Architektur von Informationssystemen

Hochschule für angewandte Wissenschaften

Sommersemester 2016

Nils Löwe / nils@loewe.io / @NilsLoewe

3. Praktikum

Praktikum 3: Architekturentwurf

Fragen?

4. Praktikum

Frage: Alle Präsentationen in der 12. Vorlesungswoche?

Wiederholung

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Adaptive Systeme

Mikrokernel

Reflexion

Dependency Injection

Mikrokernel

Ziel: Änderung von Systemanforderungen zur Laufzeit dynamisch begegnen.

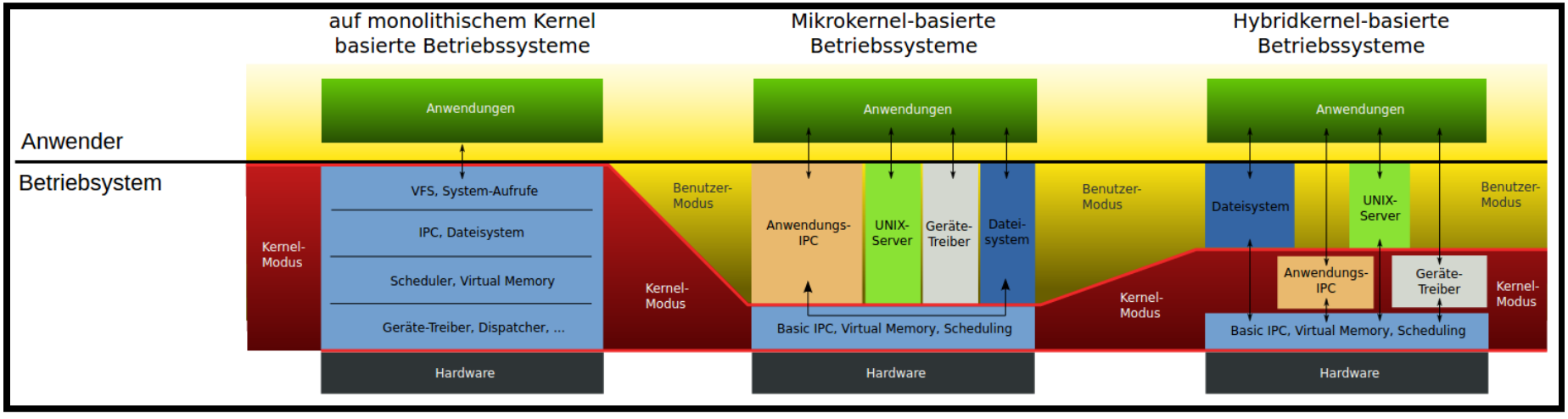
Mikrokernel

Aufgaben

- Der Mikrokernel bietet die Basis für mögliche Erweiterungen
- Der Microkernel koordiniert die Zusammenarbeit.

Microkernel

Herausforderung



Beispiele: Microkernel

- Minix Kernel
- GNU Mach
- AmigaOS
- SymbianOS

Beispiele: Monolithische Kernels

- Linux
- Android
- Windows bis Win98 (DOS Kernel)

Beispiele: Hybrid-Kernels

- MacOS X (Darwin)
- Windows NT (oft als Microkernel bezeichnet)

Mikrokernel

Vorteile

- Separierte Komponenten: Austauschbarkeit
- Treiber im Benutzer-Modus: Sicherheit
- kleine Trusted Computing Base
- Skalierbarkeit
- Zuverlässigkeit
- Transparenz

Mikrokernel

Nachteile

- Leistung
- Komplexität

Domain Driven Design

Ein Beispiel

DDD - Ein Beispiel

Eine Firma bietet Softwareentwicklung als Dienstleistung an.

http://blog.mirkosertic.de/architecturedesign/dddexample

- Aufwand wird nach Stunden verrechnet
- Es gibt festangestellte Softwareentwickler
- Es gibt einen Pool von Freelancern.
- Bisher wird die Zuordnung von Entwicklern zu Projekten in einem Excel-Sheet organisiert

