

# Web Engineering II

2016

Alexander Simianer

Duale Hochschule Baden-Württemberg  
Heidenheim

# Organisatorisches

- Vorlesungstermine:  
Donnerstag, 19.05.2016, 9:00-13:15 Uhr  
Dienstag, 07.06.2016, 9:00-12:15
- Klausur: 30.06.2016, 9:00-11:00, W102
- Lernplattform / Vorlesungsfolien  
<https://lms.dhbw-heidenheim.de/moodle/login/index.php>
- Literatur  
Zusatzinformationen können im Internet recherchiert werden. Weiter Quellen im Kurs.
- Kontakt  
Alexander Simianer  
[alexander.simianer@gmx.de](mailto:alexander.simianer@gmx.de)  
T/SMS/WhatsApp: 0157 / 79 21 22 98

# Veranstaltungsmodule

- **Vorlesung**  
Theorie / Hintergründe
- **Übungen**  
Praktische Programmierung (am System)
- **Ergebnisvorstellung und Diskussion**  
Erfahrungen / Erkenntnisse / Fragen

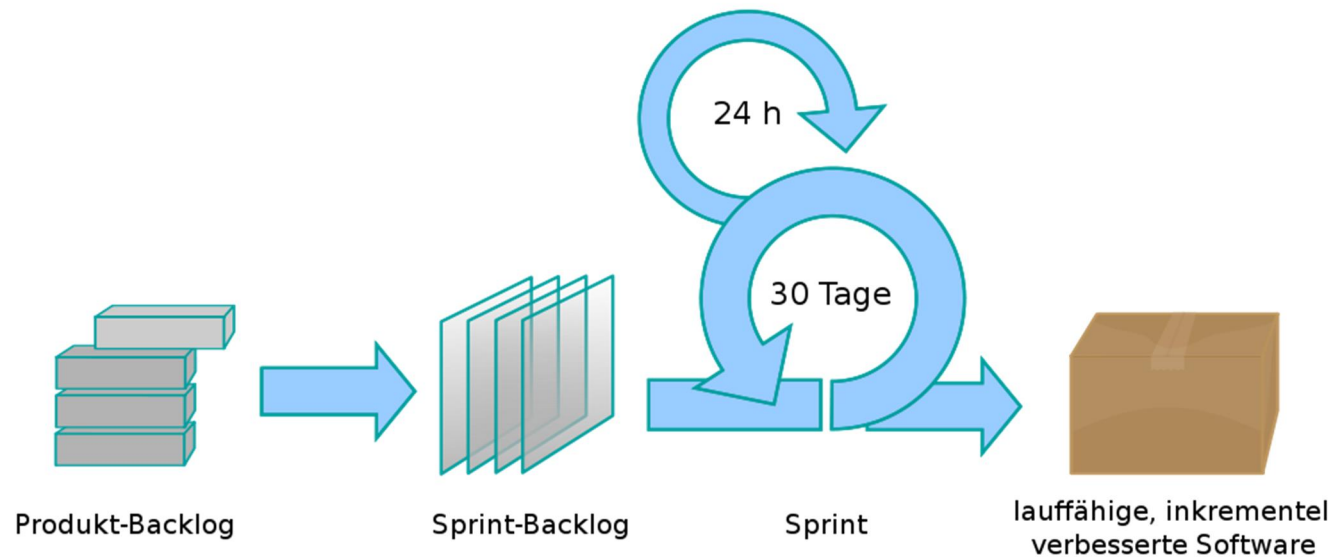
# Inhalte der Veranstaltung

- **Grundlagen**
- **Anwendungskonstellationen / Technologien / Lösungen**
- **Prakt. Anwendung und Programmierung** mit Konzepten, Werkzeugen und Verfahren des WE wie etwa:  
HTML5, CSS, JavaScript, SVG, jQuery, Angular.js, MEAN, Scaffolding, Frameworks, etc.
- **Teamprojekt**  
Produktentwicklung im Team

# Organisation: Scrum-Team (agile)

- 25 Teilnehmende => 5 Scrumteams je 5 Members
- Rollen:
  - 5 Teams
  - 5 TeamMembers (Teams bilden, Teamname!)
  - 1 ScrumMaster (ggf. gleichz. Member => bestimmen)
  - 1 ProductOwner (Dozent oder andere Person)
  - N Stakeholders

# Scrum



Quelle:  
Von Scrum\_process.svg:  
Lakeworksderivative work:  
Sebastian Wallroth (talk) -  
Scrum\_process.svg, CC BY-SA  
3.0,  
<https://commons.wikimedia.org/w/index.php?curid=10772971>

## Umsetzung:

- Sprintdauer: 1 Woche
- Sprintreview & Planungsmeeting  
(am Anfang oder am Ende des Veranstaltungstermins)

# Scrum / Ressourcenplanung

- **6 Termine** mit je 3 bis 4 Stunden Bearbeitungszeit  
die Tage zwischen den Terminen (=> Faktor 2)  
stehen zur Verfügung  
=> 6 mal 4h = 24h.  
=> 24h mal 5 Members x 2 = **240h**  
**Projektwert** = 50€ x 240h = **12.000 €**  
(Mindestens zu erwirtschaftende Kosten)
- **5 Projekte => 60.000 €**
- **Inkrementell** => Zum Sprintende steht jeweils eine lauffähige Programm-Version zur Verfügung.

# Projekt 1: Web-App „Planning Poker“

Web-App, zur Unterstützung des **PlanningPoker in ScrumTeams**.

