APPE

SoDa
- Initialisierung
- Konzeption *
- Realisierung *
- Einführung
* Scrum

Meilenstein
- def. Ergebnisse
- def. Zeitpunkt

Vision → PMP
→ SysSpec

Backlog – DEEP
- Detailed Apropriately (Grooming)
- Estimated (Geschätzt)
- Emergent (Entwickelt sich)
- Prioritized (Priorisiert)

User Stories
Als Anwender in einer definierten Rolle benötige ich eine bestimmte Funktionalität um damit einen Geschäftsnutzen zu erzielen.
Definition of Done
+ Akzeptanztest

Daten / Rolle / Prozess
Szenario User CRUD
Termin Prod.

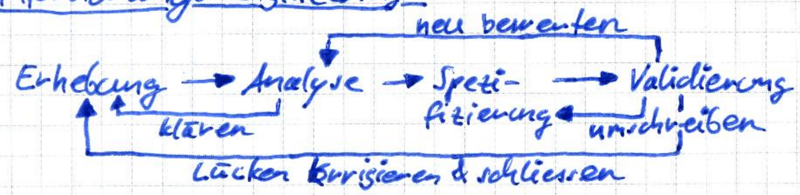
User Stories – INVEST
- Independent – unabhängig
- Negotiable – zerlegbar, änderbar
- Valuable – wirtschaftlich
- Estimatable – geschätzt Aufw.
- Small – kleingenug → Sprint
- Testable – Akzeptanzkriterien

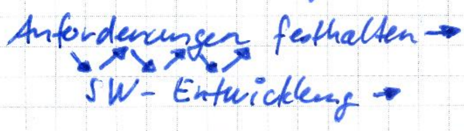
Anforderungsanalyse
- PO sammelt Anforderungen
- detailliert PO + Team
- Aufwand schätzen
- Product Backlog pflege
- priorisiert (PO)

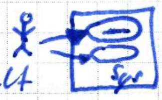
Anforderungen
- Needs
- Kunden-Anf.
- Software-Anf.

Nichtfunktionale-Anf.
- Performance / Effizienz
- Zuverlässigkeit / Verfügbark.
- Sicherheit / Wartbark.
- Portierbarkeit / Erweiterbark.

Gute Anforderungen
- Verständlich, klar
- Eindeutigkeit / Vollständig
- Konsistenz / Korrektheit
- Verfolgbar / Testbarkeit
- Gültigkeit
- Machbarkeit
- Bewertet (Prio, Aufwand)

Anforderungs-Engineering


Just in time Anforderungs-Engineering


Use Case / Jacobson
Request → Validate → Change Data → Result


Kontext-Diagramm


Lastenheft
- Vision des Kunden
- Kundenanforderung

Pflichtenheft
- Spezifikation
- SW-Anforderung

Modell
Abstraktion, Vereinfachung einer Sache (Beschreibung)

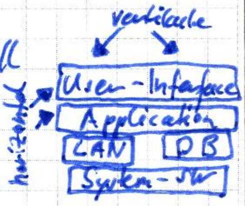
Modell-Typen
- statisch / dynamisch Modell
- Konzept / Architektur Ebene
↳ Implementation Ebene

OOA-Modell
- wesentliches
- Klassenmodell
- Nomen: Klassen
- Verben: Methoden

Diagramm
Möglichkeit Modell, Teilbereiche aus versch. Sichten grafisch darst.

SAD
- Software Architektur Dokument
- Topics: protokollierte Arch. Unterschiede
- Sichten: versch. Brillen auf Architektur

Prototyp
- funktionsfähiges Versuchsmodell
- Experimenteller Prototyp
- Teilprodukt (GUI-Proto)
- Evolutionäres Prototyping
+ Entw. Risiko reduktion
+ neue Lösungsansätze
+ GUI-Proto: Diskussion mit Kunde
− Kosten / Illusion nach Aussen, Prj fertig



Vorteile Layers
+ Wiederverwendbarkeit
+ Verständlichkeit / Lesbarkeit
+ Erweitern- / Testbarkeit
+ leichteres Refactoring
+ vereinfachte Wartung

Java Persistence API
```
import javax.persistence.*;
@Entity
@Table(name="vorlesung")
@NamedQueries({
  @NamedQuery(name="xy",
    query="select v from
           JVorlesungen v")})
public class JVorlesungen implements
                 Serializable {
  @Id
  @GeneratedValue(strategy=
     GenerationType.IDENTITY)
  @Basic(optional=false) (1)
  @Column(name="VorlNr")
  private Integer jVorlNr;
```
(1) Null not allowed

M zu 1 in JVorlesungen
```
@JoinColumn(name="gelesenVon",
   referencedColumnName=
   "PersNr")
@ManyToOne
private JProfessoren jGelesenVon;
```
Umkehrung in JProfessoren
```
@OneToMany(mappedBy=
   "jGelesenVon")
private Collection
   <JVorlesungen> jVorlesungen;
```

Vererbung in JPerson
```
@Inheritance(strategy=
   InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name=
   "DType", discriminatorType=
   DiscriminatorType.STRING)
```

