Contents

[**General** 3](#_Toc484365671)

[**Problem with traditional software methodologies, why did they came up with new? Whats the difference** 3](#_Toc484365672)

[**Agile manifesto: 4 Lines remember and explain** 3](#_Toc484365673)

[**Principles not by heart but knowing** 3](#_Toc484365674)

[**Developing Software : 4 Variables** 4](#_Toc484365675)

[**Cost of change diagrams, explain** 4](#_Toc484365676)

[**5 Values of XP** 4](#_Toc484365677)

[**XP Practices unterstand what any mean** 4](#_Toc484365678)

[**User Stories** 5](#_Toc484365679)

[**What US are for and relationship concerning verbal and written communication (US seeking balance between them, flexible during development process)$** 5](#_Toc484365680)

[**Why important balance between business and development side?** 5](#_Toc484365681)

[**What is a US?** 5](#_Toc484365682)

[**What are the 3 parts of US?** 5](#_Toc484365683)

[**What is a theme? What an epic?** 5](#_Toc484365684)

[**Why do we need user roles? How do they help?** 6](#_Toc484365685)

[**INVEST Attributes, Attributes of good US** 6](#_Toc484365686)

[**Estimation and planing** 7](#_Toc484365687)

[**Why do wee need planing? Does Agile dev need planing?** 7](#_Toc484365688)

[**What is a conditional satisfaction? What is the role of conditional satisfaction in US / Release** 7](#_Toc484365689)

[**What are story points?** 8](#_Toc484365690)

[**What is velocity?** 8](#_Toc484365691)

[**What is planing poker, what does it work? Whay do peaople heide cards until everyone's done?** 8](#_Toc484365692)

[**What is the difference date driven and feature driven project'** 8](#_Toc484365693)

[**Who does the estiamtes of SU?** 8](#_Toc484365694)

[**How do you prioritize US? What criteria do you use? Diagrams.** 9](#_Toc484365695)

[**Value, Cost, knowledge, Risk** 10](#_Toc484365696)

[**Kano Model of Customer Satisfaction, example in pfg chapter of book** 10](#_Toc484365697)

[**Scrum** 11](#_Toc484365698)

[**Describe Scrum** 11](#_Toc484365699)

[**What does SCRUM require?** 11](#_Toc484365700)

[**Stacey Model (project noise level)** 12](#_Toc484365701)

[**What is a backlog, What kind of backlogs do we have in Scrum?** 12](#_Toc484365702)

[**What is a sprint. Why do scrum use sprints?** 12](#_Toc484365703)

[**Scrum roles, ceremonies, artifacts** 12](#_Toc484365704)

[**Taskboard, burndown smells** 13](#_Toc484365705)

[**Advanced Scrum** 14](#_Toc484365706)

[**Sprint review Mechanics** 14](#_Toc484365707)

[**Definition of done. is it static?** 14](#_Toc484365708)

[**Definition of ready? for US and Sprint** 15](#_Toc484365709)

[**Why is self organization important in Scrum. Boundries of self organizing team(cloud picture)** 15](#_Toc484365710)

[**Architecture and architect** 16](#_Toc484365711)

[**Definition of Software architecture in my words.** 16](#_Toc484365712)

[**4 + 1 Architecturial Views Kruchten** 16](#_Toc484365713)

[**Why do we need SW architecture** 17](#_Toc484365714)

[**Architecture Styles** 18](#_Toc484365715)

[**For each architecture style : components, connectors, example, benefits, drawbacks** 18](#_Toc484365716)

[**OR Mapping with JPA** 19](#_Toc484365717)

[**What is a domain model?** 19](#_Toc484365718)

[**What is the problem of domain model representing in database? (Ausgangslage Slides)** 19](#_Toc484365719)

[**What is Objektrelationales Mapping in general? OR Mismatch** 19](#_Toc484365720)

[**Promises of OR Mapping** 20](#_Toc484365721)

[**Conceptual Problems in OR Mapping** 20](#_Toc484365722)

[**TechnologieStack** 21](#_Toc484365723)

[**how does JPA (Java Persistence API)work in general** 21](#_Toc484365724)

[**What is an entity manager?** 22](#_Toc484365725)

[**What is a persistence unit?** 22](#_Toc484365726)

[**Lifecycle of Entities in JPA** 23](#_Toc484365727)

[**What is the difference between relationships between relational world and ORM world?** 23](#_Toc484365728)

[**Bidirektionale Beziehung, convenience methods for updating** 23](#_Toc484365729)

[**Cascading, transitive Persistenz** 24](#_Toc484365730)

[**OR Mapping with JPA2** 25](#_Toc484365731)

[**States of objects = Life cycles(evtl drawing)** 25](#_Toc484365732)

[**When does an Object receive an ID in JPA?** 25](#_Toc484365733)

[**What is a find method?** 25](#_Toc484365734)

[**How do you find an entity?** 25](#_Toc484365735)

[**What take care of for rollback** 26](#_Toc484365736)

[**What does clear method do on entity manager** 26](#_Toc484365737)

[**What does it mean to navigate the object graph** 26](#_Toc484365738)

[**What is JPQL?** 27](#_Toc484365739)

[**Named vs positional Parameters (param binding slide)** 27](#_Toc484365740)

[**SOLID** 28](#_Toc484365741)

[**Single Responsibility Principle SRP:** 29](#_Toc484365742)

[**Open Closed Principle:** 30](#_Toc484365743)

[**Liskov Substitution Principle:** 31](#_Toc484365744)

[**Interface Segregation Principle:** 32](#_Toc484365745)

[**Dependency Inversion Principle** 33](#_Toc484365746)

[**Hollywood Prinzip (Don't call me, I'll call you)** 34](#_Toc484365747)

# **General**

## **Problem with traditional software methodologies, why did they came up with new? Whats the difference**

Waterfall: zu viel planing, keine Änderungen während der Entwicklung

Unified Process

Iterative and Incremental

Use case drivenArchitecture centric

Risk focused

- can be "process" and "artifacts" heavy

**Problems**

* Project Failure Rate is Too High
* Despite the application of (heavyweight) methodologies, technology projects still regularly fail more than would be acceptable in any other industry.
* The evolution of technology (hardware) and tools (IDEs etc.) has outstripped methodologies by an order of magnitude.

