5.– 8. September 2011
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

3D

Domain Driven Design on top

Tim de Buhr

Open knowledge GmbH

offenkundiggut

Agenda

1. Agile Entwicklung - Short backup
2. Domain Driven Design - Grundlagen
3. On Top - DDD im Entwicklungsprozess
4. Domain Driven Design - Modellieren
5. On Top - DDD im Code

Agile Entwicklung

Ziele:

- Schärfung/Fokussierung auf Kundenbedürfnisse
- Flexibler und schlanker Prozess
- Frühzeitige Risikoerkennung
- Hohe Qualität

Agile Entwicklung

Stichwort:

- Iterativer Prozess (Sprints)
- Daily Scrum
- Product Backlog
- Sprint Backlog

Agenda

1. Agile Entwicklung - Short backup
2. Domain Driven Design - Grundlagen
3. On Top - DDD im Entwicklungsprozess
4. Domain Driven Design - Modellieren
5. On Top - DDD im Code

Domain Driven Design

„For most software projects, the primary focus should be on the domain and domain logic“

„Complex domain design should be based on a model“

Eric Evans, Domain-Driven Design, Addison-Wesley, © Eric Evans, 2004

Begriffsdefinitionen

- Domäne
- Ubiquitous Language – Gemeinsame Sprache

Ubiquitous Language – Gemeinsame Sprache

- Fachsprache als Basis
- Eindeutige Begriffe für Domänen-Konzepte
- Verbindliche Nutzung der Sprache

Begriffsdefinitionen

- Domäne
- Ubiquitous Language – Gemeinsame Sprache
- Domänen-Modell

Domänen-Modell

- Objekte sind Teil der gemeinsamen Sprache
- Nomen der Sprache entsprechen i.d.R. Objekten
  →Es gibt für jeden Sachverhalt genau **einen** Begriff
- Änderungen der Sprache sind Änderungen im Modell
- Alle Beteiligten entwickeln das Modell weiter

Domänen-Modell

„Mit einer gemeinsamen Sprache ist das Modell nicht mehr nur ein Design-Artefakt oder eine technische Anbindung an die Datenbank.

Die Sprache wird integraler Bestandteil von allen Aktivitäten von Entwicklern und Domänen-Experten.“

Konzepte

- Kontinuierliches Lernen
- Austausch mit Domänen-Experten
- Knowledge Crunshing
- Breakthrough

Modellgetriebenes Design

- Verwendung des Modells in allen Bereichen
- Das Modell muss
  - bei der Implementierung einfach zu verwenden sein
  - die gemeinsame Sprache widerspiegeln
- Probleme bei der Verwendung deuten auf konzeptionelle Probleme hin

Modellgetriebenes Design

*„Das Domänen-Modell ist das **Bindeglied** zwischen den Domänen-Experten und den Entwicklern“*

Modellierung vs. Implementierung?

- Enge Kopplung zwischen Code und Modell
  - Probleme bei der Verwendung führen zu tieferer Einsicht in die Domäne
  - Berücksichtigung technischer Einschränkungen bei der Modellierung
- Modellierung und Implementierung Hand in Hand

  →Jeder Designer des Modells sollte auch einmal mit Code zu tun haben

  →Jeder Entwickler muss sich für das Modell verantwortlich fühlen

  →Jeder Entwickler muss in die Entwicklung des Modells bis zu einem gewissen Grad eingebunden sein

Modellierung vs. Implementierung?

*„Der Austausch zwischen Domänen-Experten und Entwicklern ist **bidirektional** und muss von beiden Seiten gestartet werden“*

Agenda

1. Agile Entwicklung - Short backup
2. Domain Driven Design - Grundlagen
3. On Top – DDD im Entwicklungsprozess
4. Domain Driven Design - Modellieren
5. On Top - DDD im Code

Design

- Investition nicht nur langfristig
- häufig der schnellste Weg zum Ziel
  - Den Überblick behalten
  - Strategische Wichtigkeit von Teilen erkennen
    - Wer benutzt diesen Teil
    - Wie häufig ändert sich dieser Teil?