Funktionen:

- Pokerteam festlegen / Einloggen mit Team ID (o.ä.)
- An und Abmelden
- Backlogitems erstellen
- Backlogitem anzeigen
- Backlogitem zur Planung freigeben
- Pokerzahlen anbieten
- Poker starten
- Warten bis alle abgestimmt haben.
- Anzeigen, wenn alle abgestimmt haben.
- Moderator kann aufdecken / Im Team „ 1, 2, 3 => Zeigen“.
- Karte kann angezeigt werden.
- Wenn alle abgestimmt haben => Signal und anzeigen.
- Nächstes Item anbieten.
- Poker ohne konkretes Item zulassen.

Mögliche Tools /  
Werkzeuge / Konzepte:

- **HTML**
- **CSS**
- **JavaScript**
- Datenbank  
**MySQL** oder  
MongoDB
- **Bootstrap** oder  
Foundation
- **jQuery**
- **Less**
- **PHP**
- Apache oder node
- **Git**
- **XAMPP**



# Projekt 2: „WebSite from Scratch“

## WebSite from Scratch für **Fotoagentur „ClickStar“**

- Single-Pager
- Für alle Geräte (responsive!)
- Kontaktformular
- Bildergalerie
- Geschützter Kundenbereich  
(Fotos vom Fotostudio, Fotoupload  
für Retuschieraufträge, ...)
- Wer wir sind
- Was wir tun
- Teamvorstellung
- Leistungsübersicht
- Impressum/Datenschutz
- Adminzugang zum Hochladen von Bildern in die Galerie

### Mögl. Tools/Werkzeuge/Konzepte:

- **HTML**
- **CSS**
- **JavaScript**
- **Bootstrap**
- **jQuery**
- Notepad++
- **Git**
- XAMPP
- **PHP**
- ...

**Anmerkung:** Site kann auch mit Typo3 realisiert werden, wenn Vorkenntnisse vorhanden sind.

# Projekt 3: „BikeStore“

## Webstore für Fahrräder (BikeMe24)

- Animiertes Firmenlogo (drehendes Rad)
- Neues BikeMe-„Theme“
- Preise in Euro
- Kundenkonto / Warenkorb
- Bezahlmethoden: PayPal / Überweisung / Rechnung
- Kategorien:
  - Fahrräder
    - E-Bikes
    - Mountainbikes
    - Trekkingräder
    - Cityräder
  - Zubehör
- Fotos hinterlegt
- Attribute: Farbe, Sattel, Bremspaket mit angepassten Preisen
- Impressum/Datenschutz/AGB
- Anpassungen / Optimierungen

Mögl.

Tools/Werkzeuge/  
Konzepte:

- **Magento 2**
- **HTML/CSS**
- **JavaScript**
- **jQuery**
- Datenbank (SQL)
- Less
- ...

**Anmerkung:** Store könnte auch „**from scratch**“ oder mit Shopify oder mit Jimdo realisiert werden...

# GRUNDLAGEN

# Web-Programmierung

- Motivation
  - Inhalte sollen nicht nur statisch, sondern dynamisch sein!
  - Serverseitig sollen Inhalte „on demand“ erzeugt werden können (algorithmisch / nach Benutzerinteraktion / aus Datenbank, ...)
  - Clientseitig soll auf Benutzerinteraktionen reagiert werden können
  - Clientseitig soll der Seiteninhalt verändert werden können (Neu-/Nachladen, DOM-Manipulation, ...)

# Web-Programmierung

## Ausführungsort / Sprachen

- Client
  - JavaScript
  - Plugins
- Server
  - PHP
  - ASP.NET (früher ASP)
  - JSP (Java Server Pages)
  - JavaScript (z.B. per node.exe und node.js)
  - Ruby
  - ...
  - Und: Beliebige Programme auf WebServer-Rechner...  
(per CGI, Node, ... aufgerufen)

# Web-Programmierung

## Metasprachen / Bibliotheken / Hilfssprachen

- **TypeScript** (Microsoft) => Alternative für JavaScript. Aber: Compiler erzeugt dann auch wieder JavaScript
- **WebAssembly** (Microsoft, Google, ...) – Bytecode (also nicht Quelltext wie JavaScript) zur Ausführung in WebBrowsern => Schneller, einheitlich, geschützter Quellcode
- **Angular** / AngularJS – Webframework für MVVM-Webanwendungen (Google)
- **Less** – StyleSheet Sprache. Compiler erzeugt CSS. Compiler kann serverseitig (z.B. Node) oder auch clientseitig ausgeführt werden. Bootstrap nutzt Less. Alternative: Sass, Stylus
- **SVG** - Scalable Vector Graphics. Grafiksprache.
- **WebGL** (Nutzung der Hardwarebeschleunigung)
- **Typoscript** (Konfigurationssprache für Typo3)
- uvm.

# JavaScript

- Skriptsprache
  - Mitte der 90er Jahre erstmals durch die Fa. Netscape verfügbar gemacht.
  - Streit mit Microsoft (JScript)
  - Als **ECMAScript** standardisiert (1998)
  - W3-Konsortium definiert DOM
  - Aktuell:
    - JavaScript wird durch Frameworks erweitert, die dann ggf. zukünftig in den Standard einfließen werden (z.B. jQuery).
    - JavaScript soll auch serverseitig genutzt werden können  
Z.B.:
      - „node“ (node.js / node.exe)
      - Skriptsprache für das CLI der MongoDB.
- ⇒ **Vorteil:** Eine Sprache auf Client- und Serverseite. Vermeidet häufige Mind-Set-Änderung beim Entwickler.

