Architektur von Informationssystemen

Hochschule für angewandte Wissenschaften

Sommersemester 2018

Nils Löwe / nils@loewe.io / @NilsLoewe

Wiederholung

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/

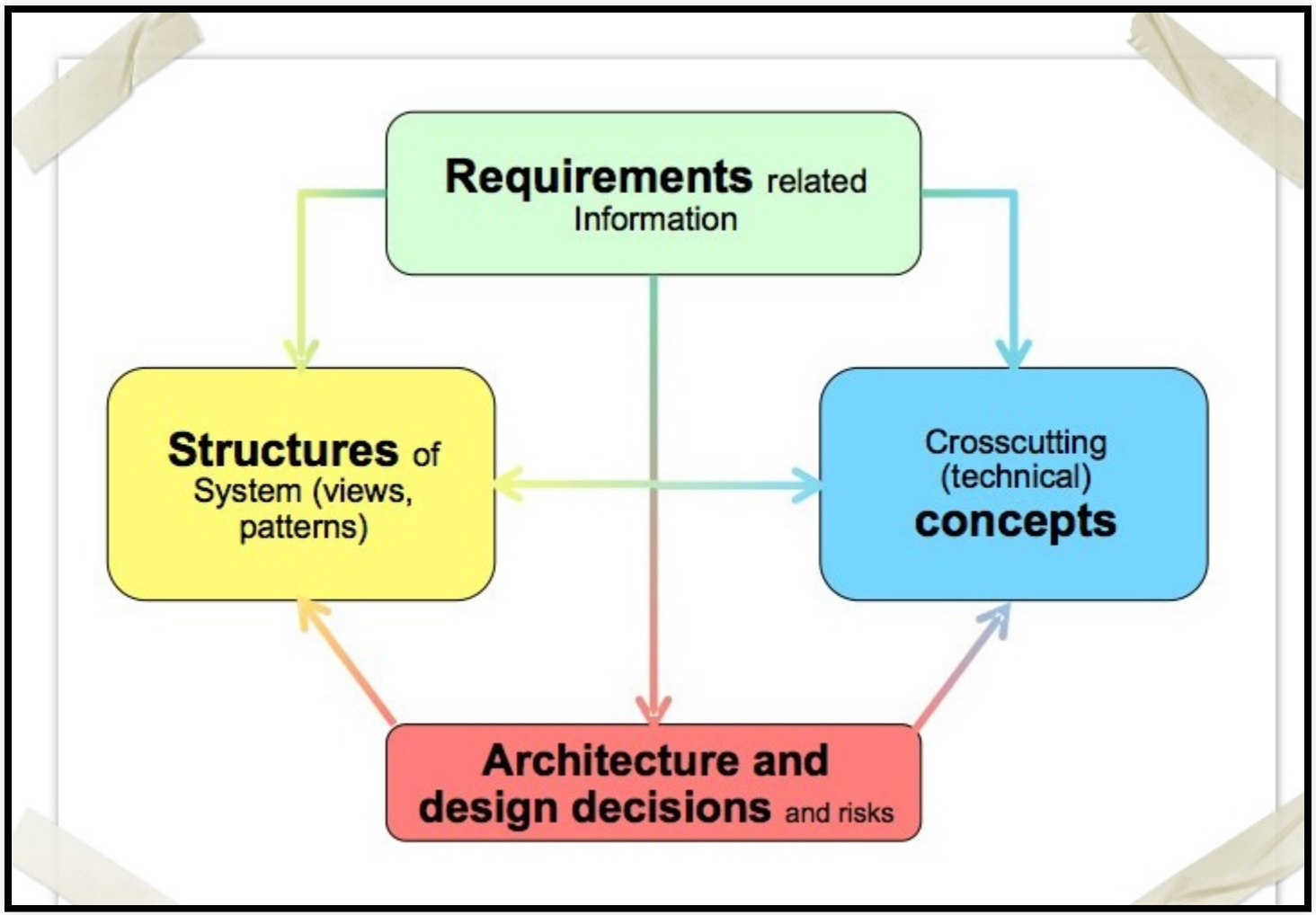
arc42 unterstützt Software- und Systemarchitekten. Es kommt aus der Praxis und basiert auf Erfahrungen internationaler Architekturprojekte und Rückmeldungen vieler Anwender.