DDD - Ein Beispiel

Probleme mit dem Excel-Ansatz

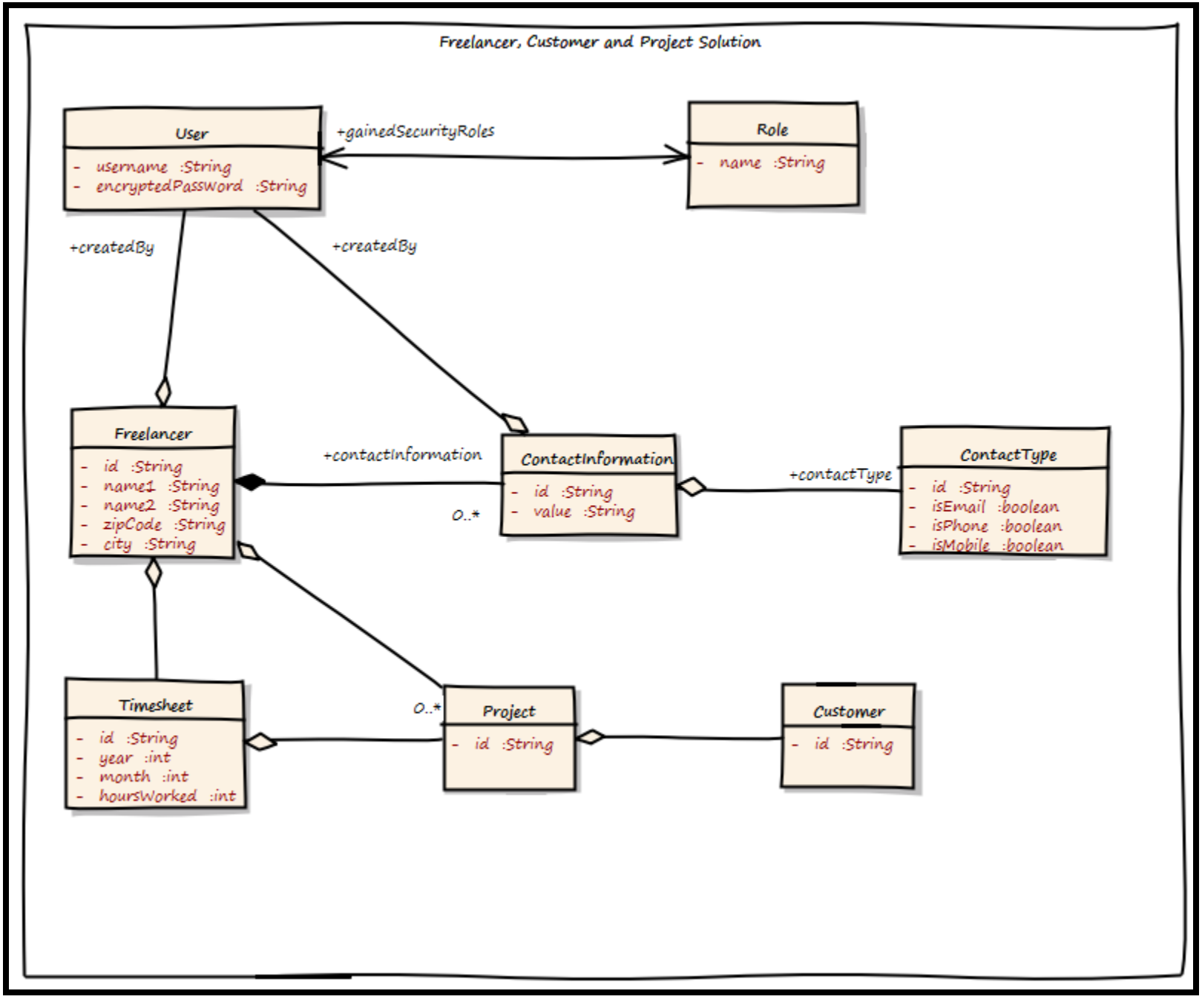
- Excel skaliert nicht auf mehrere Anwender
- Keine Sicherheit
- Kein Logging

DDD - Ein Beispiel

Lösung: Neue Software bauen

- Durchsuchbarer Katalog von Freelancern
- Mehrere Kontaktmöglichkeiten pro Freelancer
- Durchsuchbarer Katalog von Projekten
- Durchsuchbarer Katalog von Projekten
- Timesheets für jeden Freelancer (pro Projekt)

DDD - Ein Beispiel



Straight forward Ansatz

Probleme?

Straight forward Ansatz: Probleme

Großer Objektgraph: Performance Probleme unter Last

Framework wie Hibernate wäre notwendig um das zu vermeiden

Straight forward Ansatz: Probleme

Warum die bidirektionale Verknüpfung zwischen User und Rolle?

Straight forward Ansatz: Probleme

Boolean-Flags um den Kontakttypen zu unterscheiden

Straight forward Ansatz: Probleme

Projekt-Liste in der Freelancer-Klasse:

Freelancer verändern um Projekte zuzufügen

Potentielle Transaktionsprobleme unter Last (mehrere Leute legen gleichzeitig Projekte für den gleichen Kunden an)

Straight forward Ansatz: Probleme

- Kontaktinformation == Kommunikationskanal?
- Das Diagramm ist eher ein Entiy-Relationship-Diagram als ein Software Modell
- Wo ist die Businesslogik?