# JavaScript

- Wird (meist) vom Webbrowser ausgeführt
- `<script>`-Tags, direkt oder als js-Datei, z.B.: `myscript.js`
- $\Rightarrow$  Interpreter-Code
- einsehbar, oft aber „minified“
- **nicht** Java! (Aber syntaktisch ähnlich)
- Objektorientiert  
Achtung: Klassen werden ebenfalls per function deklariert! (Prototypenorientierte Vererbung)
- Läuft in „Sandbox“ = (fast) keine Zugriffe auf Daten/Ressourcen des Clientsystems.
- Eher langsam



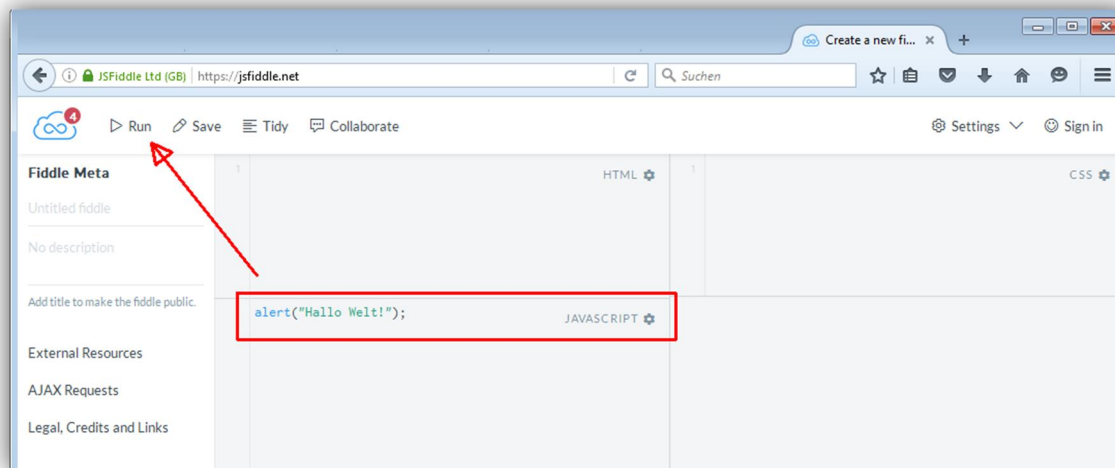
# JavaScript

- HTTP möglich
- Cookies setzen möglich
- Kann Browsereigenschaften auslesen
- Wird auch von nicht-Programmierern verwendet
- Ist mächtig
- Kann Aufgaben oft auf zu vielen Wegen lösen (historisch gewachsene Verfahren)
- Hat geschützte Zeichenketten

## Entwicklungsumgebung

(1) Online:

- **jsfiddle.net**



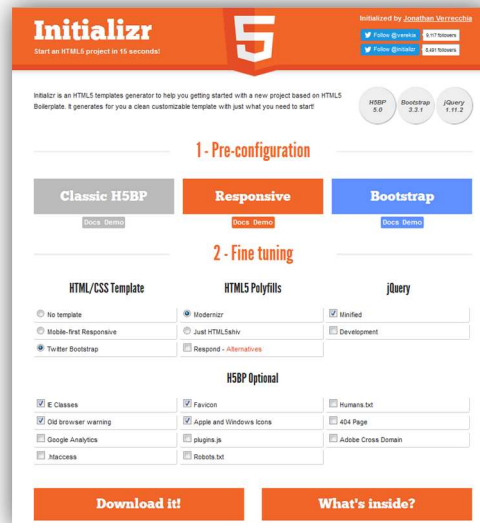
- Oder auch: **CodePen** (<http://codepen.io>),  
**Plunker** (<http://plnkr.co>), **JSBin** (<https://jsbin.com>),  
**Bootply** (<http://www.bootply.com>),  
**jsapp** (für nodejs => <http://jsapp.us>), ...