## **Agile manifesto: 4 Lines remember and explain**

* Individuals and Interactions over processes and tools
* Working software over comprehensive documentation
* Customer collaboration over contract negotiation
* Responding to change over following a plan

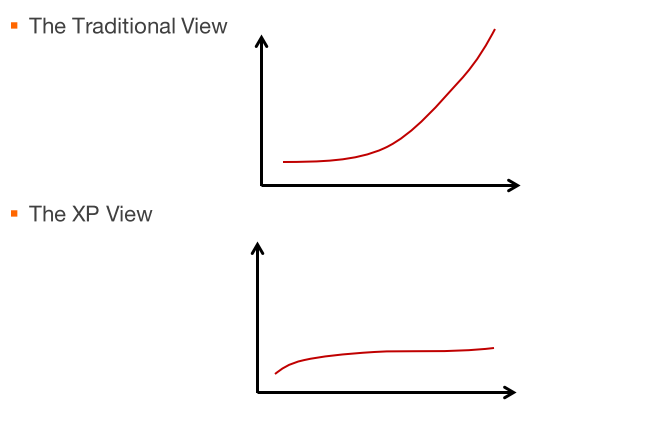
## **Principles not by heart but knowing**

* Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
* Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
* Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
* Business people and developers must work together daily throughout the project.
* Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
* The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
* Working software is the primary measure of progress.
* Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

## **Developing Software: 4 Variables**

* Time
* Resources
* Quality
* Scope

## **Cost of change diagrams, explain**



## **5 Values of XP**

## **XP Practices unterstand what any mean**

# **User Stories**

## **What US are for and relationship concerning verbal and written communication (US seeking balance between them, flexible during development process)$**

## **Why important balance between business and development side?**

=> Understandoing of each other

## **What is a US?**

## **What are the 3 parts of US?**

* Card - A written description of the user story for planning purposes and as a reminder
* Conversation - A section for capturing further information about the user story and details of any conversations
* Confirmation - A section to convey what tests will be carried out to confirm the user story is complete and working as expected

## **What is a theme? What an epic?**

Epic is summary of user stories

**Epic:**

Epics typically fall into one of two categories:

* Complex stories are intrinsically large
* Compound stories are multiple stories in one

**Theme:**

A set of related user stories may be combined and treated as a single entity. This is referred to as a theme

## **Why do we need user roles? How do they help?**

One may mistakenly ...

* assume that there’s only one user: “The user”
* write all stories from one user’s perspective
* assume all users have the same goals
  + This leads to missing stories!

Broaden the scope from looking at one user

Allows users to vary by

* What they use the software for
* How they use the software
* Background
* Familiarity with the software / computers

Use the following steps to identify and select a set of user roles:

* Brainstorm an initial set of user roles
* Organize the initial set
* Consolidate roles
* Refine the roles
* Special stories, non func req. Spikes.
* System level constraints
* Can be formulated as user stories
* Affect design and testing of many other stories
* Can be incorporated into the "Definition of Done" (later in course,

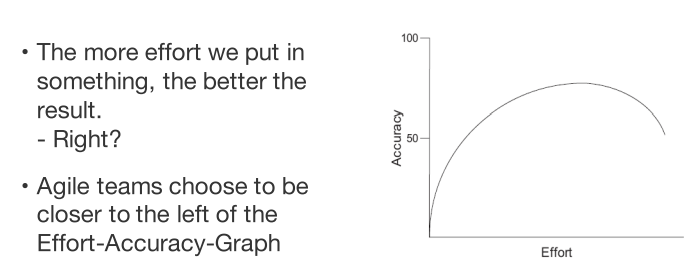
## **INVEST Attributes, Attributes of good US**

Good stories focus on six attributes:

* Independent
* Negotiable
* Valuable to users or customers
* Estimatable
* Small
* Testable

# **Estimation and planing**

## **Why do we need planning? Does Agile dev need planning?**



What is iterative planing, why is it useful?

An Agile Approach to Planing, new Capabilites and experience generated

What levels of planing do we have

=> 2 oder 3

Estimates are best derived collaboratively by the team

-> play Planning Poker

-> We tend not to know specifically who will perform a given task

Two possible estimation scales (10x range):

-> Fibonacci: 1,2,3,5, and 8

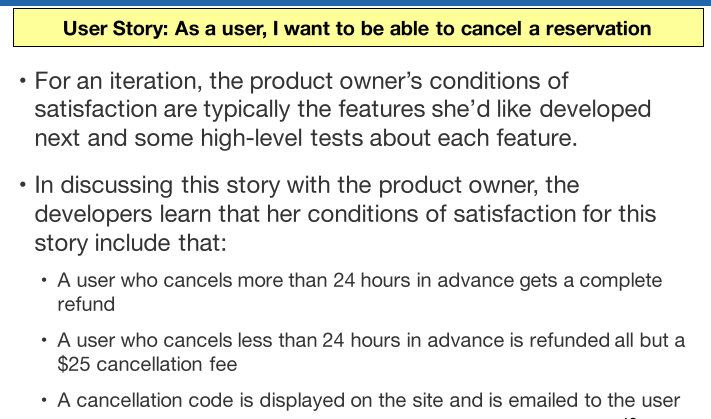
-> 2^n: 1,2,4, and 8

*Including 0 as a valid number not recommended* (Remember: 15 x 0 ≠ 0)

## **What is a conditional satisfaction? What is the role of conditional satisfaction in US / Release**

Every project is initiated with a set of objectives (E.g. create the world’s best software for the airline catering industry)

There will almost certainly be additional objectives regarding schedule, budget and quality (agile teams typically prefer to treat quality as non-negotiable)

-> These objectives are the customer’s or product owner’s **Conditions of Satisfaction**

## **What are story points?**

The estimate the amount of effort needed to develop a feature and e.g. the complexity of developing, the risk and so on.  
They are a relative measure of the complexity of a user story.

## **What is velocity?**

A measure for a team’s rate of progress. The number of story points summed up for one iteration’s completed US gives us the velocity.

