-Rich.

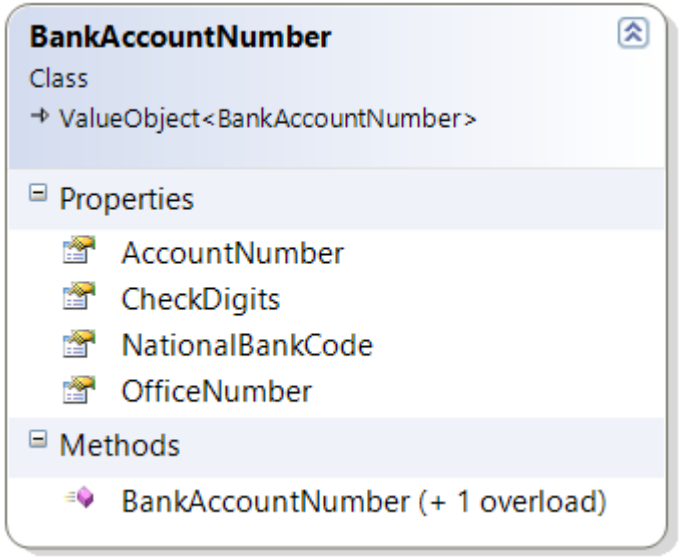


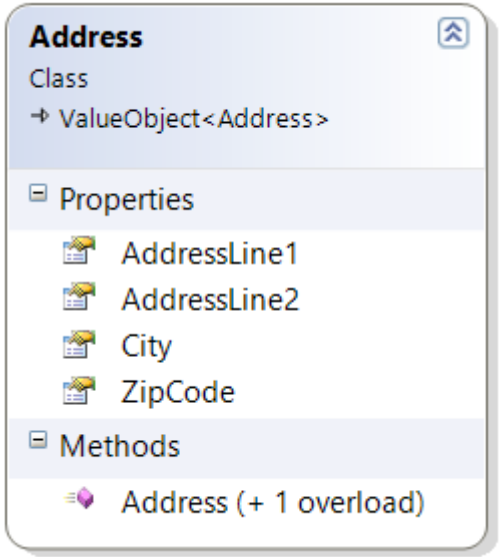
2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Value Objects

- Kleine Datenpakete
- Keine Identität
- Müssen nicht unterscheidbar sein
- Vereinfachen das Model / entfernen Komplexität
- Sind Immutable