## JavaScript

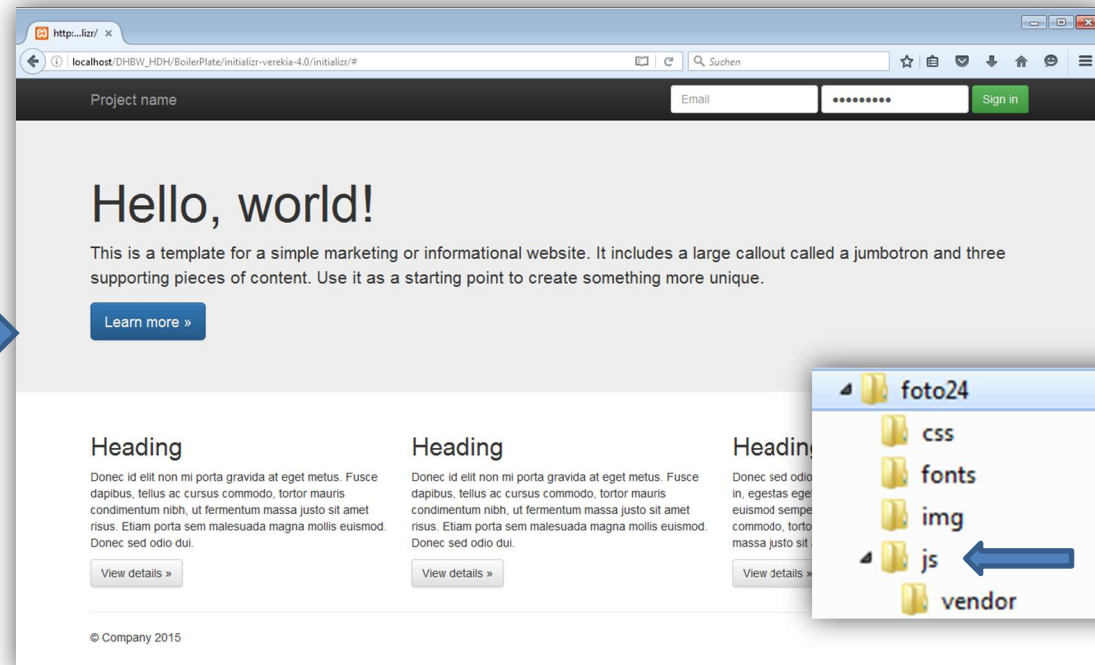
# Entwicklungsumgebung

(2) Lokal:

- XAMPP / HTML5 Skeleton / Notepad++
- Z.B.: per <http://www.initializr.com/>



Quelle:  
<http://www.initializr.com/>



## Entwicklungsumgebung

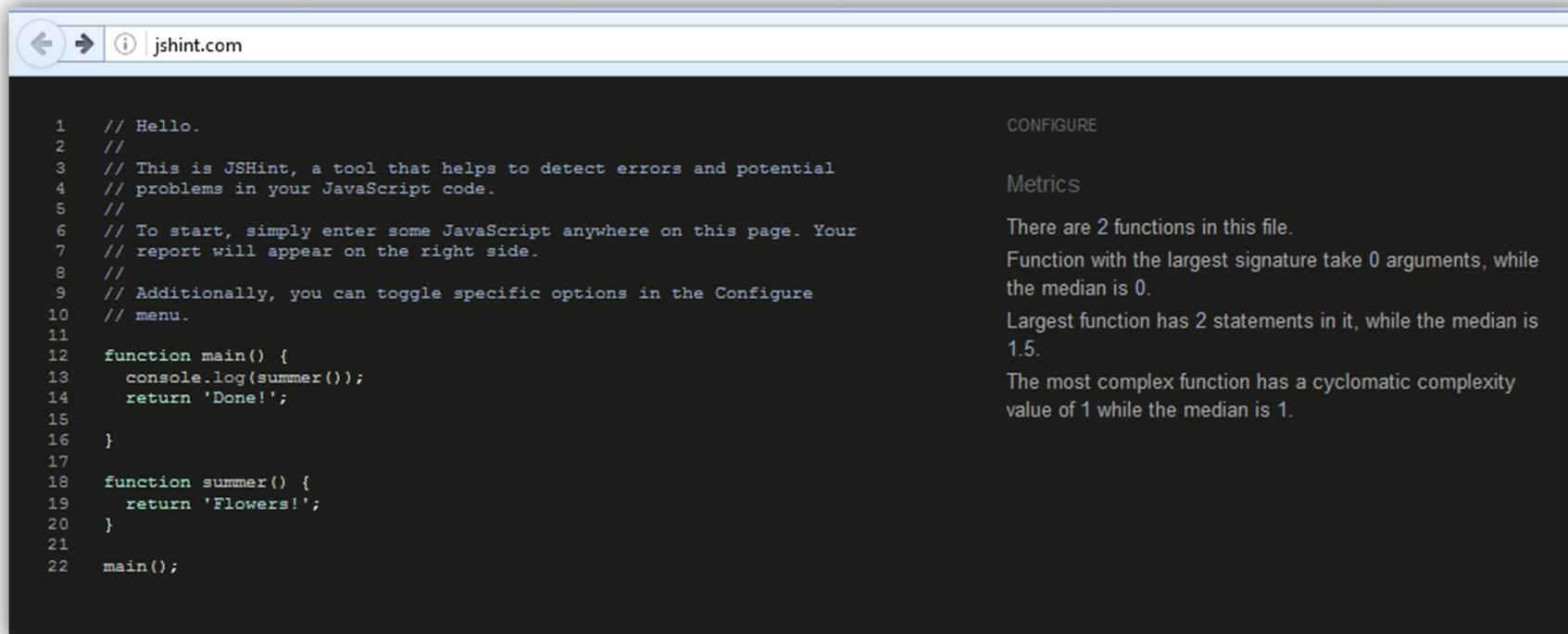
(3) Lokal, mit git-Repository auf [www.GitLab.com](https://www.gitlab.com):

- Git lokal installieren
- Account bei GitLab erstellen
- Projekt (z.B. "pr1 ") bei GitLab anlegen
- Auf Windows-PC lokales Verzeichnis (z.B. "pr1") anlegen
- Darin ausführen:
  - git init
  - git config user.name "Peter Pan"
  - git config user.email "pan@pan.de"
  - git remote add httpsorigin  
https://gitlab.com/<username>/<projektname>.git  
z.B.: https://gitlab.com/asilab/pr1.git
  - git gui
- Oder auch: **github** (<http://github.com>), lokales Git-Repository, git-Repos auf Uberspace (4Wo kostenlos), ...