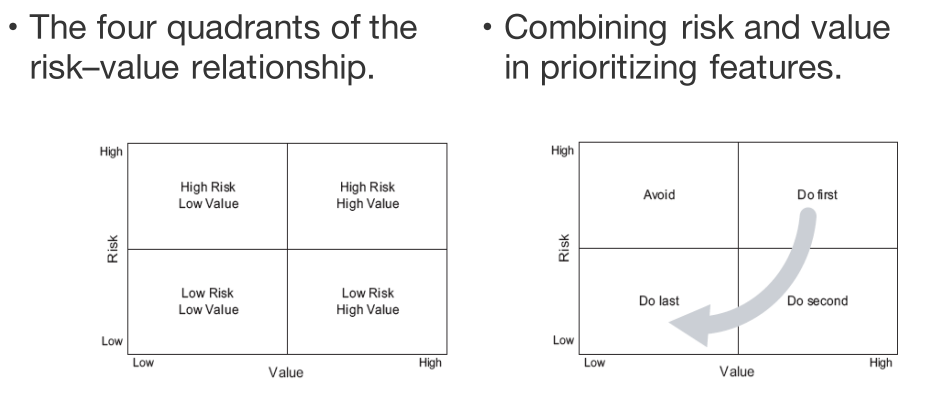
## **What is planing poker, what does it work? Whay do peaople heide cards until everyone's done?**

## **What is the difference date driven and feature driven project'**

## **Who does the estiamtes of SU?**

=> the team

## **How do you prioritize US? What criteria do you use? Diagrams.**



# **Value, Cost, knowledge, Risk**

## **Kano Model of Customer Satisfaction, example in pfg chapter of book**

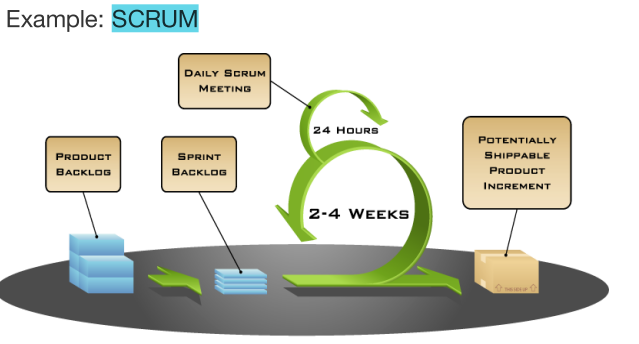
tracking: What is a burndown burnup chart?

=> down: Burning down done points.

# **Scrum**

## **Describe Scrum**

*"* *Scrum is an iterative and incremental agile software development framework for managing product development.[1][2] It defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal",[3] challenges assumptions of the "traditional, sequential approach"[3] to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines involved.*

*A key principle of Scrum is the dual recognition that customers will change their minds about what they want or need (often called requirements volatility[4]) and that there will be unpredictable challenges—for which a predictive or planned approach is not suited. As such, Scrum adopts an evidence-based empirical approach—accepting that the problem cannot be fully understood or defined up front, and instead focusing on how to maximize the team's ability to deliver quickly, to respond to emerging requirements, and to adapt to evolving technologies and changes in market conditions."*

## **What does SCRUM require?**

|  |  |  |
| --- | --- | --- |
| TDD | User Acceptance Testing | Sprint |
| Continuous Integration | Sprint Planning | Retrospective |
| Daily Scrum | Backlog | Demo to customer |
| Loose coupling | Refinement | Dedicated QA resources |
|  | Planning Poker | Product Backlog |

## **Stacey Model (project noise level)**

=> Knowledge of Understanding req(WHAT) and Certainty of Technology(HOW)

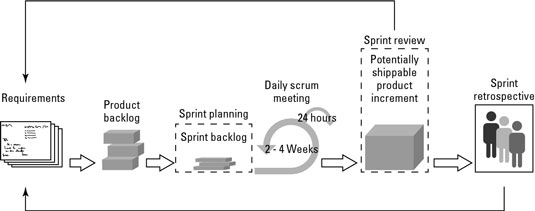
## **What is a backlog, What kind of backlogs do we have in Scrum?**

The agile product backlog in Scrum is a prioritized features list, containing short descriptions of all functionality desired in the product.

* Features
* Bugs
* Technical work
* Knowledge acquisition

## **What is a sprint. Why do scrum use sprints?**

The scrum approach includes assembling the project’s requirements and using them to define the project. You then plan the necessary sprints, and divide each sprint into its own list of requirements. Daily scrum meetings help keep the project on target as do regular inspections and reviews. At the end of every sprint, you hold a sprint retrospective to look for ways to improve the next sprint. The process looks something like this:



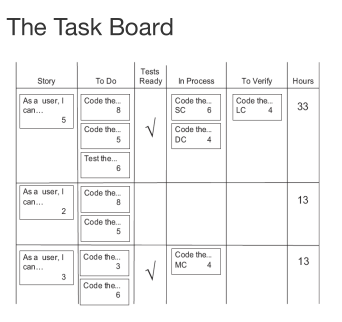
Within each sprint, the development team builds and tests a functional part of the product until the product owner accepts it and the functionality becomes a potentially shippable product. When one sprint finishes, another sprint starts. Scrum teams deliver product features in increments at the end of each sprint. A product release occurs at the end of a sprint or after several sprints.

## **Scrum roles, ceremonies, artifacts**

Roles: Product Owner & Scrum Master

Ceremonies: Daily Stand-up, Sprint planning, sprint retrospective.

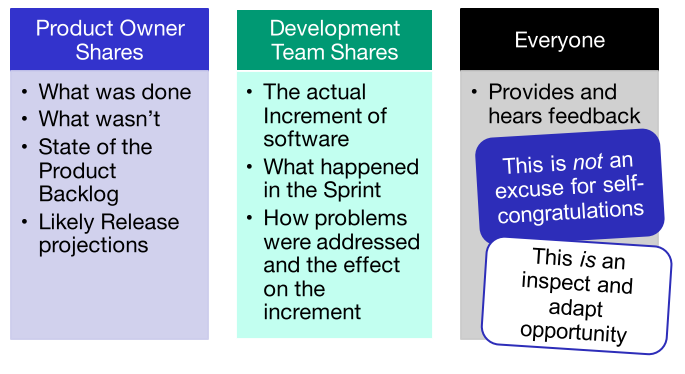
## **Taskboard, burndown smells**



# **Advanced Scrum**

## **Sprint review Mechanics**

* A collaborative working session (Not a demo)
* The Scrum Team shows the Increment
* Feedback is heard from all present (Feedback used to guide the next Increment)
* Working software, no slides
* 4 hours for 4 week Sprint (Usually less for shorter Sprints)
* The full Scrum Team participates (Complemented by the stakeholders)
* Srpitn Retrospective . What should result of it?



Results of a sprint review should be actionable improvements.

## **Definition of done. is it static?**

“Complete as mutually agreed to by all parties and conforming to an organization’s standards, conventions, and guidelines. When something is reported as “done” at the Daily Scrum or demonstrated as “done” at the Sprint review meeting, it must conform to this agreed definition.” – Agile Project Management with SCRUM

So Definition of Done is…

* Product quality standard?
* A list of activities for producing potential shippable product increment?
* Anything else?