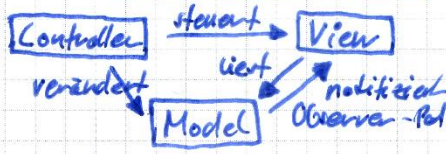
In JAssistent extends JPerson
```
@Entity
@DiscriminatorValue("assi")
```

CascadeType:
ref. Obj. implizit speichern
FetchType:
Laden der ref. Objekte

```
@OneToMany(mappedBy=
  "jGelesenVon",
  cascade=CascadeType.PERSIST,
  fetch=FetchType.EAGER)
```

Schichten / Tiers
- Schichten physisch trennt: tiers
- Abhängigkeiten nur nach unten
- Bildung: Logisch, technologisch, Abstraktion
- Präsentation / Business / Persistence
- GUI: MVC, UI-Logik
- BL: Logic / Modell
- Pers: Datenlogik / OR-Mapper

MVC – Model View Controller



+ wart / erweiterbarkeit ; Wiederverwendbar

APPE

Domain Logic Pattern
Transaction Script
- logical unit of work
- abgeschl. Funktion / Request
- EVA in einer Funktion
+ Gute Performance
- Wartbarkeit, Redundanz, reuse

Distribution Patterns
- Var 1: Attribute: ineffizient, viele Schritte
- Var 2: Objekte: grosse Datenmenge

Data Transfer Obj
- DTO
- nur Daten
- nur 1 aufruf

Remote Facade
- keine Businesslogik
- effiziente Kommunik.
- Durchlauferhitzer

Table Module
- pro Entität eine Klasse
- CRUD - Operationen
- starke Kopplung an DB
- bei grossen Sys. langsam

Domain Model
- Abstraktion Bus. Proz / Data
- vollständige Trennung phys. DBMS
+ Vorteile 3 Schicht-Architekt.
- OR Mapping Overhead

Service Layer
- Ergänzung von oben
- BL kapseln für übergeordnete Services
- z.B. Webservices

Solid – Prinzip
+ besseres Design + Vorteile Layers
- SRP Single Responsibility Principle
  eine Klasse, eine Aufgabe, keine Gott-Klassen
- OCP Open Closed Principle
  Offen f. Erweiterungen; Close f. Änderungen; Strategy Pattern
- LSP Liskov Substitution Principle
  überschriebene Methoden nur erweitern, nicht ändern
- ISP Interface Segregation Principle
  Schnittstellen trennen; hohe Kohäsion, Kopplung minimal
- DIP Dependency Inversion Principle
  High-Level-Klassen nicht von Low-Level Klassen abhängig, sondern von Interfaces; Details von Interfaces

Testarten
- Unit, Komponenten, Integration, Systemtests
- Funktionell, Performance, Stress, Sicherheitstest
⇒ Testfälle gerade so komplex wie nötig, so einfach wie möglich.
- n-Tier: Unit-Test testet alles darunter ⇒ Stubs / Mocks

Kriterien Clean Code
- Grösse, Komplexität
- Lebensdauer der App

Stubs
- vordefinierte statische Werte
- State Testing (State konst.)
- spezielle Implementationen explizit für Tests

Mockup
- Alternative Implementation
- dynamische Werte
- Behavior-Testing
- spez. Mock-Framework

Proxy-Pattern

Unit Test (saubere)
- Klarheit
- Einfachheit
- Ausdrucksdichte
- nur 1 assert pro Test
- SLA, SRP, SOC, FIRST
- nur 1 Konzept pro Test

SoC Separation of Concern

TDD: Test Driven Development
1. zuerst Unit-Test, dann produktiven Code dazu
2. Unit-Test minimal Code: compilieren, als test failed
3. minimal Code (prod) damit Test erfolgreich ist

F.I.R.S.T. Prinzip
- Fast, Independent, Repeatable, Self-Validating, Timely

Test-First rechtzeitig

Persistence.xml
```
<persistence-unit name=
  "appUniPU" transaction-type=
  "RESOURCE_LOCAL">
  <provider> org.eclipse.per...</>
  <class> uppcuni.JProfessoren </class>
  <properties> <....>
    <property name="..." value:"%
    <.....>
  </properties>
</persistence>
```

Test-Heuristiken
- Unzureichende Tests; Coverage-Werkzeuge; triviale Test ✓ ignorierte Test ⇒ Mehrdeutigkeit; Grenzbedingungen testen; Fehler: Nachbarschaft testen. Error: Muster err Diagnose nutzen
Coverage Patterns beachten; Tests sollen schnell sein

Entity Manager
```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("appUniPU");
EntityManager em = emf.createEntityManager();
List<JProfessoren> profList = em.createQuery("select p from JProfessoren p")
                                .getResultList();
EntityTransaction ta;
try{
  ta = em.getTransaction();
  p = new JProfessoren(); // set values...
  em.persist(p);          // .persist  speichern
  ta.commit();            // .merge    update
  em.refresh(p);
} catch (Exception e) { ta.rollback(); }

Query q = em.createNamedQuery("JVorlesung.xy");   // where
q.setParameter("vorlnr", 5001);                   // x = :myPara
JVorlesung v = (JVorlesung) q.getSingleResult();
```