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

ARC42

1. Einführung und Ziele

- Aufgabenstellung
- Qualitätsziele
- eine Kurzfassung der architekturrelevanten Anforderungen (insb. die nichtfunktionalen)
- Stakeholder

ARC42

2. Randbedingungen

Welche Leitplanken schränken die Entwurfsentscheidungen ein?

- Technische Randbedingungen
- Organisatorische Randbedingungen
- Konventionen

ARC42

3. Kontextabgrenzung

- Fachlicher Kontext
- Technischer- oder Verteilungskontext

ARC42

4. Lösungsstrategie

Wie funktioniert die Lösung? Was sind die fundamentalen Lösungsansätze?

ARC42

5. Bausteinsicht

Die statische Struktur des Systems, der Aufbau aus Implementierungsteilen.

ARC42

6. Laufzeitsicht

Zusammenwirken der Bausteine zur Laufzeit, gezeigt an exemplarischen Abläufen ("Szenarien")

ARC42

7. Verteilungssicht

Deployment: Auf welcher Hardware werden die Bausteine betrieben?

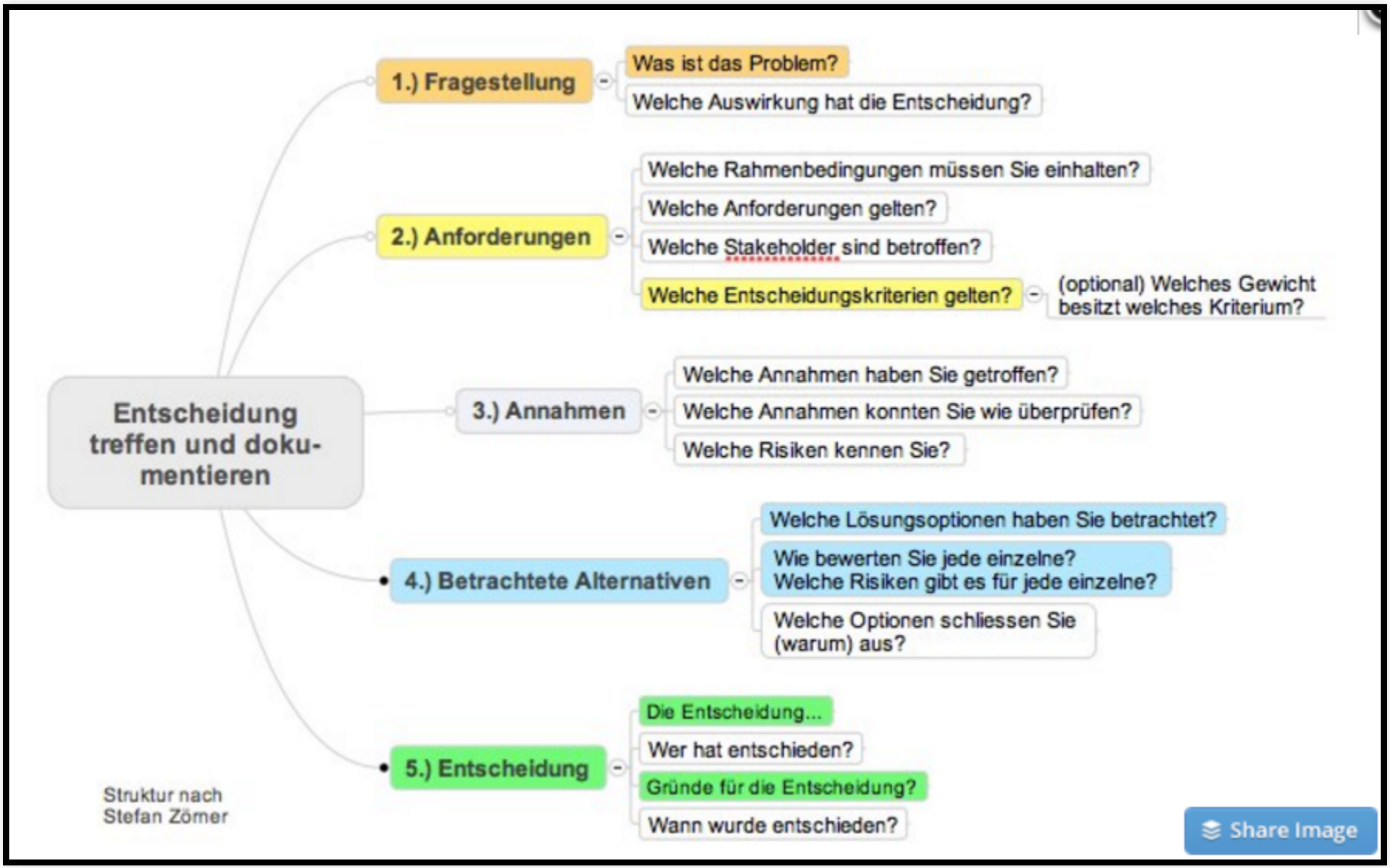
ARC42

8. Querschnittliche Konzepte und Muster

- Wiederkehrende Muster und Strukturen
- Fachliche Strukturen
- Querschnittliche, übergreifende Konzepte
- Nutzungs- oder Einsatzanleitungen für Technologien
- Oftmals projekt-/systemübergreifend verwendbar!

ARC42

9. Entwurfsentscheidungen



ARC42

10. Qualitätsszenarien

Qualitätsbaum sowie dessen Konkretisierung durch Szenarien

ARC42

11. Risiken

Eine nach Prioritäten geordnete Liste der erkannten Risiken

”Risikomanagement ist Projektmanagement für Erwachsene”