Problems without defining "Done"

* Technical debts – “Pay me now or pay me later”
* The undone work accumulate in a nonlinear manner
* Illusion of progress (velocity) and unpredictable ship date
* Team over-commit the amount of work they can do in a sprint
* The product owner surprises with the delivery during the sprint review session

## **Definition of ready? for US and Sprint**

User Story:

* User Story defined
* User Story Acceptance Criteria defined
* User Story dependencies identified
* User Story sized by Delivery Team
* Scrum Team accepts User Experience artefacts
* Performance criteria identified, where appropriate
* Person who will accept the User Story is identified
* Team has a good idea what it will mean to Demo the User Story

Sprint:

* The Sprint Backlog is prioritized
* The Spring Backlog contains all defects, User Stories and other work that the team is committing to
* No hidden work
* All team members have calculated their capacity for the Sprint
* Fulltime on project = X hours per day
* All User Stories meet Definition of Ready

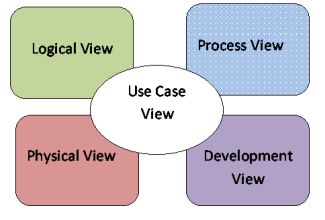
## **Why is self organization important in Scrum. Boundries of self organizing team(cloud picture)**

* Select the work to complete
* Determine how best to meet requirements
* Get help with external disruptions (Impediments)
* Select their own Scrum Master



# **Architecture and architect**

## **Definition of Software architecture in my words.**

* Software architecture is the blueprint for a software systemʼs construction and evolution
* Design decisions encompass the following aspects of the system under development
  + Structure
  + Behavior
  + Interaction
  + Non-functional properties (!)

=> components, connection, organisation between components, different views.

## **4 + 1 Architecturial Views Kruchten**

*Logical View*

The logical view is concerned with the functionality that the system provides to end-users. UML Diagrams used to represent the logical view include Class diagram, Communication diagram, Sequence diagram.

*Development View*

The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. It uses the UML Component diagram to describe system components. UML Diagrams used to represent the development view include the Package diagram.

*Process View*

The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML Diagrams to represent process view include the Activity diagram.

*Physical View*

The physical view depicts the system from a system engineer's point-of-view. It is concerned with the topology of software components on the physical layer, as well as communication between these components. This view is also known as the deployment view. UML Diagrams used to represent physical view include the Deployment diagram.

*Scenarios*

The description of an architecture is illustrated using a small set of use cases, or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. UML Diagram(s) used to represent the scenario view include the Use case diagram.

## **Why do we need SW architecture**

* Even though every stakeholder wants to do her/his best
  + overall results are bad (too often!)
* Some reasons:
  + overall structure inadequate
  + interfaces to the outside inappropriate
  + interfaces between parts donʼt fit
  + functionality falls in “gap” between the components
* Architecture can help
* Communication among stakeholders
  + basis for discussions and decisions
  + common understanding about the system
  + get consensus
* document design decisions
  + guidelines for implementation
  + basis for evolution and iterations
  + helps structuring team and planning
  + checkpoints for goal achievement
* abstraction of the system (as built)
  + for building product lines and homogeneous systems
  + for outsourcing or acquiring parts

=> communitcation with stakeholders, document designdecisions, abstraction

# **Architecture Styles**

## **For each architecture style : components, connectors, example, benefits, drawbacks**

# **OR Mapping with JPA**

## **What is a domain model?**

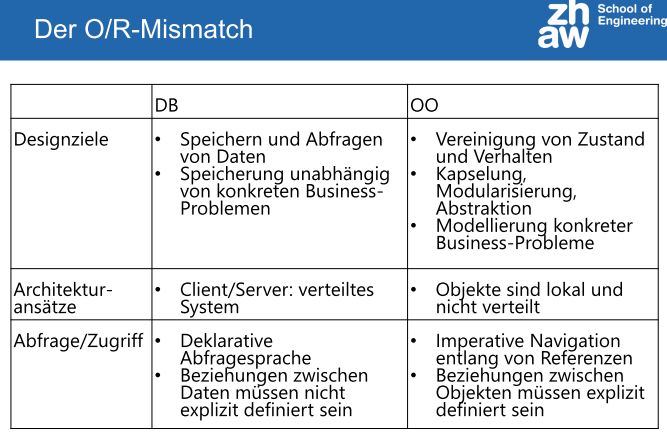
A domain model is a system of abstractions that describes selected aspects of a sphere of knowledge, influence or activity (a domain). The model can then be used to solve problems related to that domain.

## **What is the problem of domain model representing in database? (Ausgangslage Slides)**

De facto Standard-Konstellation für Enterprise-Applikationen:

* OO-Technologie für die Applikations-entwicklung
* Relationale Datenbanken für die Persistenz der Daten
* An dieser Ausgangslage wird sich mittelfristig kaum etwas ändern.
* Der OO-Ansatz und der relationale Ansatz weisen grundsätzliche Unterschiede auf
* Aus diesen Unterschieden resultiert der sog. Object-Relational-Mismatch

## **What is Objektrelationales Mapping in general? OR Mismatch**

Object-relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language.

## **Promises of OR Mapping**

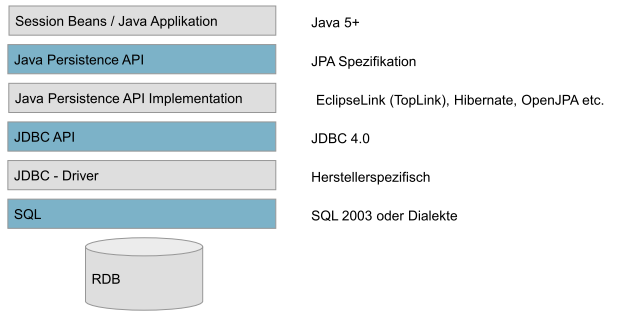
* Die Applikation wird von der DB entkoppelt
  + Applikationsentwickler muss kein SQL beherrschen
  + Das relationale Modell der Datenbank hat keinen Einfluss auf das OO-Design
* Automatische Persistenz
  + Automatisierte Abbildung der Objekte in die relationalen Strukturen
  + Der Applikationsentwickler muss sich nicht um die “low-level”-Details des CLI (z.B. connections) kümmern.
* Transparente Persistenz / Persistence Ignorance
  + Die Klassen des Domain-Models wissen nicht dass sie persistiert und geladen werden können und haben keine Abhängigkeit zur Persistenz-Infrastruktur.
* Abstraktion
  + Abstraktion ist eines der wichtigsten Werkzeuge um Komplexität zu bewältigen
* Separation of Concerns
  + Bei der Applikationsentwicklung kann man sich ausschliesslich auf die Geschäftsprobleme konzentrieren
  + Infrastruktur-Probleme können separat gelöst werden und beeinflussen das Design und die Implementation der Geschäftslogik nicht.