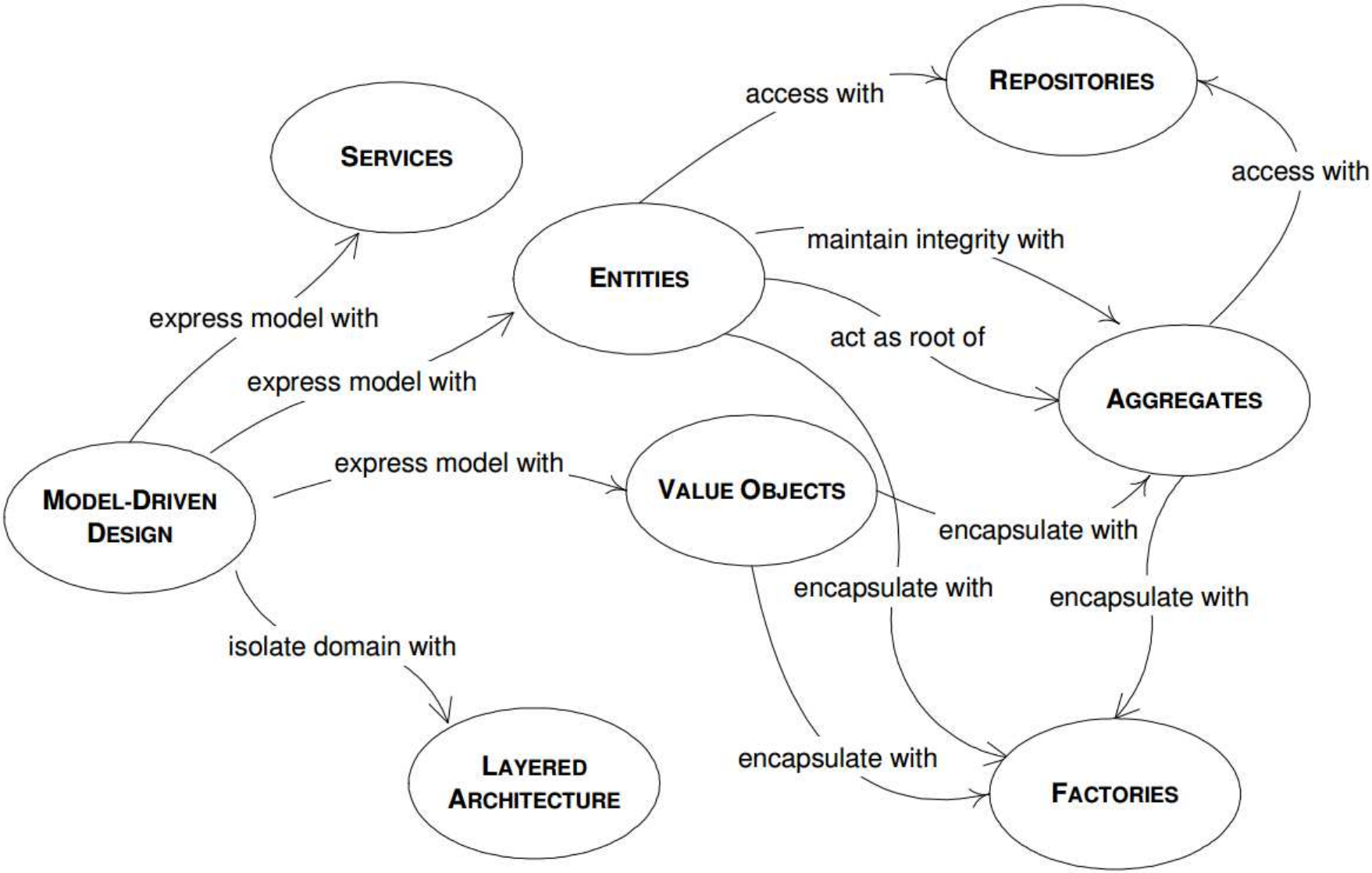




2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

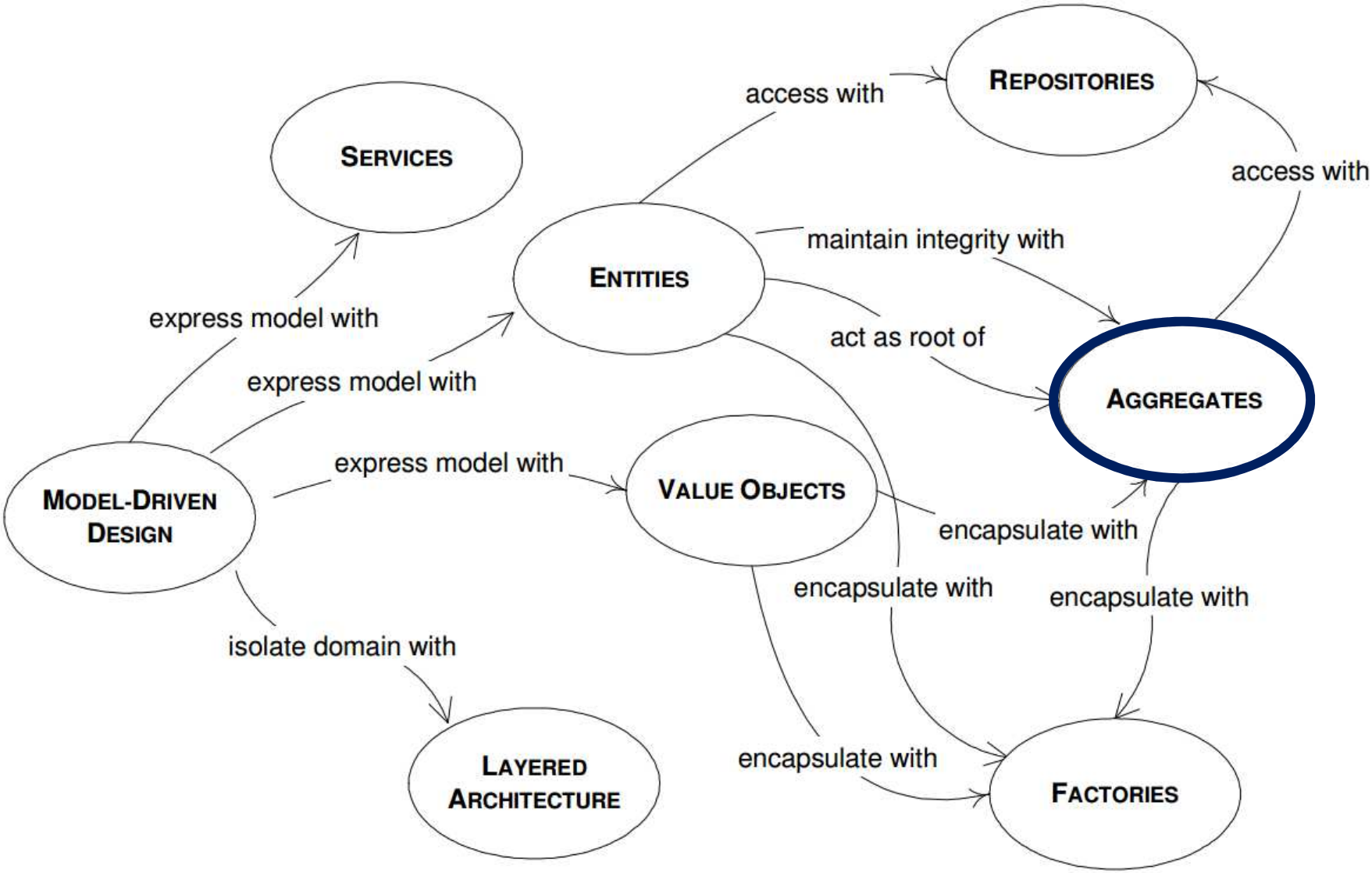
DDD Building Blocks



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

DDD Building Blocks

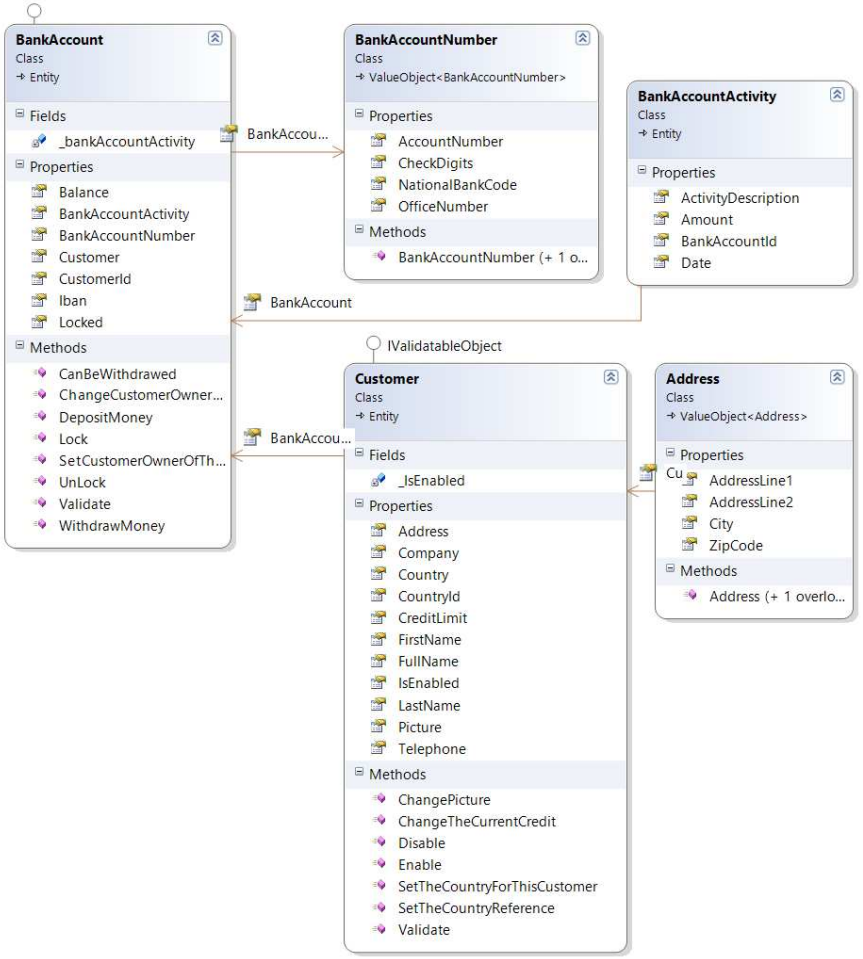


2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Aggregates

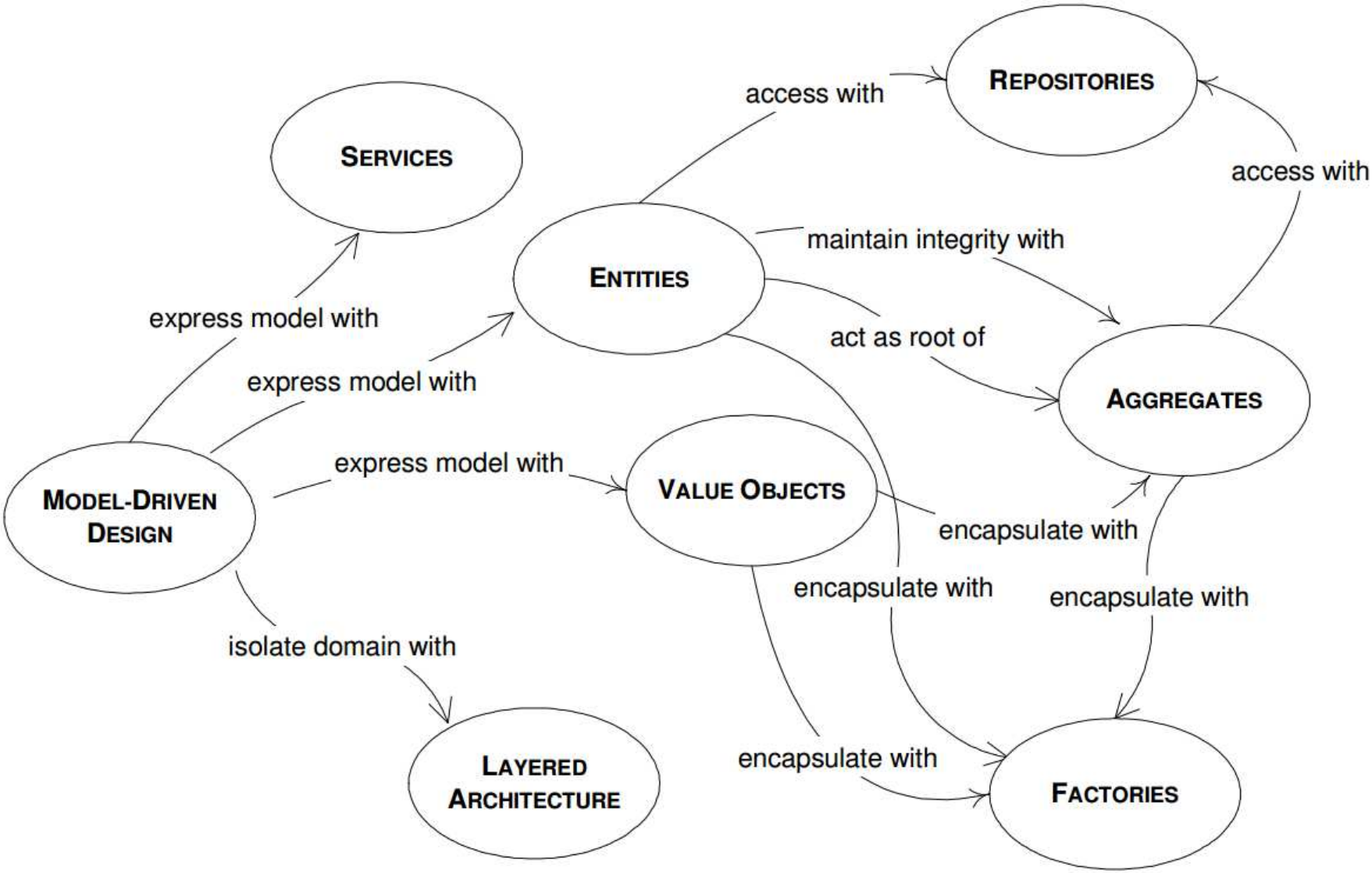
- Fachlich sinnvolle Konstrukte aus Entitäten und Value Objects
- Haben immer 1 Root Entity
- Ausserhalb des Aggregates kann nur die Root Entity referenziert werden
- Die Root Entity ist verantwortlich für die Konsistenz des Aggregates.



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

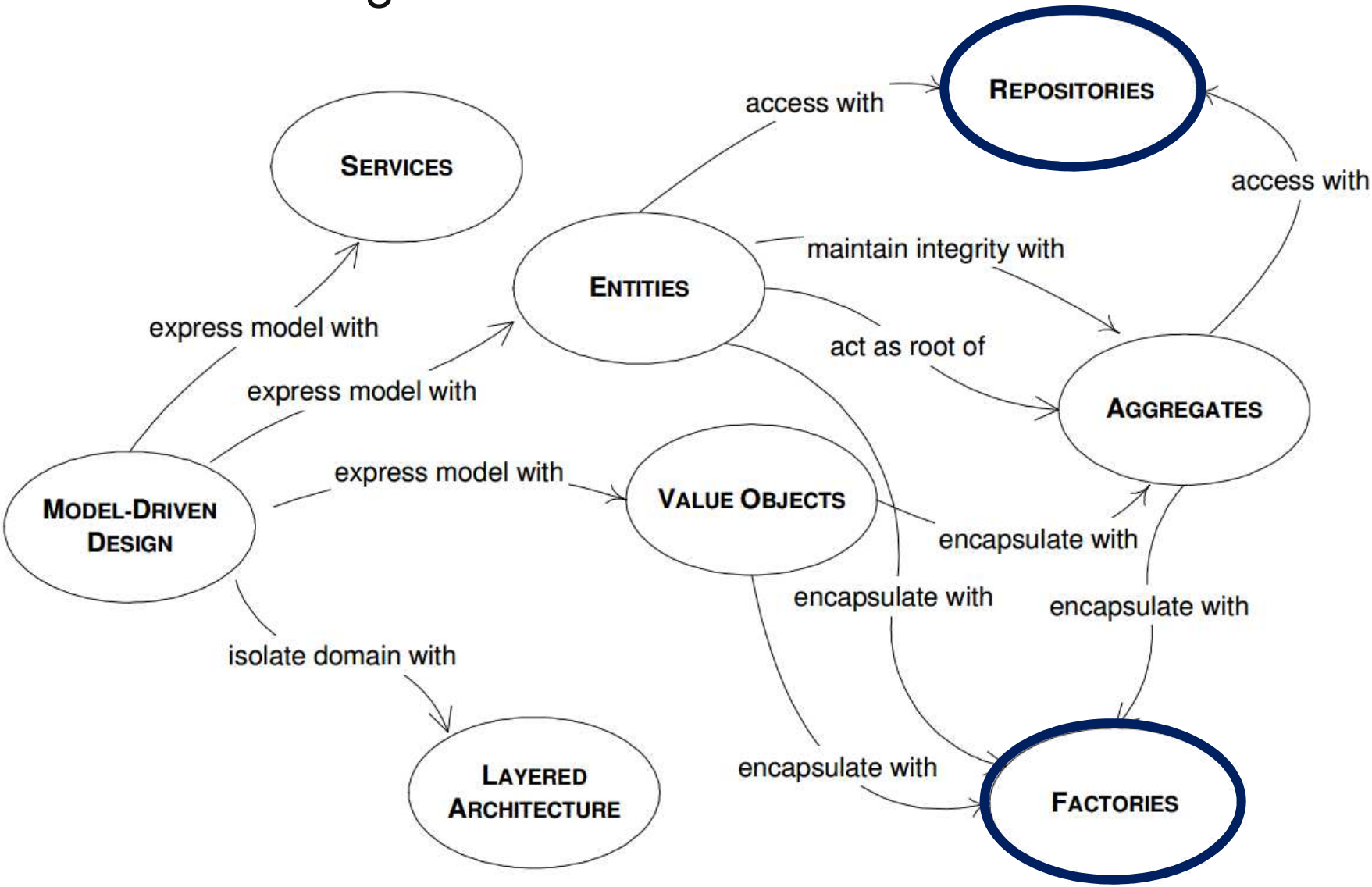
DDD Building Blocks



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

DDD Building Blocks

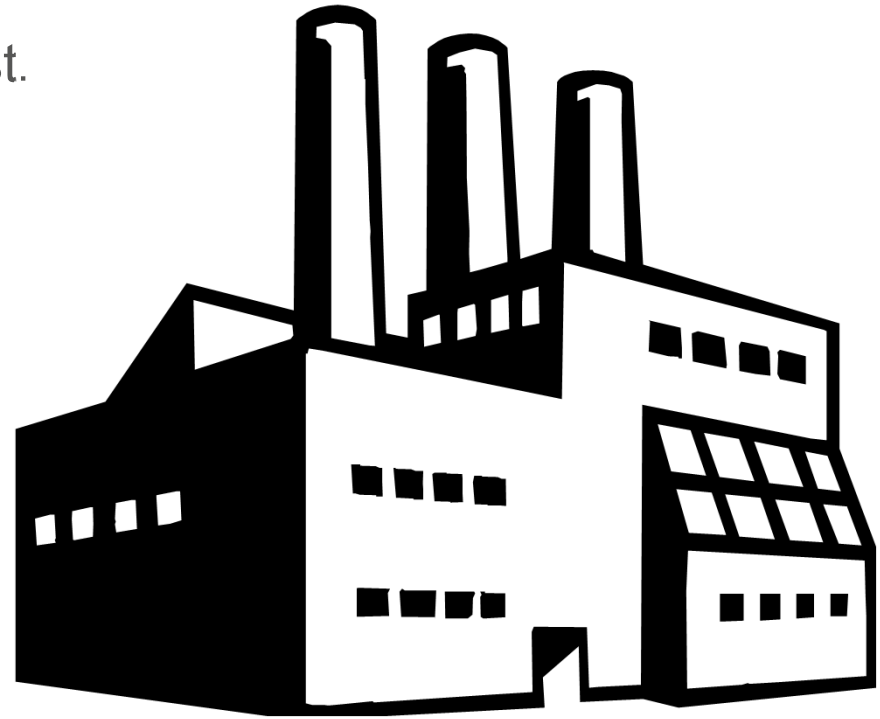


2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Factories

- Erstellen Entities/Aggregates
- Benützen FACTORY Patterns
- Kapseln die Logik, welche zum erstellen komplexer Objekte nötig ist.



Repositories

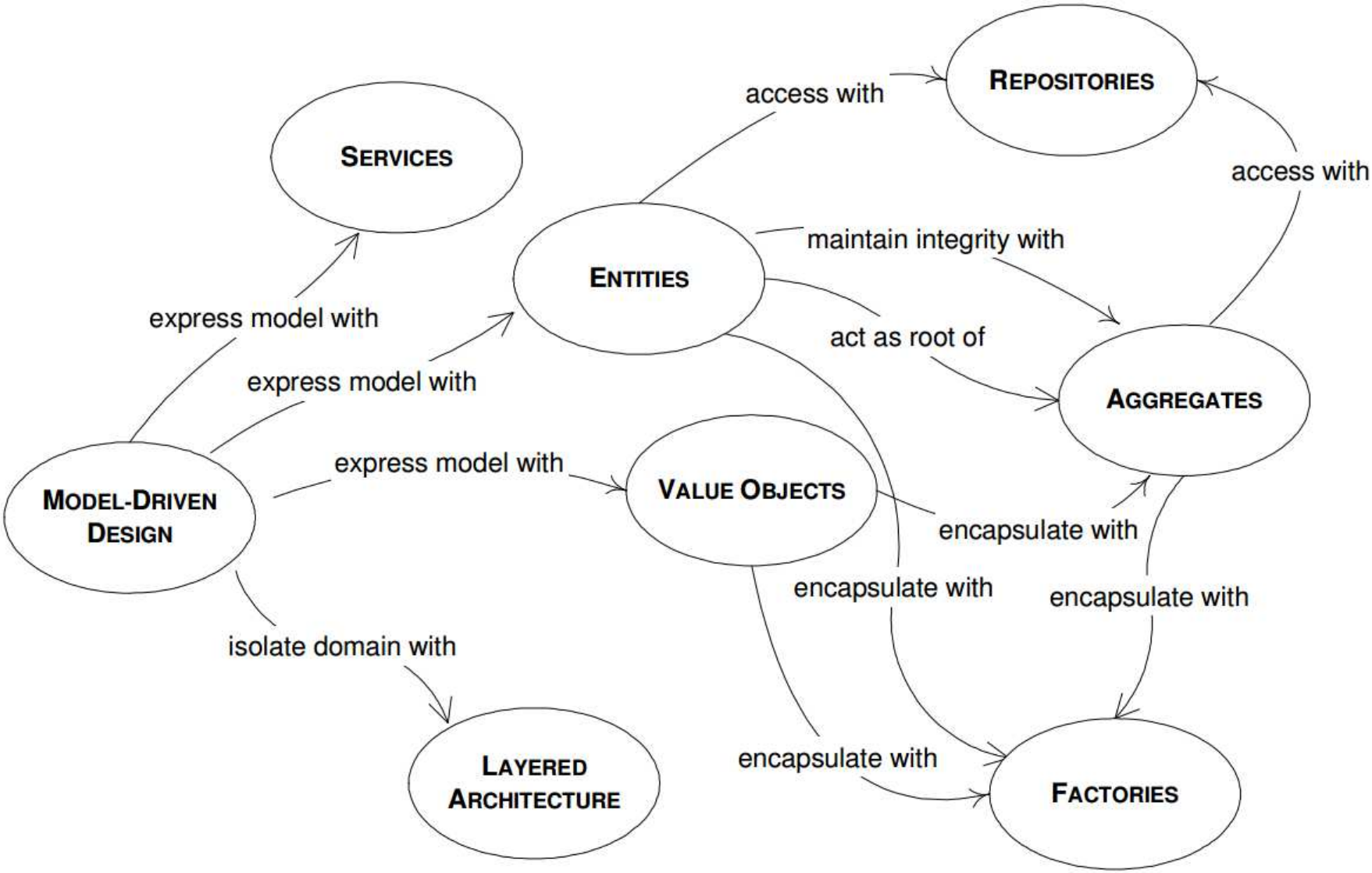
- Kapseln die Persistierung von Entities/Aggregates

