DHBW Stuttgart

Verteilte Systeme
Web-Programmierung

Diplom Wirtschaftsinformatiker (BA) Thomas Pohl & Christian Romeyke

Version: 02.09

Ihre Erwartungshaltung?

Aus der Modulbeschreibung

Kerninhalte:

- Methoden und Werkzeuge für die Entwicklung von Web-Anwendungen: z. B. Modellierungs- und - Implementierungswerkzeuge, integrierte Entwicklungsumgebungen, Frameworks, Architekturen, Infrastruktur
- Übertragungsprotokolle und APIs zwischen Client und Server (z.B. HTTP, HTTPS, WebSockets, XMLHttpRequest, Fetch API)
- HTML, CSS, JavaScript als clientseitige Web-Technologien und aktuelle APIs (z.B. HTML5 und verwandte Technologien)
- Kommunikation zwischen einzelnen Komponenten Web-basierter Anwendungen
- Optimierung von Webseiten für verschiedene Zielsysteme

Zusatzinhalte:

- Vertiefung von Frameworks
- Fallbeispiel zu RESTful Webservices
- Dynamische serverseitige Erzeugung von Webseiten

Zielstellung in unseren Worten:

- Funktionsweise von Web Anwendung verstehen
- Auswirkungen und Konzepte von verteilten Systemen verstehen
- Frameworks für Webentwicklung kennen lernen
- **Web Anwendungen auf Basis von JavaScript und Java Technologie selbstständig erstellen**
- Fehler in einer Anwendung analysieren

Portfolioprüfung Bestandteile

- **Programmentwurf**
  - Wann: 6.10.
  - Erreichbare Punkte: 40
  - Einzelleistung
  - Bewertungskriterien:
    - Erfüllung der Aufgabenstellung
    - **Erläuterung der Umsetzung**
    - Erstellen eines Sequenzdiagramms
- **Präsentation**
  - Gruppenleistung (3er Gruppen)
  - Wann: 28.09. – vormittags
  - Erreichbare Punkte: 30

Rollen in der Softwareentwicklung

Benutzer
- Nutzt das System → daraus ergibt sich der Wert der Anwendung
- Hat Anforderungen an das System
- Auftraggeber und Benutzer sind meist separate Rollen

Architekt
- Entwirft Konzeption für die Anwendung
- Stellt Umsetzung der Funktionalen und nicht-Funktionalen Anforderungen sicher
- Trifft grundsätzliche Technologieentscheidungen

Entwickler
- Entwirft Anwendungsdesign
- Entwickelt die Anwendung und führt erste grundlegende Tests durch

Administrator
- Installation der Anwendung
- Überwachung des Betriebs

DevOps

Weitere Rollen: Projektmanager, Product Owner, Tester, Auftraggeber (meist nicht der Benutzer), …

Definition

Ein verteiltes System ist ein System,

- das aus räumlich verteilten und vernetzten Computern (Hardware und Betriebssysteme) besteht, die miteinander kommunizieren.

Ein verteiltes System ist,

- Eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen.

Quelle: A. Tanenbaum, M. Stehen - „Verteilte Systeme“, Pearson Studium 2008

„Sie wissen, dass Sie eines haben, wenn der Absturz eines Computers, von dem Sie nie zuvor gehört haben, verhindert, dass Sie Ihre Arbeit erledigen können.“

Quelle: Leslie Lamport

Beispiele

- Skype
- Google Suche
- Web-Shop (Amazon, eBay …)
- ERP System eines Unternehmens – z.B. SAP
- Vernetzte Prozessoren im Auto
- Smart Home

„Gegenbeispiele“

- MS Word, PowerPoint etc.
- Host basierte Terminalanwendung
- Lokales Computerspiel
- Einfaches Java Programm

Dahinter verbergen sich verschiedene Probleme die für verteilte Systeme gelöst werden müssen:

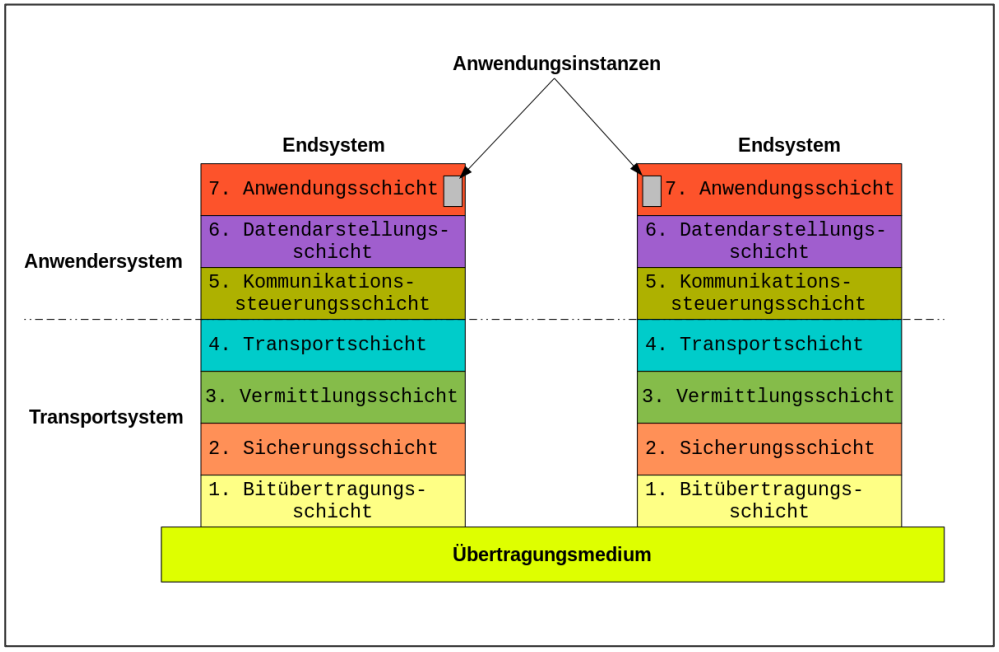
- **Synchronisierung** von Datenverarbeitungsvorgängen (Nebenläufigkeit)
- **Fehlertoleranz**, z.B. bei Ausfall einer Komponente oder Verbindung → Single Point of Failure vermeiden
- **Sicherheit** gegen ungewollte Zugriffe oder Veränderungen
- **Konsistenz** der verwendeten Daten
- **Skalierbarkeit**, z.B. für große Nutzerzahlen oder Datenmengen

Quiz:

- Wie alt ist das Internet?
- Wer hat‘s erfunden?
- Warum wurde es erfunden?
- Welche Farbe hatte das Internet ursprünglich?
- Was konnte man mit dem Internet anfangs machen?

