

# Web-Entwicklung

Tools im Entwicklungsprozess

# Inhalte der Vorlesung

- Tools im Entwicklungsprozess
  - Erzeugung der Projektstruktur
  - Debugging und Profiling
  - Überprüfung der Code-Qualität
  - Präprozessoren
  - Bundling, Minifikation und Obfuskation
  - Unit-Tests
  - Build-Tools

# Der Entwicklungsprozess

Build Tool	Einrichtung (Scaffolding)	Erzeugung der Ordnerstruktur
		Anlegen von Konfigurationsdateien
	Implementierung	
	Prüfung	Debugging, Profiling
		Überprüfung der Code-Qualität (Linting)
		(Unit-)Tests
	Erstellung	Präprozessoren
		Bundling, Minifikation, Obfuskation
		Dokumentationsgenerierung
	Deployment	(S)FTP, SCP, ...

**Erzeugung der Projektstruktur**

# Scaffolding

- engl. *scaffold*: Gerüst
- Automatische Generierung einer Projektstruktur als Ausgangspunkt für eine Neuentwicklung
  - Ordnerstruktur
  - Obligatorische Dateien (z.B. Datei mit Einstiegspunkt, Konfigurationsdateien, ...)
- Festlegung der Projektstruktur über Templates oder Generatoren, abhängig von
  - Laufzeitumgebung (Server/Client)
  - verwendeten Frameworks und Bibliotheken
  - Konventionen im Unternehmen
  - eigenen Präferenzen

# Gängige Konventionen

- Auf oberster Ebene separate Ordner für...
  - *webapp*, *client* oder *frontend* für Browser-Anwendung
  - *server* oder *backend* für Server-Anwendung
- Innerhalb dieser separate Ordner für...
  - Quellcode: *src*
  - Build: *dist*, *build* oder *release*
  - Unit-Tests: *test*
  - Dokumentation: *doc(s)*
- Innerhalb der clientseitigen (Browser-)Anwendung Unterordner je Ressourcentyp
  - JavaScript: *js* oder *scripts*
  - CSS: *styles* oder *css*
  - Bilder, Fonts, etc.: *assets*
  - HTML-Vorlagen: *templates*
  - ...

# Scaffolding Tools



Yeoman



Lineman



Slush

# Scaffolding Tools

"Für das Generieren von Codegerüsten oder initialen Projektstrukturen habe ich in der Vergangenheit meistens [Yeoman](#) verwendet. Das Tool kann viel mehr als ein einfacher Projektgenerator und hat das allgemeine Ziel, den Workflow beim Entwickeln von JavaScript-Anwendungen zu vereinheitlichen.

Yeoman nimmt [Grunt](#) als Build-Tool und [Bower](#) als Package Manager zur Grundlage und lässt sich über den Node Package Manager als Node.js-Modul installieren. Das Erzeugen der Projektgerüste wiederum übernehmen sogenannte Generatoren. Mit ihrer Hilfe lassen sich beispielsweise Projektstrukturen für [AngularJS](#)-, [Spring-Boot/AngularJS](#)-, [WordPress](#)-Projekte et cetera generieren.

Allerdings empfinde ich die Anzahl und den Umfang der erzeugten Artefakte je nach verwendetem Generator teils als zu groß. Darum sind wir in unserem Team nun verstärkt dazu übergegangen, für unterschiedliche Anwendungsfälle (mobile App, Express nutzende Webanwendung etc.) unsere Projektvorlagen in Form von Git-Repositories vorzuhalten und sie bei Bedarf zu klonen. So sind wir eher der Herr über den Quelltext."

Quelle: Philip Ackermann  
<http://www.heise.de/developer/artikel/Aus-der-Werkzeugkiste-Teil-1-Philip-Ackermann-3215454.html>

Möchte man eine neue Anwendung oder ein neues Modul erstellen, ist zunächst die Projektstruktur aufzusetzen. Früher habe ich dafür in der Regel [grunt-init](#) verwendet, inzwischen erledige ich das eher per Hand.



Golo Roden

Das JavaScript- und Node.js-Ökosystem entwickelt sich demmaßen schnell weiter, dass man regelmäßig gezwungen ist, seine Vorlagen zu aktualisieren. Da mir das zu aufwendig und zeitintensiv war, veralteten sie schnell. Daher musste ich die erzeugte Struktur häufig im Nachgang anpassen, was den Sinn einer Vorlage ad absurdum führt.

Das Geheimnis beim Verzicht auf Tools zur Codegenerierung ist, Projekte nicht zu groß werden zu lassen. Für kleine Projekte, die nur aus wenigen Dateien bestehen, lässt sich die Struktur rasch von Hand erstellen. Nebenbei bemerkt ist das mein größter Kritikpunkt an den zahllosen Vorlagen, die man online findet: Sie enthalten zu viel Code, bei dem man nicht genau weiß, was er bewirkt.

Große Anwendungen und Module stellen für mich ein Antipattern dar. Den Bedarf an einem Tool zur Codegenerierung sehe ich daher nicht als Hinweis darauf, dass noch ein Element in der Werkzeugkiste fehlt, sondern als Warnung, dass die Projektstruktur falsch ist und die Entwickler nicht genug über Modularisierung nachgedacht haben.

Quelle: Golo Roden  
<http://www.heise.de/developer/artikel/Aus-der-Werkzeugkiste-Teil-2-Golo-Roden-3242442.html>



# Empfehlung

- Manuelles Anlegen der Ordnerstruktur
  - kein unnötiger Ballast
  - häufig schneller (da kein Einarbeitungs- und Konfigurationsaufwand)
- Hat sich eine Ordnerstruktur etabliert, kann diese bei Bedarf
  - als Vorlage in einem Repository archiviert werden
  - durch ein Build-Skript erzeugt werden

# Grundgerüst einer Web-Anwendung

Ordner-/Dateistruktur einer sehr einfachen Web-Anwendung:

- lib
  - src
- server
  - src
- **webapp**
  - **src**
    - **assets**
    - **js**
      - **Main.mjs**
    - **styles**
      - **style.less**
    - **index.html**
  - **dist**

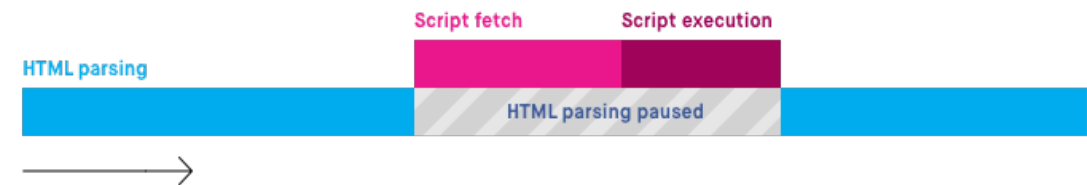
# Grundgerüst einer Browser-Anwendung

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Webapp Example</title>
  <link rel="stylesheet" href="style.css" />
  <script defer src="bundle.js"></script>
</head>
<body>
  <header></header>
  <main></main>
  <footer></footer>
</body>
</html>
```

# Einbindung der JavaScript-Datei

- script-Tag blockiert das lineare Parsen des HTML-Dokuments
- Sofortiger Download und Ausführung der JavaScript-Datei
- Probleme:
  - JavaScript referenziert evtl. HTML-Elemente, die noch nicht geparkt wurden
  - Längere Wartezeit