ARC42

12. Glossar

Die wichtigsten Begriffe der Software-Architektur in alphabetischer Reihenfolge

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

Software Guidebook: Ein Beispiel

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

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

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

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

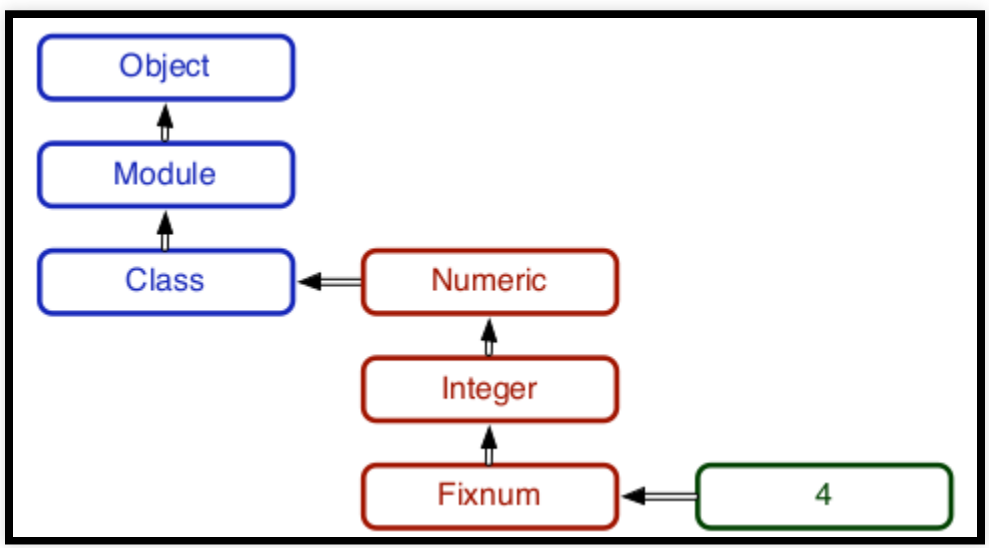
Ruby: Programming Model

```
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: Literaturempfelungen

- Programming Ruby (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)
- Version 5.2 (2018)

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

Ruby on Rails: Grundlagen

„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

Ruby on Rails: Grundlagen

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

Webserver-Unterstützung

- Integrierter Application-Server: WEBrick
- Für Produktion: Apache, Nginx, Lighttpd, ... z.B. mit Phusion Passenger

Schichten: Model

- Typischerweise basierend auf einer relationalen Datenbank
- Zugriff mit Hilfe von ActiveRecord hergestellt (ORM-Schicht)
- --> Klasse <> Tabelle
- --> Attribut <> Spalte
- --> Objekt <> Zeile
- Support für: SQLite, DB2, Informix, MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Sybase, ...
- Alternativen: Sequel, Datamapper, XML, ...

Schichten: Controller

- Basiert auf *ActionController*, Bestandteil von *ActionPack*
- Ein Controller kapselt die Geschäftslogik im Model von der Darstellung der View
- Ruby-Code

Schichten: View

- Präsentationsschicht *ActionView*, Bestandteil von *ActionPack*
- Diverse Ausgabeformate/Template-Engines werden unterstützt:
- -> HTML. ERB, Slim, Haml, Sass, CoffeeScript, ...
- -> XML – z. B. für XHTML und Web Services
- -> JSON
- -> JavaScript – RJS-Templates
- -> Binärdaten

Ruby on Rails: Ein Beispiel

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

Fragen?

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Ruby on Rails

Spring Boot (moovel Group GmbH)

NodeJS

AMQP

Docker (Akra GmbH)

AngularJS

Microservices (Wer liefert was GmbH)

Twitter Bootstrap

Node.js

Node.js

Node.js ist eine serverseitige Plattform für Netzwerk-Anwendungen, die auf der Google Chrome JavaScript Engine (V8 Engine) basiert.

- easily build fast and scalable network applications
- event-driven, non-blocking I/O model
- lightweight, efficient, perfect for data-intensive real-time applications
- large JavaScript Library

Node.js

Node.js = Runtime Environment + JavaScript Library

https://github.com/nodejs/node