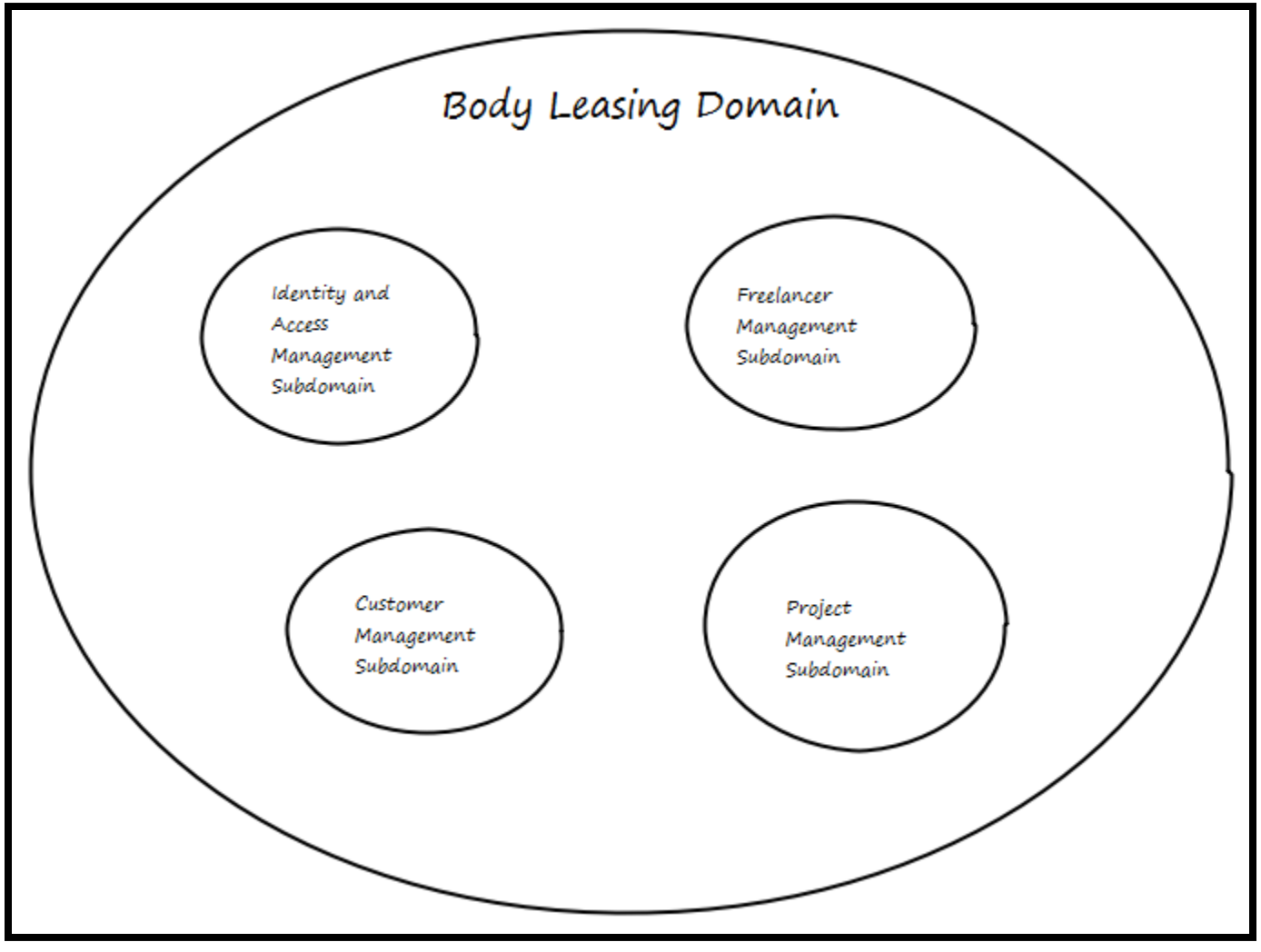
DDD nutzen

Anforderungen: ”Body Leasing Domain”

Context-Map: Komplexität reduzieren durch Subdomains

- Identität und Access Management Subdomain
- Freelancer Management Subdomain
- Kundenmanagement Subdomain
- Projektmanagement Subdomain

DDD - Context Map



DDD - Bounded Context

- Zuordnen von Subdomains zu Teilen der Lösung
- *Building blocks* nutzen (Design Pattern anwenden)
- Die DDD Architekturpattern sind nicht technologieabhängig!

DDD - Ein erster Ansatz

- Jeder Bounded Context enthält Aggregates und Wertobjekte
- Aggregates sind Objekthierarchien
- Nur das Root-Objekt eines Aggregates ist von außen zugreifbar
- Jeder Zugriff auf ein Objekt passiert durch die Aggregates: Bessere Kapselung
- Aggregates und Entites besitzen eine ID
- Wertobjekte haben keine ID und können ihren Zustand nicht ändern
- Jede Zustandsänderung erzeugt ein neues Wertobjekt: Vermeiden von Seiteneffekten

Vollständiger UseCase mit Persistenz

- Persistenz in DDD wird mit "Repositories" umgesetet
- Ein Repository ist durchsuchbar, kann Instanzen liefern und löschen, sowie neue Instanzen ablegen
- Es sollte ein Repository für jedes Aggregate geben

Vollständiger UseCase mit Persistenz

- Ein Client ist ein abstraktes Konzept
- Ein Client kann alles von einem Frontend über einen SOAP Webservice zu einer REST Ressource sein
- Ein Client sendet Befehle an den ApplciationService