# Entwicklungsumgebung

Statische Codeanalyse. Z.B. mit:

- JS Hint. Code Analyzer  
Online hier: <http://jshint.com/>



The screenshot shows the JSHint website interface. On the left is a code editor with the following JavaScript code:

```
1 // Hello.
2 //
3 // This is JSHint, a tool that helps to detect errors and potential
4 // problems in your JavaScript code.
5 //
6 // To start, simply enter some JavaScript anywhere on this page. Your
7 // report will appear on the right side.
8 //
9 // Additionally, you can toggle specific options in the Configure
10 // menu.
11
12 function main() {
13     console.log(summer());
14     return 'Done!';
15 }
16
17
18 function summer() {
19     return 'Flowers!';
20 }
21
22 main();
```

On the right side, there is a 'CONFIGURE' section and a 'Metrics' section. The 'Metrics' section contains the following text:

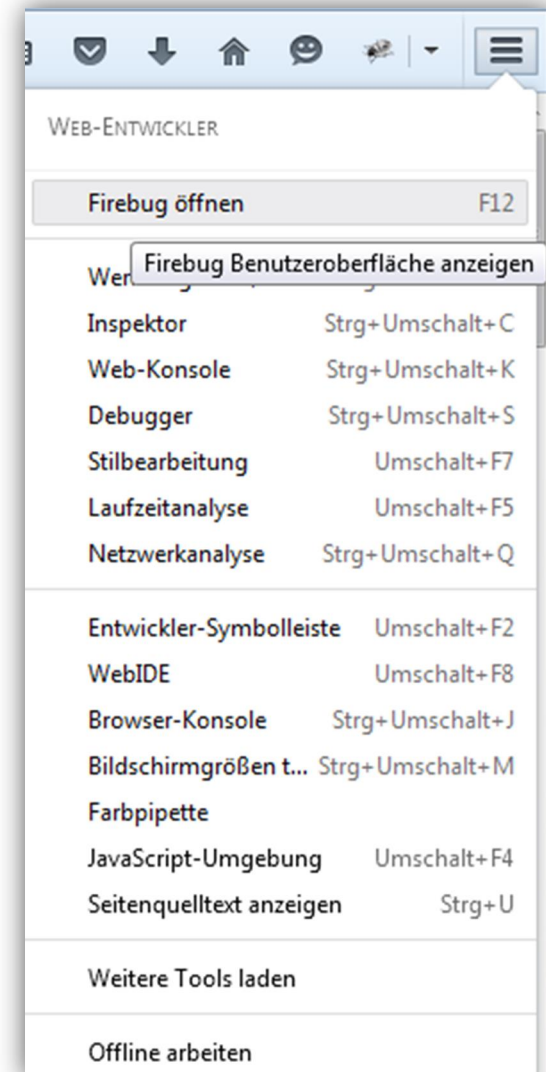
There are 2 functions in this file.  
Function with the largest signature take 0 arguments, while the median is 0.  
Largest function has 2 statements in it, while the median is 1.5.  
The most complex function has a cyclomatic complexity value of 1 while the median is 1.

## Entwicklungsumgebung

### Dynamische Codeanalyse (1)

=> JavaScript debuggen

- Entwicklersicht im Browser öffnen... (Chrome ist gut!)  
=> Debugging Optionen der Online-Umgebung verwenden...
- Man kann auch FireBug nutzen... (<http://getfirebug.com/>)
- Generell: Debugging Option der eingesetzten Sandbox nutzen



# Entwicklungsumgebung

## Dynamische Codeanalyse (2)

=> JavaScript „tracen“

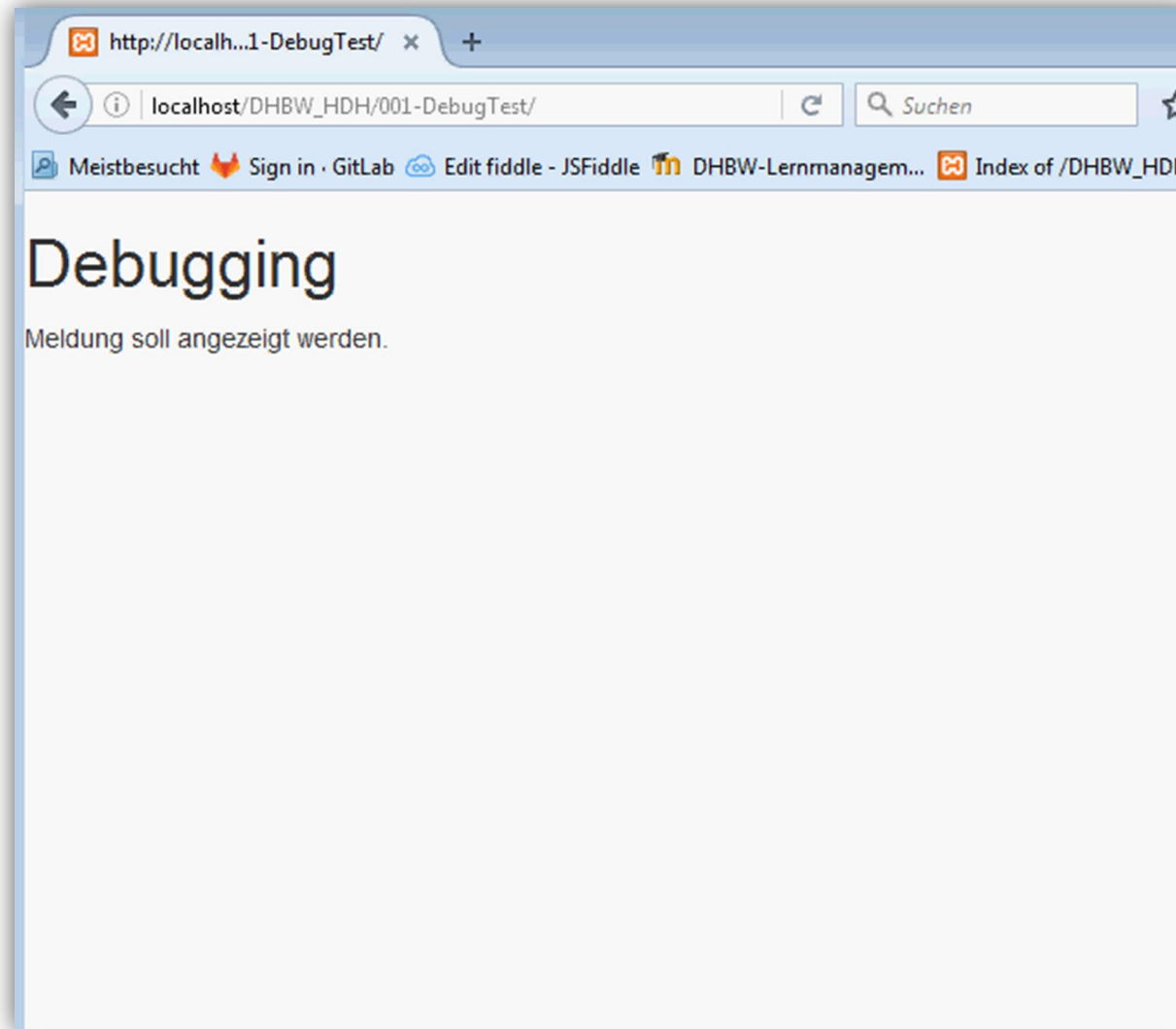
- Web-Console verwenden!
- per `console.log(...)` in die Console schreiben.