## **Conceptual Problems in OR Mapping**

Folgende konzeptionelle Probleme sollten beim O/R-Mapping beachtet werden:

* Location Transparency
  + Alle Daten immer verfügbar
  + Für die Applikation sollte es keinen Unterschied machen, ob die sich Daten im lokalen Speicher oder auf der entfernten Datenbank befinden.
  + Wieviele Daten werden geladen?
    - Zuviele: Speicher, Bandbreite -> Ladezeit!
    - Zuwenige: Konstantes Nachladen -> Ladezeit!
    - Stichworte: Lazy-Loading / Eager-Loading
* Optimiertes SQL
  + SQL ist nicht so performant wie handgeschriebens, getuntes SQL
    - Dynamisch generiertes SQL muss auch nicht gewartet werden!
    - Ausgereifte Frameworks generieren gut optimiertes SQL
  + Performance
    - Stored Procedures sind nicht performanter als dynamisches SQL!
* Mengen-Manipulationen
* Relationale Suche / Reports / OLAP
  + Probleme:
  + Karthesisches Produkt
    - In der DB sehr effizient umgesetzt
    - In der OO-Welt keine Entsprechung
  + Die Struktur des Resultats der Anfrage existiert nicht in der OO-Welt
    - Es gibt keinen Typ mit den Attributen (Vorname, Nachname, Produktname)
    - Die relationale Welt erlaubt flexible (untypisierte) Sichten auf die Daten (Deklarative ad-hoc formulierte Anfragen)
    - Resultat einer Anfrage kann völlig entkoppelt sein von der Struktur wie die Daten gespeichert werden.
    - Die OO-Welt verlangt definierte, stark typisierte Strukturen
  + Das ‚Flachdrücken‘ / ‚Denormalisieren‘ muss in der OO-Welt manuell ausgeführt werden
  + Alle beteiligten Objekte müssen geladen werden (Performance!)

# **TechnologieStack**



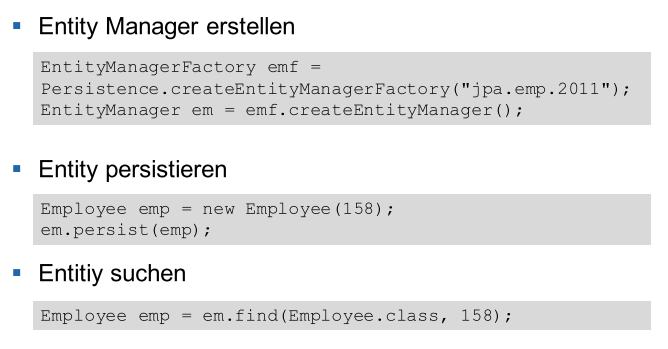
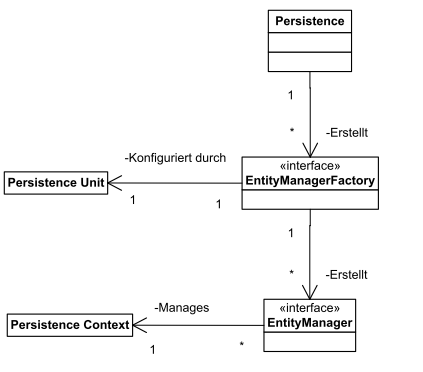
## **how does JPA (Java Persistence API)work in general**

*Entity*

* Eine Entity ist persistierbar. Der Zustand kann in einer Datenbank abgespeichert und später wieder darauf zugegriffen werden
* Wie jedes andere Java Objekt hat eine Entity eine Objektidentität. Zusätzlich besitzt sie eine
* Datenbankidentität (Primary Key)
* In Zusammenhang mit der Datenbank werden die Entities transaktional verwendet. Die Erstellung, Änderung und das Löschen wird in einer Transaktion durchgeführt

=> Annotations, entities columns, relationships

## **What is an entity manager?**



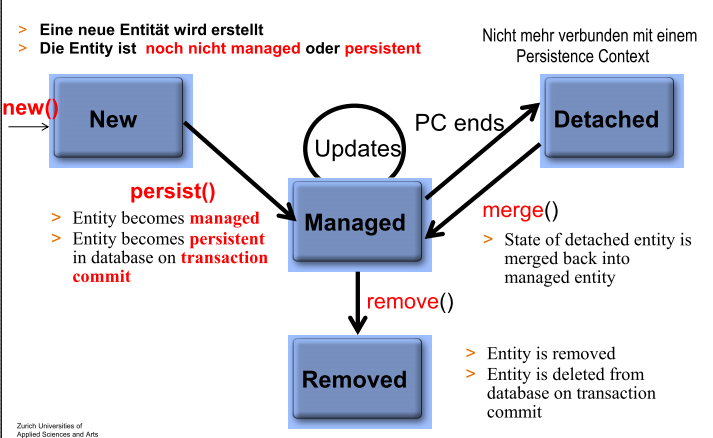
## **What is a persistence unit?**

* Eine Persistence Unit ist eine logische Einheit von Entities. Sie wird beschrieben durch:
  + Einen Namen
  + Die zu dieser Unit gehörenden Entity-Klassen
  + Angaben zum Persistence Provider
  + Angabe zum Transaktionstyp
  + Angaben zur Datenquelle
  + Weitere Properties
  + Namen von XML O/R-Mapping Files
* Technisch wird die Beschreibung einer Persistence Unit in der Datei META-INF/persistence.xml abgelegt.
* Persistence Archive = JAR

**What is difference lazy and eager fetching?**

* Die Idee von Lazy Loading ist es, die Daten erst dann von der DB zu laden, wenn sie auch wirklich in der Applikation benötigt werden.
  + Das Laden sollte für den Client transparent sein
  + Dem Programmierer wird viel Arbeit erspart
* Nachteile:
  + Traversieren eines Objekt-Graphen kann in vielen einzelnen DB-Queries resultieren
  + “N+1 select problem”: Für eine Parent-Child Beziehung wird für jedes Kind ein eigenes DB-Query abgesetzt
* Das Gegenteil von Lazy Loading ist Eager Loading