IRepository<TEntity>
Generic Interface
IDisposable

Properties
- *UnitOfWork*

Methods
- *Add*
- *AllMatching*
- *Get*
- *GetAll*
- *GetFiltered*
- *GetPaged<KProperty>*
- *Merge*
- *Modify*
- *Remove*
- *TrackItem*

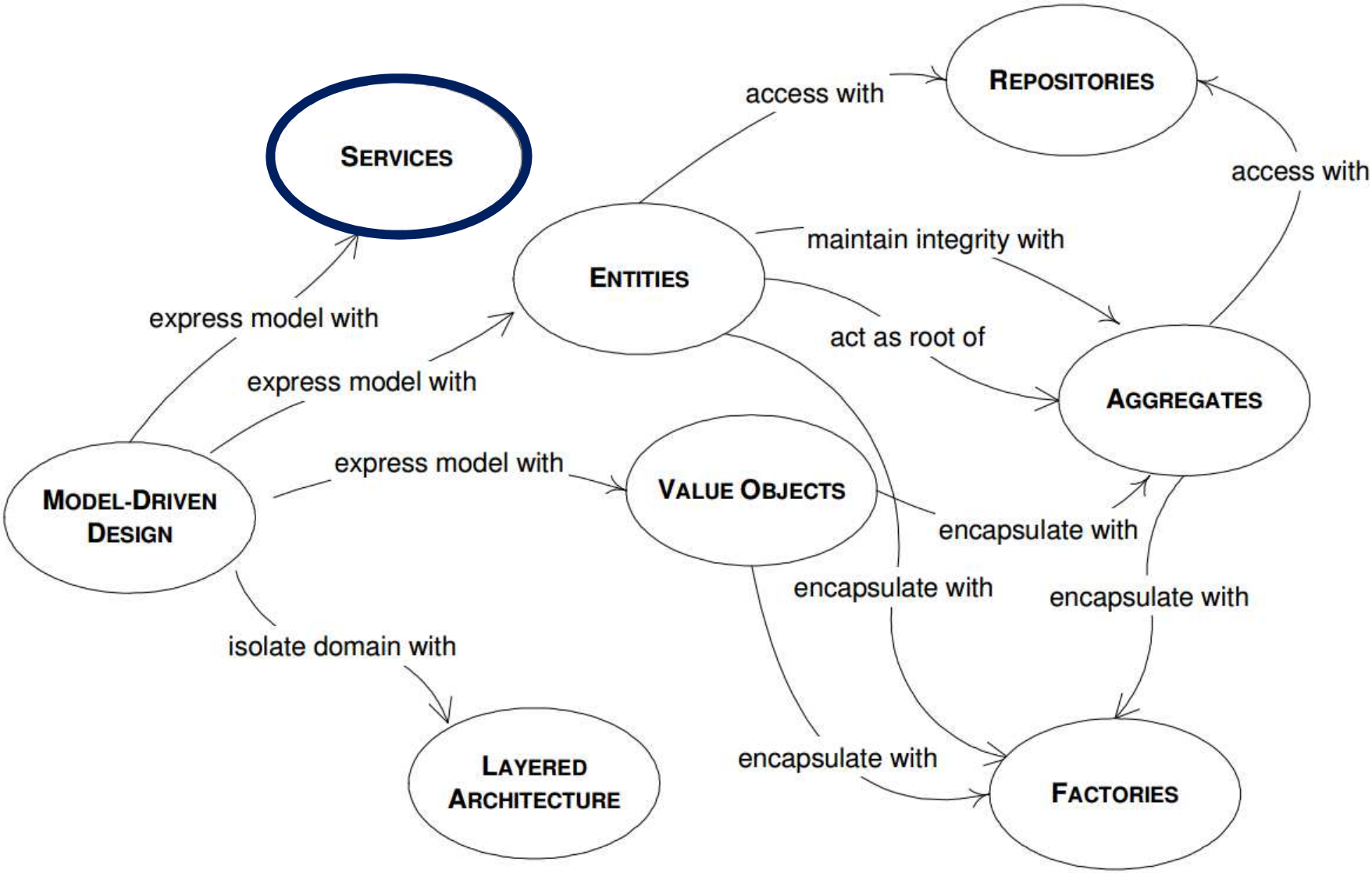
DDD Building Blocks



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

DDD Building Blocks



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Services

- Kapseln Logik, welche nicht bestimmten Entities zugewiesen werden kann.
- Stateless by Design
- Domain vs. Application vs. Infrastructure Services.

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

DDD Konzepte

- „Making implicit Concepts explicit“
- Supple Design
  - Intention-Revealing Interfaces
  - Side Effect Free Functions
  - Assertions (pre-, postconditions, invariants)
- Bounded Context / Context Map & Continuous Integration
- Distillation
  - Generic SubDomains
  - Domain Vision Statement.

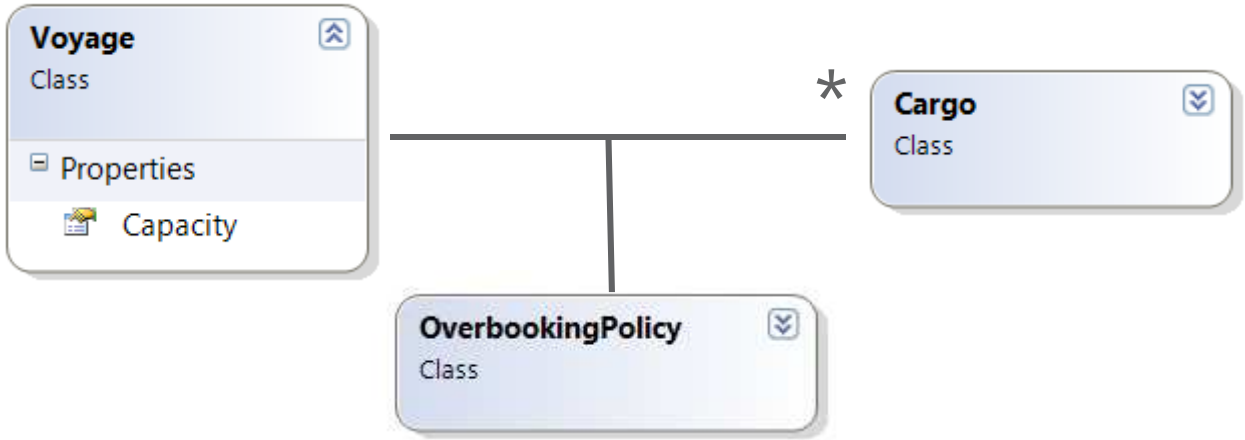
2014 © Trivadis

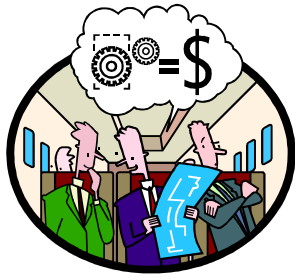
20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

DDD Konzepte

- „Making Implicit Concepts Explicit“

Die Capacity darf nicht überschritten werden!





Ubiquitous Language

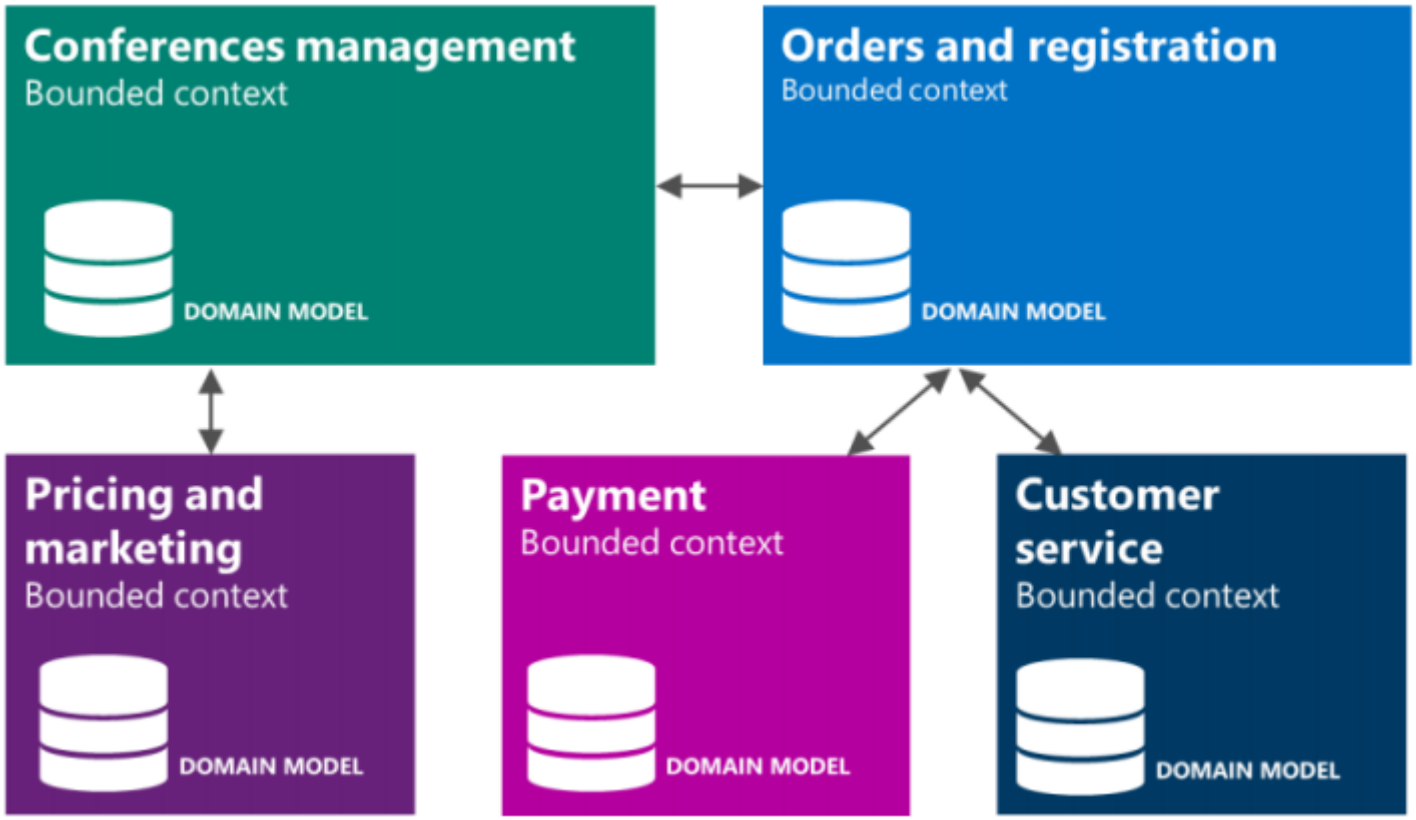
DDD Konzepte

Bounded Context/Context Map/Continuous Integration

- Definition des Kontexts, in welchem ein Model gültig ist.
- Grenzen: Teams, Teile der Applikation, Code, Infrastruktur
- Definition der Beziehungen zwischen Models
- Die Context Map zeigt das Big Picture
- Continuous Integration stellt sicher, dass die verschiedenen BCs zueinander passen