```
<html>
<head>
  ...
  <script src="bundle.js"></script>
</head>
<body>
  ...
</body>
</html>
```



## Einbindung der JavaScript-Datei

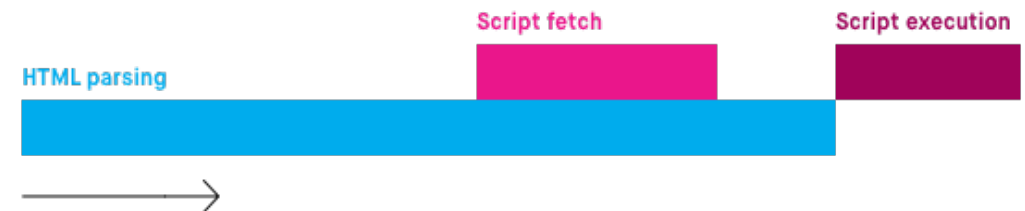
- Alter Lösungsansatz: Verschieben des script-Tags aus dem head ans Ende des body
- Problem: Längere Wartezeit (da serieller Download)

```
<html>
<head>
    ...
</head>
<body>
    ...
    <script src="bundle.js"></script>
</body>
</html>
```

# Einbindung der JavaScript-Datei

- Attribut defer startet parallelen Download, verzögert aber Ausführung bis HTML geparkt

```
<html>
<head>
  ...
  <script defer src="bundle.js"></script>
</head>
<body>
  ...
</body>
</html>
```



# Einstiegspunkt einer Browser-Anwendung

- `Main.mjs` und alle direkt und indirekt abhängigen Module werden später im Zuge des Build-Prozesses zu einer einzigen Datei `bundle.js` gebündelt (z.B. mithilfe von `esbuild`)
- Da beim Bundling alle Module in jeweils eine Funktion gekapselt werden, werden globale Variablen vermieden

```
import { SingleWord, ThreeWords } from 'lib';

SingleWord.print();
ThreeWords.print();

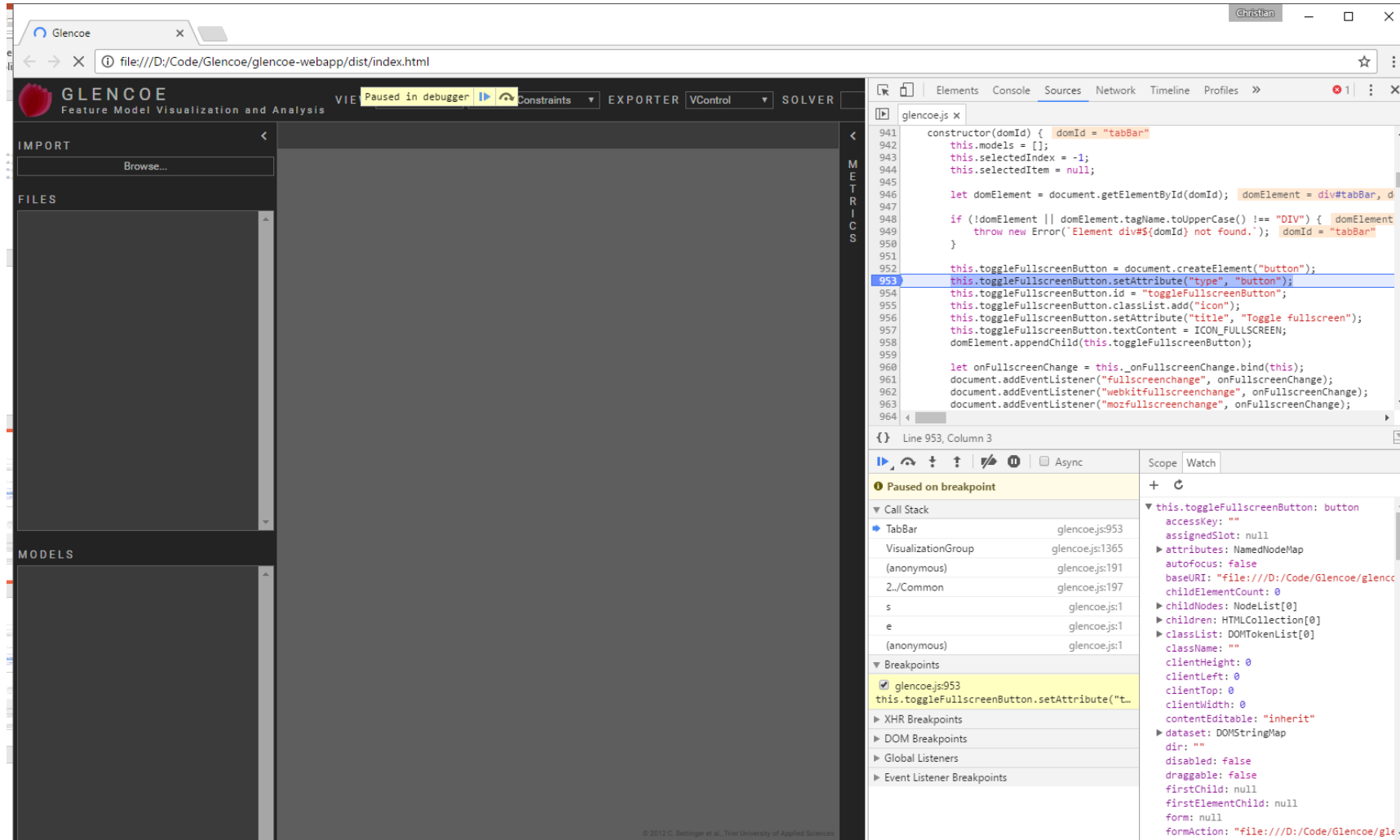
greet();

function greet() {
    console.log('Hello, World!');
}
```

## Debugging und Profiling



# Debugging



# Debugging

Glencoe

file:///D:/Code/Glencoe/glencoe-webapp/dist/index.html

**Glencoe**  
Feature Model Visualization and Analysis

VIEW Circle All Constraints EXPORTER VControl SOLVER

IMPORT

Browse...

FILES

- 20-1.xml
- 20-2.xml
- 20-3.xml
- 20-4.xml
- 30-1.xml
- 30-2.xml
- 30-3.xml
- 30-4.xml
- Test.xml**

MODELS

- 1 Feature
- 2 Features
- 3 Features
- 4 Features (1 dead)
- 6 Features
- 81E1E4CC-CDD8-ACDA-7EE8-E96232BE46F3
- Optional / Mandatory
- Feature Types
- Extra Large Feature Name Lorem ipsum dolor s...
- Common Constraints**
- Inconsistent Model
- Smoothie
- Truck
- Anhänger

Common Constraints

© 2012 C. Bettenger et al., Trier University of Applied Sciences

Elements Console Sources Network Timeline Profiles

```
<!DOCTYPE html>
<html lang="en">
<head>...</head>
<body>
  <div id="topBar">...</div>
  <div id="leftSideBar" class="sideBar">...</div>
  <div id="rightSideBar" class="sideBar hidden">...</div>
  <main id="visualizationGroup" class="right">
    <div id="tabBar">...</div>
    <div id="visualization">
      <div id="Circle" class="view" style="left: 0px; top: 0px;">
        <svg width="50637" height="50787">
          <g id="circle" transform="translate(317.3505859375, 312.9011535644531)">
            <circle cx="0" cy="0" r="231.9155134996809" />
          </g>
          <g id="nodes" transform="translate(317.3505859375, 312.9011535644531)">...</g>
          <g id="constraints" transform="translate(317.3505859375, 312.9011535644531)">...</g>
        </svg>
      </div>
    </div>
  </main>
  <script src="glencoe.js"></script>
</body>
</html>
```

html body main#visualizationGroup.right div#visualization div#Circle.view svg g#circle circle

Styles Event Listeners DOM Breakpoints Properties

Filter

element.style {