Modellierung

- Domain Driven Design startet nicht mit einem fertigen Modell
- Das Modell wächst iterativ mit dem Projektfortschritt
- Ein gutes Modell ist die Basis für eine langfristig wartbare und erweiterbare Software

Fachlichkeit

- keine Fragmentierung der Software
- einheitliches Benutzergefühl
- Koordination und Konsistenz von User-Storys

Zeithorizont

Was ist der Fokus agiler Software-Entwicklung?

- Umsetzung einer User-Story?
- Umsetzung einer sinnvollen Menge an User-Stories für das Release am Ende des Sprints?

→ Liefere am Ende des Sprints ein Release ab, auf dessen Basis die Software für weitere Releases erweitert werden kann

Risiko-Management

- Große, aber billige Schritte (am White-Board)
- Unterscheidung zwischen Exploration und Entscheidung
- Iterative Modell-Entwicklung
- Code-Probe (Spike)

Refactoring

Agile Entwicklung

- Wenn sich eine Anforderung nicht umsetzen lässt, wird refaktoriert

Domain Driven Design

- Refaktorierung bereits, wenn Probleme mit dem Modell festgestellt werden
  - nicht handhabbar
  - passt nicht zur Fachlichkeit

→ Je früher refaktoriert wird, desto billiger wird es

Planung

- Release-Planung
  - Initiales Domänen-Modell in Iteration Zero
  - Refactoring in jeder Iteration
- Sprint-Planung
  - Etablieren einer gemeinsamen Sprache beim Schreiben der User-Stories
  - Domänen-Zusammenhänge bei Bewertung berücksichtigen

Umsetzung

- Sprint-Durchführung
  - Kurze Design-Phase einplanen
  - User-Stories werden in Verbindung gesetzt
- Daily Scrum
  - Etablieren der gemeinsamen Sprache
  - Ansprechen von Problemen mit dem Modell

Vorteile von Domain Driven Design im agilen Entwicklungsprozess

- Gesamtkonzept (einheitliches Benutzergefühl)
- Koordination und Konsistenz von User-Stories
- Verhindern von Fragmentierung und Doppelentwicklung
- Trennung von „Exploration“ und Entscheidung
- Verbesserung von Risiko-Management
  - Durch früheres und weitgehenderes Refaktorieren
  - Weniger Gefahr einer „Sackgasse“

Agenda

1. Agile Entwicklung - Short backup
2. Domain Driven Design - Grundlagen
3. On Top - DDD im Entwicklungsprozess
4. Domain Driven Design - Modellieren
5. On Top - DDD im Code

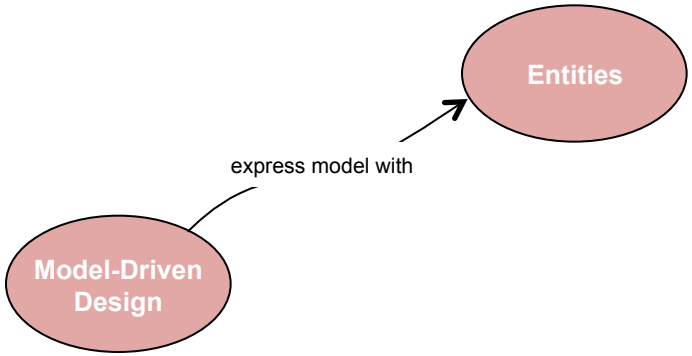
Modellierungsbausteine

- Jedes Objekt im Domänen-Modell hat eine bestimmte Rolle
- Je nach Rolle unterscheidet sich die Art der Umsetzung
- Jede mögliche Umsetzungsart erhält einen klar definierten Namen

Modellierungsbausteine

No more
- Util
- Helper
- Manager
- …

Modellierungsbausteine



Eric Evans, Domain-Driven Design, Addison-Wesley, © Eric Evans, 2004

Entitäten 1/2

- Kontinuität über einen Zeitraum
- Existenz einer eindeutigen Identität (z.B. User-Name)
  →künstlicher Schlüssel sinnvoll
- Identität bleibt konstant
- Änderung der Attribute ändern die Identität nicht