Historie des Internet

- 1969 - **Arpanet** (ein Projekt der Advanced Research Project Agency (ARPA) des US-Verteidigungsministeriums) – Vorläufer des Internet
  - Nutzung der knappen Rechenkapazitäten
- 1971 – **E-Mail**
- Andere Dienste:
  - Telnet: textbasierte Fernsteuerung von Rechnern
  - FTP: Dateiaustausch
- …
- 1989 – **World Wide Web (WWW)** - Projekt an der Forschungseinrichtung CERN, Tim Berners-Lee
  - Kernstandards: HTTP, HTML, URLs
  - 1993 – Mosaic Browser



Quelle: http://de.wikipedia.org

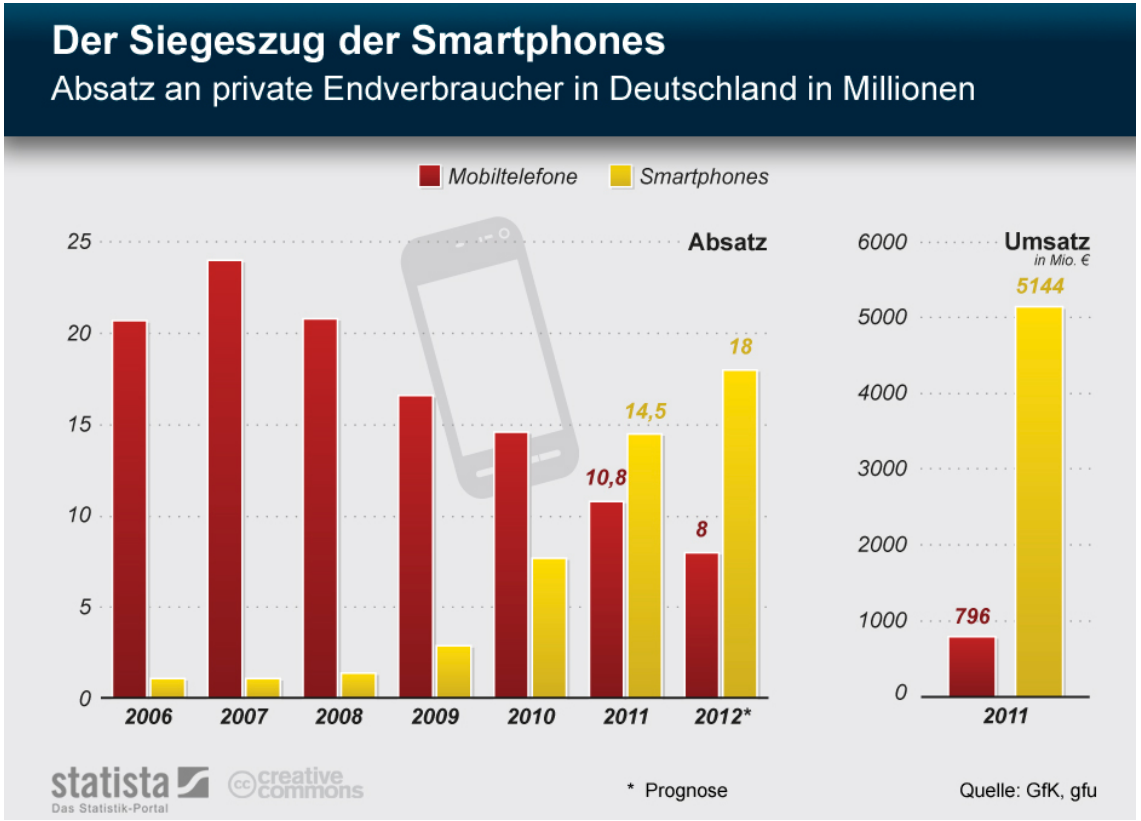
Entwicklungsstufen des WWW (1/2)

- Statischer Inhalt
  - Über Hyperlinks vernetzte Inhalte
  - Wirtschaftliche Nutzung: „Visitenkarte im Netz“
- Web Anwendungen
  - Dateneingabe und Übermittlung an einen Server
  - Wirtschaftliche Nutzung: Web-Shops
  - 1994 - Gründung von Amazon
- ~ 2003 - „Web 2.0“
  - „user generated content“ – z.B. Kommentare & Bewertungen
  - Neue Technologien, die Web Seiten interaktiver gestalteten

Entwicklungsstufen des WWW (2/2)