```
circle[Attributes Style] {
  cx: 0;
  cy: 0;
  r: 231.916;
}
```

\* { user agent stylesheet

```
transform-origin: 0px 0px 0px;
```

Inherited from g#circle

```
#Circle #circle {
  fill: none;
  stroke: #444444;
  stroke-width: 5px;
}
```

Inherited from body

```
body {
  width: 100%;
}
```

margin -

border -

padding -

auto x auto

Filter

Show all

- color
- cursor
- cx
- cy
- display
- fill
- font-family

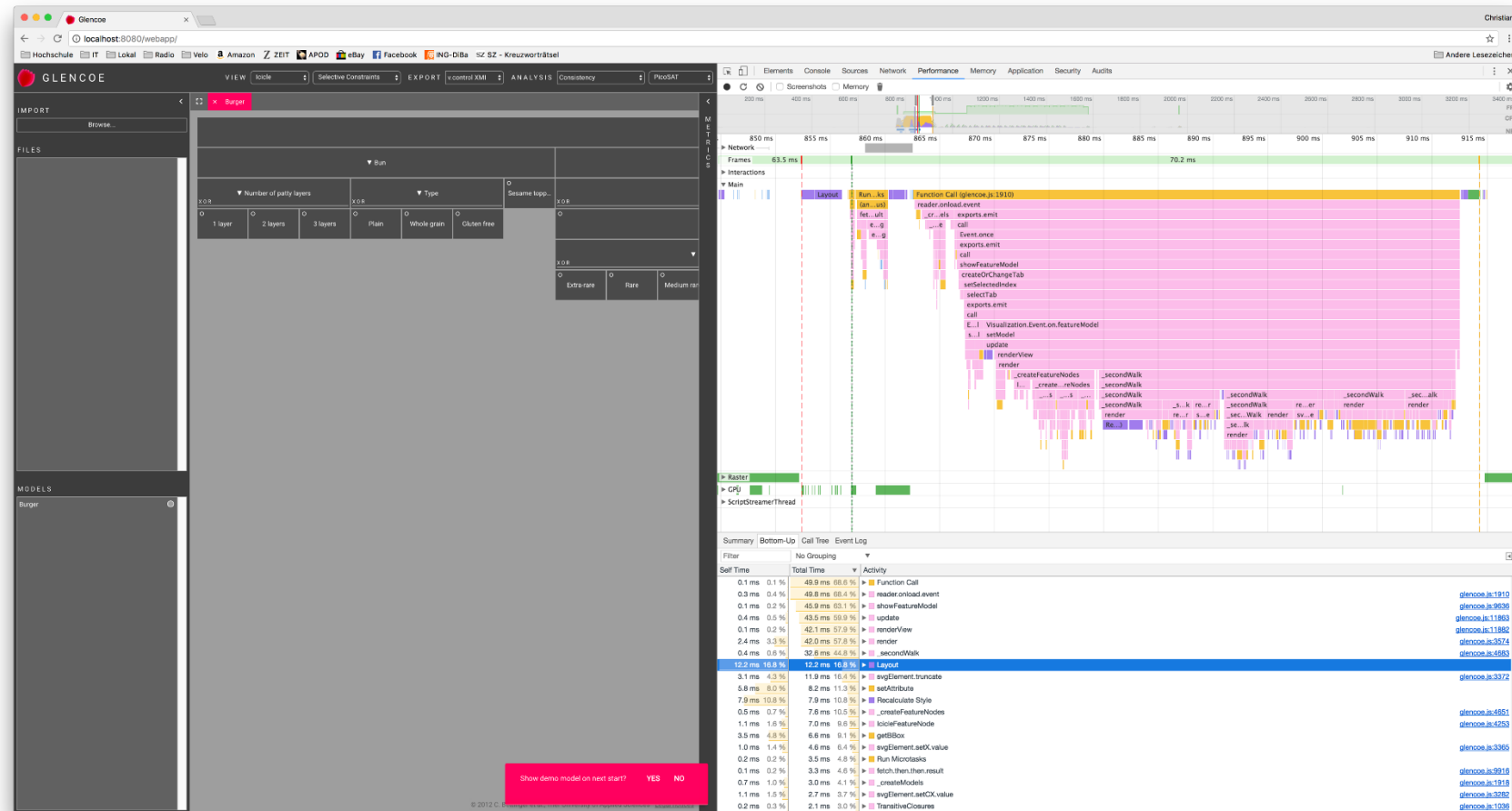
# Profiling

The screenshot displays the Glencoe web application interface. The browser address bar shows `localhost:8080/webapp/`. The application has a dark theme and a sidebar on the left with sections for 'IMPORT', 'FILES', and 'MODELS'. The 'MODELS' section shows a 'Burger' model. The main content area is titled 'Run' and contains a table for configuring the burger model. The table has columns for 'Number of patty layers', 'Type', and 'Sesame toppings'. The 'Number of patty layers' column has options for 1, 2, and 3 layers. The 'Type' column has options for Plain, Whole grain, and Gluten free. The 'Sesame toppings' column has options for Extra rare, Rare, and Medium rare. The 'Run' button is highlighted in red. Below the table, there is a red button that says 'Show demo model on next start? YES NO'. On the right side of the interface, there is a network waterfall chart showing the loading of various resources. The chart has a timeline at the top with markers for 10 ms, 20 ms, 30 ms, 40 ms, 50 ms, 60 ms, 70 ms, 80 ms, 90 ms, 100 ms, and 110 ms. The chart shows the following resources and their loading times:

Name	Status	Type	Initiator	Size	Time	Waterfall
webapp/	200	document	Other	1.1 KB	3 ms	
css?family=RobotoMaterial+icons	200	stylesheet	(index)	1.0 KB	32 ms	
zifox.css	200	stylesheet	(index)	22.8 KB	5 ms	
glencoe.css	200	stylesheet	(index)	14.5 KB	4 ms	
glencoe.js	200	script	(index)	347 KB	6 ms	
glencoe-small.css	200	stylesheet	(index)	646 B	2 ms	
list	200	fetch	glencoe.js:9911	860 B	3 ms	
glencoe-512.png	200	png	Other	48.3 KB	4 ms	
Demo.xml	200	fetch	glencoe.js:9859	7.0 KB	4 ms	
okMqT2MqDnOryGdIn2ZdsunDn1pK8aKai.peZSc5A.worf2	200	font	(index)	10.2 KB	7 ms	
2kuYYPa7p86g4L3H-Y5UfW0S80gJawQr3FEmqeworf2	200	font	(index)	46.6 KB	10 ms	

At the bottom of the network waterfall chart, there is a summary: '11 requests | 500 KB transferred | Finish: 83 ms | DOMContentLoaded: 91 ms | Load: 106 ms'. Below the chart is a console window showing the following message: 'top Filter Info 1 item hidden by filters'.

## Profiling



**Überprüfung der Code-Qualität**

# Style Guides

- Fixierung von Konventionen bezüglich des Code-Stils
  - auch *Code Conventions*
  - betrifft Code-Formatierung, Wahl von Bezeichnern, Groß-/Kleinschreibung, ...
- Erleichtern das Lesen und Pflegen von Code – insbesondere im Team
- Für die Entscheidung für bzw. gegen bestimmte gibt es manchmal harte Kriterien, z.B.
  - Platzeinsparung durch öffnende Klammern in gleicher Zeile oder Weglassen von Leerzeichen in Parameterlisten
  - verbesserte Lesbarkeit durch Leerzeichen in Parameterlisten
- Einsatz von Style Guides ist wichtiger als die Auswahl eines konkreten Style Guides
  - i.d.R. Vorgabe durch das Unternehmen

# Linting

- Style Guides umfassen i.d.R. keine Tools zur automatisierten Prüfung auf Einhaltung der Konventionen
- Linting-Tools prüfen auf
  - Einhaltung von Code-Konventionen
  - Existenz von fehlerhaftem oder potentiell fehleranfälligem Code (engl. *lint*: Fussel, Fluse)

# ESLint



- Prüfungen können aus einem umfangreichen Katalog zusammengestellt werden
- Unterstützt ES6+
- Erweiterung möglich durch
  - Plugins (z.B. eslint-plugin-sonarjs)
  - Definition eigener Regeln
- <http://eslint.org>
- Ähnliche Tools: JSLint, JSHint, Google Closure Linter, ...
- Plugins für gängige IDEs, z.B.
  - <https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint>



# ESLint

The screenshot shows the Visual Studio Code editor with a file named `controlstructures.js` open. The Explorer sidebar on the left shows the file structure, including `controlstructures.js` and several other JavaScript files. The main editor area displays the code for `controlstructures.js`, which includes a switch statement, a while loop, a do-while loop, and two for loops. The bottom of the editor shows the ESLint error list, which contains 17 errors. The first four errors are related to irregular whitespace, and the next four are related to the variable `i` being already defined.