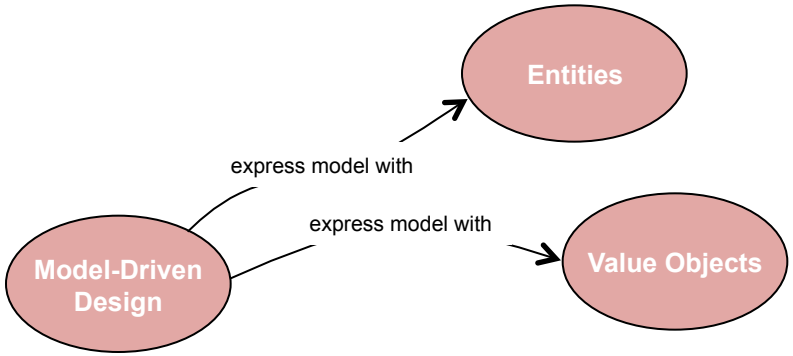
Entitäten 2/2

- Gleichheit von Entitäten
  - Entitäten mit gleichen Attributen sind nicht zwingend gleich
  - Entitäten mit verschiedenen Attributen können dennoch identisch sein

  →Vorsicht bei Operationen, die Entitäten über Attribute vergleichen

- Die Daten einer Entität sind zu jedem Zeitpunkt konsistent

Modellierungsbausteine



Eric Evans, Domain-Driven Design, Addison-Wesley, © Eric Evans, 2004

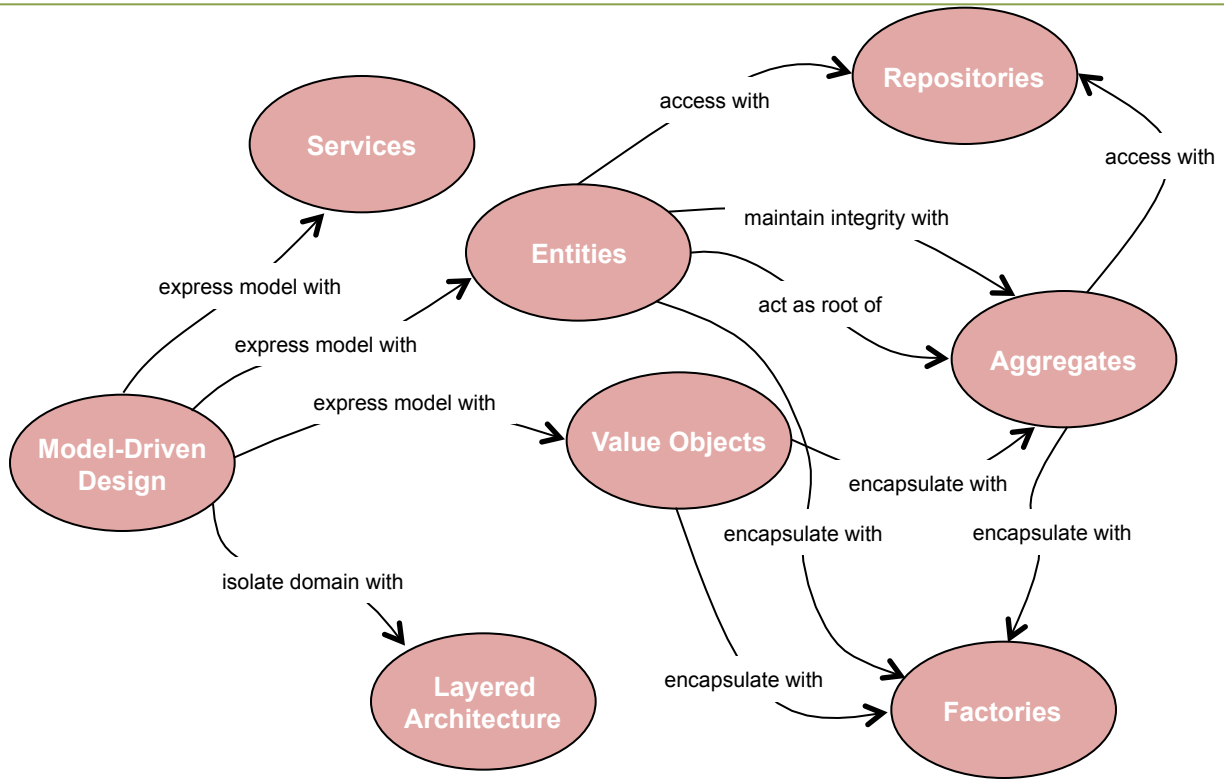
Value Objects 1/2