DDD Konzepte

Bounded Context/Context Map/Continuous Integration

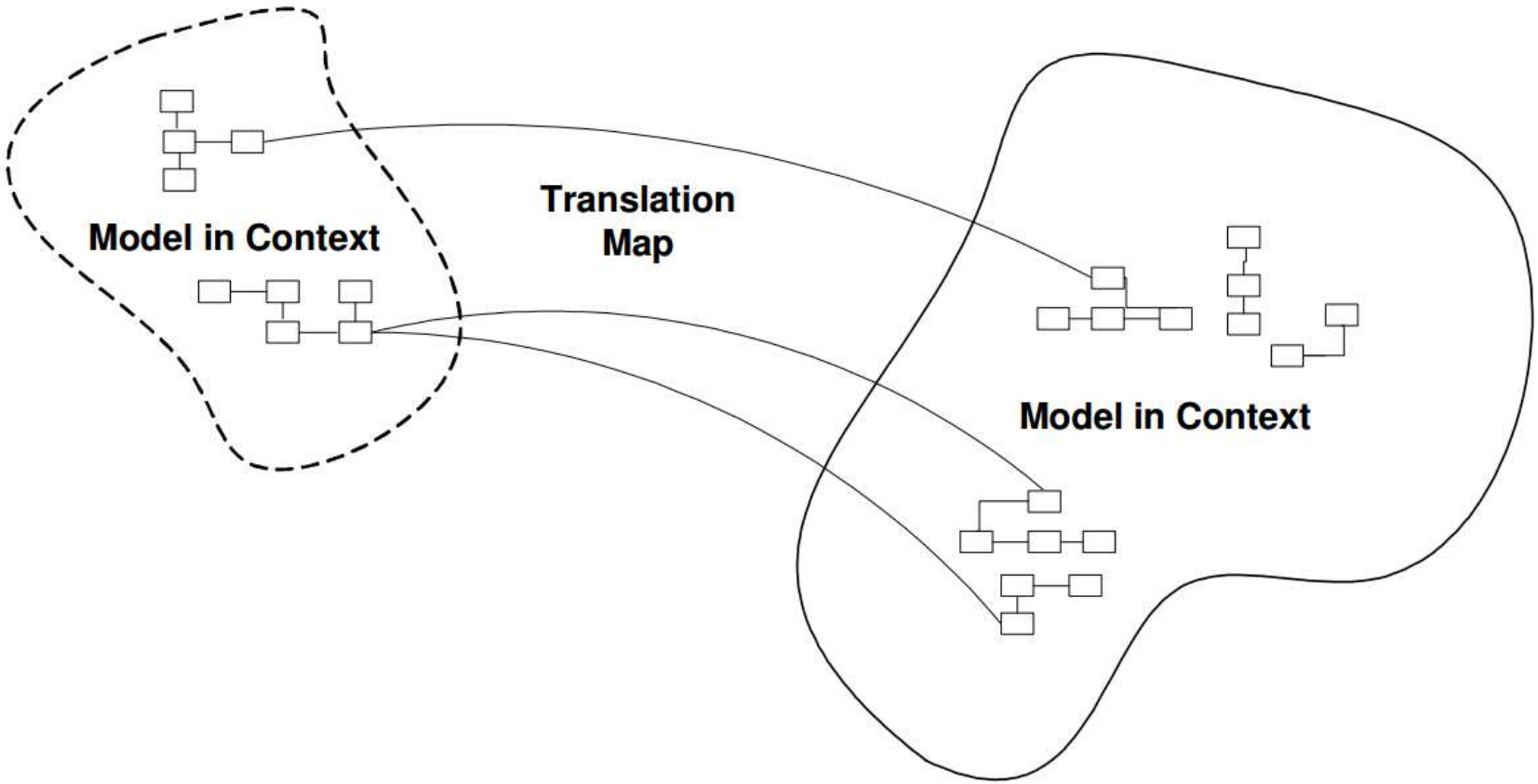


2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

DDD Konzepte

Context Map



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

DDD Konzepte: Distillation

- „Wie gewinnt man die Essenz?“
- Generische vs. Essentielle Konzepte
  - Generic SubDomains
- Dokumentation
  - Domain Vision Statement (1p)
  - Highlighted Core (3-7p).

2014 © Trivadis

4 DDD Best Practices von Eric Evans

1. Stay Hands-On: Modelers need to code!
2. Focus on concrete scenarios
3. Do NOT apply DDD to everything. Draw a context map!
4. Experiment a lot and make lots of mistakes.

DDD Anti-Patterns

- Anemic Domain Model

> “Ein Model, in welchem die Entitäten nur Daten transportieren”

- Smart UI
- Data Driven Design (Bottom-Up).

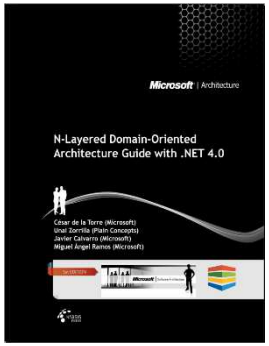
Von Data-Driven zu Domain-Driven in .NET

20 JAHRE TRIVADIS We love IT.

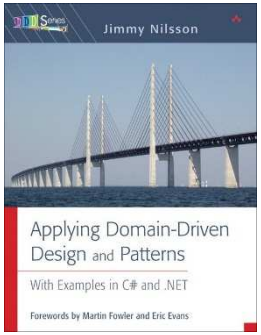
trivadis makes IT easier.

DDD in .NET

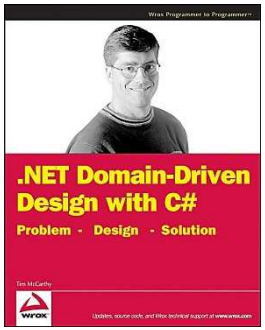
- N-Layered Domain-Oriented Architecture Guide with .NET 4.0
  - Cesar de la Torre, Architect Advisor, Microsoft



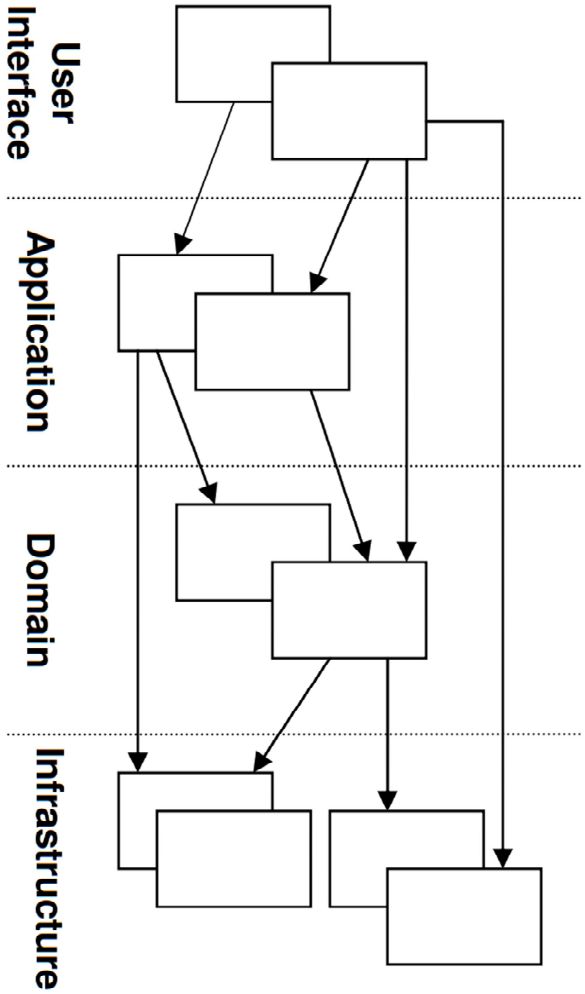
- Applying Domain Driven Design and Patterns
  - Jimmy Nilsson



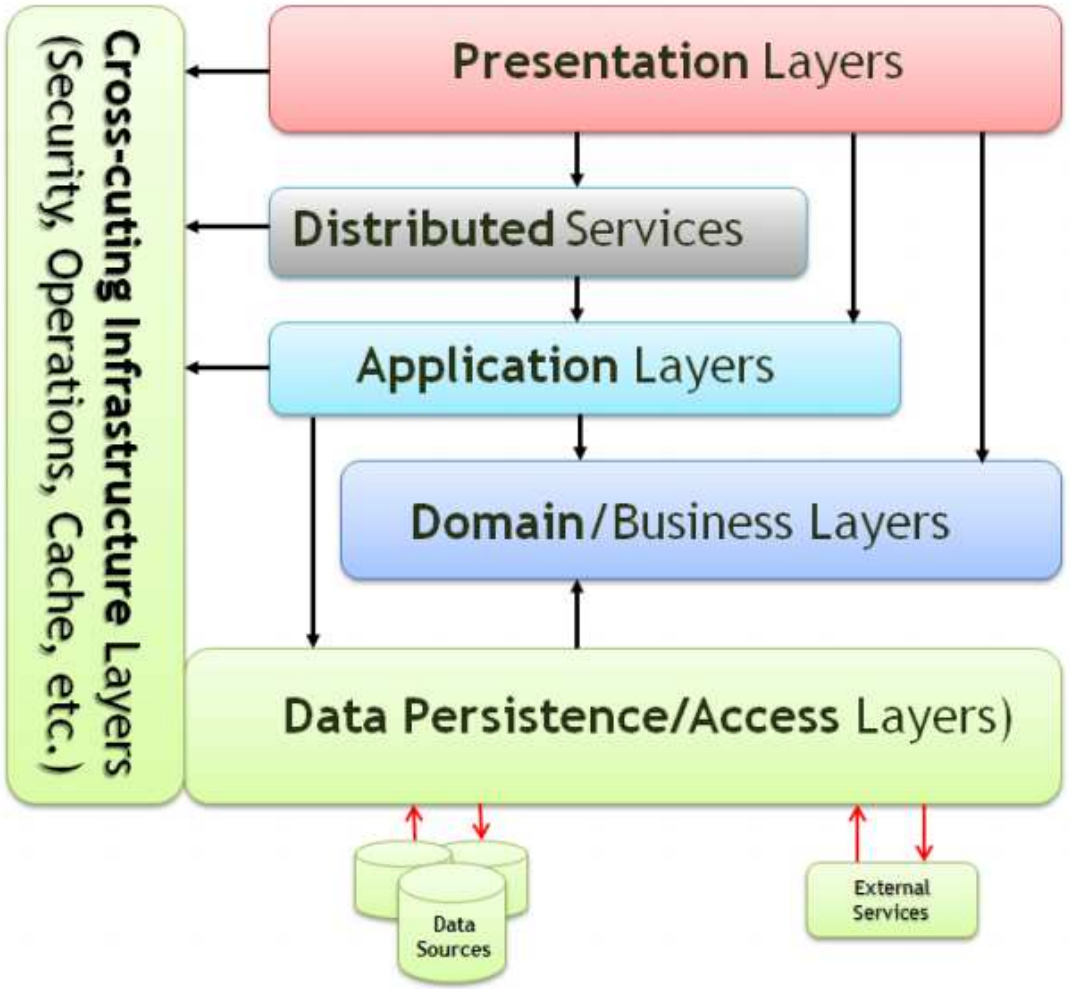
- .NET Domain-Driven Design with C#: Problem – Design – Solution
  - Tim McCarthy