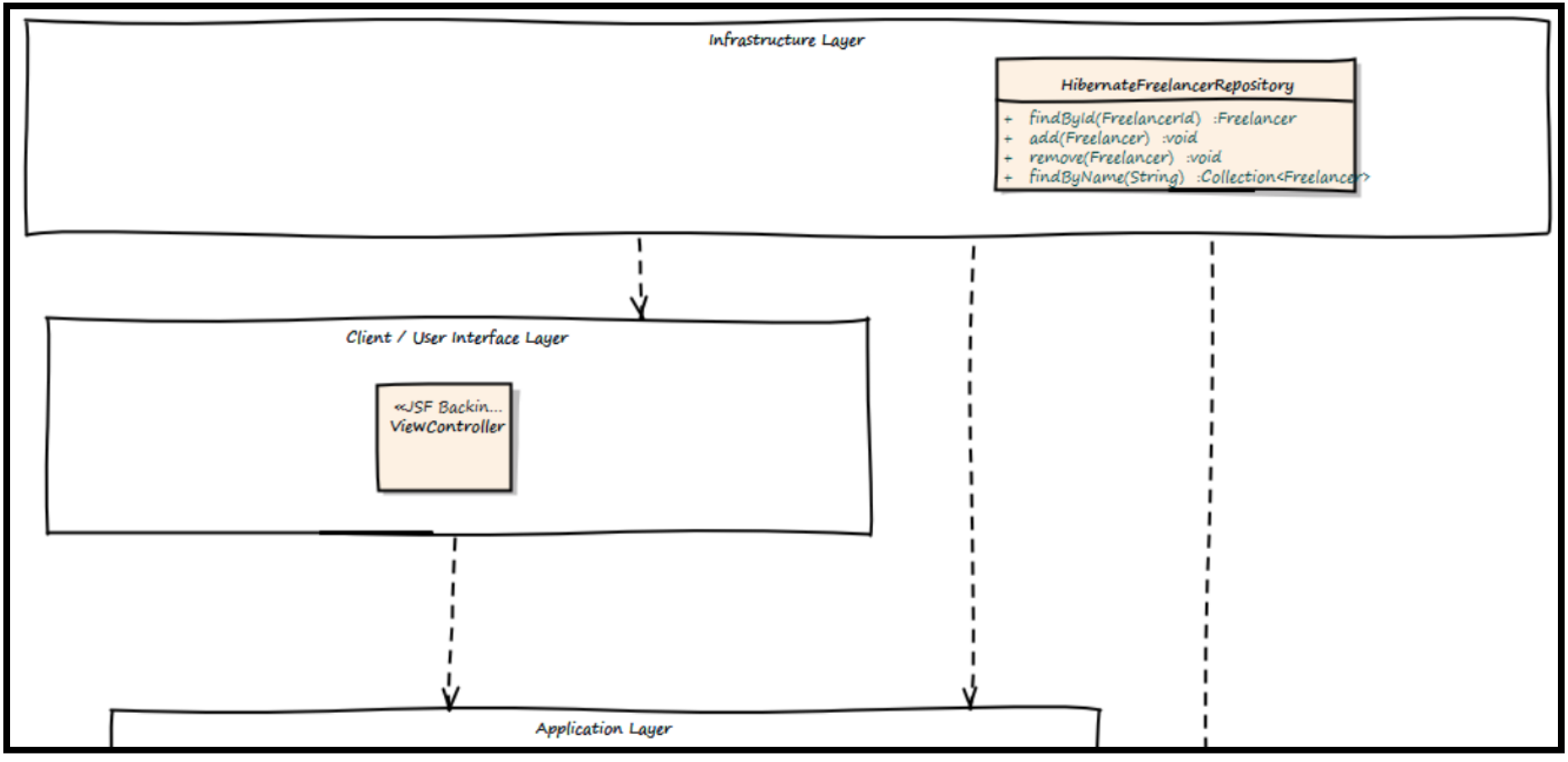
Vollständiger UseCase mit Persistenz

- Der ApplicationService setzt die Befehle in UseCases um
- Der FreelancerApplicationService lädt dasFreelancer Aggregate aus dem FreelancerRepository und ruft moveTo() auf dem FreelancerAggregate auf
- Der FreelancerApplicationService bildet dabei die Transactionsgrenzen.
- Jeder Aufruf erzeugt eine neue Transaktion

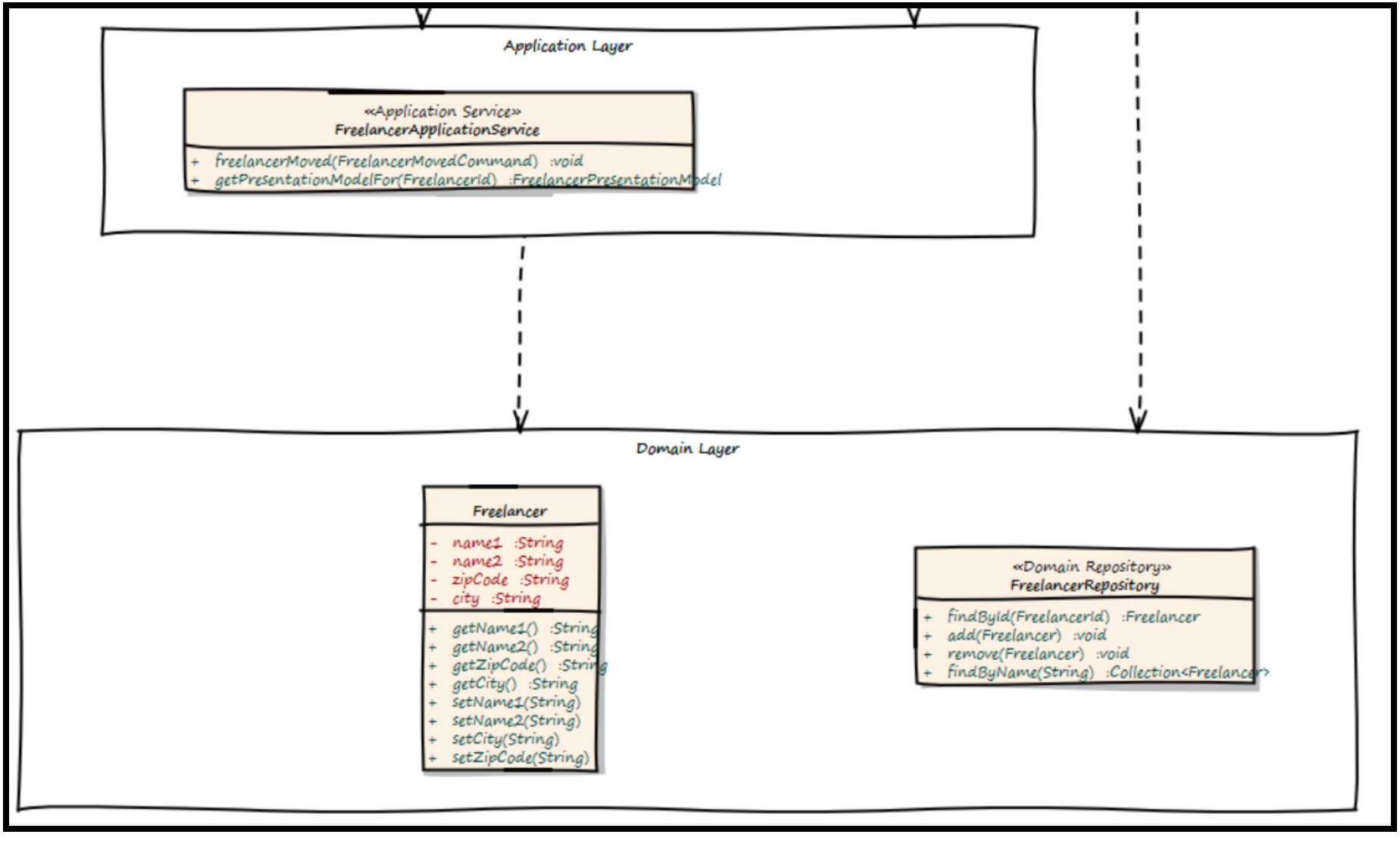
DDD - Applikations-Architektur

- Jeder Bounded Context sollte eine "Deployment Unit" bilden, z.B. ein Java WAR file oder ein EJB JAR
- Die Bounded Contexts sind unabhängig designt, sie sollten daher auch unabhängig implementiert werden