- Keine konzeptionelle Identität
- Beschreibung von Charakteristika
- Gleichheit ist über die Gleichheit der Attribute definiert
- Value Objects sind immutable
- Die Daten eines Value Objects sind zu jedem Zeitpunkt konsistent

Value Objects 2/2

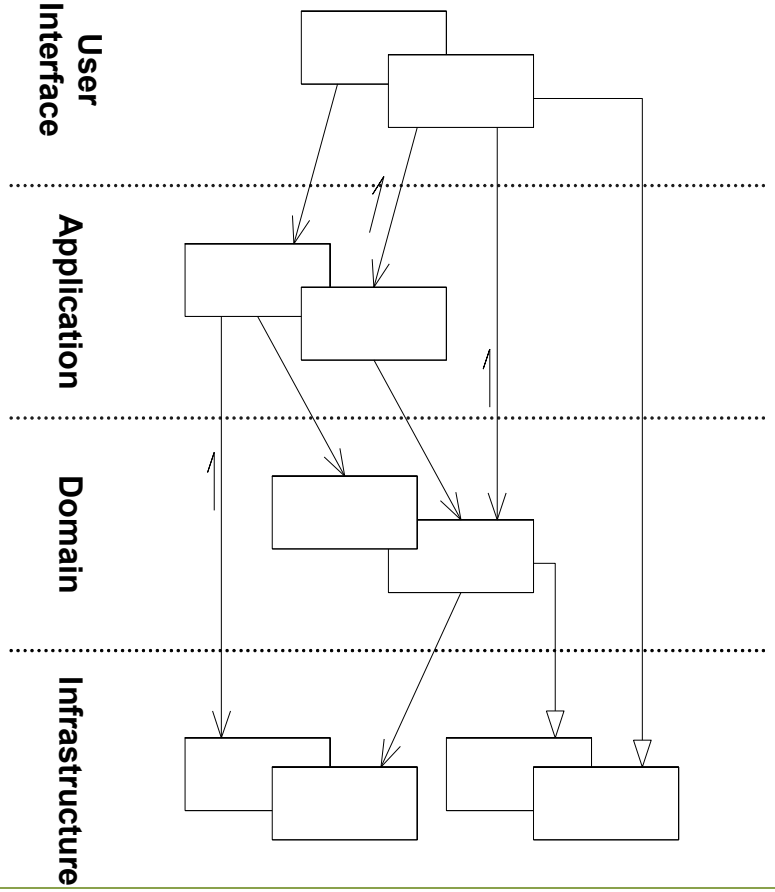
- Operationen auf Value Objects
  - In sich abgeschlossen
  - Rückgabe-Wert ist eine neue Instanz desselben Value Objects, das den neuen Wert enthält
- Beispiele
  - java.lang.String
  - java.math.BigInteger
  - java.math.BigDecimal