N-Layered DDD Architecture



Eric Evans

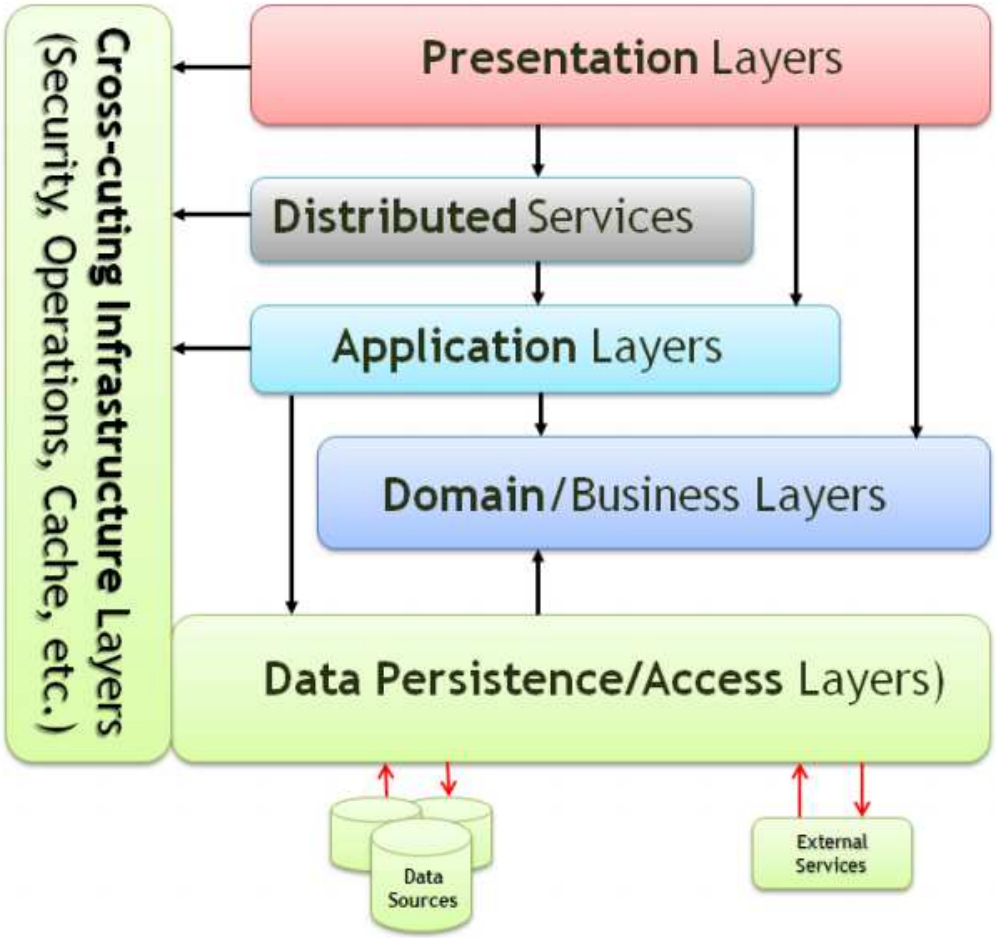


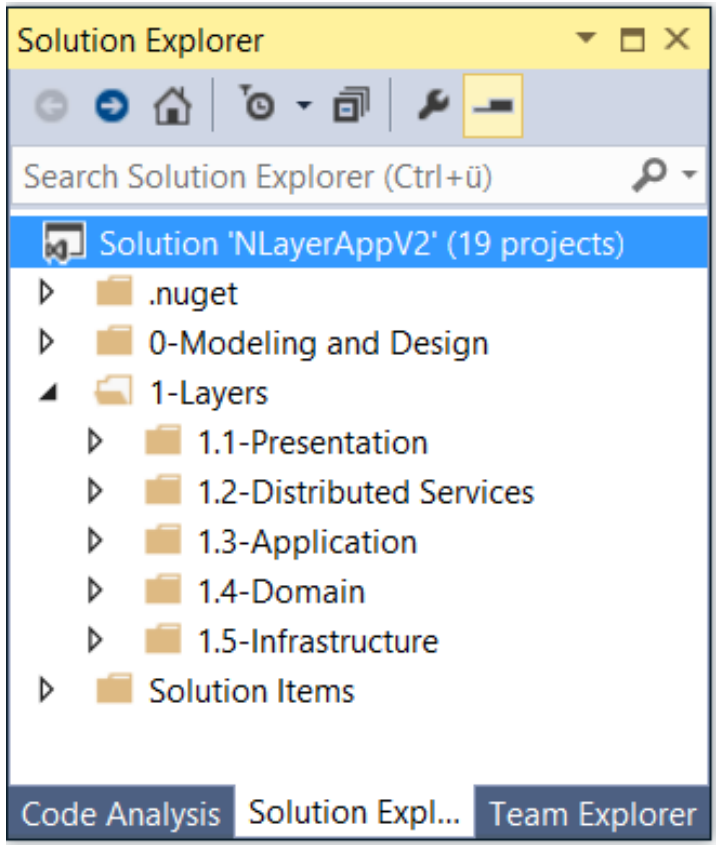
C. De la Torre

2014 © Trivadis

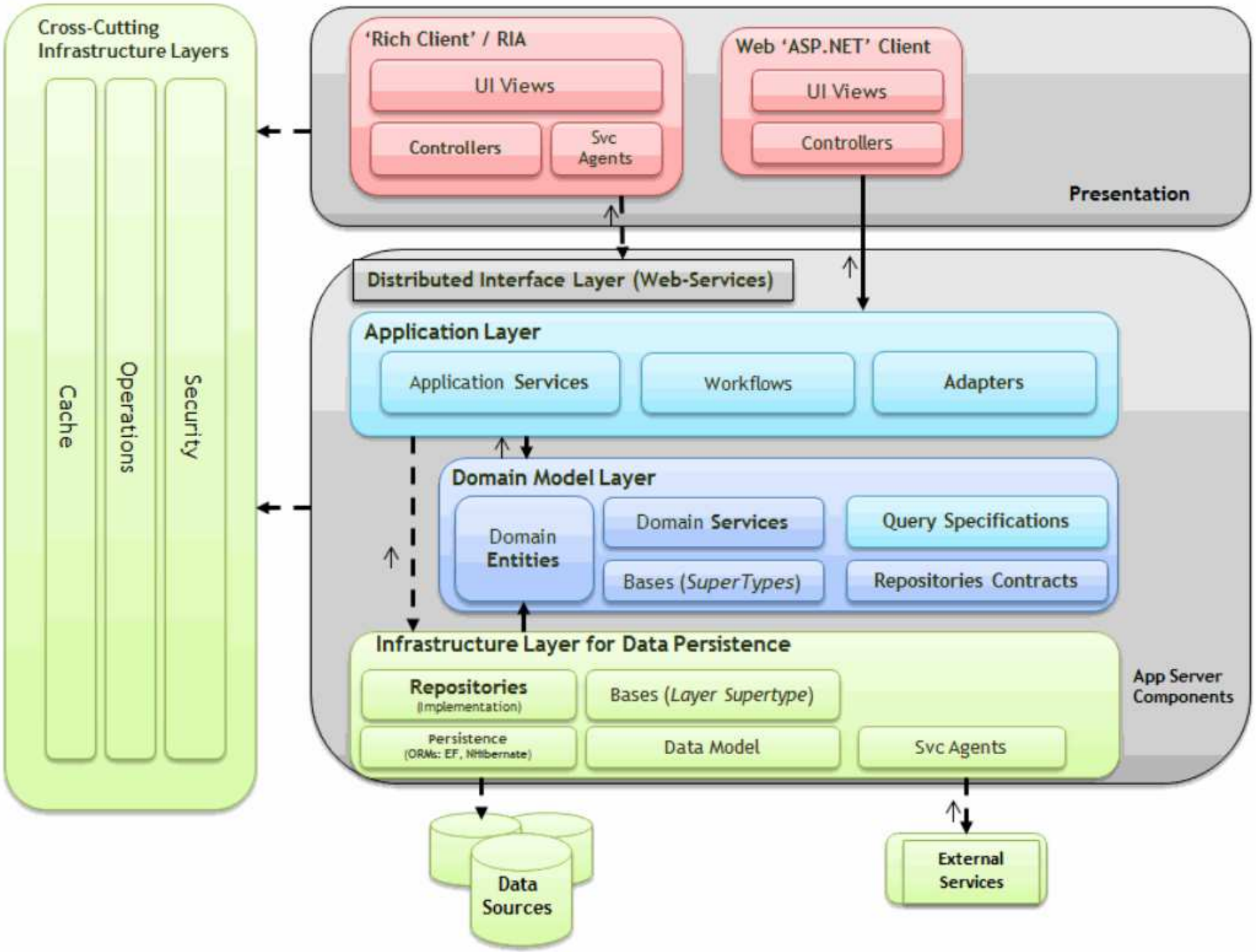
20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

N-Layered DDD Architecture

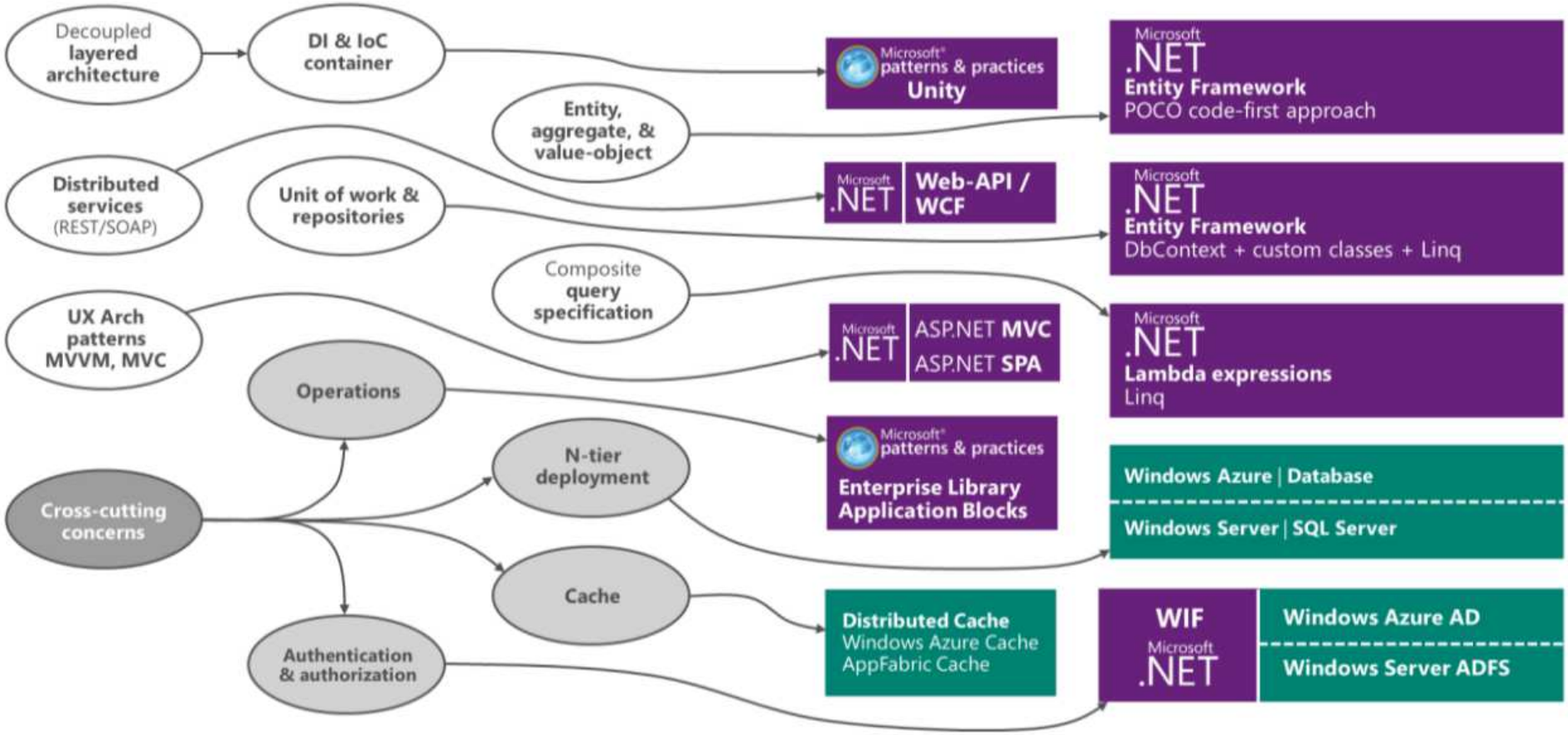




C. De la Torre


Cross-Cutting Infrastructure Layers
Cache
Operations
Security
'Rich Client' / RIA
UI Views
Controllers
Svc Agents
Web 'ASP.NET' Client
UI Views
Controllers
Presentation
Distributed Interface Layer (Web-Services)
Application Layer
Application Services
Workflows
Adapters
Domain Model Layer
Domain Entities
Domain Services
Query Specifications
Bases (SuperTypes)
Repositories Contracts
Infrastructure Layer for Data Persistence
Repositories
(Implementation)
Bases (Layer Supertype)
Persistence
(ORMs: EF, NHibernate)
Data Model
Svc Agents
App Server Components
Data Sources
External Services

N-Layered DDD: Technology Map



MS Technology Guide for Business Applications
(http://www.microsoft.com/net/nettechnologyguidance)

2014 © Trivadis

Exemplarisches Vorgehen (gemäss Tim McCarthy)