Node.js

Releases

- Current: Released from active development branches of this repository
- LTS: Releases that receive Long-term Support, with a focus on stability and security
- Nightly: Versions of code in this repository on the current Current branch

Node.js

LTS Releases

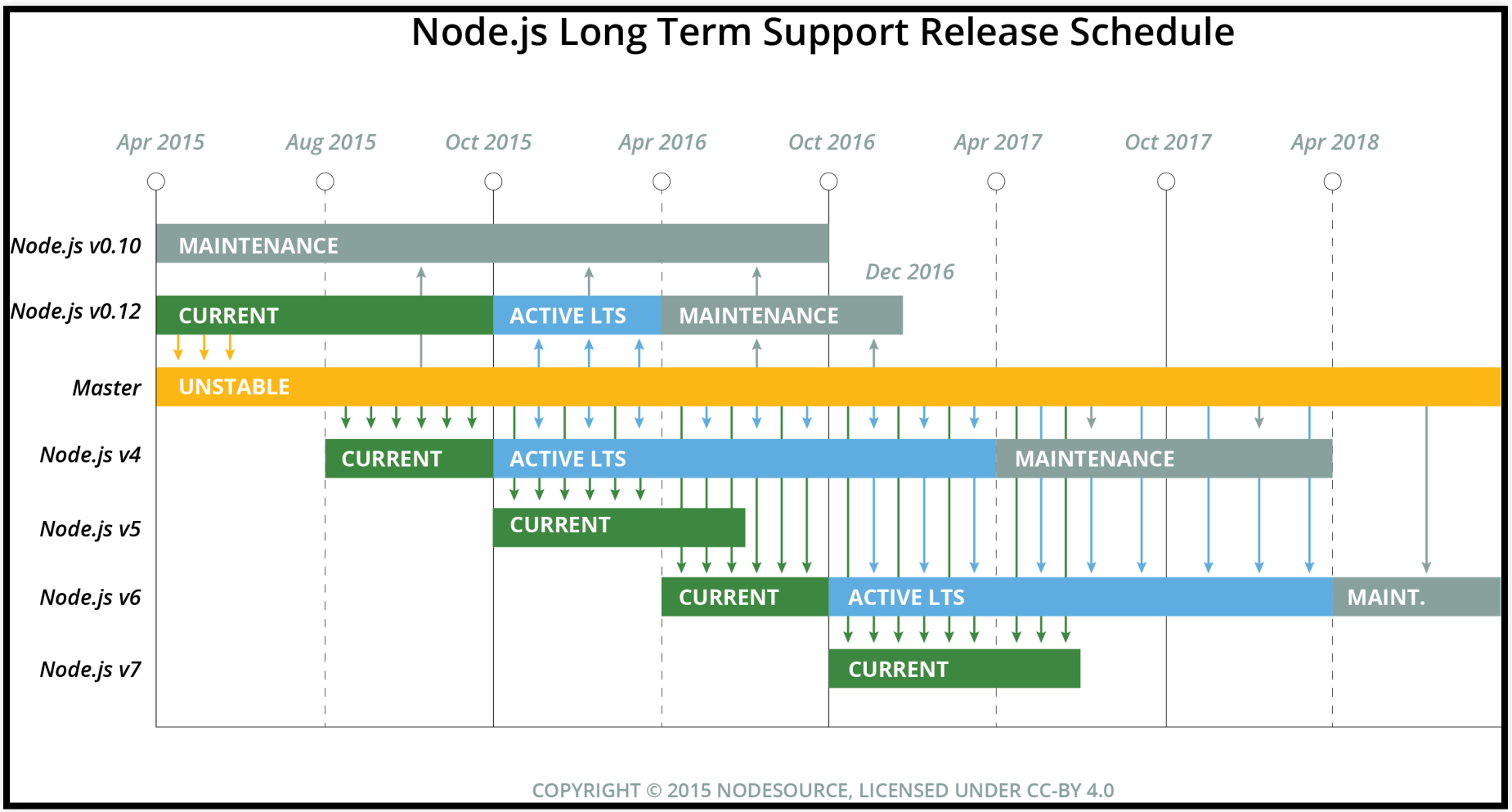
- Every second Current release line (major version) will become an LTS line
- 18 months of Active LTS support
- Additional 12 months of Maintenance
- LTS releases are less frequent and will attempt to maintain consistent major and minor version numbers
- There are no breaking changes or feature additions, except in some special circumstances

Node.js

Releases

- 06/2009: Version 0.0.3
- 59 unstable Releases
- 2013: Version 0.10.x
- 2015: Version 0.12.x
- 2015: Version 1.0.x
- 2015: Version 2.0.x
- 2015: Version 3.0.x
- 2015: Version 4.0.x (LTS)
- 2015: Version 5.0.x
- 2016: Version 6.0.x (LTS)

Node.js Long Term Support Release Schedule



Node.js

Chrome V8

Google's high performance, open source, JavaScript engine.

- Open source, high-performance JavaScript engine
- Written in C++
- Used in Google Chrome
- It implements ECMAScript as specified in ECMA-262, 3rd edition
- Runs on Windows XP or later, Mac OS X 10.5+, and Linux systems that use IA-32, ARM or MIPS processors
- V8 can run standalone, or can be embedded into any C++ application

Features of Node.js

- Asynchronous and Event Driven
- Very Fast, being built on Google Chrome's V8 JavaScript Engine
- Single Threaded but Highly Scalable: Single threaded model with event looping
- No Buffering: Node.js applications never buffer any data
- Node.js is released under the MIT license.

Who Uses Node.js?

- eBay
- General Electric
- GoDaddy
- Microsoft
- PayPal
- Uber
- Wikipins
- Yahoo!
- ...

Node.js

Anwendungsfälle

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications

Node.js

Grenzen

- CPU intensive applications

Node.js

Installation

- Install the Distro-Stable Version (linux, MacOS)
- Install Using NVM (linux, MacOS)
- Install from source/tar (linux, MacOS, Windows, FreeBSD, OpenBSD, AIX, ARM, ...)

Node.js

REPL: Read Eval Print Loop

- Read - Reads user's input, parse the input into JavaScript data-structure and stores in memory.
- Eval - Takes and evaluates the data structure
- Print - Prints the result
- Loop - Loops the above command until user press ctrl-c twice.

Node.js

REPL: Read Eval Print Loop

- REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.
- REPL can be started by simply running node on shell/console without any argument as follows.