Modellierungsbausteine



Eric Evans, Domain-Driven Design, Addison-Wesley, © Eric Evans, 2004

Layered Architecture

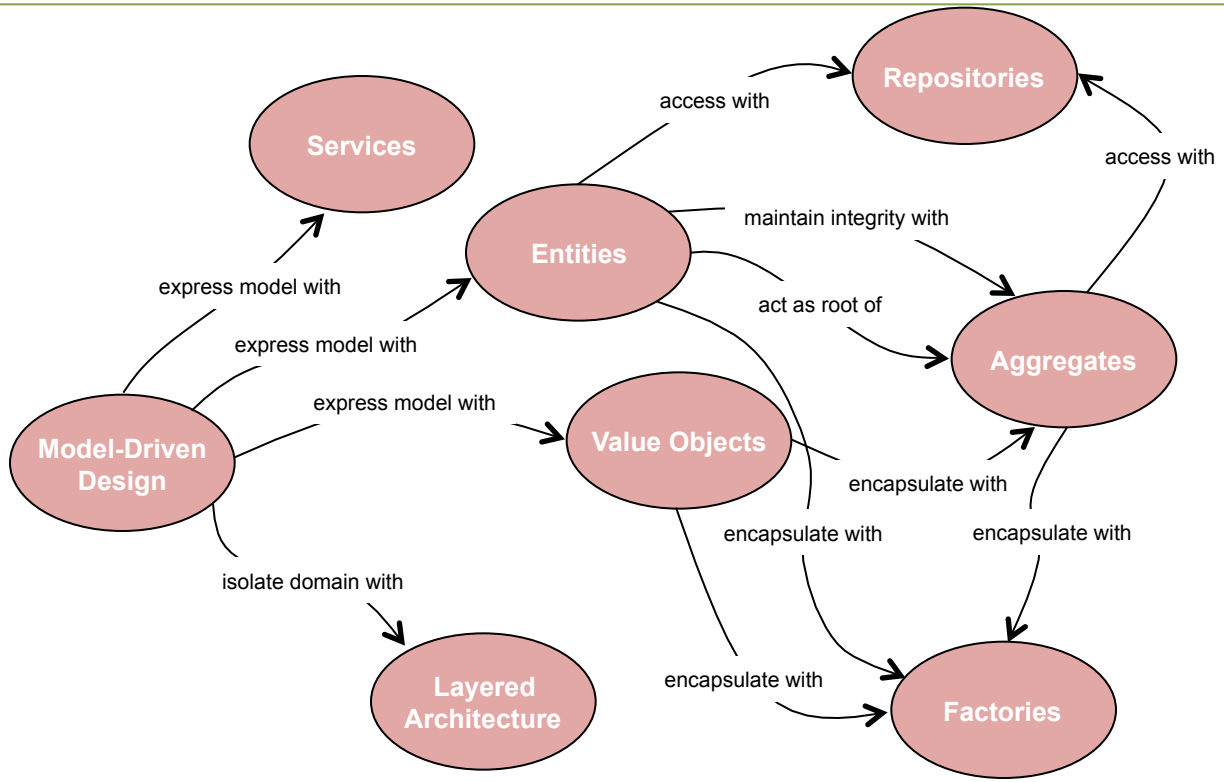


Eric Evans, Domain-Driven Design, Addison-Wesley, © Eric Evans, 2004