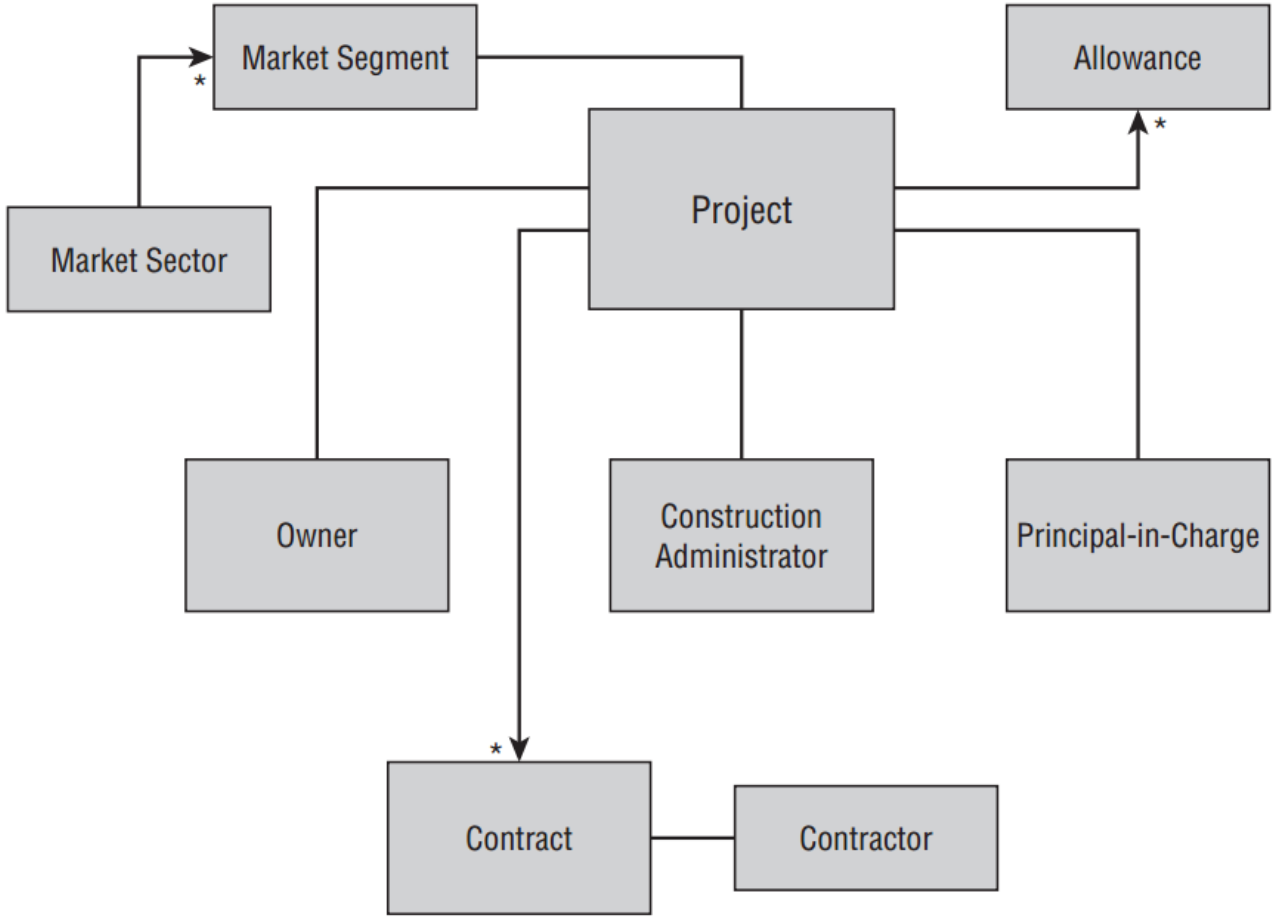
1. Das Domain Model designen
2. Die Aggregates bestimmen
3. Die Aggregate Boundaries bestimmen
4. Die Repositories designen
5. Tests dazu schreiben
6. Die Repositories implementieren.

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Exemplarisches Vorgehen (gemäss Tim McCarthy)

1. Das Domain Model designen

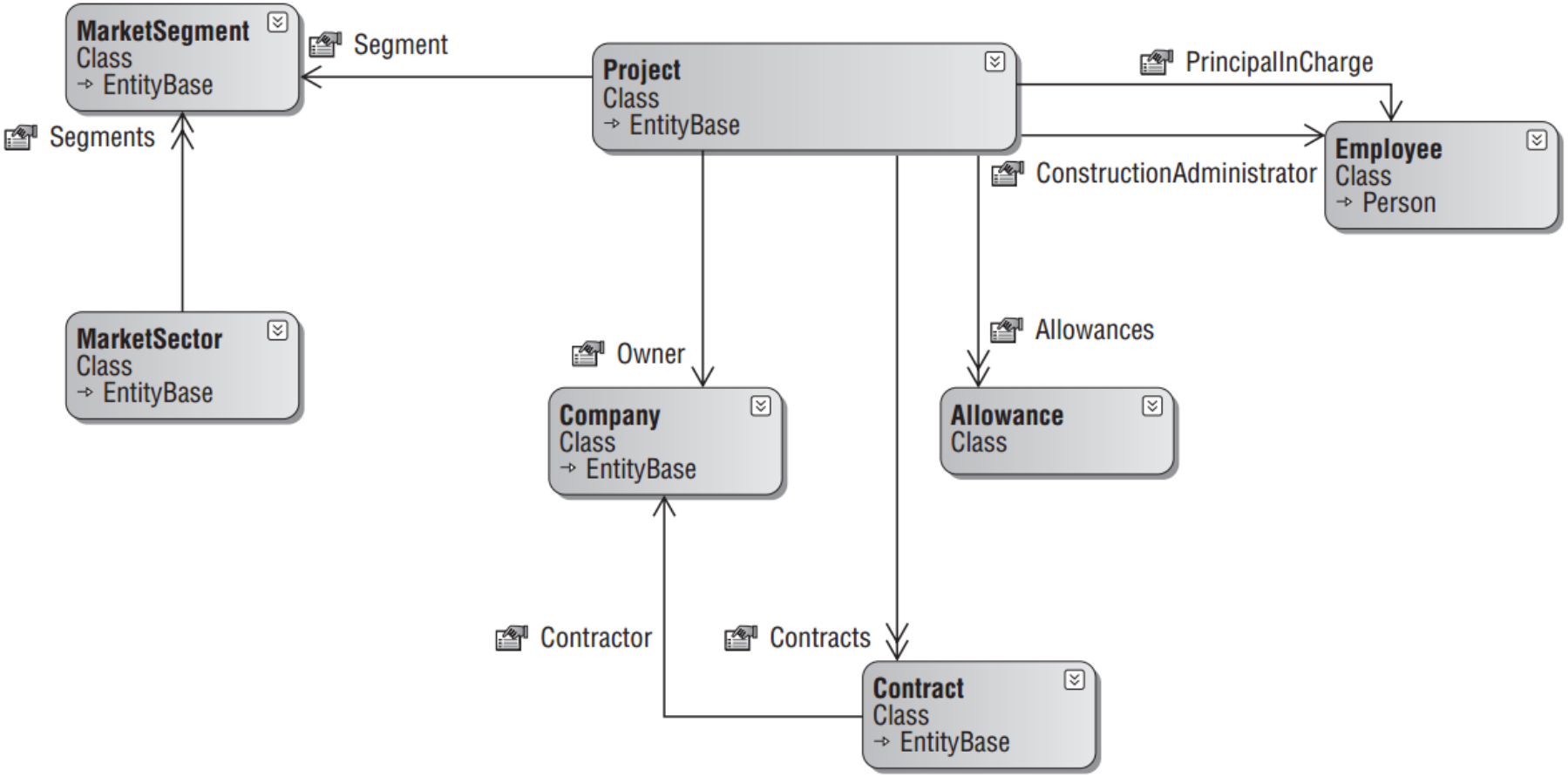


2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Exemplarisches Vorgehen (gemäss Tim McCarthy)

2. Die Aggregates bestimmen

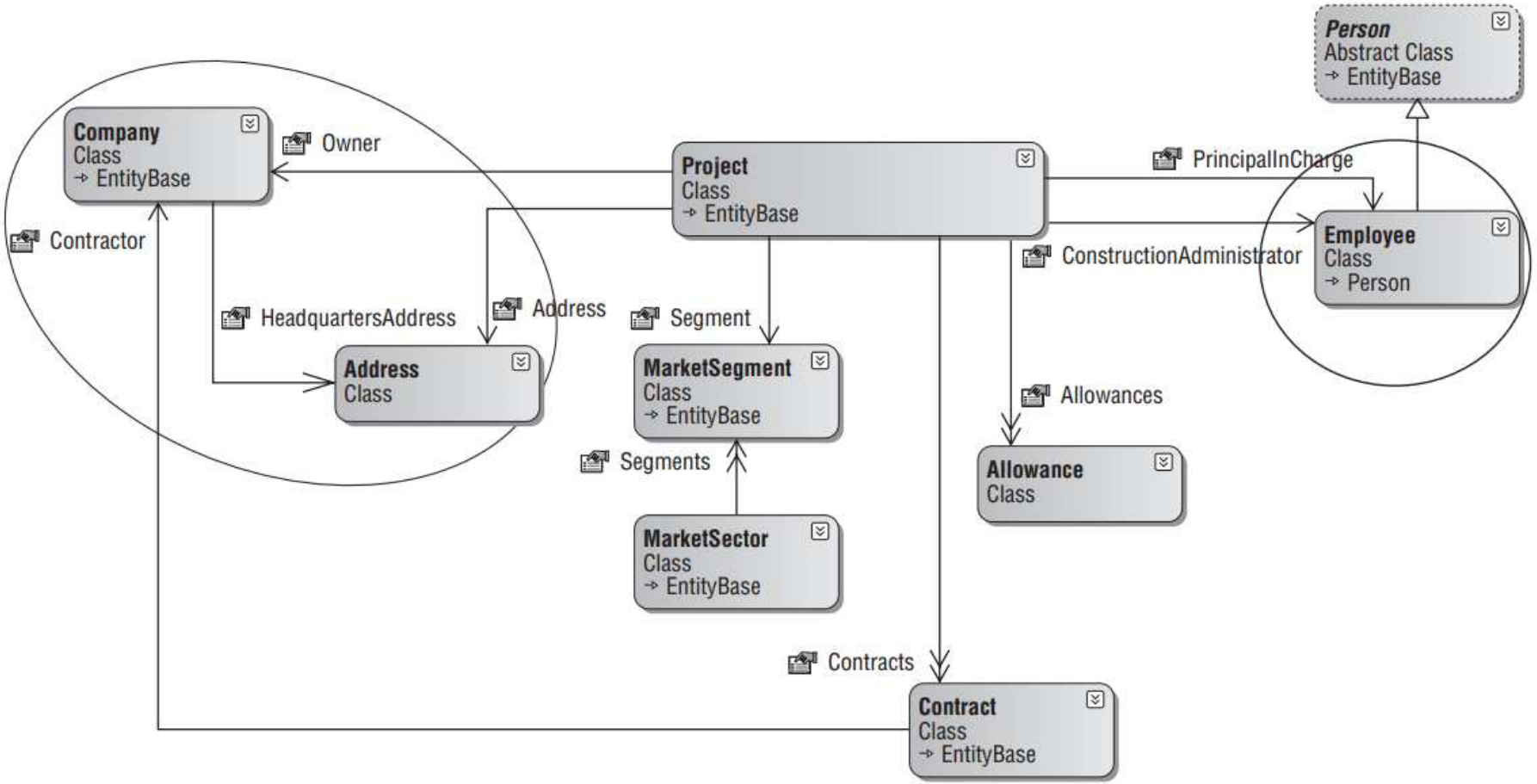


2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Exemplarisches Vorgehen (gemäss Tim McCarthy)