DDD - Layers


Infrastructure Layer
HibernateFreelancerRepository
+ findById(FreelancerId) :Freelancer
+ add(Freelancer) :void
+ remove(Freelancer) :void
+ findByName(String) :Collection<Freelancer>
Client / User Interface Layer
«JSF Backin...
ViewController
Application Layer

DDD - Layers


Application Layer
«Application Service»
FreelancerApplicationService
+ freelancerMoved(FreelancerMovedCommand) :void
+ getPresentationModelFor(FreelancerId) :FreelancerPresentationModel
Domain Layer
Freelancer
- name1 :String
- name2 :String
- zipCode :String
- city :String
+ getName1() :String
+ getName2() :String
+ getZipCode() :String
+ getCity() :String
+ setName1(String)
+ setName2(String)
+ setCity(String)
+ setZipCode(String)
«Domain Repository»
FreelancerRepository
+ findById(FreelancerId) :Freelancer
+ add(Freelancer) :void
+ remove(Freelancer) :void
+ findByName(String) :Collection<Freelancer>

DDD - Vorteile

- Der Domain Layer basiert nicht auf anderen Teilen der Architektur
- Die Repository Implementierung kann getauscht werden, ohne die Businesslogik zu beeinflussen

DDD - Domain Layer

- Enthält die Businesslogik, keine Abhängigkeiten der Infrastruktur
- Die Modelle sollten nach dem CQS(Command-Query-Separation) Prinzip entworfen werden
- --> Query Methoden geben lediglich Daten zurück ohne Zustände zu ändern
- --> Command Methoden ändern den State

DDD - Application Layer

- Der Application Layer nimmt Kommandos des User Interface Layer an
- Der Application Layer ruft UseCase Implementierungen im Domain layer auf
- Der Application Layer biete Transactioskontrolle für Business Operationen

DDD - Infrastructure Layer

Der Infrastructure Layer bietet Infrastrukturabhängige Teile für alle anderen Layer

DDD - User Interface Layer

- Der User Interface Layer konsumiert Application Services und ruft Funktionalität der Businesslogik auf diesen Services auf.
- Jeder Aufruf ist eine neue Transaktion
- Der User Interface Layer kann beliebig implementiert sein, z.B. ein SOAP webservice, eine REST Resource oder eine Swing/AWT GUI

DDD - Zusammenfassung

Domain-driven Design is object oriented programming done right. *(Eric Evans)*

Fragen?

Rückblick auf Architekturmuster

Chaos zu Struktur / Mud-to-structure

Verteilte Systeme

Interaktive Systeme

Adaptive Systeme

Domain-spezifische Architektur

Chaos zu Struktur / Mud-to-structure

- Organisation der Komponenten und Objekte eines Softwaresystems
- Die Funktionalität des Gesamtsystems wird in kooperierende Subsysteme aufgeteilt
- Zu Beginn des Softwareentwurfs werden Anforderungen analysiert und spezifiziert
- Integrierbarkeit, Wartbarkeit, Änderbarkeit, Portierbarkeit und Skalierbarkeit sollen berücksichtigt werden

Chaos zu Struktur / Mud-to-structure

Layers

Pipes und Filter

Blackboard

Domain-driven Design

Verteilte Systeme

- Verteilung von Ressourcen und Dienste in Netzwerken
- Kein "zentrales System" mehr
- Basiert auf guter Infrastruktur lokaler Datennetze

Verteilte Systeme

Serviceorientierte Architektur (SOA)

Peer-to-Peer

Client-Server

Interaktive Systeme

- Strukturierung von Mensch-Computer-Interaktionen
- Möglichst gute Schnittstellen für die Benutzer schaffen
- Der eigentliche Systemkern bleibt von der Benutzerschnittstelle unangetastet.

Interaktive Systeme

Model View Controller (MVC)

Model View Presenter

Presentation-Abstraction-Control (PAC)

Adaptive Systeme

- Unterstützung der Erweiterungs- und Anpassungsfähigkeit von Softwaresystemen.
- Das System sollte von vornherein mögliche Erweiterungen unterstützen
- Die Kernfunktionalität sollte davon unberührt bleiben kann.

Adaptive Systeme

Mikrokernel

Reflexion

Dependency Injection

Fragen?

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Dokumentation von Architekturen

Nutzen von Templates

Beispiele:

- arc42
- Normen
- Software Guidebook

ARC42

(Dr. Gernot Starke / Dr. Peter Hruschka)