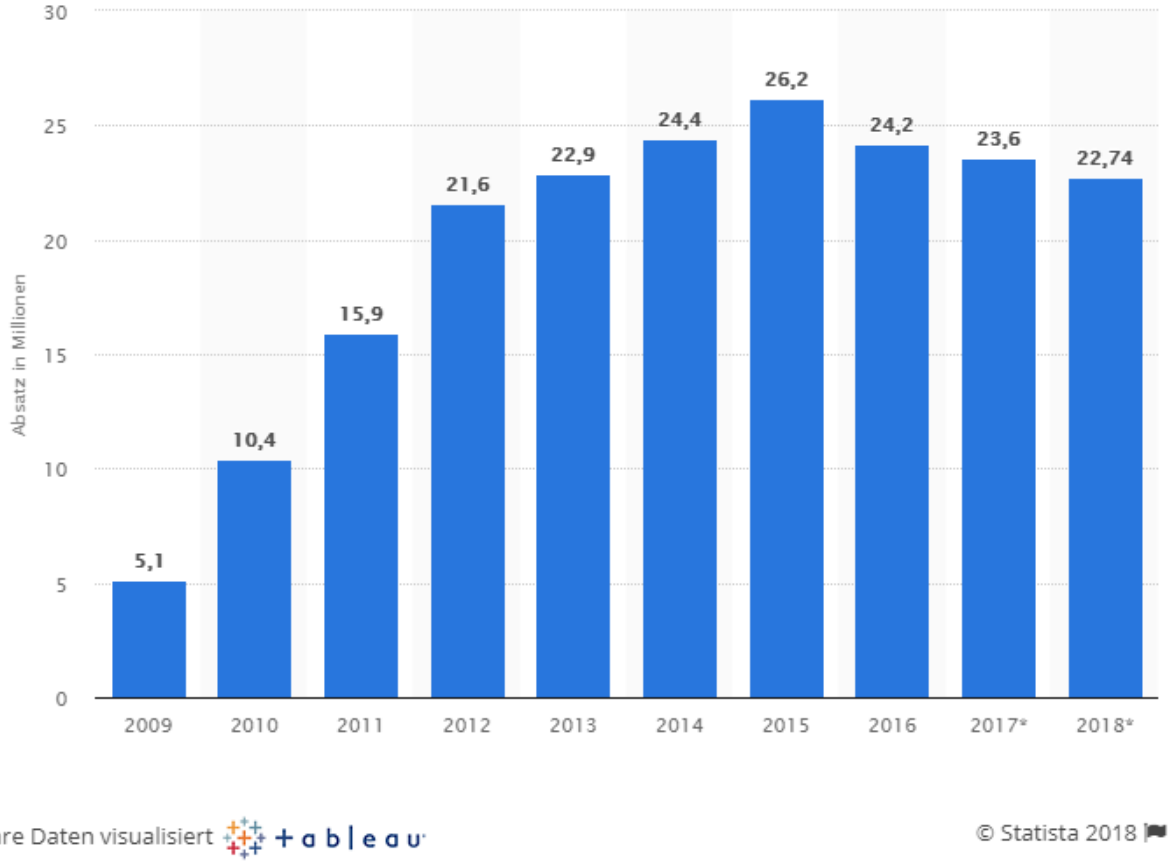
- Google (1998)
  - Einfache Suchseite
  - Ausgefeilter Algorithmus zur Priorisierung von Suchergebnissen
- Wikipedia (2001)
- Social Media
  - Facebook (2004), Blogs, YouTube (2005), Instagram (2010)
- Mobile
  - Ausweitung der Netznutzung auf SmartPhones, TV-Geräte, Fahrzeuge, ….
- Cloud, Internet of Things (IoT)
- Cognitive Anwendungen
- …

Die rasante Verbreitung von Smartphones veränderte die Nutzung des Internets.



- Beispiel: iPhone von Apple (weltweit)
  - 2007 ~1,4 Mio.
  - 2008 ~11,5 Mio.
  - 2013 ~150 Mio.
- Weltweit Smartphone
  - 2014 1,3 Mrd.
- Tablets, Wearables

Absatz von Smartphones in Deutschland in den Jahren 2009 bis 2018 (in Millionen Stück)



Entwicklung geht weiter mit Wearables, Persönlichen Assistenten (Amazon Alexa, Google Home), Smart Home …

Ist das Internet ein Verteiltes System?

HTML im Überblick

Die **Hypertext Markup Language** (HTML) ist

eine textbasierte Auszeichnungssprache zur Strukturierung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten.

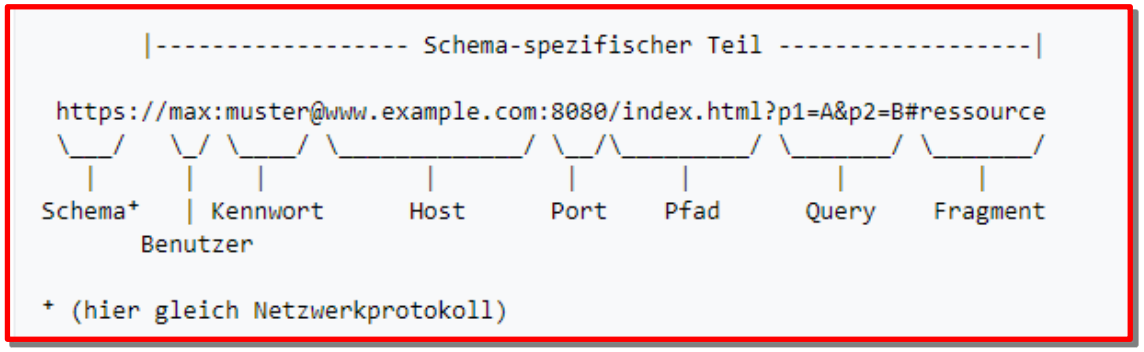
Quelle: http://de.wikipedia.org/wiki/Html

- Standard entwickelt durch **World Wide Web Consortium (W3C)** und der **Web Hypertext Application Technology Working Group (WHATWG)**
- Aktuelle Version **HTML5** (https://www.w3.org/TR/2014/REC-html5-20141028/)
- Ersetzt HTML 4.01 und den XHTML Standard (folgt den XML Syntaxregeln und ist dadurch leichter zu parsen)
  - Seit Oktober 2014 verabschiedet
- HTML eine rein deskriptive Sprache
- Wesentliche Eigenschaft von HTML ist die „Vernetzung“ von Dokumenten durch URLs

URL – „Uniform Resource Locator“

- Identifiziert und lokalisiert eine Ressource, zum Beispiel eine Webseite
- URLs haben protokollübergreifend den gleichen Aufbau (HTTP, FTP, SMB, MAILTO)
- HTTP URLs erlauben das „Deep-Linking“ in Web-Anwendungen

Aufbau einer URL



Quelle: http://de.wikipedia.org/wiki/URL

- Beispiel:
  - mailto:susi@dhbw.de?cc=max@dhbw.de&subject=Hitzefrei&body=Freut%20euch