```
39     case 12:
40     case 1:
41     case 2:
42         console.log('Winter');
43         break;
44     default:
45         console.error(' WTF?! ');
46     }
47
48     // while
49     var i = 1;
50
51     while (i <= 50)
52     {
53         console.log(i);
54         i++;
55     }
56
57     // do-while
58     var i = 1;
59
60     do
61     {
62         console.log(i);
63         i++;
64     } while (i <= 50);
65
66     // for
67     for (var i = 1; i <= 50; i++)
68     {
69         console.log(i);
70     }
71
72     for (var i = 1; i <= 10; i++)
73     {
74         var line = '';
75         for (var j = 1; j <= 10; j++)
76         {
77             line += ((j*i) + '\t');
```

17 ERRORS

- [eslint] Irregular whitespace not allowed (no-irregular-whitespace) (36, 6)
- [eslint] Irregular whitespace not allowed (no-irregular-whitespace) (39, 6)
- [eslint] Irregular whitespace not allowed (no-irregular-whitespace) (40, 6)
- [eslint] Irregular whitespace not allowed (no-irregular-whitespace) (41, 6)
- [eslint] 'i' is already defined (no-redeclare) (58, 5)
- [eslint] 'i' is already defined (no-redeclare) (67, 10)
- [eslint] 'i' is already defined (no-redeclare) (72, 10)
- [eslint] 'i' is already defined (no-redeclare) (83, 10)

Zeile 58, Spalte 5 Tabulargröße: 4 UTF-8 CRLF JavaScript

# ESLint

- npm-Modul
  - Kommandozeile
  - Integration in beliebige Build-Tools
- Installation:  
`npm install -D eslint (eslint-plugin-sonarjs)`
- Konfigurationsdatei `.eslintrc.json` im Projektwurzverzeichnis erstellen:  
`eslint --init`  
oder manuell (<http://eslint.org/docs/user-guide/configuring>):  
{  
 "parserOptions": { "ecmaVersion": 2022, "sourceType": "module" },  
 "env": { "browser": true, "commonjs": false, "es2022": true, "mocha": true, "node": true },  
 "plugins": ["sonarjs"],  
 "extends": ["eslint:recommended", "plugin:sonarjs/recommended"],  
 ...  
}

# ESLint

- Regelkatalog: <http://eslint.org/docs/rules/>
- Weitere Regeln aktivieren bzw. überschreiben

```
{  
  ...  
  "rules": {  
    "no-console": "off",           // off, warn, error  
    "indent": ["error", 4],        // Optionen via Array  
    "linebreak-style": ["error", "unix"],  
    "quotes": ["warn", "double"],  
    ...  
  }  
}
```

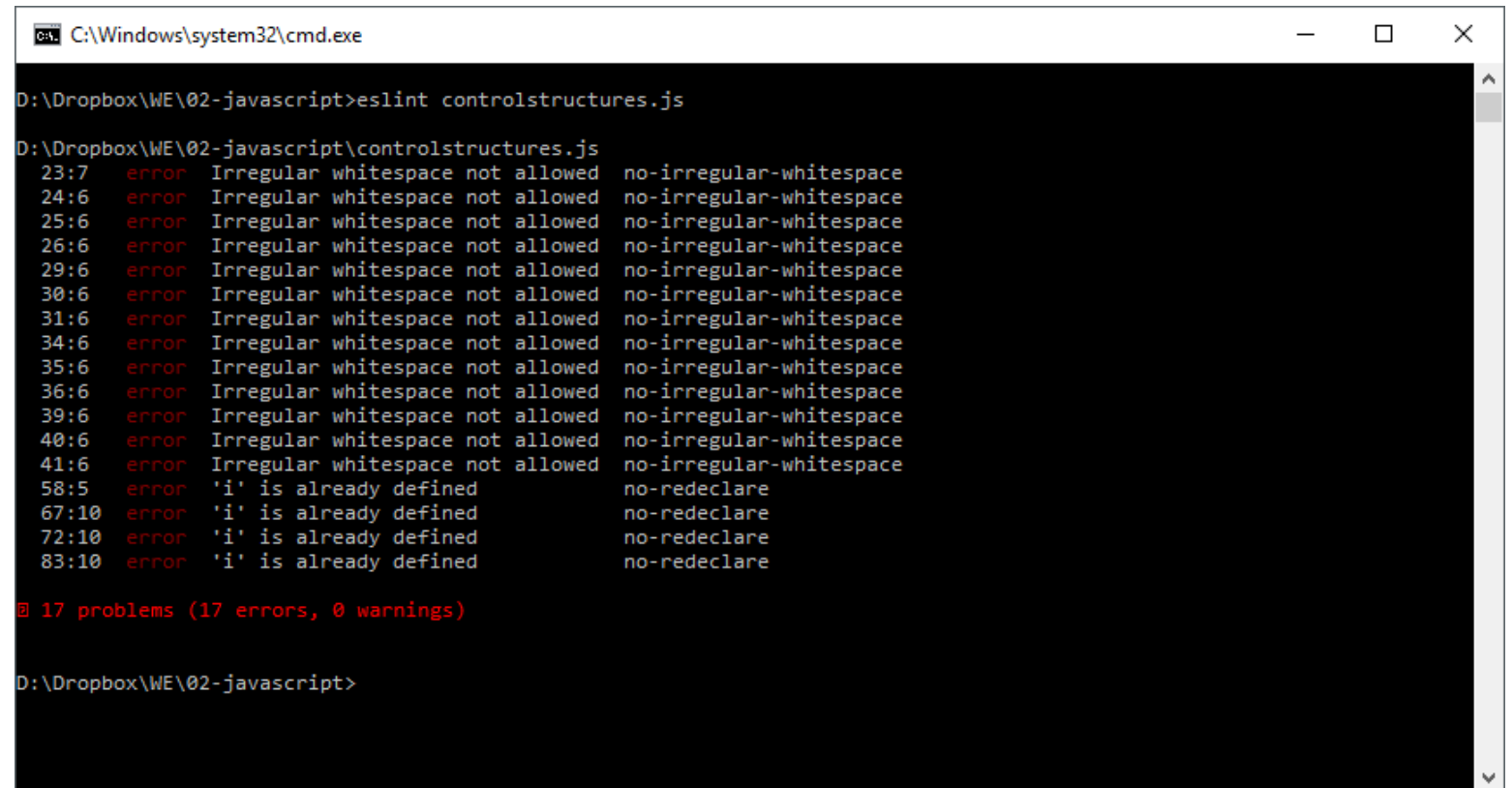
# ESLint

- Verwendung via Kommandozeile:

```
eslint Main.mjs
```

```
eslint *.mjs
```

```
eslint webapp/src/js/**
```



```
C:\Windows\system32\cmd.exe

D:\Dropbox\WE\02-javascript>eslint controlstructures.js

D:\Dropbox\WE\02-javascript\controlstructures.js
 23:7   error  Irregular whitespace not allowed  no-irregular-whitespace
 24:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 25:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 26:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 29:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 30:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 31:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 34:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 35:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 36:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 39:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 40:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 41:6   error  Irregular whitespace not allowed  no-irregular-whitespace
 58:5   error  'i' is already defined            no-redeclare
 67:10  error  'i' is already defined            no-redeclare
 72:10  error  'i' is already defined            no-redeclare
 83:10  error  'i' is already defined            no-redeclare






































@ 17 problems (17 errors, 0 warnings)

D:\Dropbox\WE\02-javascript>
```

# standard und semistandard

- <https://standardjs.com/>
- npm-Module auf Basis von ESLint
- Vorkonfigurierter, nicht veränderbarer Regelsatz
- Vermutlich am weitesten verbreitete Code Convention für JavaScript
- Unterschied
  - standard: Am Ende einer Zeile darf kein Semikolon stehen
  - semistandard: Am Ende einer Zeile muss ein Semikolon stehen
- Bei der Hausarbeit muss semistandard beachtet werden – sonst Abwertung!

Who uses JavaScript Standard Style?

## standard und semistandard



- Installation:  
`npm install -D semistandard (Alternativ: standard)`
- Verwendung via Kommandozeile:  
`semistandard`  
`semistandard Main.mjs`  
`semistandard server/src/**/*.mjs webapp/src/**/*.mjs test/**/*.mjs`
- Automatische Code-Formatierung (nicht automatisch anwendbare Regeln werden ausgegeben):  
`semistandard --fix`  
`semistandard --fix Main.mjs`  
`semistandard --fix server/src/**/*.mjs webapp/src/**/*.mjs test/**/*.mjs`
- Empfehlung: Kombination mit npm-Modul `snazzy` zur Formatierung

## snazzy

- Ausgabe von standard/semistandard kann mit dem npm-Modul snazzy verbessert werden:

```
npm install -D semistandard snazzy
```

```
semistandard --verbose | snazzy
```

```
semistandard --fix --verbose | snazzy
```

```
standard: Use JavaScript Standard Style (https://github.com/feross/standard)
/Users/feross/test1.js:2:1: Multiple blank lines not allowed.
/Users/feross/test1.js:3:1: Expected error to be handled.
/Users/feross/test1.js:3:10: handleRequest is defined but never used
/Users/feross/test1.js:3:23: Missing space before function parentheses.
/Users/feross/test2.js:1:1: Strings must use singlequote.
/Users/feross/test2.js:1:4: Extra semicolon.
/Users/feross/test2.js:3:1: "alert" is not defined.
```

```
standard: Use JavaScript Standard Style (https://github.com/feross/standard)

/Users/feross/test1.js
  2:0  error  Multiple blank lines not allowed
  3:0  error  Expected error to be handled
  3:9  error  handleRequest is defined but never used
  3:22 error  Missing space before function parentheses

/Users/feross/test2.js
  1:0  error  Strings must use singlequote
  1:3  error  Extra semicolon
  3:0  error  "alert" is not defined

✖ 7 problems
```

# Präprozessoren



# JavaScript-Präprozessoren

- In manchen Projekten wird nicht unmittelbar in JavaScript implementiert, sondern in einer Programmiersprache, die anschließend nach JavaScript transpiliert wird, z.B.
  - TypeScript (Microsoft)
  - CoffeeScript
  - Dart (Google)
- Vorteile
  - Typsicherheit zur Entwicklungszeit
  - "Klassische" Syntax
  - Kompatibilität mit älteren Browsern durch Ersetzung neuer Sprachfeatures (ES6+) durch Workarounds mit "alten" Sprachfeatures (sog. polyfills)
    - Babel

# CSS-Präprozessoren

- CSS-Dateien sind statisch, d.h. sie erlauben zur Laufzeit einer Web-Anwendung keinerlei Interpretation
- Problem: Hohe Redundanz
  - Gleiche Werte in vielen CSS-Anweisungen (z.B. Farbangaben, Abstände)
  - Abhängigkeiten zwischen mehreren CSS-Anweisungen
- Lösung: CSS-Präprozessoren
  - Modellierung der Stylesheets mithilfe von Konstanten, Variablen, Funktionen, Mixins, Operatoren, Schleifen, ...
  - Erzeugung einer regulären CSS-Datei
    - im Rahmen des Build-Prozesses
    - zur Laufzeit
  - Gängig: Less, Sass/SCSS, Stylus

# Less

```
@main-background-color: #888888;
```

```
@topbar-height: 50px;
```

```
@sidebar-width: 300px;
```

```
@font-size: 9pt;
```

```
@font-color: #FFFFFF;
```

```
main {  
    background-color: @main-background-color;  
    position: absolute;  
    left: @sidebar-width;  
    right: @sidebar-width;  
    top: @topbar-height;  
}
```

```
.h {  
    font-color: @font-color;  
    letter-spacing: 0.25em;  
    white-space: nowrap;  
    margin: 0;  
}  
h1 {  
    .h;  
    font-size: @font-size + 6;  
    font-weight: 400;  
    text-transform: uppercase;  
}  
h2 {  
    .h;  
    font-size: @font-size - 1;  
    font-weight: normal;  
}
```

# Less



- Plugins für gängige IDEs, z.B.
  - <https://marketplace.visualstudio.com/items?itemName=mrcrowl.easy-less>
- npm-Modul
  - Kommandozeile
  - Integration in beliebige Build-Tools
- Installation:  
`npm install -D less`
- Nutzung:  
`lessc webapp/src/styles/style.less webpp/dist/style.css`

# lesshint

- Linting-Tool für Less
- Installation:  
`npm install -D lesshint (lesshint-reporter-stylish)`
- Konfigurationsdatei `.lesshintrc.json` im Projektwurzverzeichnis erstellen  
(<https://github.com/lesshint/lesshint/blob/master/docs/user-guide/configuration.md>):  