http://www.arc42.de/

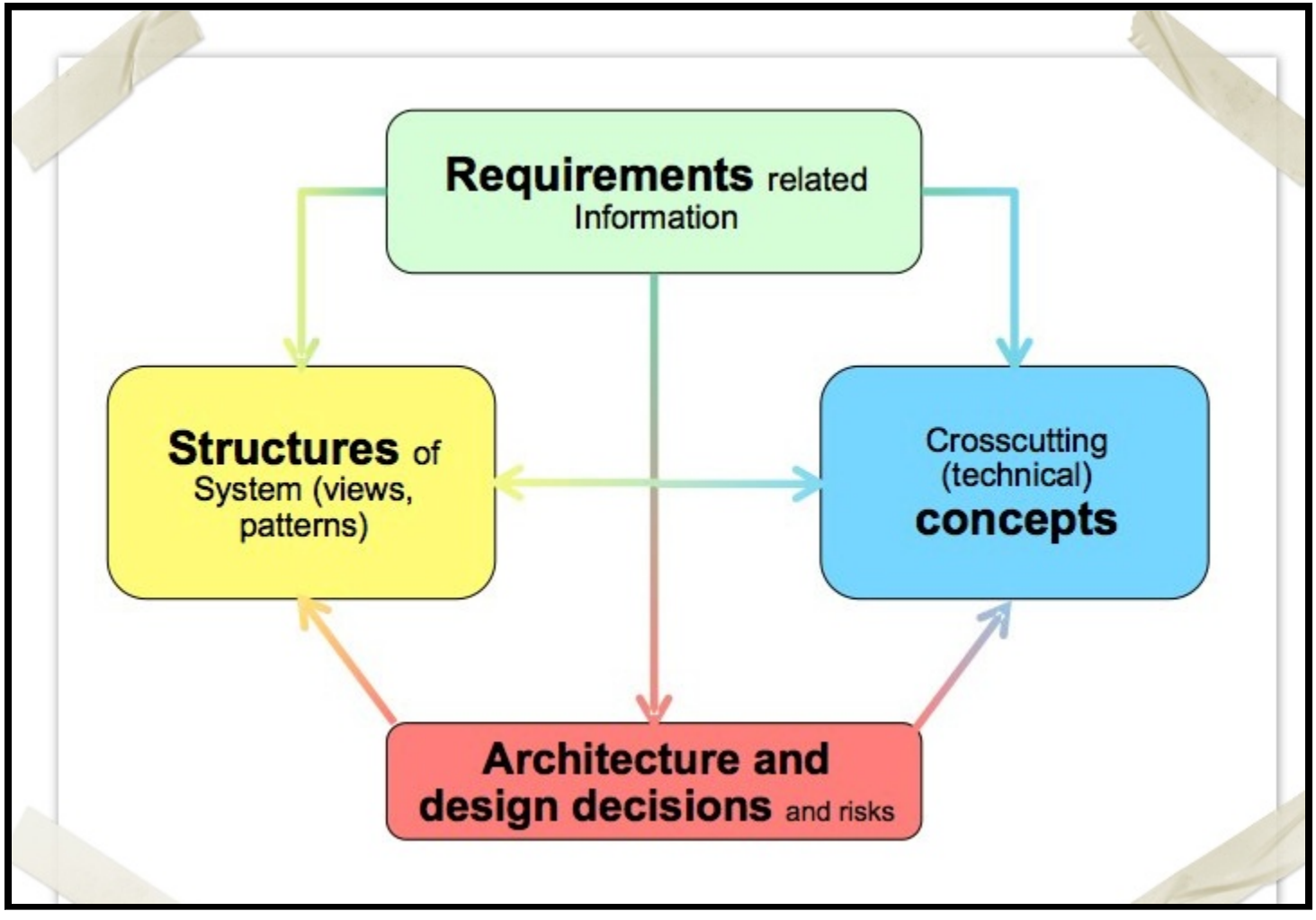
arc42 unterstützt Software- und Systemarchitekten. Es kommt aus der Praxis und basiert auf Erfahrungen internationaler Architekturprojekte und Rückmeldungen vieler Anwender.

Dokumentation von Architekturen

ARC42

1. Einführung und Ziele
2. Randbedingungen
3. Kontextabgrenzung
4. Lösungsstrategie
5. Bausteinsicht
6. Laufzeitsicht
7. Verteilungssicht
8. Querschnittliche Konzepte/Muster
9. Entwurfsentscheidungen
10. Qualitätsszenarien
11. Risiken
12. Glossar

ARC42


Requirements related Information
Structures of System (views, patterns)
Crosscutting (technical) concepts
Architecture and design decisions and risks

IEEE Standards

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards or implementations thereof.

IEEE Standards

- IEEE 802: LAN
- IEEE 802.3: Carrier sense multiple access with collision detection (CSMA/CD)
- IEEE 802.11: Wireless LAN
- IEEE 830: Recommended Practice for Software Requirements Specifications
- IEEE 1394: FireWire/i.Link Bussysteme
- IEEE 1471: Recommended Practice for Architectural Description of Software-Intensive Systems
- IEEE 9945: Portable Operating System Interface (POSIX®)

IEEE Standards - Kosten

- IEEE 830: 171$
- Journals: 26.500$ / Jahr
- Standards Library: *custom quote*

Software Guidebook

- Template von Simon Brown aus dem Buch *"Software Architecture for Developers"*
- Buch: https://leanpub.com/software-architecture-for-developers
- Beispiel: https://leanpub.com/techtribesje (kostenlos)

Software Guidebook

Welche Informationen wünsche ich mir, wenn ich in ein neues Projekt komme?

- Karten
- Sichten
- Geschichte
- Praktische Informationen!

Software Guidebook

Product vs project documentation

Software Guidebook

1. Context
2. Functional Overview
3. Quality Attributes
4. Constraints
5. Principles
6. Software Architecture
7. External Interfaces
8. Code
9. Data
10. Infrastructure Architecture
11. Deployment
12. Operation and Support
13. Development Environment

Fragen?

Vorbereitung auf Klausuraufgaben

- Was waren die Gründe für Softwarearchitektur?
- Was sollte eine Kontext-Sicht enthalten?
- In welcher Beziehung stehen Architektur und Design?
- Was besagt 'Conways Law'?
- Nennen und erläutern Sie drei Arten von Architekturmustern
- Für welche Systeme wird das MVC Muster typischerweise verwendet?

Fragen?

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Software Guidebook: Ein Beispiel

- Beispiel: https://leanpub.com/techtribesje
- Website: https://techtribes.je

Fragen?

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Frameworks

Was ist ein Framework?

Frameworks

Ein Framework ist kein fertiges Programm, es stellt einen Rahmen zur Verfügung.

Frameworks

- Ein Framework ist eine semi-vollständige Applikation.
- Es stellt für Applikationen eine wiederverwendbare, gemeinsame Struktur zur Verfügung.
- Entwickler integrieren dass Framework in ihre eigene Applikation ein, und erweitern es um die Applikationslogik.
- Frameworks stellen eine kohärente Struktur zur Verfügung, anstatt eine einfache Menge von Hilfsklassen anzubieten.