Grundgerüst einer HTML Seite

```
<!DOCTYPE html>
```

Doctype

```
<html>

<head>
    <meta charset="UTF-8" />
    <title>Sample HTML5 Grundgerüst</title>
    <meta name="description" content="Beschreibung" />
    <link href="style.css" type="text/css" rel="stylesheet" />
</head>
```

Header

```
<body>

</body>
```

Body

```
</html>
```

Grundelement von HTML

→ Übung html01

Elemente werden durch sog. Start- und Ende-Tags „geklammert“; z.B. **<html>** und **</html>**

Titel des Dokuments (Head)

Überschrift im Text

Textabsatz

Attribute

```
<!DOCTYPE html>
<html>
  <head>
     <title>HTML-Beispiel</title>
  </head>
  <body>
    <h1>Hallo Welt!</h1>
      <p>Dies ist ein Standard-Text</p>
       <a href="http://link-wo-hin.de">Linktext</a>
      <button type="input"/>
  </body>
</html>
```

Cascading Style Sheets

Grundlegende Idee: **Formatierungsinformationen unabhängig vom Inhalt** definieren. D.h. beispielsweise alle Tabellenköpfe sehen gleich aus, obwohl das Format nur an einer Stelle definiert wird. Zentrale Style-Sheets werden auch zur Realisierung einer „Corporate Identity“ genutzt.

CSS kann: Schriftarten festlegen, Rahmen und Schatten gestallten, Elemente positionieren, Text ausrichten, das Seitenlayout bestimmen, Hintergrundbilder einbinden ...

Style Sheets können wie folgt definiert werden:

- „inline“ als Attribut eines Tags → gültig für das eine Tag
- „internal“ als <style>-element im html-Head → gültig für die ganze Seite
- „external“ als extra css-Datei → gültig für mehre Seiten

Cascading Style Sheets

Aufbau von CSS Statements:

- **BEI WEM** soll **WELCHES ELEMENT** welchen **WERT** annehmen:

```
h1,th        {color:          red;}
p            {margin-left:    20px;}
```

  - Überschriften (h1) und Tabellenköpfe (th) haben die Farbe rot
  - Absätze (p) sind von Links eingerückt

- „Bei Wem“ = Selektor kann sich auf verschiedene Dinge beziehen

- „Welches Element welchen Wert“ = Deklaration – in der geschweiften Klammer können verschieden Elemente mit Semikolon getrennt aufgelistet werden.

Cascading Style Sheets - Selektoren

→ Übung html02

Der Selektor bestimmt auf welche HTML-Elemente die nachfolgende Formatierung angewendet wird. Es gibt verschiedene Typen von Selektoren:

- Html-Typenselektoren
  - Beispiel: h1 {color:red;}
  - Wird auf alle HTML-Elemente dieses Typs angewendet (z.B. <h1>Überschrift</h1>)
- ID-Selektoren
  - Beispiel: #1234 {color:red;}
  - Wird auf das HTML-Elemente mit dieser ID angewendet. Die ID kann frei gewählt werden, muss jedoch innerhalb eines HTML-Dokumentes eindeutig sein.
  - z.B. text</h1>
- Class-Selektoren
  - Beispiel: .roter_text {color:red;}
  - Wird alle HTML-Elemente mit dieser Class angewendet. Die Class kann frei gewählt werden.
  - z.B. text
- ... und einige mehr z.B. dynamische Pseudoklassen :hover
- Selektoren können kombiniert werden!

Responsive Webdesign

→ Übung html03, html04

- ... Bedeutet, dass die das Design einer Webseite (gleicher Code!) sich an die Eigenschaften der Darstellung z.B. Bildschirmgröße anpaßt
- Wird erreicht z.B. Durch:
  - Spezifische CSS definitionen welche durch eine Media-query gekennzeichnet sind („@media“)
  - JavaScript
- **Responsive Webdesign im Vgl. zu gerätespezifischem Design**
  - **Vorteil**
    - Geringerer Pflegeaufwand, da nur **eine** Implementierung für die Unterstützung unterschiedlicher Geräte und Auflösungen notwendig ist
  - **Nachteil**
    - höherer Testaufwand
    - Ggf. sehr unterschiedliche Anforderungen der Nutzer (Mobile vs. Desktop) nicht abbildbar
- Beispiel:
  - https://wiki.selfhtml.org/wiki/CSS/Anwendung_und_Praxis/mehrspaltige_Layouts

Dynamisches HTML – Abgrenzung

- Statische Websites liefern die HTML-Dateien und zugehörigen **Ressourcen** so aus, wie sie auf dem Server gespeichert sind. Es erfolgt keine Anpassung zur Laufzeit.
  - jede Änderung am Inhalt erfordert einen Eingriff des Administrators
  - Beispiel: die Homepage vom „Rasthaus zum Goldenen Löwen“
- Dynamische Websites / Web-Applikationen generieren durch **serverseitige Logik** (Zugriff auf Datenbanken etc.) dynamisch Teile des Inhalts
  - Inhaltliche Änderungen sind daten- oder benutzergetrieben
  - nur strukturelle Änderungen erfordern einen Eingriff des Administrators
  - Beispiel: Blogs, Wikipedia, Amazon
  - Beispiel: das Gästebuch vom „Rasthaus zum Goldenen Löwen“
- (D)ynamisches HTML fasst Techniken zusammen, mit denen eine im Browser geladene Seite **clientseitig manipuliert** werden kann. DHTML verhält sich orthogonal zu statischen und dynamischen Websites
  - Beispiel: Aufklappen von Sektionen eines Wikipedia-Eintrags
  - Beispiel: Endless Scrolling auf Twitter
  - Beispiel: das Flyout-Menü vom „Rasthaus zum Goldenen Löwen“
- Die Endausbaustufe von DHTML sind **Single-Page-Apps** (SPAs), die bis auf ein leeres Gerüst sämtliches HTML im Browser erzeugen
  - Beispiel: Google Maps

Dynamisches HTML – JavaScript (Historie)