```
$ node
>
```

Node.js: NPM

Node Package Manager

- https://www.npmjs.com/
- Find, share, and reuse packages of code from hundreds of thousands of developers
- 3 million developers and thousands of companies use npm

Node.js: NPM

Statistiken (Stand 13.05.2016)

- 283,432 total packages
- 57,724,266 downloads in the last day
- 1,040,776,238 downloads in the last week
- 4,169,158,104 downloads in the last month

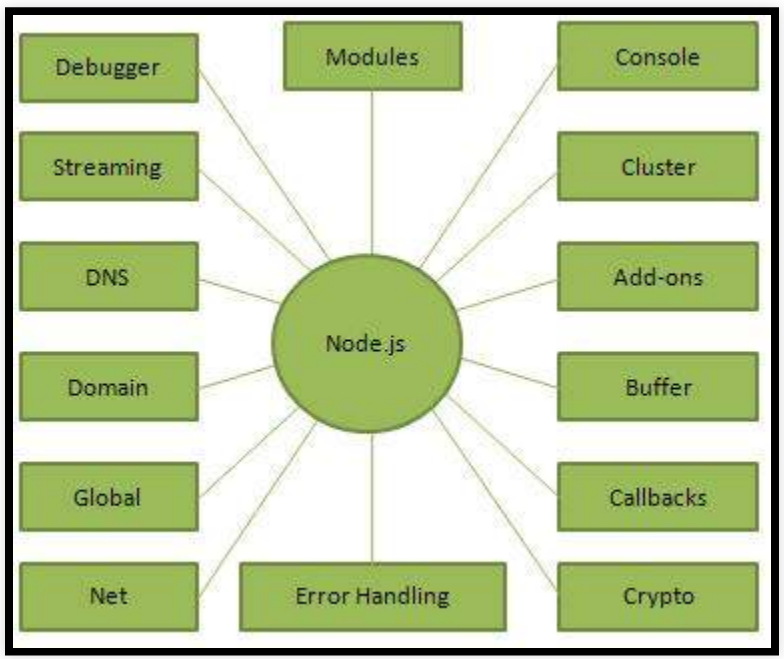
Node.js: NPM

Geschäftsmodell

- Open Source: FREE
- Private Account: $7 per user / month
- Orgs: $16 per user / month
- Enterprise: $2000 per year
- npm Enterprise Pro

Node.js

Concepts



Node.js

package.json

- Manage locally installed npm packages
- Document project dependencies
- Specify the versions used packages
- Make a build reproducable

Node.js

package.json: Minimum

- "name"
- "version"