Frameworks

- Ein Framework gibt in der Regel die Anwendungsarchitektur vor.
- Ein Framework definiert den Kontrollfluss der Anwendung
- Ein Framework definierte die Schnittstellen für die Applikation.

Frameworks

Eine allgemeingültige Definition von Frameworks gibt es aufgrund der hohen Anzahl von Diversitäten nicht.

Frameworks

Vorteile

- Wiederverwendung von Code
- Grundfunktionalität muss nicht immer wieder implementiert werden
- Es existieren genormte Schnittstellen z.B. zu Datenbanken
- Frameworks erleichtern die Programmierarbeit und sparen Entwicklungszeit
- Frameworks können den Stil entscheidend verbessern

Frameworks

Nachteile

- Frameworks erhöhen die Komplexität der Anwendung
- Frameworks stecken voller Know-How und eine effiziente Anwendung erfordert Profiwissen
- Frameworks nehmen nicht das Verständnis der Grundlagen ab, auch wenn oft so gearbeitet wird
- Dokumentationen sind größtenteils unzureichend

Frameworks

Wie wähle ich ein Framework aus?

Popularität und Community

Wie wahrscheinlich finde ich Hilfe und Entwickler?

Philosophie

A tool developed by professionals for their own needs will obviously meet the demands of other professionals.

Sustainability / Nachhaltigkeit

Kann das Framework "mitwachsen"?

Support

Gibt es professionelle Hilfe neben der Community?

Technik

Wie gut ist das Framework implementiert?

Security

Wie schnell werden Sicherheitslücken reportet und geschlossen?

Dokumentation

Wie gut, ausführlich und verständlich ist das Framework dokumentiert?
Wie aktuell ist die Doku?

Lizenz

Ein Framework unter GPL Lizenz verlangt z.B., dass die Anwendung auch unter der GPL steht. MIT dagegen nicht.

Entwickler-Kapazität

Wie wahrscheinlich werde ich Entwickler finden?

Hosting Requirements

Wie einfach kann ich die Anwendung deployen?

Einfache Installation?

Wie schnell ist ein neues Projekt eingerichtet?

Lernkurve

Wie komplex ist das Framework?

Inhalte / Funktionen?

- AJAX
- Authentication
- Authorization
- Caching
- Data Validation
- Templating engine
- URL mapping / rewriting
- ...?

DB Abstraktion / ORM

Wie einfach/mächtig ist das Object Relational Mapping?

JS Library

Welche JS Bibliothek ist per default dabei?

Unit Testing

Wie sehr ist TDD Teil der Philosophie, wie ist der Tool-Support?

Skalierbarkeit?

Wie einfach lässt sich die Anwendung bei Bedarf skalieren?

Ausprobieren!

Reviews lesen reicht nicht, Erfahrungen und das look&feel zählen!

Wann brauche ich ein Framework?

- Die Anwendung basiert im Wesentlichen auf CRUD Operationen
- Die Anwendung wird relativ groß
- UI und Anwendungslogik sollen getrennt werden
- Authentication und andere Grundfunkionen werden intensiv genutzt
- Zeitdruck + Das Framework ist bereits bekannt

Wann brauche ich KEIN Framework?

- Ich brauche nur einen kleinen Teil des Frameworks (z.B. ORM)
- Zeitdruck + Das Framework ist nicht bekannt
- "Frameworks lösen jedes Problem"

Überblick über aktuelle Frameworks

https://en.wikipedia.org/wiki/Comparison_of_web_frameworks

Vorstellung konkreter Technologien & Frameworks

- Ruby on Rails
- Spring Boot (moovel Group GmbH)
- Docker (Akra GmbH)
- Microservices (Wer liefert was GmbH)
- NodeJS
- AngularJS
- Twitter Bootstrap

Fragen?

Ruby on Rails

Ruby on Rails

Ruby: Überblick

- Erste Version 1993
- Verbreitet seit 2006 (durch Rails)
- Objektorientiert
- Interpretiert
- Dynamisch getypt
- "Script Sprache"

Ruby on Rails

Ruby: Tradeoffs

- Flexibilität vs. Sicherheit
- Laufzeit-Effizienz vs. Produktivität

Ruby on Rails

Ruby: Beispielcode

```
>> properties = ['object oriented', 'duck typed', 'productive', 'fun']
=> ["object oriented", "duck typed", "productive", "fun"]
>> properties.each {|property| puts "Ruby is #{property}."}
Ruby is object oriented.
Ruby is duck typed.
Ruby is productive.
Ruby is fun.
```

Ruby on Rails

Ruby: Installation

Installation via

- OS Paketmanager
- rvm
- rbenv

Ruby on Rails

Ruby: Programming Model

```
>> 4
=> 4
>> 4.class
=> Fixnum
>> 4 + 4
=> 8
>> 4.methods
=> ["inspect", "%", "<<", "singleton_method_added", "numerator", ...
"*", "+", "to_i", "methods", ...
]
```

Alles ist ein Objekt

Ruby on Rails

Ruby: Programming Model

```
>> x = 4
=> 4
>> x < 5
=> true
>> x <= 4
=> true
>> x > 4
=> false
>> false.class
=> FalseClass
>> true.class
=> TrueClass
```

Alles ist ein Objekt

Ruby on Rails

Ruby: Programming Model

```
>> 4 + 'four'
TypeError: String can't be coerced into Fixnum
from (irb):51:in `+'
from (irb):51
>>
=>
>>
=>
4.class
Fixnum
(4.0).class
Float
>> 4 + 4.0
=> 8.0
```