- Wurde 1995 von Brian Eich bei Netscape erfunden als Scripting-Sprache für den Netscape Navigator
- Erst unter dem Namen LiveScript, dann aus Marketinggründen umbenannt
- Außer dem Namen und einer ähnlichen Syntax keinerlei Verwandtschaft zu Java
- 1997 standardisiert von der European Computer Manufacturers Association als EcmaScript
- Trotz mehrerer Versuche (VBScript, ActionScript, Silverlight) bis heute die defacto-Standardsprache im Browser und dadurch eine der verbreitetsten Sprachen überhaupt
- Heute auch außerhalb des Browsers weit verbreitet, (=> Node.JS)

Dynamisches HTML – JavaScript Grundkonzepte

- Die Einbindung in eine HTML-Seite erfolgt über das <script> Tag

```
<html>
  <head>
    <!-- Externes Script -->
    <script src="http://myserver.com/myscript.js"></script>
    <!-- Inline Script -->
    <script>
      function hello() {
        // Ausgabe in den Browser Developer Tools sichtbar
        console.log("Hallo Welt");
      }
    </script>
  </head>
  <!-- Funktion wird unmittelbar nach dem Laden der Seite gerufen -->
  <body onload="hello()">
  </body>
</html>
```

Dynamisches HTML – JavaScript Grundkonzepte

- Viele Java-Sprachkonstrukte sind leicht auf JavaScript übertragbar.

```
if (value != 3) {
  // ...
} else {
  throw new Error("oops");
}

while (index < 10 ) {
  //..
}

for (i = 0; i<10; i++) {
  // ...
}
```

Dynamisches HTML – JavaScript Grundkonzepte

- JavaScript ist nicht typisiert.

```
function summe(a, b) {
  console.log(a + b);
}

summe(1, 3);
summe('3', '4');

function printLength(c) {
  console.log(c.length);
}

var thing = 'foo';
printLength(thing);
thing = ['foo', 'bar'];
printLength(thing);
thing = 42;
printLength(thing);
```

Dynamisches HTML – JavaScript Grundkonzepte

- “Object literal” Notation vereinfacht das Erstellen von Objekten.

JavaScript

```
let person = {
  firstName: 'Tony',
  lastName: 'Stark',
  address: {
    street: 'Malibu Point 10880',
    postalCode: '90265'
  }
};

console.log(person.address.street);
```

Java

```
Person person = new Person();
person.setFirstName("Tony");
person.setLastName("Stark");
Address address = new Address();
address.setStreet("Malibu Point 10880");
address.setPostalCode("90265");
person.setAddress(address);

System.out.println(person.getAddress().getStreet());
```

Dynamisches HTML – JavaScript Grundkonzepte

- Funktionen sind “First Class Citizens” (können als Übergabeparameter und Rückgabewerte von Funktionen verwendet werden)

```
function scream(text) {
  return text.toUpperCase();
}

function print(str, modifier) {
  console.log(modifier(str));
}

print('hallo', scream);
```

Dynamisches HTML – JavaScript Grundkonzepte

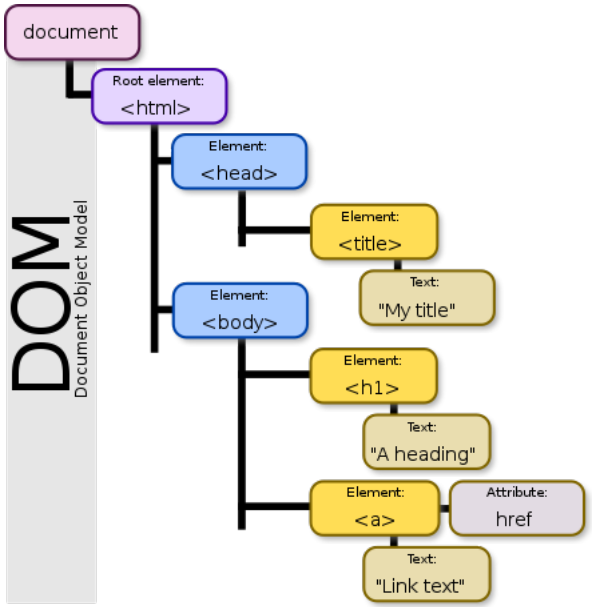
- JavaScript ist single-threaded und event-basiert. Callbacks sind allgegenwärtig

```
function doStuff() {
  console.log('OK');
}

button.onclick(function() {
  doStuff();
});

// oder:

button.onclick(() => doStuff());

button.onclick(doStuff);

button.onclick(doStuff());
```

Dynamisches HTML – Document Object Model

- ist eine standardisierte(!) Programmierschnittstelle für die Ver- und Bearbeitung von HTML- (und XML)-Dokumenten
- Darstellung von Dokumenten in einer Baumstruktur mit unterschiedlichen Knotentypen (Element, Attribut, Text)
- Die Spezifikation ist sprachunabhängig, aber im Browser kommt faktisch ausschließlich JavaScript zum Einsatz

```
<html>
  <head>
    <title>My title</title>
    <body>
      <h1>A heading</h1>
      <a href="...">Link text</a>
    </body>
  </head>
</html>
```



Dynamisches HTML – Document Object Model

- Über das “document” Objekt kann JavaScript auf das aktuelle Dokument zugreifen und darin navigieren, Knoten auffinden, deren Inhalte auslesen und verändern sowie Knoten hinzufügen oder löschen
- Die jQuery Library vereinfacht das API und liefert nützliche Zusatzfunktionen (wird von fast 80% der meist besuchten 10 Millionen Websites verwendet, auch wenn die Bedeutung allmählich sinkt)
- Referenzen:
  - https://wiki.selfhtml.org/wiki/JavaScript/DOM
  - https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
  - https://api.jquery.com/

Dynamisches HTML – Document Object Model

→ Übung html06

```
<html>
  <head>
    <script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
  </head>
  <body>
    <p id="special" class="item">Eins</p>
    <div class="container">
      <p class="item">Zwei</p>
      <p class="item">Drei</p>
      <p class="item">Vier</p>
    </div>
  </body>
</html>
```

Standard: `document.getElementById('special').innerHTML`
jQuery: `$('#special').text()`