```
{
  "name": "my-cool-package",
  "version": "1.0.0"
}
```

package.json: Attribute

- name
- version
- description
- homepage
- author
- contributors
- dependencies - *npm automatically installs all the dependencies mentioned here in the node_module folder*
- repository - *repository type and url of the package*
- main - *entry point of the package*
- keywords

Node.js

Create a module

```
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sane defaults.

See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg> --save' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (webmaster)
...

$ npm adduser
Username: abcdefg
Password:
Email: (this IS public) abcdefg@gmail.com
```

Node.js

Callbacks

- Asynchronous equivalent for a function
- A callback function is called at the completion of a given task
- All APIs of Node are written is such a way that they supports callbacks.

Node.js

Callbacks: Blocking Example

main.js

```
var fs = require("fs");

var data = fs.readFileSync('input.txt');

console.log(data.toString());
console.log("Program Ended");
```

input.txt

```
This is the input.txt file
```

```
$ node main.js
This is the input.txt file
Program Ended
```

Node.js

Callbacks: Non-Blocking Example

main.js

```
var fs = require("fs");

fs.readFile('input.txt', function (err, data) {
if (err) return console.error(err);
console.log(data.toString());
});

console.log("Program Ended");
```

input.txt

```
This is the input.txt file
```

```
$ node main.js
Program Ended
This is the input.txt file
```

Node.js

Concurrency

- Node js is a single threaded application
- Concurrency is implemented via events and callbacks
- As every API of Node js is asynchronous and single threaded
- Async function calls maintain the concurrency.
- Node uses the observer pattern for events

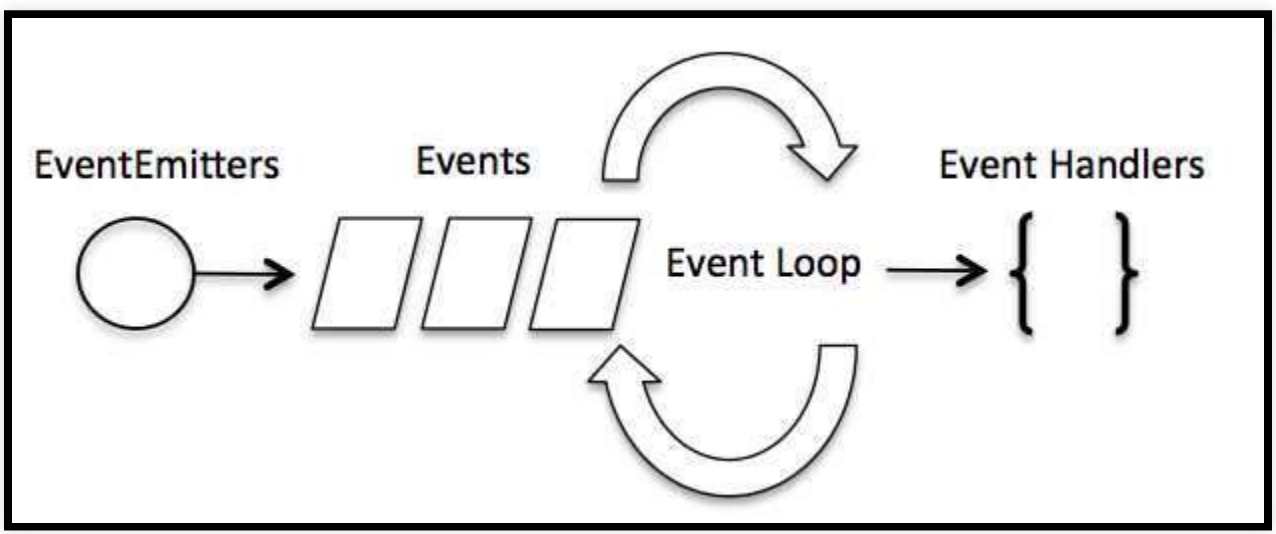
Node.js

Event Driven Programming

In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.

Node.js

Event Driven Programming



Node.js

Event Driven Programming

- The functions which listens to events act as Observers
- When an event gets fired, its listener functions start executing
- Node.js has multiple in-built events available through the **events module** and **EventEmitter class**

Node.js

Event Loop

```
// Import events module
var events = require('events');

// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();

// Bind event and even handler as follows
eventEmitter.on('eventName', eventHandler);

// Fire an event
eventEmitter.emit('eventName');
```

Node.js

Event Loop

All objects which emit events are instances of events.EventEmitter.

```
// Import events module
var events = require('events');
// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();
```