=> Man kann so auch Objekte inspizieren...

z.B. `console.log("This is %o", this);`

**Tipp:** Generell in die Console schauen, wenn JavaScript – Fehler vermutet werden

# Entwicklungsumgebung / Debuggen





# Quellcode. Wohin?

Typische Varianten:

- **Im HTML-Element**

```

```

- **Eingebettet in eine HTML-Seite**

```
<script>  
    alert("Hallo Welt");  
</script>
```

- **Skript-Datei, z.B.: `myscript.js`**

...wird in HTML-Datei angesprochen

```
<script src="js/ myscript.js"></script>
```

## JavaScript

# Wo genau soll eingebunden werden?

Head? Body? Mittendrin?

## Regel

- **CSS** im **<head>** einbinden
- **JavaScript** nach CSS und optimalerweise **direkt vor** dem schließenden **</body>** einbinden

## Beispiel

```
<html>  
  <head>  
    <link href="style.css">  
  </head>  
  <body>  
    <h1>Hallo Welt!</h1>  
    ...  
    <!-- JavaScript -->  
    <script src="jquery.min.js"></script>  
    <script>alert("Hello JavaScript!"); </script>  
  </body>  
</html>
```

# Wann wird JavaScript ausgeführt?

## Standard

- Browser lädt alle Seitenelemente nacheinander
- Trifft der Browser auf `<script>`, so liest er zunächst den JavaScript-Code bis `</script>` ein, und führt die Anweisungen anschließend sofort aus
- Während der Ausführung des JavaScript-Codes ist das Einlesen (Parsen) der Seite blockiert!

**=> Blockierungsproblem!**

## Wann wird JavaScript ausgeführt?

### Lösung des Blockierungsproblem

- `<script>`-Anweisungen „nach unten“ setzen
  - Dann wird JavaScript erst am Schluss geladen, wenn der DOM-Baum vollständig geladen ist.
  - Die Seite erscheint dann früher
- `<script src="blubb.js" defer></script>`  
Mit **defer** wird...
  - ...„paralleles“ Parsen von HTML-Code und JavaScript-Code aktiviert. HTML-Parser ist **nicht** mehr **blockiert**.
  - ...sichergestellt, dass der geparsete JavaScript-Code erst **nach** vollständigem **Laden** der HTML-Seite ausgeführt wird.
- `<script src="blubb.js" async></script>`  
Mit **async** wird...
  - ...„paralleles“ Parsen von HTML-Code und JavaScript-Code aktiviert. HTML-Parser ist **nicht** mehr **blockiert**.
  - ... der JavaScript-Code **weiterhin unmittelbar** nach dem Laden des Codes ausgeführt. Hierbei wird das Parsen der HTML-Seite **unterbrochen** : (