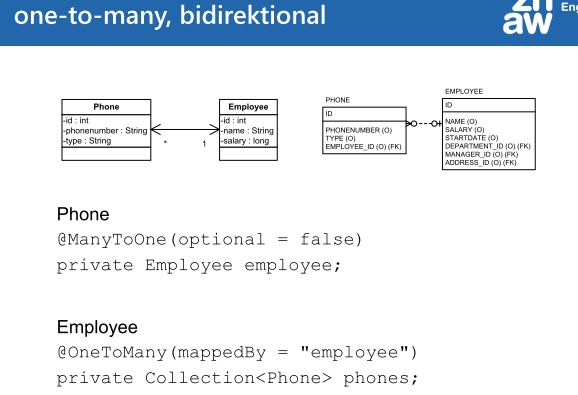
## **Lifecycle of Entities in JPA**

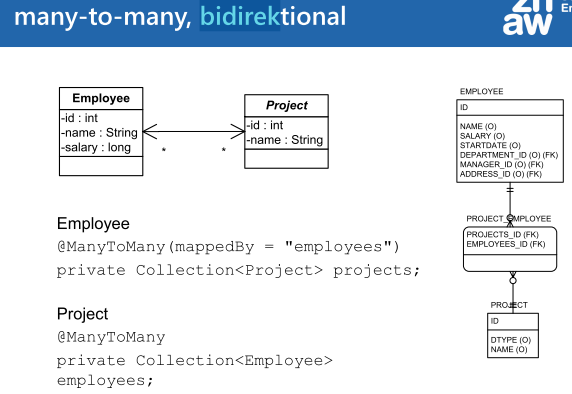


## **What is the difference between relationships between relational world and ORM world?**

=> realtion : navigate from both sides, ORM navigate just one side, upateproblem

## **Bidirektionale Beziehung, convenience methods for updating**

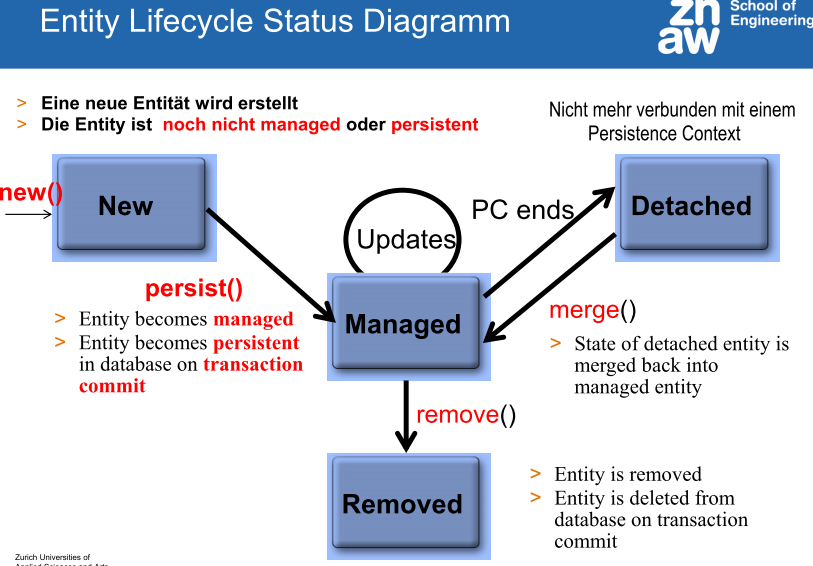




## **Cascading, transitive Persistenz**

# **OR Mapping with JPA2**

## **States of objects = Life cycles(evtl drawing)**



## **When does an Object receive an ID in JPA?**

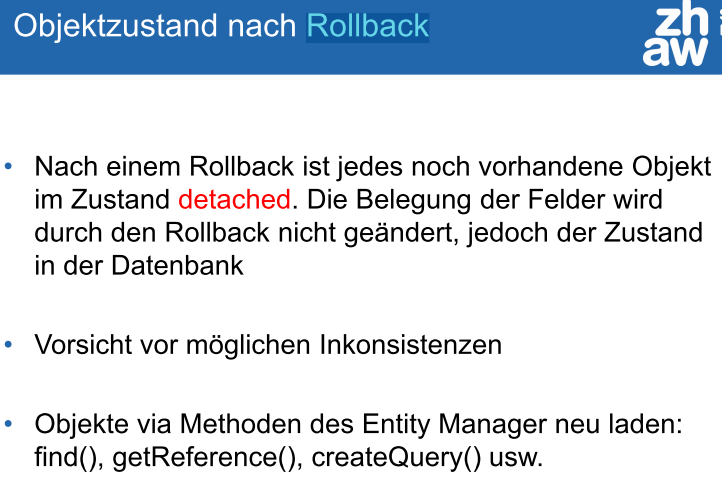
=> em.persist + em.flush or em.commit

## **What is a find method?**

## **How do you find an entity?**

=> give entity name and ID

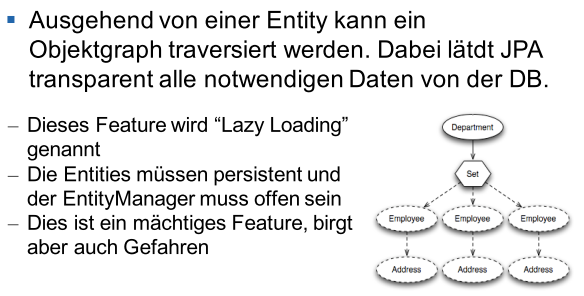
## **What take care of for rollback**



## **What does clear method do on entity manager**

* Ab und zu kann es vorkommen, dass der Persistence Context gelöscht werden soll
* Dies kann mit der Methode clear() des EntityManager erreicht werden
* Alle Entities kommen in den Zustand detached
* Vorsicht: enthält der Persistence Context Änderungen welche noch nicht mit commit() gespeichert wurden, gehen diese verloren

## **What does it mean to navigate the object graph**



=> something.other.any , dot notation like java JOIN attribute

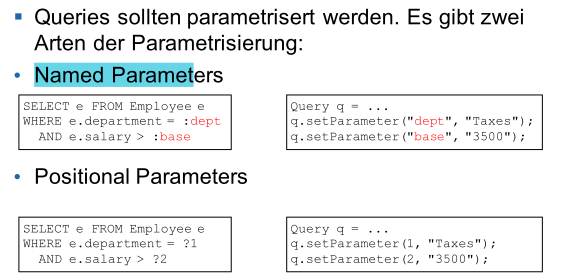
## **What is JPQL?**

Queries in der Java Persistence Query Language (JPQL)

JPQL ist eine mächtige Abfragesprache basierend auf dem Entitätenmodell:

* Stark an SQL angelehnt
* Unabhängig von der darunterliegenden Datenbank
* Abfragen basieren auf dem Klassenmodell (Entitäten), nicht auf dem Datenmodell (Tabellen)
* Unterstützt OO-Konstrukte wie Vererbung, Polymorphismus und Pfadausdrücke

## **Named vs positional Parameters (param binding slide)**



Inheritance in ORM, not details but the 4 points

=> supported, different ways of saving, single table with lot of columns with empty or each object in own table

# **SOLID**

**Single Responsibility Principle (SRP):** Jedes Modul soll genau eine einzige Verantwortung übernehmen.

**Open Closed Principle (OCP):** Offen für Erweiterung, geschlossen für Änderung

**Liskov Substitution Principle (LSP):** Eine Instanz einer Klasse soll durch eine Instanz einer abgeleiteten Klasse ersetzt werden künnen, ohne dass sich das Verhalten ändert.