```
{  
  "fileExtensions": [".less", ".css"],  
  "excludedFiles": [],  
  "attributeQuotes": { "enabled": true, "severity": "error" },  
  "colorVariables": { "enabled": false, "severity": "error" },  
  ...  
}
```
- Nutzung:  
`lesshint (--reporter lesshint-reporter-stylish) webapp/src/styles`

**Bundling, Minifikation und Obfuskation**

# Bundling

- Es ist i.d.R. sinnvoll und üblich, den JavaScript-Code der eigenen Anwendung auf mehrere Dateien zu verteilen
  - Bei OOP: Jeder Prototyp in eigene Datei
  - *Separation of Concerns*
- Bei der Verwendung eigener oder fremder Bibliotheksmodule kommen weitere Dateien hinzu
- Probleme:
  - Overhead für Initiierung vieler einzelner Anfragen durch den Browser
  - Aufgrund der asynchronen Downloads ist es kompliziert, zu jedem Zeitpunkt die Auflösung der Abhängigkeiten sicher zu stellen
- Lösung: Bundling
  - Zusammenfassen des gesamten benötigten (auch fremden) Codes in eine einzige Datei im Zuge des Build-Prozesses

- Automatisierte und standardisierte Umsetzung des Module-Entwurfsmusters auf Grundlage von ESM- oder CommonJS-Abhängigkeiten (<https://esbuild.github.io/>)
  - Analysiert Abhängigkeiten und erstellt einen Abhängigkeitsgraph
    - Benötigen mehrere Module ein anderes Modul, so wird dieses nur einmal importiert
  - Packt den Einstiegspunkt sowie jedes direkt oder indirekt benötigte Modul in jeweils eine IIFE, sodass diese nach außen abgeschlossen bleiben
  - Kopiert alle IEFES in eine Datei
    - Dies ist die einzige JS-Datei, die in der HTML-Datei referenziert wird
- Erlaubt die Verwendung (fast) der gesamten npm-Moduldatenbank (<https://www.npmjs.com>) auch im Browser
- Ähnliche Tools: browserify, webpack, rollup, snowpack, parcel, broccoli, ...



# esbuild

- npm-Modul
  - Kommandozeile
  - Integration in beliebige Build-Tools
- Installation:  
`npm install -D esbuild`
- Nutzung:  
`esbuild webapp/src/js/Main.mjs`  
`--log-level=warning --bundle`  
`--outfile=webapp/dist/bundle.js`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
    <script defer src="bundle.js"></script>
    ...
</head>
<body>
    ...
</body>
</html>
```

# Minifikation

- Ziel: Reduzierung des zu übertragenden Datenvolumens
- Dateiinhalte, die die Entwicklung erleichtern oder die Lesbarkeit verbessern, zur Ausführung aber unnötig sind, können im Build-Prozess entfernt werden, z.B.
  - Leerzeichen, Tabulatoren, Zeilenumbrüche (engl. *whitespace*)
  - Kommentare
  - Unnötige Semikola oder Klammern
- Tools beherrschen häufig weitere Optimierungen, z.B.
  - Verkürzte Schreibweise für if-Verzweigungen durch Short-Circuit-`&&` bzw. den ternären Bedingungsoperator
  - Zusammenfassung mehrerer Variablendeklarationen zu einer Anweisung

# Obfuskation

- JavaScript wird nicht kompiliert und in lesbarer Form verteilt
  - Kopieren des Quellcodes leicht
  - Bei Open Source-Projekten unproblematisch und sogar erwünscht
  - N.B.: Kompilierter Code kann oft dekompiliert und so auch kopiert werden
- Ziel: Absichtliche Verschlechterung der Lesbarkeit
  - z.B. durch Ersetzung von langen, sprechenden Bezeichnern (Namen von Klassen, Methoden, Variablen usw.) durch kürzere, nicht-sprechende
  - Nebeneffekt: Weitere Verringerung der Dateigröße
  - Kein effektiver Kopierschutz!

# Terser



- ES6+-fähiger Minifikator/Obfuskator
- npm-Modul
  - Kommandozeile
  - Integration in beliebige Build-Tools
- Installation:  
`npm install -D terser`
- Ähnliche Tools:
  - babel-preset-minify (Plugin für Babel-Präprozessor)
  - Google Closure Compiler

# Terser



- Nutzung:

```
terser webapp/dist/bundle.js  
  
  --compress           // Minifikation  
  --mangle             // Obfuskation  
  --comments=false    // Entfernt auch Lizenzkommentare  
  -o webapp/dist/bundle.js
```

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="utf-8" />  
  <link rel="stylesheet" href="style.css" />  
  <script defer src="bundle.js"></script>  
  ...  
</head>  
<body>  
  ...  
</body>  
</html>
```

# Minifizierte und obfuskierte JavaScript-Datei

```
(function d(g,c,p){function h(I,b){if(!c[I]){if(!g[I]){var F="function"==typeof require&&require;if(!b&&F)return F(I,!0);if(C)return C(I,!0);var T=new Error("Cannot find module '"+I+"'");throw T.code="MODULE_NOT_FOUND",T}var S=c[I]={exports:{}};g[I][0].call(S.exports,function(L){var A=g[I][1][L];return h(A?A:L)},S,S.exports,d,g,c,p)}return c[I].exports}for(var C="function"==typeof require&&require,E=0;E<p.length;E++)h(p[E]);return h})(({1:[function(d,g,c){const p=d("izitoast");c.showToast=function(h,C=!1){h&&p.show({title:C?"Error":"Info",message:h,backgroundColor:C?"darkred":"#FF2262",titleColor:"white",messageColor:"white",progressBarColor:"white",animateInside:!1,close:!1})},c.showDialog=function(h,C=[]){h&&p.show({title:"Info",message:h,backgroundColor:"#FF2262",titleColor:"white",messageColor:"white",timeout:"disabled",animateInside:!1,close:!1,position:"center",buttons:C})},c.truncateTextToLength=function(h,C){const E="\u2026";let I=h;return h.length>C&&(I=h.substr(0,C-E.length)+E),I},c.getFileExtension=function(h){return h.substr(h.lastIndexOf(".")+1)},c.degToRad=function(h){return h*Math.PI/180},c.traverseParentNodes=function(h,C){for(let E=h;E;){if(C(E))return E;E=E.parentNode}return null}},{izitoast:62}],2:[function(d){function p(){let Ce=document.getElementById("leftSideBar"),fe=document.getElementById("leftSideBarButton"),Ee=document.getElementById("leftSideBarLabel"),Ie=document.getElementById("leftSideBarContent"),be=document.getElementById("visualizationGroup");...
```

# Minifikation von CSS-Dateien



- Less-Plugin
- Installation:  
`npm install -D less-plugin-clean-css`
- Nutzung:  
`lessc --clean-css`  
    `webapp/src/styles/style.less`  
    `webapp/dist/style.css`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
    <script defer src="main.js"></script>
    ...
</head>
<body>
    ...
</body>
</html>
```