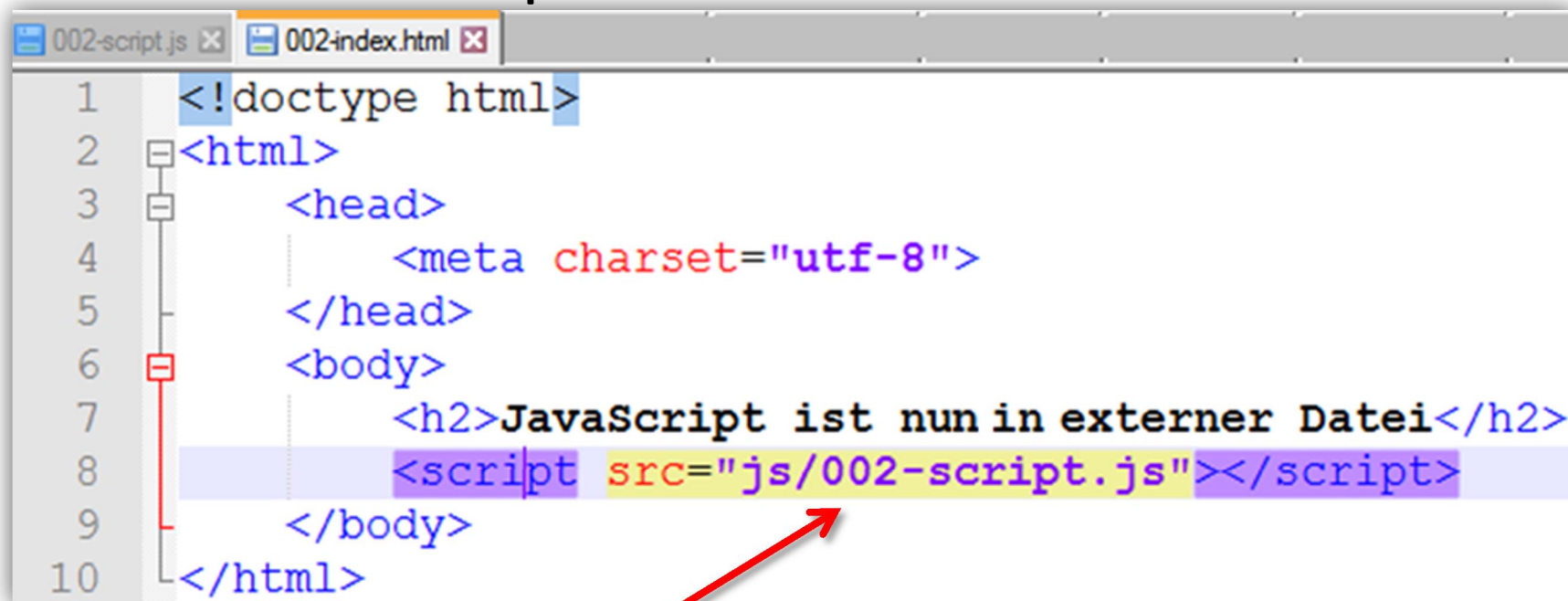
# Beispiel: Hallo Welt

"Hallo Welt"-Nachricht in Mitteilungsfenster:

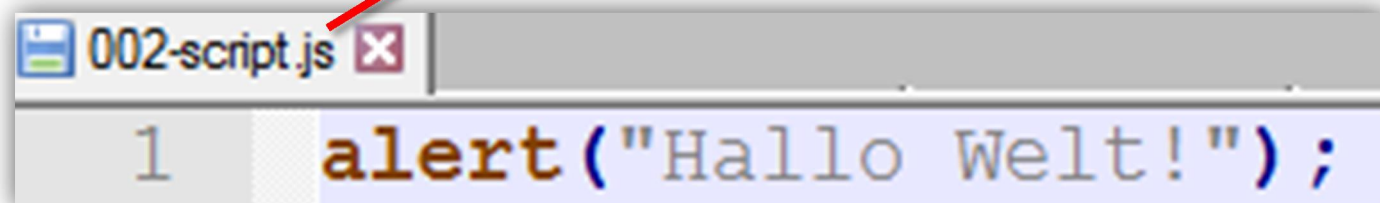
```
1  <!doctype html>
2  <html>
3      <head>
4          <meta charset="utf-8">
5      </head>
6      <body>
7          <script>
8              alert("Hallo Welt!");
9          </script>
10     </body>
11 </html>
```

# Beispiel: Hallo Welt (extern)

Externe JavaScript-Datei:



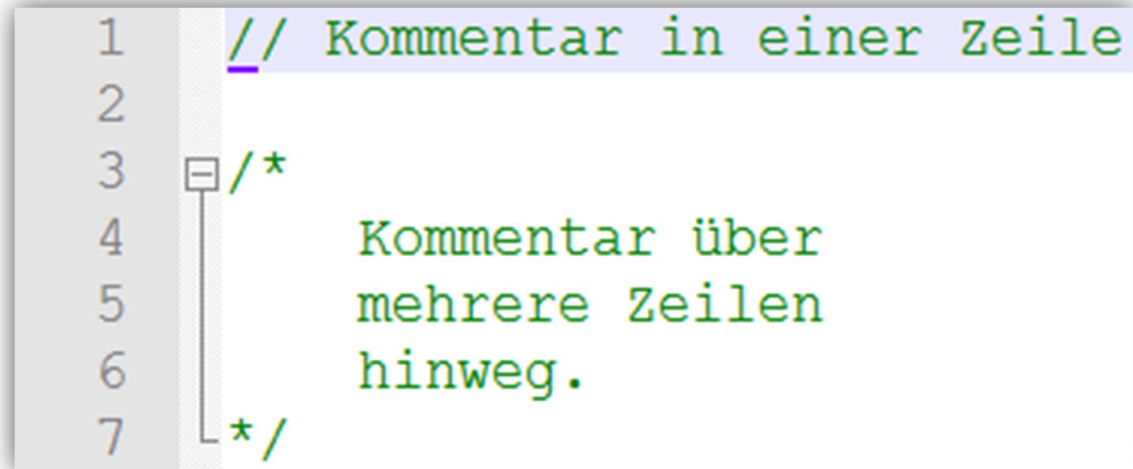
```
1 <!doctype html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5   </head>
6   <body>
7     <h2>JavaScript ist nun in externer Datei</h2>
8     <script src="js/002-script.js"></script>
9   </body>
10 </html>
```



```
1 alert("Hallo Welt!");
```

# Kommentieren

Varianten:



```
1 // Kommentar in einer Zeile
2
3 /*
4     Kommentar über
5     mehrere Zeilen
6     hinweg.
7 */
```

The image shows a code editor with line numbers 1 through 7 on the left. Line 1 contains a single-line comment `// Kommentar in einer Zeile`. Line 2 is empty. Line 3 starts a multi-line comment with `/*`. Line 4 contains the text `Kommentar über`. Line 5 contains `mehrere Zeilen`. Line 6 contains `hinweg.`. Line 7 ends the multi-line comment with `*/`. A vertical line connects the `/*` on line 3 to the `*/` on line 7, indicating the scope of the multi-line comment.