3. Die Aggregate Boundaries bestimmen

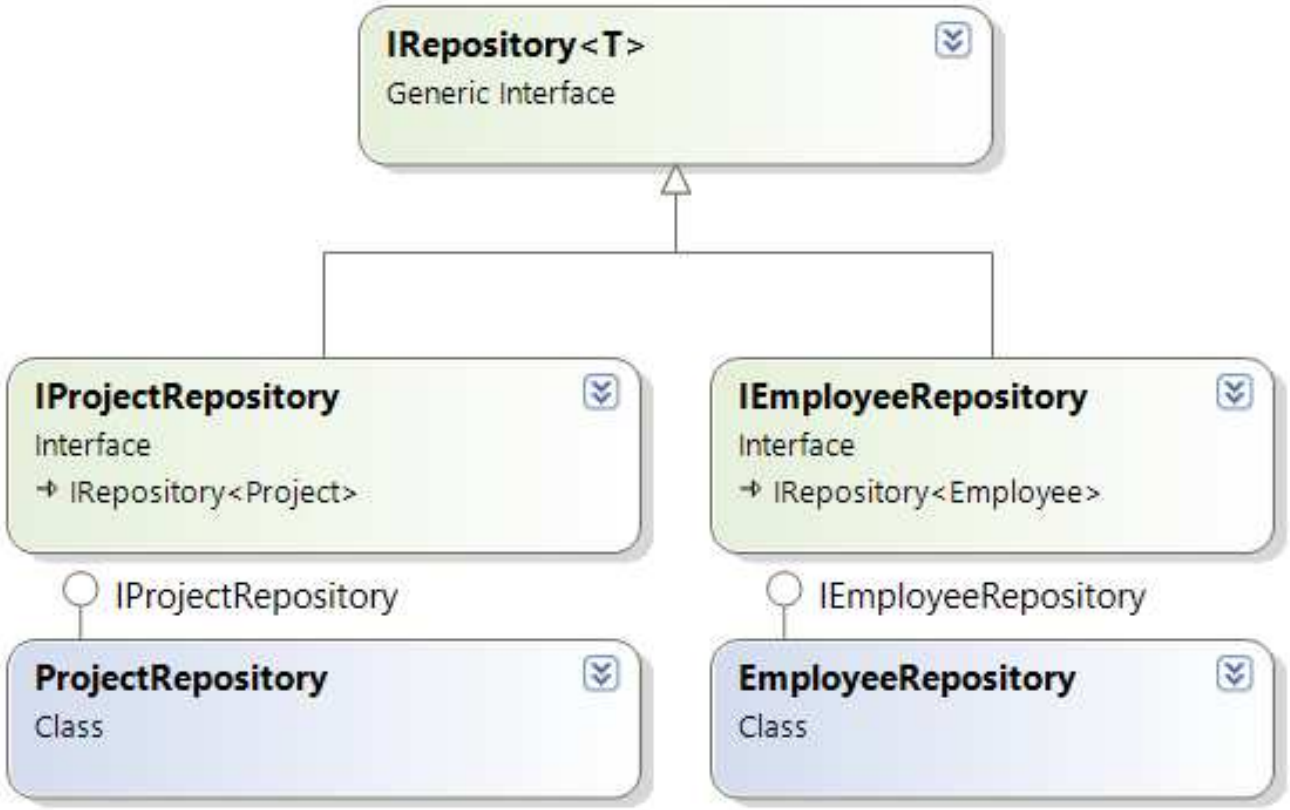


2014 © Trivadis

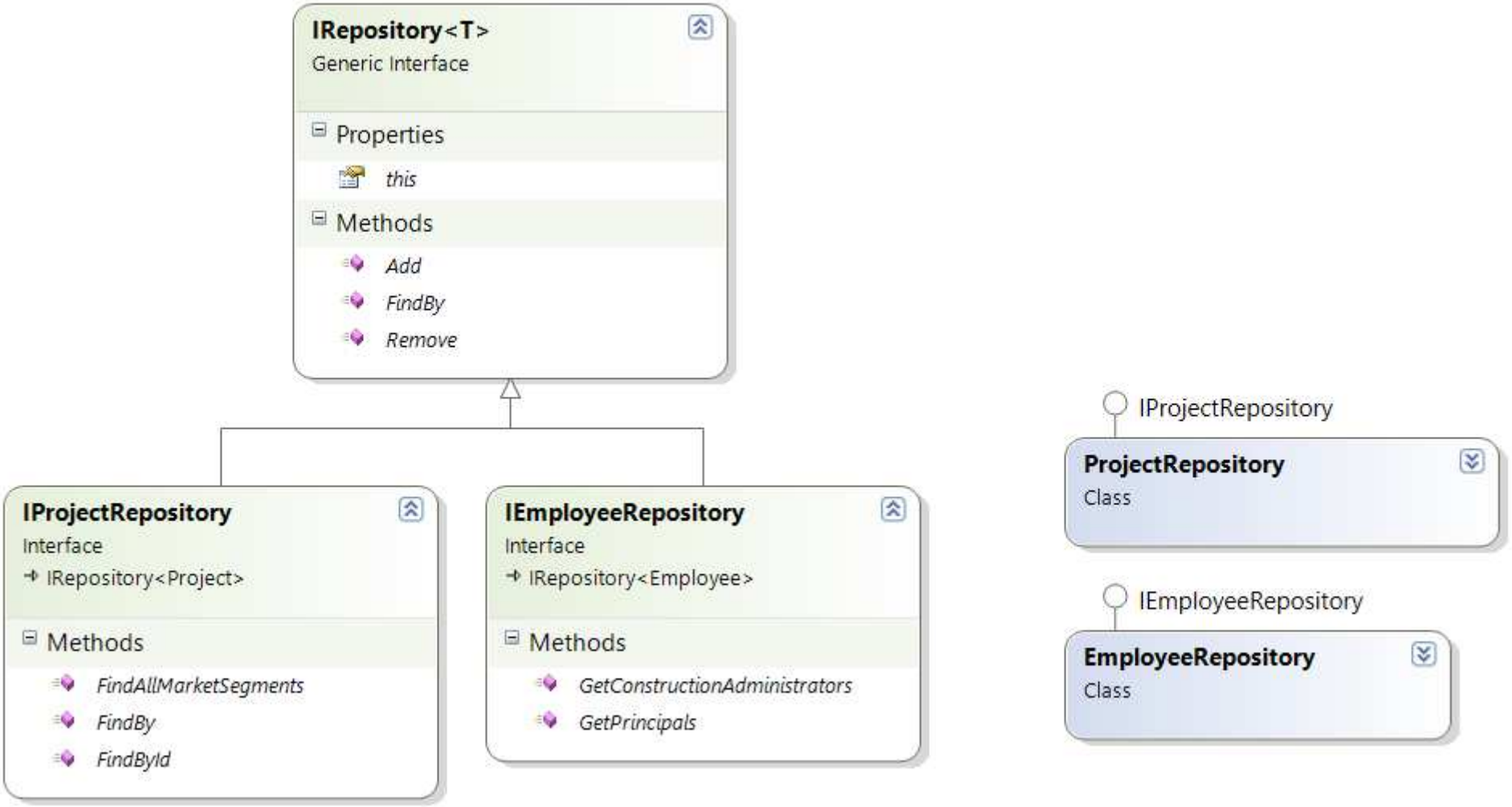
20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Exemplarisches Vorgehen (gemäss Tim McCarthy)

4. Die Repositories designen



Exemplarisches Vorgehen (gemäss Tim McCarthy)



2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

Exemplarisches Vorgehen (gemäss Tim McCarthy)

5. Tests dazu schreiben

```
/// < summary >
/// Use TestInitialize to run code before running each test
/// < /summary >
[TestInitialize()]
public void MyTestInitialize()
{
    this.unitOfWork = new UnitOfWork();
    this.repository = RepositoryFactory.GetRepository<IProjectRepository,
                                                       Project>(this.unitOfWork);
}

/// < summary >
///A test for FindBy(string projectNumber)
/// < /summary >
[TestMethod]
public void FindByProjectNumberTest()
{
    string projectNumber = "12345.00";
    Project project = this.repository.FindBy(projectNumber);
    Assert.AreEqual("My Project", project.Name);
}
```

DDD Praxis Tipps

- Beim designen von Entitäten den Datenzugriff ausblenden, später ein Tool wie z.B. EF Code First benützen für den Datenzugriff.
- Entities:
  - Property Setter privat markieren, Datenmodifikation über Methoden kontrollieren.
  - Instanzierung in Factories auslagern
  - Parameterlosen Konstruktor verstecken
- Nicht essentielle Aufgaben über Bounded Context auslagern und z.B. mit simplem CRUD oder gar OTS lösen.
- Simple Domänenobjekte identifizieren und mit Value Objekts abbilden
- Beziehungen identifizieren und vereinfachen.

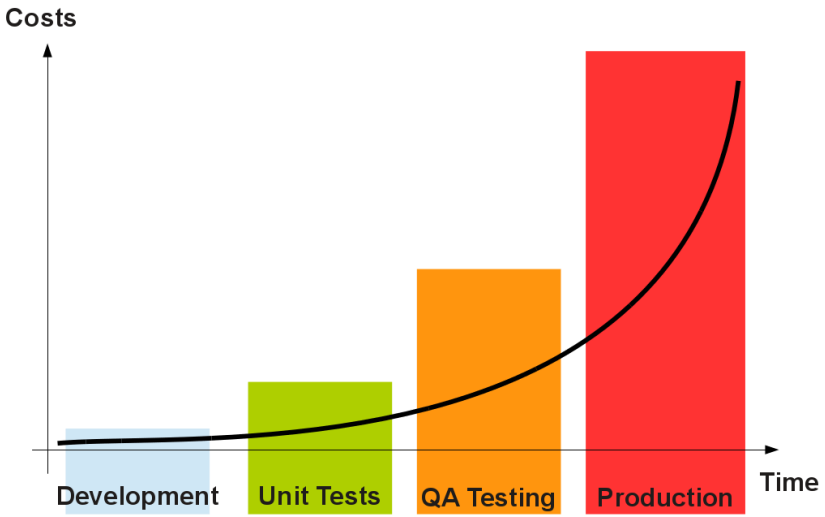
20 JAHRE TRIVADIS We love IT.

trivadis makes IT easier.

Zusammenfassung

Zusammenfassung

- Kosten & Legacy

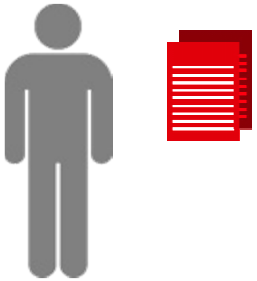


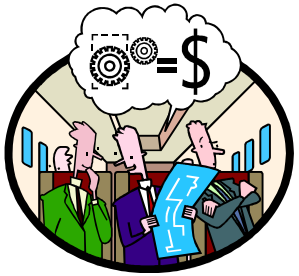
20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

SOLID Principles

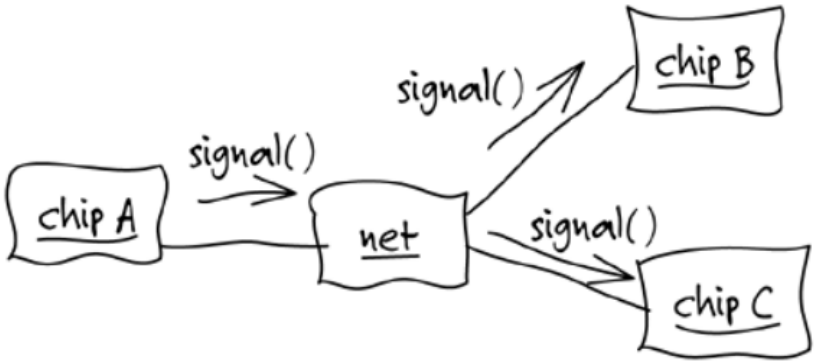
- **S**RP: Single Responsibility Principle
- **O**CP: Open Closed Principle
- **L**SP: Liskov Substitution Principle
- **I**SP: Interface Segregation Principle
- **D**IP: Dependency Inversion Principle

Domain Driven Design





Ubiquitous Language

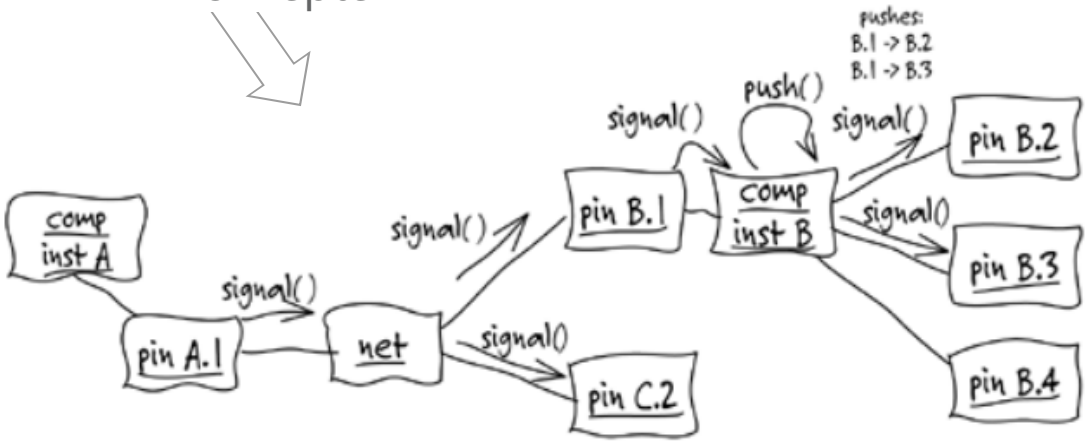


"Make implicit concepts explicit»