# Minifizierte CSS-Datei

```
body{width:100%;height:100%;margin:0;padding:0;font-family:Roboto,Helvetica,Arial,sans-serif;font-size:9pt;color:#fff;-webkit-font-smoothing:antialiased;text-rendering:optimizeLegibility;-moz-osx-font-smoothing:grayscale;background-color:#888;cursor:default;overflow:hidden;user-select:none;-webkit-user-select:none;-moz-user-select:none;-ms-user-select:none}.flex{display:flex;display:-webkit-flex;display:-ms-flexbox;align-items:center;-webkit-align-items:center;-ms-flex-align:center}.flexV{display:flex;display:-webkit-flex;display:-ms-flexbox;align-items:center;-webkit-align-items:center;-ms-flex-align:center;-ms-flex-direction:column;flex-direction:column}.fill{flex:1}.border{border-width:1px;border-color:#888;border-style:solid;border-radius:0}.icon{font-family:'Material Icons';font-weight:400;font-style:normal;display:inline-block;text-transform:none;letter-spacing:normal;word-wrap:normal;white-space:nowrap;direction:ltr;-webkit-font-smoothing:antialiased;text-rendering:optimizeLegibility;-moz-osx-font-smoothing:grayscale;font-feature-settings:'liga'}.h{letter-spacing:.25em;white-space:nowrap;margin:0}h1{letter-spacing:.25em;white-space:nowrap;margin:0;font-size:15pt;font-weight:400;text-transform:uppercase}h2{letter-spacing:.25em;white-space:nowrap;margin:0;font-size:8pt;font-weight:400}h3{letter-spacing:.25em;white-space:nowrap;margin:0;font-size:9pt;font-weight:400;text-transform:uppercase}#upload,button,select{border-width:1px;...
```



**Unit-Tests**

# Unit-Tests

- Testen von Software auf der Ebene der Module, Klassen und Methoden ist ein wesentlicher Aspekt der professionellen Entwicklung
  - Insbesondere (aber nicht nur) bei Bibliotheksprojekten
- Automatisierung beugt Regressionen vor ("Regressionstest")
- Unit-Tests: Abgleich zwischen spezifiziertem und tatsächlichem Verhalten
- Theoretische Grundlagen
  - Black Box-Testing, Äquivalenzklassenverfahren, ...
  - siehe Modul "Software-Qualitätssicherung"

# Mocha



- npm-Modul
  - Kommandozeile
  - Integration in beliebige Build-Tools
- Unterstützt u.a.
  - Node.js und Browser
  - asynchronen Code
  - ES6+-Features
- <https://mochajs.org>
- Ähnliche Tools: QUnit, Jasmine, ...
- Installation:  
`npm install -D mocha`
- Verwendung  
`mocha lib/test/Common.test.mjs`  
`mocha lib/test`  
`mocha`

# Mocha

```
export function isNaturalNumber (value) {  
  return value !== null &&  
    Number.isInteger(value) && value >= 0;  
}
```

- Hierarchische Gruppierung von Testfällen mittels `describe()`
- Definition eines Testfalls mittels `it()`
- Zum Vergleich wird ein sog. Assertion-Modul benötigt
  - Node: z.B. Standardmodul `assert`
  - Browser: `Chai`, `should.js`, `expect.js`, ...

```
import assert from 'assert';  
  
import { isNaturalNumber } from '../src/Common.mjs';  
  
describe('isNaturalNumber()', () => {  
  describe('wrong type', () => {  
    it('should return false if argument is not of type number', () => {  
      assert.strictEqual(isNaturalNumber('SomeString'), false);  
    });  
  });  
});
```

# assert

The screenshot shows the Node.js v16.17.0 documentation for the `assert` module. The browser address bar shows `nodejs.org/docs/latest-v16.x/api/assert.html`. The left sidebar contains a navigation menu with the following items: Node.js, About this documentation, Usage and example, Assertion testing (highlighted), Asynchronous context tracking, Async hooks, Buffer, C++ addons, C/C++ addons with Node-API, C++ embedder API, Child processes, Cluster, Command-line options, Console, Corepack, Crypto, Debugger, Deprecated APIs, Diagnostics Channel, DNS, Domain, Errors, Events, File system, Globals, HTTP, and HTTP/2. The main content area is titled "Node.js v16.17.0 documentation" and includes a "Table of contents" link. The "Table of contents" section is expanded, showing a list of items under the "Assert" heading: Strict assertion mode, Legacy assertion mode, Class: `assert.AssertionError` (with a sub-item `new assert.AssertionError(options)`), Class: `assert.CallTracker` (with sub-items `new assert.CallTracker()`, `tracker.calls([fn][, exact])`, `tracker.report()`, and `tracker.verify()`), `assert(value[, message])`, `assert.deepEqual(actual, expected[, message])` (with a sub-item "Comparison details"), `assert.deepStrictEqual(actual, expected[, message])` (with a sub-item "Comparison details"), `assert.doesNotMatch(string, regexp[, message])`, `assert.doesNotReject(asyncFn[, error][, message])`, `assert.doesNotThrow(fn[, error][, message])`, `assert.equal(actual, expected[, message])`, `assert.fail([message])`, `assert.fail(actual, expected[, message[, operator[, stackStartFn]]])` (marked as **deprecated**), `assert.ifError(value)`, `assert.match(string, regexp[, message])`, `assert.notDeepEqual(actual, expected[, message])`, `assert.notDeepStrictEqual(actual, expected[, message])`, and `assert.notEqual(actual, expected[, message])`.

# Mocha

The screenshot shows the Visual Studio Code editor with a file named `CommonTest.mjs` open. The file contains Mocha test code for a library. The tests are organized into `describe` blocks for `isNaturalNumber` and `createUID`. The `isNaturalNumber` tests cover various inputs: wrong type, non-integer, negative integer, positive integer, and zero. The `createUID` tests cover length, grouping, and allowed characters. The code uses `import assert from 'assert';` and `import { isNaturalNumber, createUID } from '../src/Common.mjs';`.

```
1 import assert from 'assert';
2
3 import { isNaturalNumber, createUID } from '../src/Common.mjs';
4
5 describe('isNaturalNumber()', function () {
6   describe('wrong type', function () {
7     it('should return false if argument is not of type number', () => {
8       assert.strictEqual(isNaturalNumber('SomeString'), false);
9     });
10  });
11
12  describe('not integer', function () {
13    it('should return false if argument is not a integer', () => {
14      assert.strictEqual(isNaturalNumber(1.5), false);
15    });
16  });
17
18  describe('negative integer', function () {
19    it('should return false if argument is a negative integer', () => {
20      assert.strictEqual(isNaturalNumber(-1), false);
21    });
22  });
23
24  describe('positive integer', function () {
25    it('should return true if argument is a positive integer', () => {
26      assert.strictEqual(isNaturalNumber(1), true);
27    });
28  });
29
30  describe('zero', function () {
31    it('should return true if argument is 0', () => {
32      assert.strictEqual(isNaturalNumber(0), true);
33    });
34  });
35 });
36
37 describe('createUID()', function () {
38   describe('length', function () {
39     it('should return a string of length 36', () => {
40       assert.strictEqual(createUID().length, 36);
41     });
42   });
43
44   describe('grouping', function () {
45     it('should be grouped by '-' at the right positions', () => {
46       assert.deepStrictEqual(indicesOf(createUID(), '-'), [8, 13, 18, 23]);
47     });
48   });
49
50   describe('allowed chars', function () {
51     const ALLOWED_CHARS = '0123456789ABCDEF';
52
53     it('should only contain allowed chars', () => {
54       const uid = createUID();
55       for (const char of uid) {
56         assert.ok(ALLOWED_CHARS.includes(char), 'Invalid char: ${char}');
57       }
58     });
59   });
60 });
61
62 function indicesOf(string, char) {
63   const indices = [];
64   for (let i = 0; i < string.length; i++) {
65     if (string[i] === char) {
66       indices.push(i);
67     }
68   }
69   return indices;
70 }
71
```

The right sidebar shows the test results for the `lib@1.0.0 test` command. It lists the test suites and their status:

- `isNaturalNumber()`
  - `wrong type`: ✓ should return false if argument is not of type number
  - `not integer`: ✓ should return false if argument is not a integer
  - `negative integer`: ✓ should return false if argument is a negative integer
  - `positive integer`: ✓ should return true if argument is a positive integer
  - `zero`: ✓ should return true if argument is 0
- `createUID()`
  - `length`: ✓ should return a string of length 36
  - `grouping`: ✓ should be grouped by '-' at the right positions
  - `allowed chars`: ✓ should only contain allowed chars
- `Point`
  - `x`: ✓ should return value which was given to constructor
  - `y`: ✓ should return value which was given to constructor

The summary shows **10 passing (7ms)**. The terminal at the bottom shows the command `cb0FURY:/mnt/d/Code/ME/code/05-Tools/lib$ npm run test` and the output `> lib@1.0.0 test` and `> mocha`.