Standard: `document.querySelectorAll('.container .item').item(0)`
jQuery: `$('.container .item').get(0)`

Standard:
```
var p = document.createElement('p');
p.innerHTML = 'Vier';
document.querySelector('.container').appendChild(p);
```
jQuery: `$('.container').append('<p>Vier</p>')`

Kommunikation zwischen Anwendungen

	Synchron „Telefon-Metapher“	**Asynchron** „Brief/Mail-Metapher“
	Der Sender wartet auf die Antwort des Empfängers und ist in dieser Zeit blockiert. → enge Kopplung	Der Sender wartet die Antwort des Empfängers *nicht* ab. → lose Kopplung
Antwortzeit		
Verfügbarkeit		

Fragen:

- Welches Kommunikationsmodell ist schneller?
- Welches Kommunikationsmodell ist für große Datenmengen besser geeignet?
- Kann man beide Modelle kombinieren?
- Was sind Auswirkungen bei der Verwendung in Anwendungen die von einem Benutzer bedient werden?

Kommunikation zwischen Anwendungen

	1:1 „Anfrage-Metapher“	**1:n** „Aushang-Metapher“
	Der Sender spricht dediziert einen Empfänger an.	Der Sender wendet sich an eine Gruppe von Empfängern
Verantwortlichkeit		
Kopplungsgrad		

Kommunikation zwischen Anwendungen

	1:1	1:n
Synchron		
Asynchron		

HTTP Grundlagen

- HTTP - Hypertext Transfer Protocol
- gehört der Anwendungsschicht bei Netzwerkmodellen (ISO/ OSI Schichtenmodell)
- Zustandsloses Protokoll
  - Session wird durch Session-ID auf Anwendungslevel verwaltet
- Request-Response Kommunikation (synchron)
- (Vorwiegend für die Transport von Webseiten (HTML) genutzt)
- Sichere Datenübertragung durch HTTPS
- Aufgrund der Verbreitung von HTTP gibt es eine leistungsfähige Netzinfrastruktur welche das Vorteile von HTTP nutzen:
  - Browser mit lokalem Cache
  - Proxy/ Reverse-Proxy Server
  - Zugriff auf Port 80 (http) bzw. 443 (https) fast überall geöffnet
  - → Was bedeutet das für den Entwickler von Verteilten- bzw. Webanwendungen?
- Mai 2015: neue HTTP 2.0
  - Schnellere Übertragung Bündelung von Anfragen und Kompression
  - Push-Übertragung (Server-initiiert)

Aufbau einer HTTP-Nachricht

- Gilt für Request und Response
- http-header:
  - Steuerinformationen z.B. angefragt Seite, Kodierung, Security-Informationen
- Body:
  - Nutzlast z.B. im Response die HTML-Seite

Request

GET /infotext.html **HTTP/1.1**
Host: www.example.net

Response

HTTP/1.1 200 OK
Server: Apache/1.3.29 (Unix) PHP/4.3.4
Content-Length: (Größe von infotext.html in Byte)
Content-Language: de (nach RFC 3282 sowie RFC 1766)
Connection: close
Content-Type: text/html

(Inhalt von infotext.html)