Supple Design

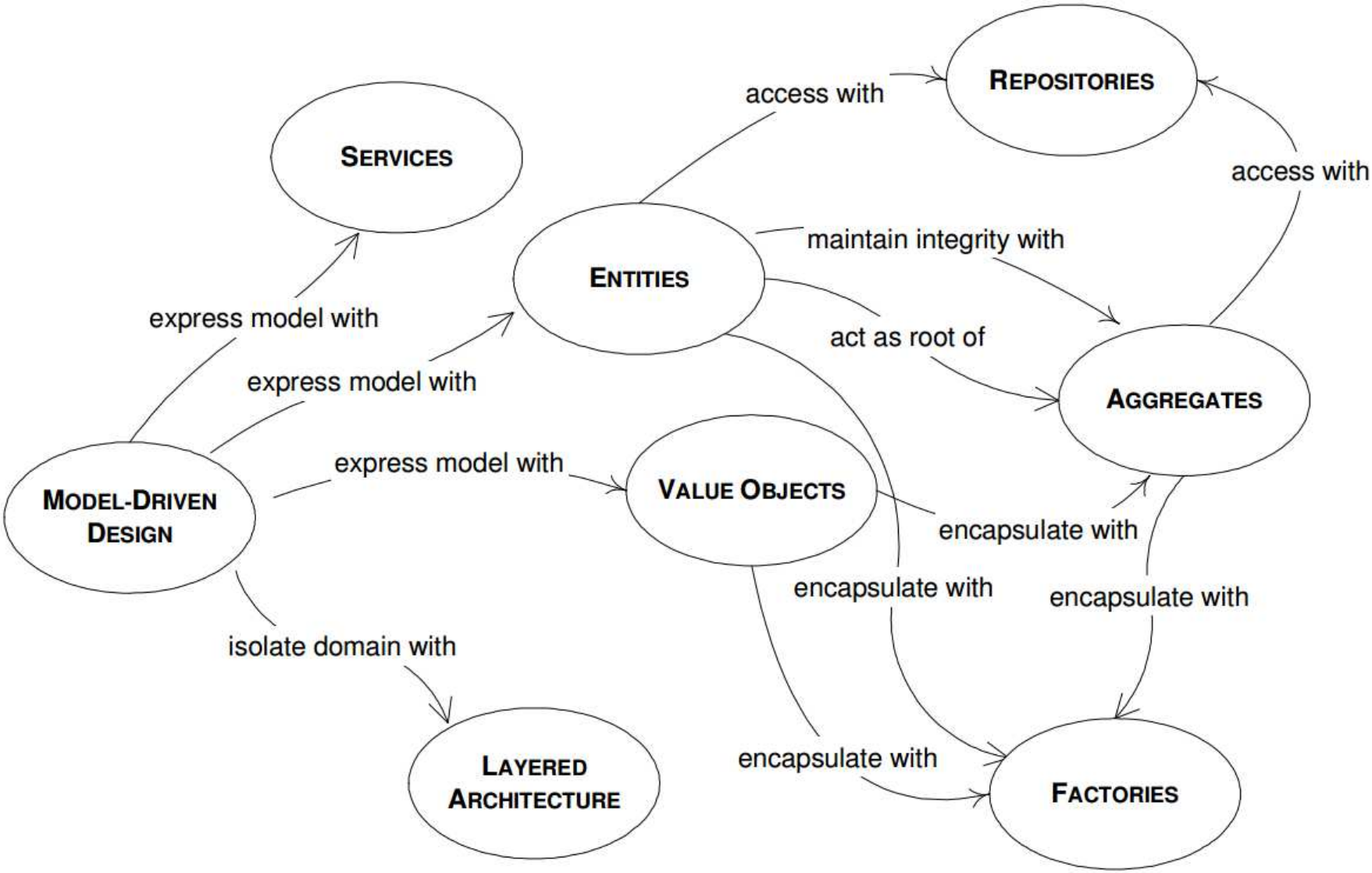
Persistence Ignorance.

DDD Konzepte



Knowledge-Rich Model

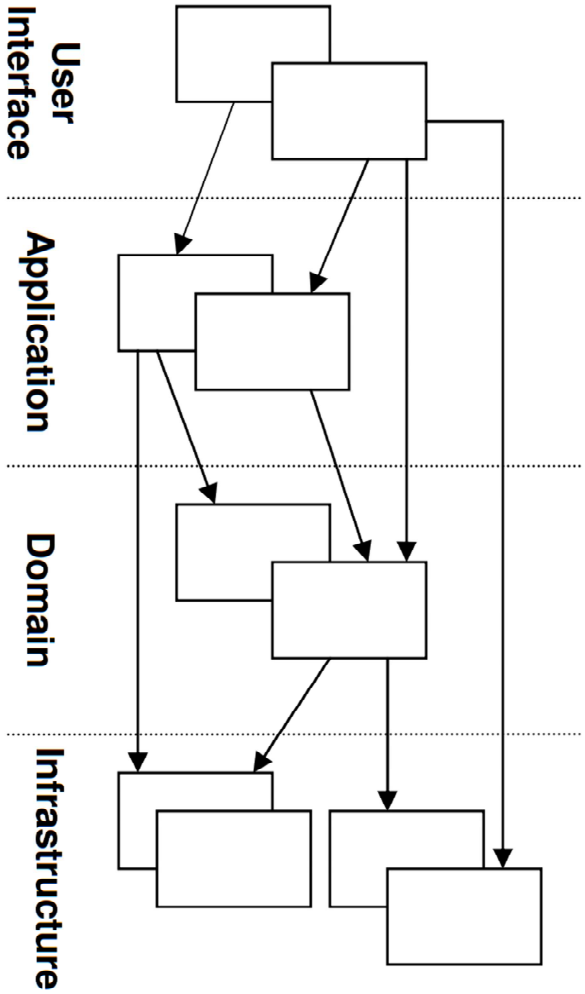
DDD Building Blocks



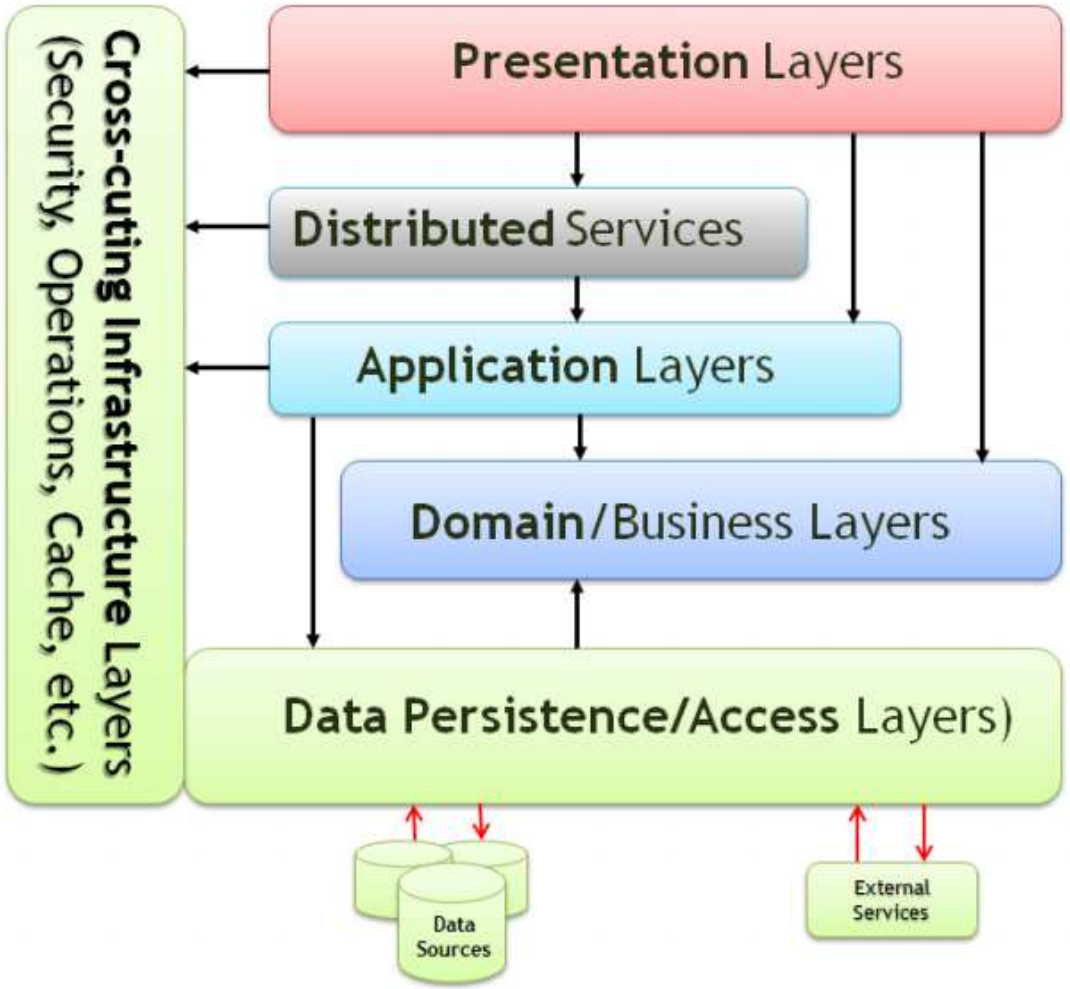
2014 © Trivadis

20 JAHRE TRIVADIS We love IT.
trivadis makes IT easier.

N-Layered DDD Architecture

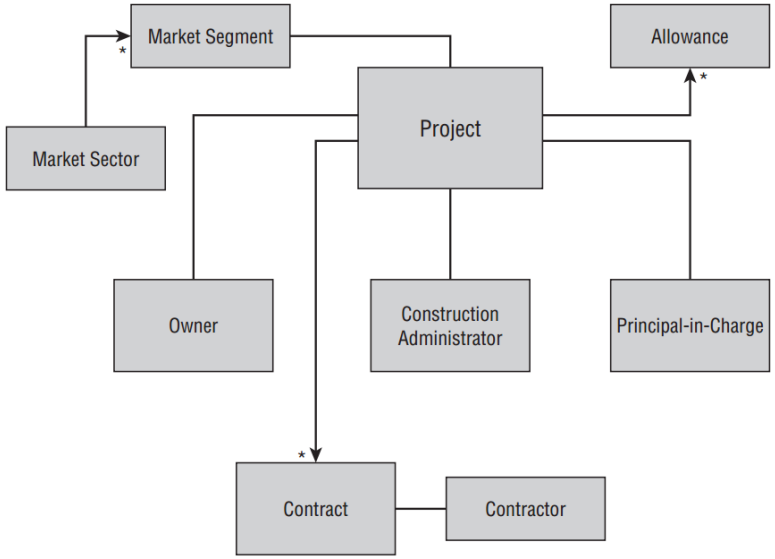


Eric Evans



C. De la Torre

Exemplarisches Vorgehen (gemäss Tim McCarthy)

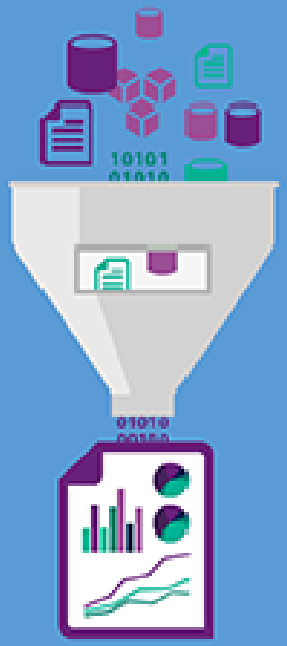



```
/// < summary >
/// Use TestInitialize to run code before running each test
/// < /summary >
[TestInitialize()]
public void MyTestInitialize()
{
    this.unitOfWork = new UnitOfWork();
    this.repository = RepositoryFactory.GetRepository<IProjectRepository,
                                                       Project>(this.unitOfWork);
}

/// < summary >
///A test for FindBy(string projectNumber)
/// < /summary >
[TestMethod]
public void FindByProjectNumberTest()
{
    string projectNumber = "12345.00";
    Project project = this.repository.FindBy(projectNumber);
    Assert.AreEqual("My Project", project.Name);
}
```

Zusammenfassung

.NET Software-Architektur
- S.O.L.I.D. Prinzipien
- Domain Driven Design

Enterprise-Lösungen
- Wartbarkeit
- Erweiterbarkeit



Vielen Dank!

Manuel Meyer

Senior Consultant Trivadis AG

manuel.meyer@trivadis.com

BASEL BERN BRUGG LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

2014 © Trivadis

20 JAHRE TRIVADIS We love IT.

trivadis makes IT easier.