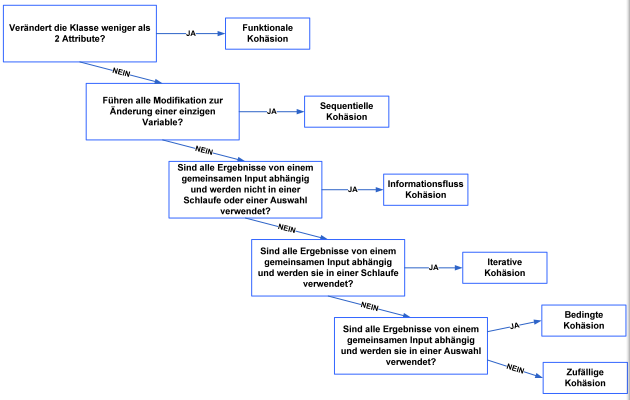
**Interface Segregation Principle (ISP):** Clients sollen nur von Schnittstellen abhängig sein, die sie auch tatsächlich benutzen

**Dependency Inversion Principle (DIP):** Umkehr der Abhängigkeiten - Schnittstellen statt konkrete Klassen

**10 Commandments from Jim Flemming**

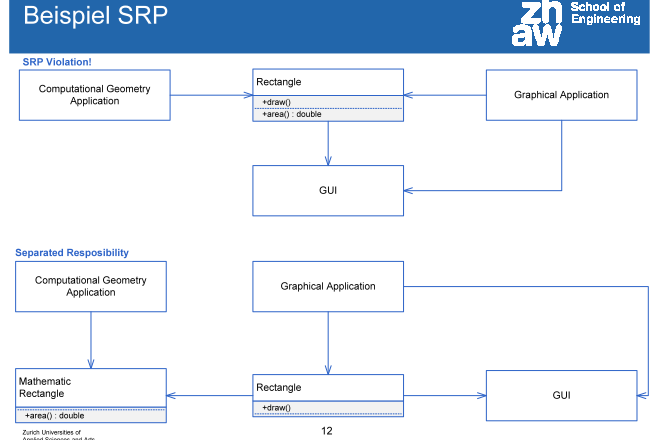
1. Als Klassenname sollten Nomen (Hauptwörter) und als Methodennamen Verben verwendet werden
2. Kommentare sind genauso wichtig wie Code und sollte dementsprechend eingestuft werden.
3. Vererbung sollte neuem Entwicklungspersonal helfen, den “Unterschied zu programmieren”
4. Die Kapselung von Allem – Sourcecode eingeschlossen – ist wesentlich
5. Verben (Methodennamen) sollten sehr sorgfältig ausgewählt werden, um Polymorphismus zu unterstützen.
6. Wiederverwendung kann nur dann erreicht werden, wenn die Standardklassen verständlich sind.
7. Neues Entwicklungspersonal sollte die CASE Tools der erfahrenen Leute wiederverwenden.
8. Plattformübergreifende Portabilität sollte wichtiger als andere Designaspekte sein
9. Eine neue Klasse sollte für jede neue Schnittstelle zu einer bestehenden Library verwendet werden.
10. Mehrfache Returns sollten ausschliesslich für Errorhandling und Debugging verwendet werden.

## **Single Responsibility Principle SRP:**



Um lose Kopplung zu erreichen, sind Module oder Klassen so zu gestalten, dass sie im Minimum die drei Kohäsionsarten Funktionale, Sequentielle und Informationsfluss Kohäsion erfüllen.

* Funktionale Kohäsion: Lediglich eine einzige Variable wird überhaupt verändert
* Sequentielle Kohäsion: Es werden zwei Variablen verändert, aber beide Modifikationen resultieren lediglich in einen Output.
* Informationsfluss Kohäsion: Die Änderung mehrerer Attribute wird von einem einzigen Input veranlasst



## **Open Closed Principle:**

Betrand Meyer: Module sollten sowohl offen, als auch geschlossen sein

Eine Klasse muss zwei primäre Eigenschalten aufweisen:

* **Offen für Erweiterungen:** Das Verhalten einer Klasse kann erweitert werden. Diese Eigenschaft erlaubt das nachträgliche verändern der Verhaltens oder das Hinzufügen von neuen Eigenschaften, falls entsprechende An-forderungen dies verlangen.
* **Geschlossen für Änderungen:** Es können keine Änderungen an der Klasse durchgeführt werden. Es ist nicht erlaubt, an die Sourcen der Klasse zu verändern.

Abstraktion

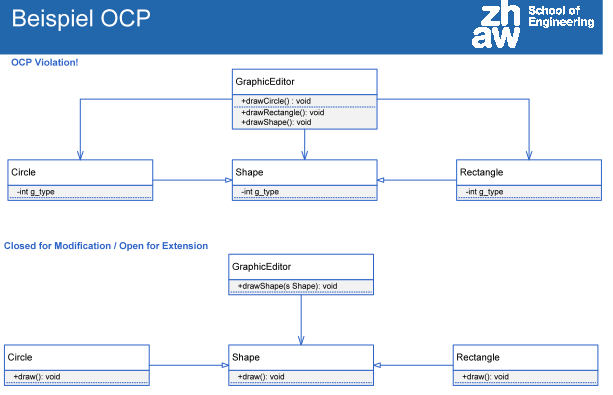
Definition: Unter Abstraktion versteht man Verallgemeinerung, das Absehen vom Besonderen und Einzelnen, das Loslösen vom Dinglichen. Abstraktion ist das Gegenteil von Konkretisierung

• **Klassifizierung:** Charakterisierung der Gemeinsamkeiten von Individuen durch Typen oder Klas-sen

• **Komposition:** Zusammensetzen einer Menge zusammenhängender Individuen zu einem Ganzen

• **Generalisierung:** Verallgemeinerung der Merkmale einer Menge ähnlicher Typen

• **Benutzung:** Nutzung von Leistungen Dritter durch ein Individuum zwecks Erbringung eigener, höherwertiger Leistungen