Agenda

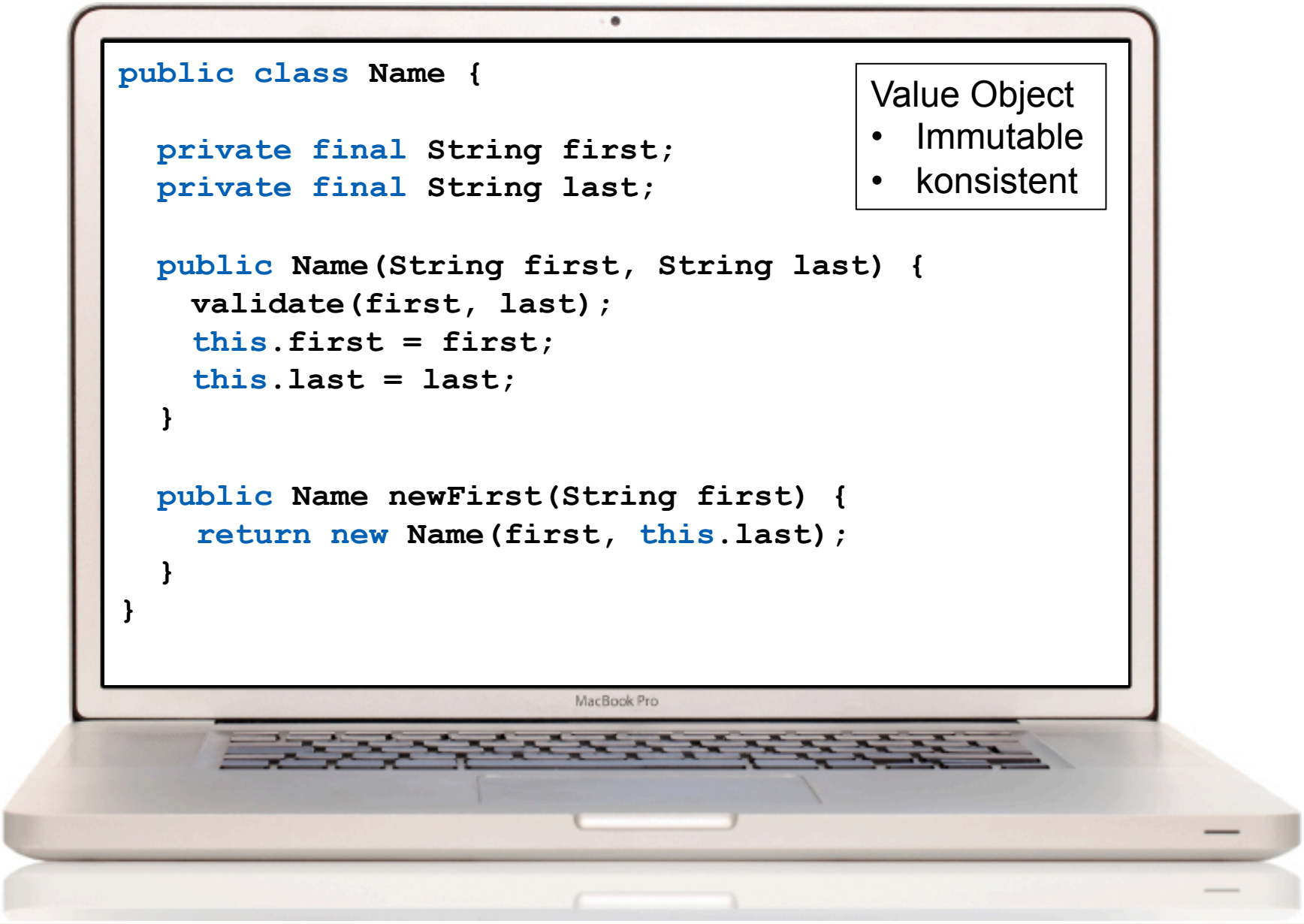
1. Agile Entwicklung - Short backup
2. Domain Driven Design - Grundlagen
3. On Top - DDD im Entwicklungsprozess
4. Domain Driven Design - Modellieren
5. On Top - DDD im Code