Duck Typing

Ruby on Rails

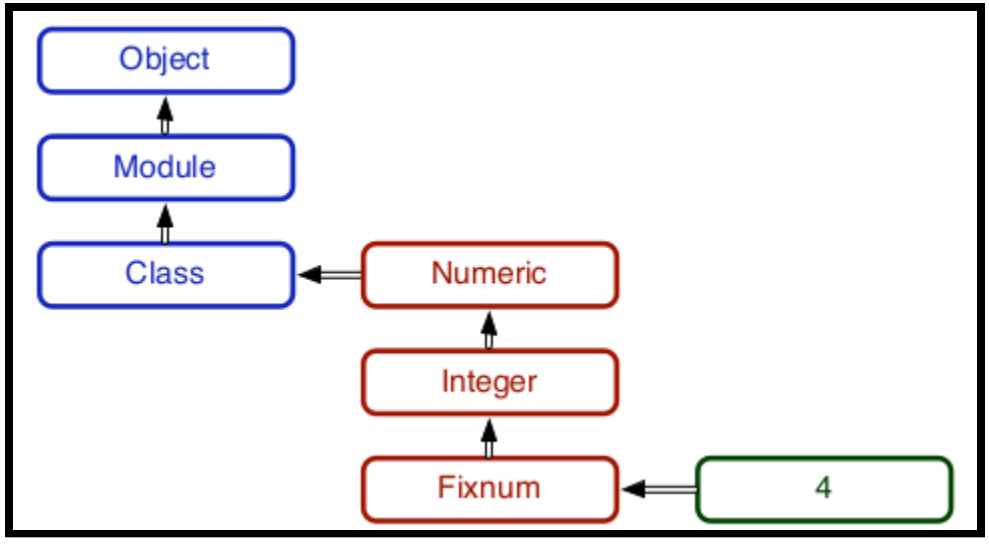
Ruby: Programming Model

```
If it walks like a duck and quacks like a duck, it's a duck.
```

- Dynamische Typisierung: Zur Ausführungszeit interpretiert
- Starke Typisierung: Typsicherheit

Ruby on Rails

Ruby: Metamodel



Ruby on Rails

Ruby: Programming Model

```ruby
class Tree
  attr_accessor :children, :node_name

  def initialize(name, children=[])
    @children = children
    @node_name = name
  end

  def visit_all(&block)
    visit &block
    children.each {|c| c.visit_all &block}
  end

  def visit(&block)
    block.call self
  end
end
```

Ruby on Rails

Ruby: Programming Model

```
module ToFile
  def filename
    "object_30853860.txt"
  end
  def to_f
    File.open(filename, 'w') {|f| f.write(to_s)}
  end
end

class Person
  include ToFile
  attr_accessor :name

  def initialize(name)
    @name = name
  end
  def to_s
    name
  end
```

Ruby: Literaturempfelungen

- Programming Ruby 1.9 & 2.0 (Dave Thomas / Andy Hunt)
- Practical Object-Oriented Design in Ruby (Sandy Metz)
- Confident Ruby: 32 Patterns for Joyful Coding (Avdi Grimm)

Ruby on Rails: Geschichte

- 2004: Entwickelt als Basis für *Basecamp*
- Version 1.0 (2005)
- Version 1.1 (2006) - Scripting Engines, Performance
- Version 1.2 (2007) - REST Support, MIME-type routing, UTF-8
- Version 2.0 (2007) - REST als Default
- Version 2.1 (2008) - Named Scopes, Migrationen mit Timestamp
- Version 2.2 (2008) - Internationalisierung, Threadsicherheit
- Version 2.3 (2009) - Template Engine
- Version 3.0 (2010) - Modularisierung: Einzelne Teile verwendbar
- Version 3.1 (2011) - Asset Pipeline
- Version 4.0 (2013)
- Version 5.0 (2016)

Ruby on Rails: Bestandteile

- Active Support: Ruby-Erweiterungen von Rails
- Active Record: Objektabstraktionsschicht (ORM)
- Action Pack: Request-Behandlung und Response-Ausgabe.
- Action View: Templates
- Action Mailer: E-Mail-Versand und -Empfang
- Active Resource: Routing, REST, XML-RPC

Ruby on Rails: Grundlagen

- Basiert auf Ruby
- Model-View-Controller Architektur
- „Don’t repeat yourself“
- „Convention over Configuration“
- Scaffolding
- Datenbankmigrationen

„Don’t repeat yourself“

- Jede Information sollte nur ein einziges Mal vorhanden sein
- z.B. ActiveRecord liest das DB-Scheme direkt aus der DB
- z.B. Rails erstellt für das Model automatisch Getter- und Setter-Methoden
- Vorteil: Informationen werden nicht inkonsistent wenn eine Stelle verändert wird

Ruby on Rails: Grundlagen

„Convention over configuration"

- Rails erwartet sinnvolle Standardwerte
- --> z.B. Primärschlüssel einer Tabelle ist ID vom Typ Integer
- --> ein Modell mit dem Namen Customer liegt in der Datei # {Rails.root}/app/models/customer.rb
- --> Die zugehörige Tabelle heißt customers heißt

Scaffolding

- Es gibt Generatoren für alle Standardfälle
- Models, Controller, Views, Mailer, Migrationen, ...
- Konventionen werden eingehalten
- Web-Anwendungen lassen sich sehr schnell entwickeln
- Wenn in der Datenbank etwa ein Feld hinzugefügt wird, erscheint es auch sofort in der entsprechenden View/New/Edit-Ansicht.
- Scaffolding ist vor allem für Prototyping gedacht

Ruby on Rails: Ein Beispiel

- https://github.com/railstutorial/sample_app_rails_4
- https://www.railstutorial.org/
- https://www.railstutorial.org/book

Fragen?

Vorbereitung auf Klausuraufgaben

- Was ist ein Framework?
- Nennen sie drei Vorteile für die Verwendung von Frameworks!
- Wann ist die Verwendung eines Frameworks sinnvoll?
- Nennen sie drei Auswahlkriterien für Frameworks!
- Für welche Systeme wird das Microkernel Architekturmuster typischerweise verwendet?
- Was ist der Grundgedanke hinter dem "Software Guidebook"?

Fragen?

Unterlagen: ai2016.nils-loewe.de