- When an instance faces any error, it emits an 'error' event.
- When new listener is added, 'newListener' event is fired
- When a listener is removed, 'removeListener' event is fired.

Node.js

Streams

Streams are objects that let you read data from a source or write data to a destination in continous fashion.

Node.js

Streams

In Node.js, there are four types of streams.

- Readable - used for read operation.
- Writable - used for write operation.
- Duplex - can be used for both read and write operation.
- Transform - A type of duplex stream where the output is computed based on input.

Node.js

Streams

Each type of Stream is an EventEmitter instance and throws several events

- data - fired when there is data is available to read.
- end - fired when there is no more data to read.
- error - fired when there is any error receiving or writing data.
- finish - fired when all data has been flushed to underlying system

Node.js

File System

- Node implements File I/O using simple wrappers around standard POSIX functions
- Every method in fs module have synchronous as well as asynchronous form
- Asynchronous methods take a parameter as "completion function callback"
- It is preferred to use asynchronous methods instead of synchronous methods

Node.js

Standard Modules

- OS Module: Provides basic operating-system related utility functions.
- Path Module: Utilities for handling and transforming file paths.
- Net Module: Servers and clients as streams
- DNS Module: DNS lookup, operating system name resolution functionalities
- Domain Module: I/O operations as a single group.
- http module: Create either HTTP client of server

Node.js

Express

- Minimal and flexible Node.js web application framework
- Robust set of features to develop web and mobile applications
- Allows to set up middlewares to respond to HTTP Requests
- Defines a routing table for actions based on HTTP Method and URL
- Allows to dynamically render HTML Pages based on passing arguments to templates

Node.js

JXcore

- Compile and distribute it a Node.js app
- Open source project
- Packaging and encryption of source files and other assets

Node.js

Beispiel

Fragen?

Vorbereitung auf Klausuraufgaben

- In welcher Beziehung stehen Architektur und Design?
- Was besagt 'Conways Law'?
- Nennen und erläutern Sie drei Arten von Architekturmustern
- Was sind zwei Grundprinzipien, die hinter Ruby on Rails stecken?
- Nennen sie drei Nachteile für die Verwendung von Frameworks!
- Wann ist die Verwendung eines Frameworks nicht sinnvoll?
- Nennen sie drei Auswahlkriterien für Frameworks!
- Für welche Systeme wird das MVC Architekturmuster typischerweise verwendet?
- Wie ist das ARC42 Template entstanden?

Fragen?

Unterlagen: ai2018.nils-loewe.de