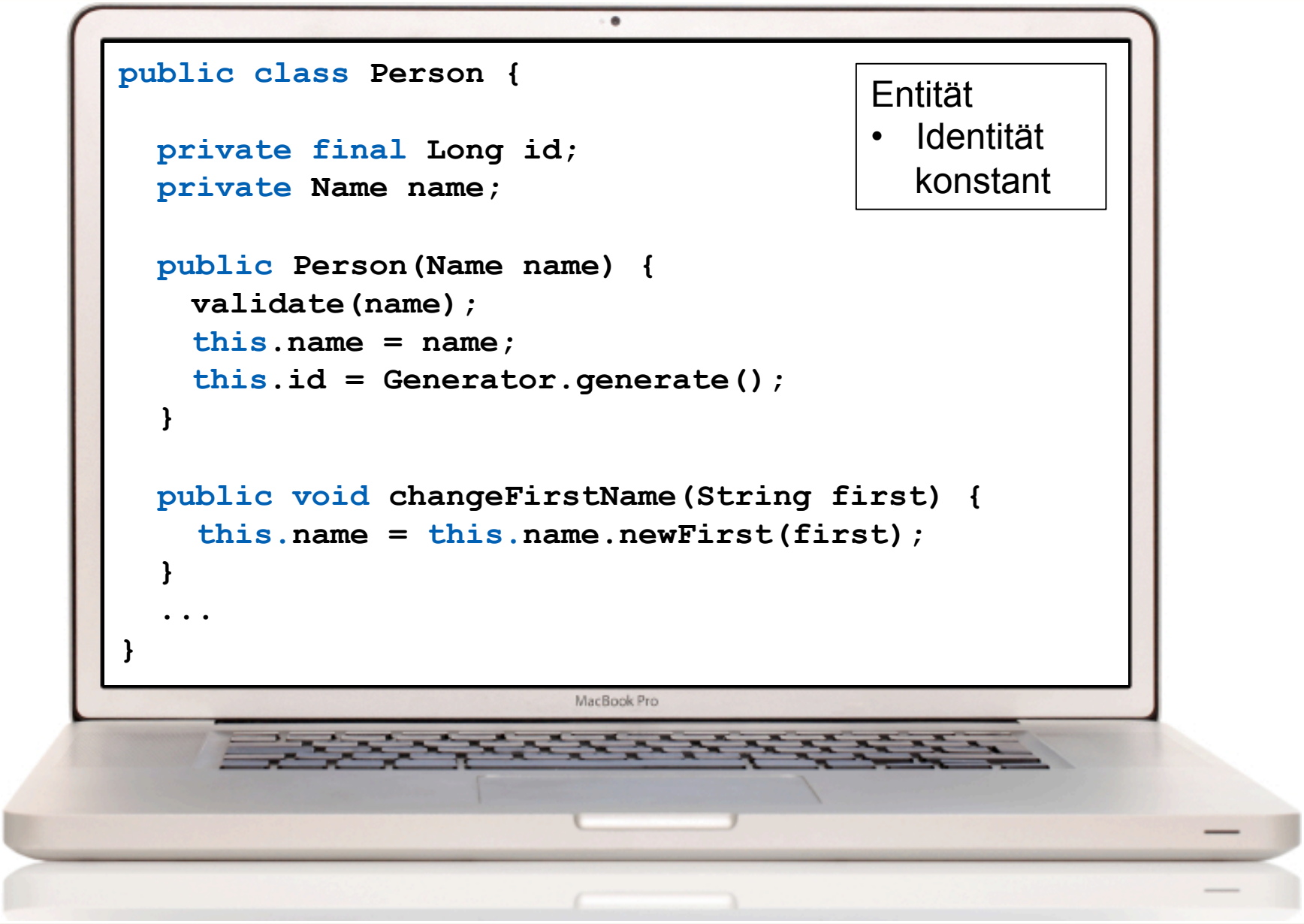
Modellierungsbausteine

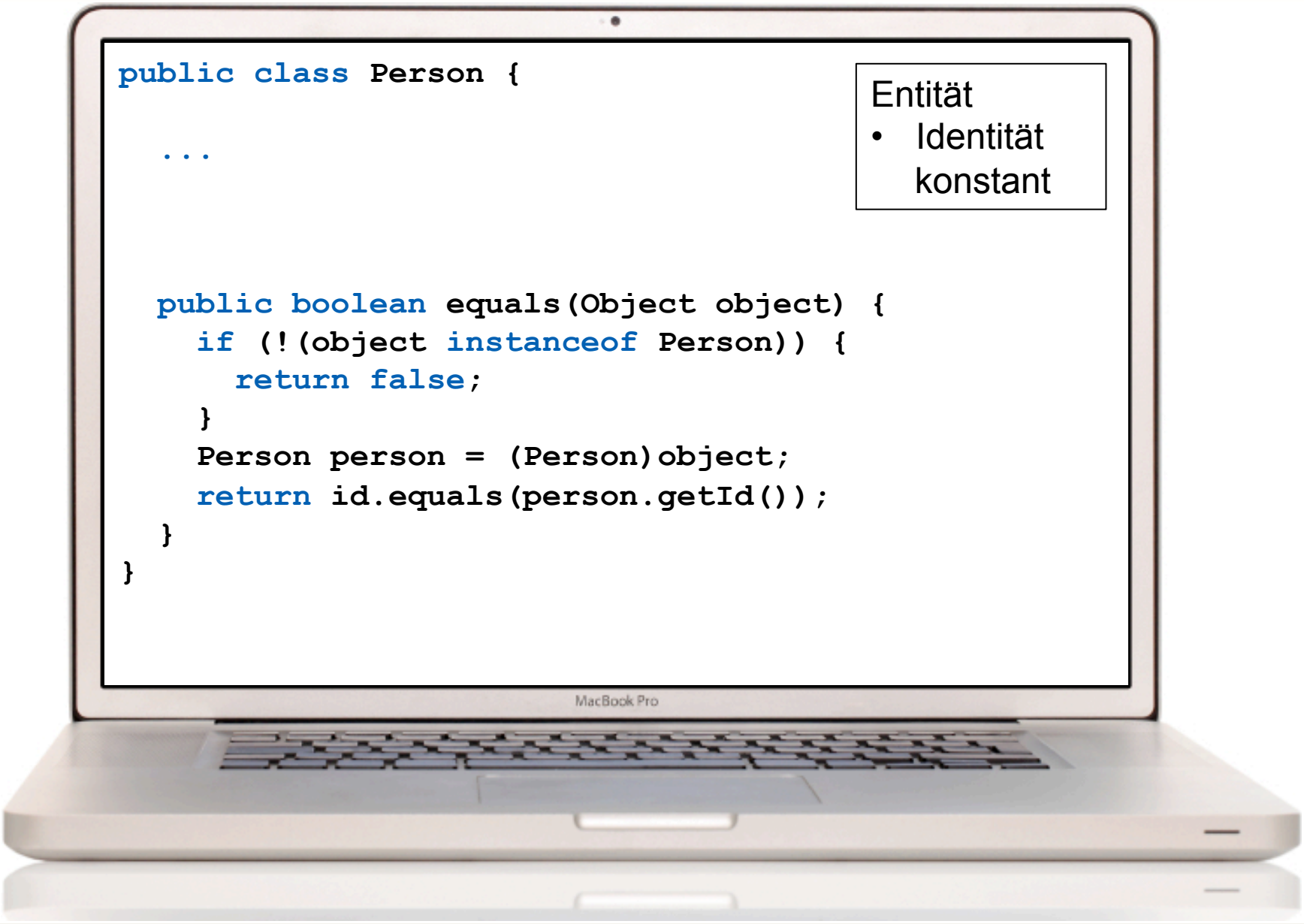


Eric Evans, Domain-Driven Design, Addison-Wesley, © Eric Evans, 2004


public class rson {
private e;
public
return
}
public void se) {
name = ne
}
publi (Obje
retu s(((Autho ()));
}
}
MacBook Pro


public class Name {
private final String first;
private final String last;
public Name(String first, String last) {
validate(first, last);
this.first = first;
this.last = last;
}
public Name newFirst(String first) {
return new Name(first, this.last);
}
}
Value Object
• Immutable
• konsistent
MacBook Pro


public class Person {
private final Long id;
private Name name;
public Person(Name name) {
validate(name);
this.name = name;
this.id = Generator.generate();
}
public void changeFirstName(String first) {
this.name = this.name.newFirst(first);
}
...
}
Entität
• Identität konstant
MacBook Pro


public class Person {
...
public boolean equals(Object object) {
if (!(object instanceof Person)) {
return false;
}
Person person = (Person)object;
return id.equals(person.getId());
}
}
Entität
• Identität konstant
MacBook Pro

Agenda

Fazit

- Domain Driven Design revolutioniert Agile Entwicklung nicht
- Agile und Domain Driven Design können Hand in Hand arbeiten
- Domain Driven Design hat an einigen Stellen die besseren Konzepte

Fazit

Domain Driven Design kann sehr gut ein Teil des agilen Entwicklungsprozesses sein.

Dadurch können die Vorteile von Domain Driven Design zum Vorteil des Gesamtprozesses und zum Vorteil der entstehenden Software genutzt werden.

Fragen?

5.– 8. September 2011
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Vielen Dank!

Tim de Buhr

Open Knowledge GmbH

offenkundiggut

Firma

open knowledge **GmbH**
Bismarckstraße 13
26122 Oldenburg

www.openknowledge.de
info@openknowledge.de

@_openknowledge
@_tooltime

offenkundiggut