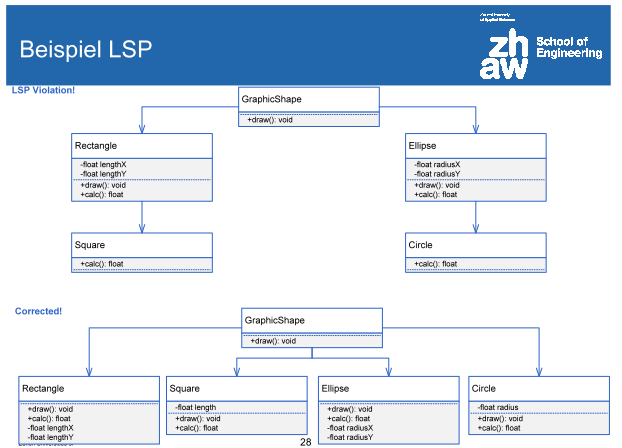
## **Liskov Substitution Principle:**

**Konsequenzen:**

* **Vorbedingungen:** Eine Unterklasse kann die Vorbedingungen für eine Operation, die durch die Oberklasse definiert wird, einhalten oder abschwächen. Sie darf die Vorbedingungen aber nicht verschärfen.
* **Nachbedingungen:** Eine Unterklasse kann die Nachbedingungen für eine Operation, die durch eine Oberklasse definiert werden, einhalten oder erweitern. Sie darf die Nachbedingungen aber nicht einschränken.
* **Invarianten:** Eine Unterklasse muss dafür sorgen, dass die für die Oberklasse definierten Invarianten immer gelten.

**Polymorphismus:**

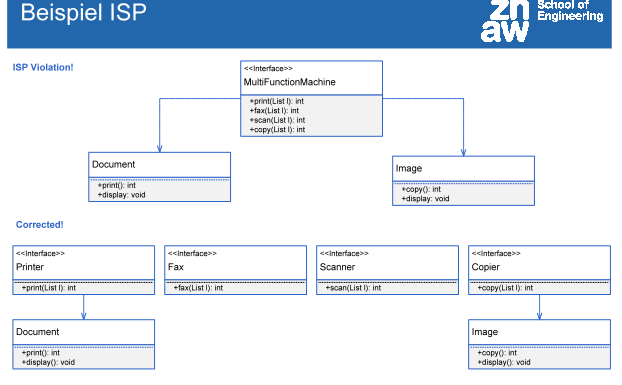
* **Dynamische Polymorphie:** Einer Variablen können Objekte verschiedenen Typen zugeordnet werden. Der Typ der Variablen fungiert dabei als Schnittstelle. Implementieren die verschiedenen Objekte dieselbe Methode – jedoch mit verschiedenem Verhalten – so kann zur Laufzeit (abhängig vom Typ des Objektes) die entsprechende Methode aufgerufen werden.
* **Statische Polymorphie (oder Überladung):** Der Aufruf einer Operation wird aufgrund eines konkreten Typs einer Variable oder einer Konstanten auf eine bestimmte Methode abgebildet. Die Auswahl des Aufrufs einer Methode erfolgt dann beispielsweise aufgrund des Parametertyps oder der Anzahl der Parameter.



## **Interface Segregation Principle:**

**Unlce Bob:** Die Nutzer einer Klasse / eines Objektes sollten nicht von Schnittstellen abhängig sein, die sie nicht selbst verwenden

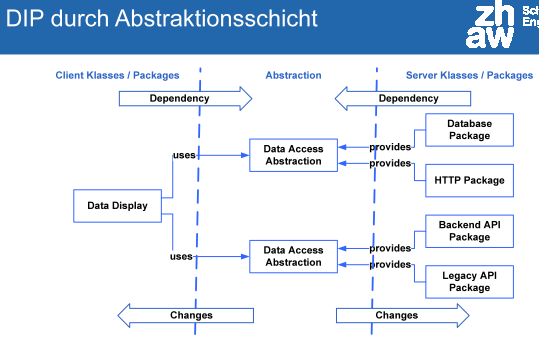
Zur Einhaltung des Prinzips wird jede Schnittstelle wird darauf hin geprüft, ob all ihre Methoden von einem bestimmten Client verwendet werden. Falls dies nicht der Fall, ist zu prüfen, ob eine Trennung der Schnittstelle in mehrere einzelne Schnittstellen sinnvoll ist.for each: explain, problems caused by not respecting it, example, example of violation and how to fix it(see praktika code)



## **Dependency Inversion Principle**

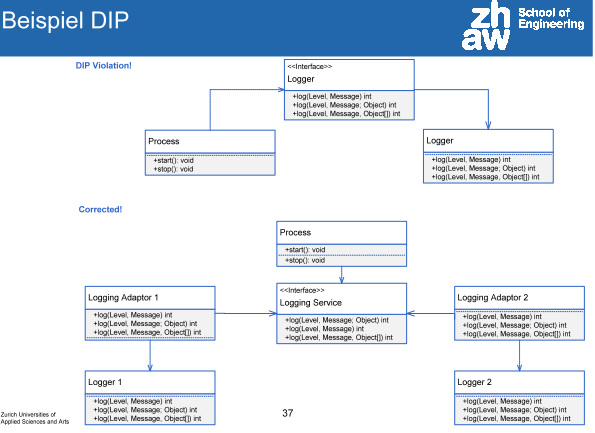
**Unlce Bob:** Module höherer Ebenen sollten nicht von Modulen) abhängen. Sie sollten beide von Abstraktionen abhängen und Abstraktionen sollten nicht von Details abhängen. Details sollten von Abstraktionen abhängen

DIP wird als wichtigstes Design-Prinzip für die Entkopplung von Komponenten eines Objektorientierten Software-Systems angesehen. Es formuliert, wie Objekte über andere abhängige Objekte Bescheid wissen sollten. Und zwar lediglich über eine Abstraktion.



## **Hollywood Prinzip (Don't call me, I'll call you)**

Die Einführung einer Abstraktionsschicht zwischen Klassen, Modulen und Packages erlaubt eine weitere Ausprägung des DIP, die Inversion of Control (IoC). Es geht darum, wer wen aufruft – also wer genau eine bestimmte Meldung initiiert. Wird beispielsweise ein bestimmtes Framework aus den Code aufgerufen, oder kann ein Codestück in ein Framework so eingefügt werden, dass es zum richtigen Zeitpunkt von Framework aufgerufen werden kann (Call-Back).



META:

3 - 4 questions. 1-2 short, 2 longer ones (explain / draw)

15 min

make schedule of