Quelle: http://de.wikipedia.org/wiki/Http

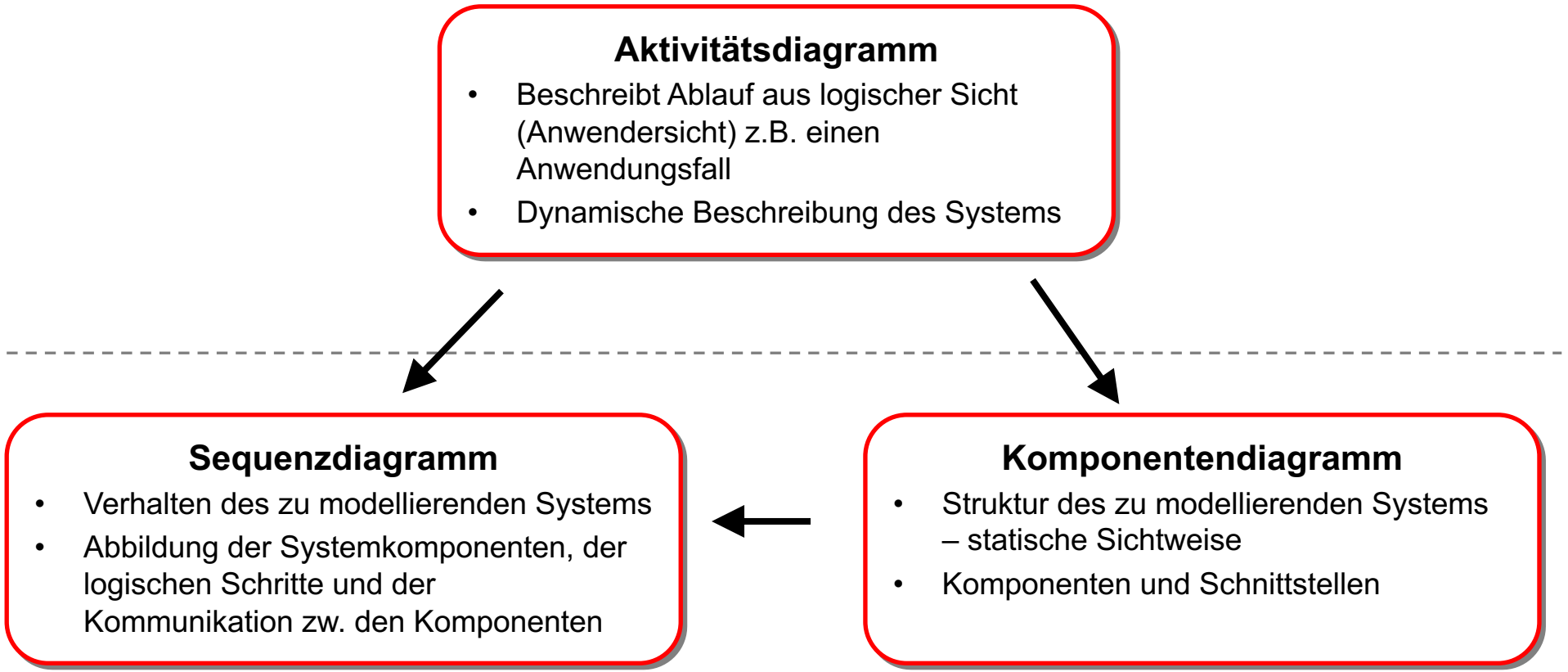
HTTP Methoden/ Antwort Codes

- Methoden im Request
  - GET
    - Liefert Daten zu einer spezifizierten Ressource
    - Response wird typischer Weise in Caches (Browser, Proxy) gespeichert
    - Aus Formularen übertragene Daten sind Teil der URL d.h. kann als Lesezeichen gespeichert werden
  - POST
    - Überträgt Daten zur Verarbeitung an einen Server z.B. als Name-Wert-Paar
    - Wird nicht in Caches gespeichert
    - Browser warnt vor einem „Reload“
  - PUT, DELETE, HEAD …
- Antwort-Code (siehe: http://www.w3schools.com/tags/ref_httpmessages.asp)
  - **2xx: Successful** → 200 OK
  - **3xx: Redirect (Umleitung)**
  - **4xx: Client Error** → 400 Bad Request | 403 Forbidden | 404 Not Found | 408 Request Timeout
  - **5xx: Server Error** → 500 Internal Server Error | 501 Not Implemented

Unified Modeling Language

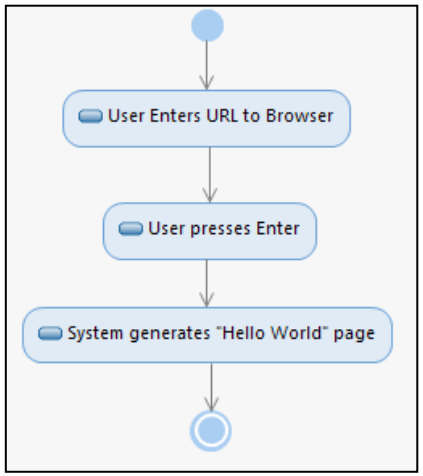
- UML allgemein
- Aktivitätsdiagramm
- Komponentendiagramm
- Sequenzdiagramm