# Variablen

## Namen

- Alles zulässig, außer Schlüsselworte
- Beispiele: myName, id\_user, UserId, ...

## Gültigkeitsbereich

- Außerhalb von Funktionen: **global**
- Innerhalb von Funktionen:
  - Bei Deklaration ohne "var": **global**
  - Bei Deklaration mit "var": **lokal**

„var counter;“ erzeugt eine nicht definierte Variable. Sie hat deshalb einen undefinierten Typ und den Wert „undefined“, der auch abgefragt werden kann:  
[if (counter === undefined) ...]

## Beispiele:

```
a = 100; // global
var b = 200; // global
function f() {
    c = "Haha"; // global
    var d = "Huhu"; // lokal
}
```



# Datentypen

**Es gibt (unter anderem):**

- **Boolean**

```
var aBoolean = true;
```

- **Number**

```
var aNumber = 10;
```

- **String**

```
var aString = "Hello";
```

- **Array**

```
var aArray = ["Hanna", "Paul", "Leon"];
```

- **Object**

```
var aObject = {name: "Hanna", age: 33};
```

# Typumwandlungen

**Datentypen werden implizit durch Operationen in andere Datentypen umgewandelt.**

- **Abarbeitung von links nach rechts:**

```
var ergebnis = "Hi"+5;    // => Hi5  
var ergebnis = "Hi"+2+3;  // => Hi23
```

- **Explizite Typumwandlung (casten)**

```
var ten = "10";  
var aNumber = Number(ten)+1;  //=> 11
```

## **Tipp:**

Explizite Typumwandlung ist oft sinnvoll, damit unvorhergesehene Ergebnisse vermieden werden!

# Typumwandlungen

## Typumwandlungen durch Funktionen

- **Umwandlung in einen String:**

```
var s1 = (false).toString();  
var s2 = (10).toString();  
var s3 = aNumber.toString();
```

- **Umwandlung in Zahl**

```
var ten = parseInt("10");  
var flo = parseFloat(aFloatStr);
```

# Operatoren

## Operationen auf Werten:

- + (Addition)
- - (Subtraktion)
- \* (Multiplikation)
- / (Division)
- % (Modulo)  
Restklassenbildung,  $3 \% 2 \Rightarrow 1$
- ++ (Inkrementieren)
- -- (Dekrementieren)

# Operatoren

## Zuweisung

- =

**Zuweisung plus Operation auf sich selbst**  
 **$x <op>= y$  entspricht  $x = x <op> y$ .**

- +=
- -=
- \*=
- /=
- %=

# Operatoren

## Vergleichsoperatoren

- `==` gleich, mit impliziter Typumwandlung
- `===` Wert und Typ gleich
- `!=` nicht gleich, trotz impliziter Typumwandlung
- `!==` Wert oder Typ nicht gleich
- `>`
- `<`
- `>=`
- `<=`
- `?` z.B.: `var stadt = kennzeichen == „HDH“ ? „Heidenheim“ : „andere Stadt“;`

# Ablaufsteuerung

## Bedingungen

```
1  if (<Bedingung 1>)
2  {
3      <tue dies>
4  }
5  else if (<Bedingung 2>)
6  {
7      <tue das>
8  }
9  else
10 {
11     <dann eben das>
12 }
```

# Ablaufsteuerung

## For-Schleife

```
1  var summe = 0;  
2  for (i = 1; i <= 100; i++)  
3  {  
4      summe += i;  
5  }
```

```
35  var user = {name: "Hanna", age: 33};  
36  t = "";  
37  for (v in user) {  
38      t += user[v] + " ";  
39  }  
40  alert(t);
```

Hanna 33

OK



# Ablaufsteuerung

## While-Schleife

```
1  var summe = 0;  
2  var i = 1;  
3  while (i <= 100)  
4  {  
5      summe += i;  
6      i++;  
7  }
```

```
1  var summe = 0;  
2  var i = 1;  
3  do {  
4      summe += i;  
5      i++;  
6  }  
7  while (i <= 100);  
8  alert(summe);
```

```
1  // Achtung!  
2  while true;
```

# Ablaufsteuerung

## Switch-Case

```
1 switch(wert) {  
2     case 1:  
3         // ...  
4         break;  
5     case 2:  
6         // ...  
7         break;  
8     default:  
9         // ...  
10 }
```

# Objekte

## Behälter für Eigenschaften und Methoden

```
1 // Objects
2 ▼ var book = {
3     author: "Camilla Läckberg",
4     title: "Die Schneelöwin",
5 ▼   getTitle: function() {
6       return book.title;
7     }
8 };
9
10 alert(book.author + " is " +
11 ▼   book["author"] + " " +
12     book.getTitle());
```

Camilla Läckberg is Camilla Läckberg Die Schneelöwin