Zusammenspiel der Diagrammtypen

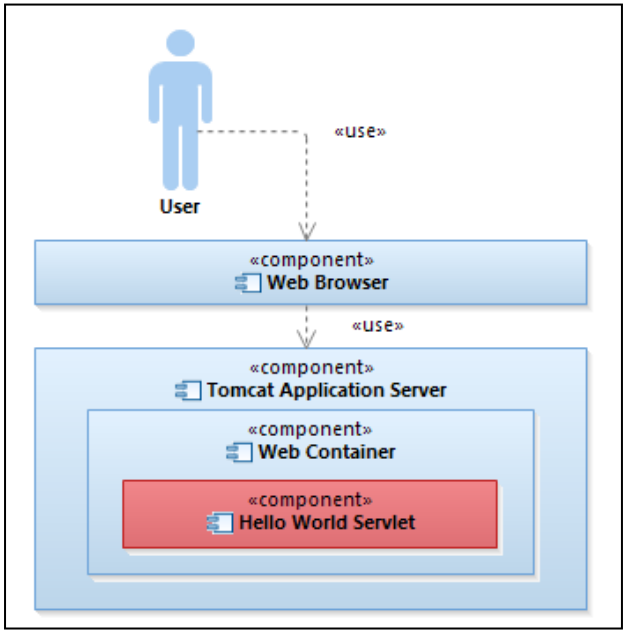


„Hello World“

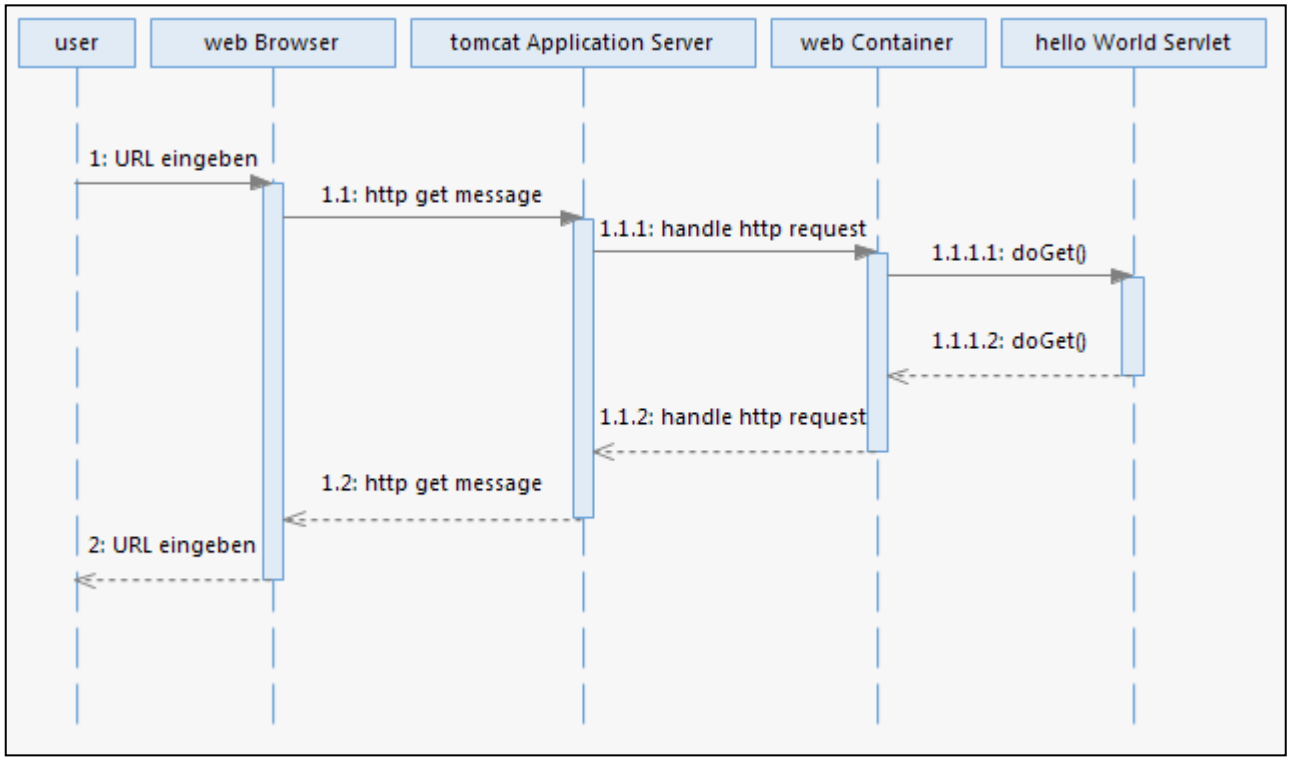
Activity Diagram



Component Model



Sequence Diagram



UML – ein Werkzeug zur strukturierten Analyse von Anforderungen und IT Architekturen.

Übung

Sequenzdiagramm zeichnen:

- Leonie schreib Lisa eine WhatsApp-Nachricht und fragt nach dem Vorlesungsplan
- Um die Frage zu beantworten ruft Lisa bei Lilly an.
- Lilly schaut auf ihrem Rechner – kann ihn aber nicht finden.
- Dann sucht Lisa auf ihrem Laptop den Moodle-Link herraus und schickt ihn Leonie via WhatsApp.
- Leonie meldet sich mit ihrem Rechner bei Moodle an und lädt den Vorlesungsplan herrunter.

Ajax

- Asynchronous JavaScript and XML
- Erlaubt den Austausch von Daten mit einem Server ohne Page Reload
- Beispiel: Suchfelder mit “Auto Suggest”
- von Microsoft entwickelt, erstmals in Internet Explorer 5 (1999)
- Durchbruch ca. 2004 (Gmail, Google Maps)

Ajax – XHR

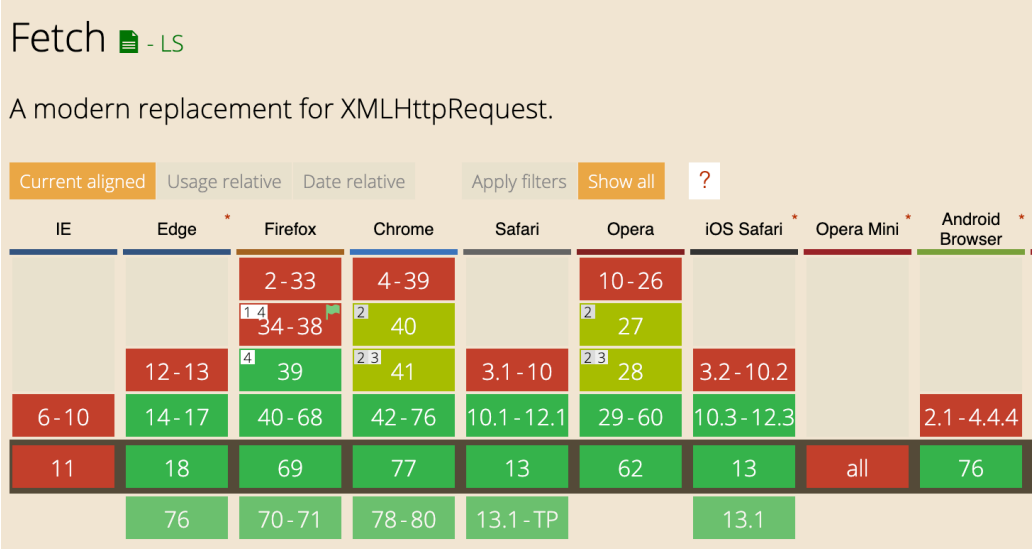
- Basiert auf dem XMLHttpRequest

```
1.  const request = new XMLHttpRequest();

2.  request.onload = () => {
5.    console.log(request.responseText);
    };

3.  request.open('GET', 'http://www.example.org/example.txt');
4.  request.send();
```

Ajax – Fetch API

- Moderne Alternative zu XHR, basierend auf **Promises**
- Erstmals in Chrome 40, inzwischen guter Browser-Support

```
fetch('http://www.example.org/example.txt')
  .then(response => response.text())
  .then(text => console.log(text))
  .catch(error => console.log('Oops!'));
```



https://caniuse.com/#feat=fetch

Ajax – JSON

- JavaScript Object Notation
- Ein Subset der Syntax von JavaScript Object Literals
- Wurde in den frühen 2000ern als leichtgewichtige Alternative zu XML entdeckt
- Heute das dominante Datenaustausch-Format für REST Interfaces (sprachunabhängig)

```xml
<author>
  <firstName>Stephen</firstName>
  <lastName>King</lastName>
  <books>
    <book isbn="978-0-385-18244-7">
      <title>Pet Sematary</title>
      <pages>374</pages>
    </book>
    <book isbn="978-0-670-81364-3">
      <title>Misery</title>
      <pages>310</pages>
    </book>
</author>
```

```json
{
  "author": {
    "firstName": "Stephen",
    "lastName": "King",
    "books": [
      {
        "isbn": "978-0-385-18244-7",
        "title": "Pet Sematary",
        "pages": 374
      },
      {
        "isbn": "978-0-670-81364-3",
        "title": "Misery",
        "pages": 310
      }
    ]
  }
}
```

Ajax – JSON

→ Übung ajax01

- In JavaScript eingebauter Support zum Parsen …

```
const json = '{ "firstName": "Stephen", "lastName": "King" }';
const author = JSON.parse(json);
console.log(author.firstName);
```

- … und Serialisieren …

```
const json = JSON.serialize(author);
// send to server ...
```

- Auch im Fetch API

```
fetch('http://www.books.org/books')
  .then(response => response.json())
  .then(json => console.log(json.author.books[0].title)
  .catch(error => console.log('Oops!'));
```