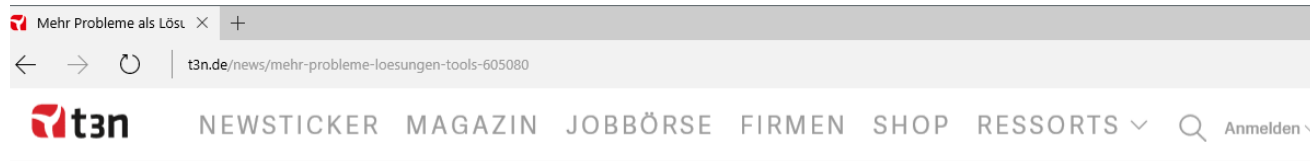
**Build-Tools**

# Build-Tools

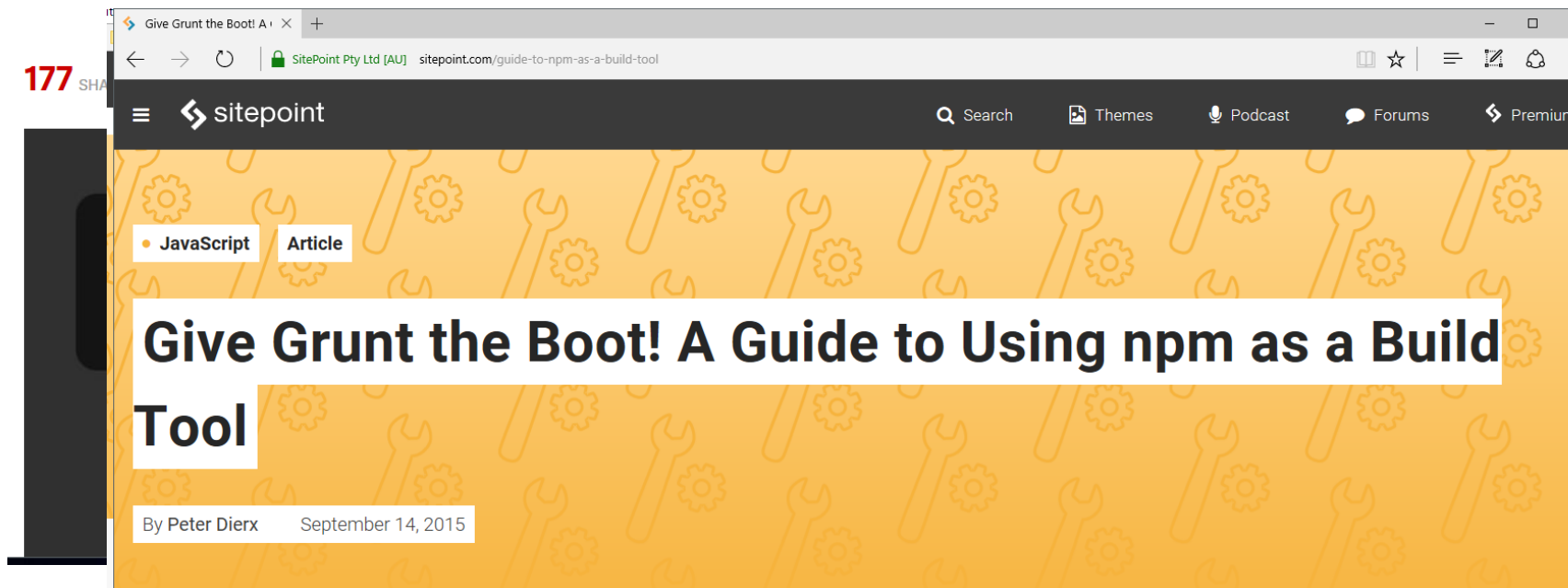
- Zentrales Werkzeug zur automatisierten Erstellung eines Software-Projekts aus dessen Komponenten
- Orchestrieren die Tools im Entwicklungsprozess
- Bekannte Build-Tools
  - make
  - Ant, Maven, Gradle
  - ...
- Bekannteste Build-Tools im JavaScript-Umfeld: Grunt, Gulp.js
  - Konfigurationen werden selbst in JavaScript implementiert
    - sog. *Domain Specific Language* (kurz: DSL)
  - Anbindung von Tools über Plugins



# Build-Tools



## Mehr Probleme als Lösungen: Warum Tools wie Grunt, Gulp und Co. eine schlechte Wahl sind



# npm als Build-Tool

- npm als Werkzeug zum Paketmanagement
- Kann auch als Build-Tool eingesetzt werden
- Anlegen einer Konfigurationsdatei  
`package.json` via `npm init`
- Zentrale Eigenschaften:
  - `dependencies`
  - `devDependencies`
  - `scripts`

```
{  
  "name": "webapp",  
  ...  
  "dependencies": {  
    ...  
  },  
  "devDependencies": {  
    ...  
  },  
  "scripts": {  
    ...  
  }  
}
```

## npm als Build-Tool (dependencies)

- Angabe aller npm-Module, die zur Laufzeit benötigt werden, im Objekt `dependencies`
  - Semantische Version (<https://semver.org>)
  - `latest`
  - Lokale Pfade
  - ...
- Eintragung und Download ins Unterverzeichnis `node_modules` via `npm install <modulname>`
  - `import ... from M` sucht im Ordner der `package.json` nach `node_modules/M`

```
{  
  "name": "lib",  
  ...  
  "dependencies": {  
    "random-curse-words": "^0.0.2"  
  },  
  ...  
}  
  
{  
  "name": "webapp",  
  ...  
  "dependencies": {  
    "browser": "^2.11.0",  
    "lib": "file:../lib"  
  },  
  ...  
}
```

## npm als Build-Tool (devDependencies)

- Ein guter Build-Prozess sollte keine global installierten Tools voraussetzen
- Angabe aller npm-Module, die nur zur Erstellung aber nicht zur Laufzeit benötigt werden, im Objekt `devDependencies`
- Eintragung und Download ins Unterverzeichnis `node_modules` via `npm install -D <modulname>`
- `npm install` lädt sowohl die Module in `dependencies` als auch die in `devDependencies` herunter

```
{  
  "name": "webapp",  
  ...  
  "devDependencies": {  
    "esbuild": "^0.14.54",  
    "http-server": "^14.1.1",  
    "less-plugin-clean-css": "^1.5.1",  
    "less": "^4.1.3",  
    "lesshint-reporter-stylish": "^3.0.0",  
    "lesshint": "^6.3.7",  
    "mocha": "^10.0.0",  
    "semistandard": "^16.0.1",  
    "snazzy": "^9.0.0",  
    "terser": "^5.14.2",  
  },  
  ...  
}
```

## npm als Build-Tool (scripts)

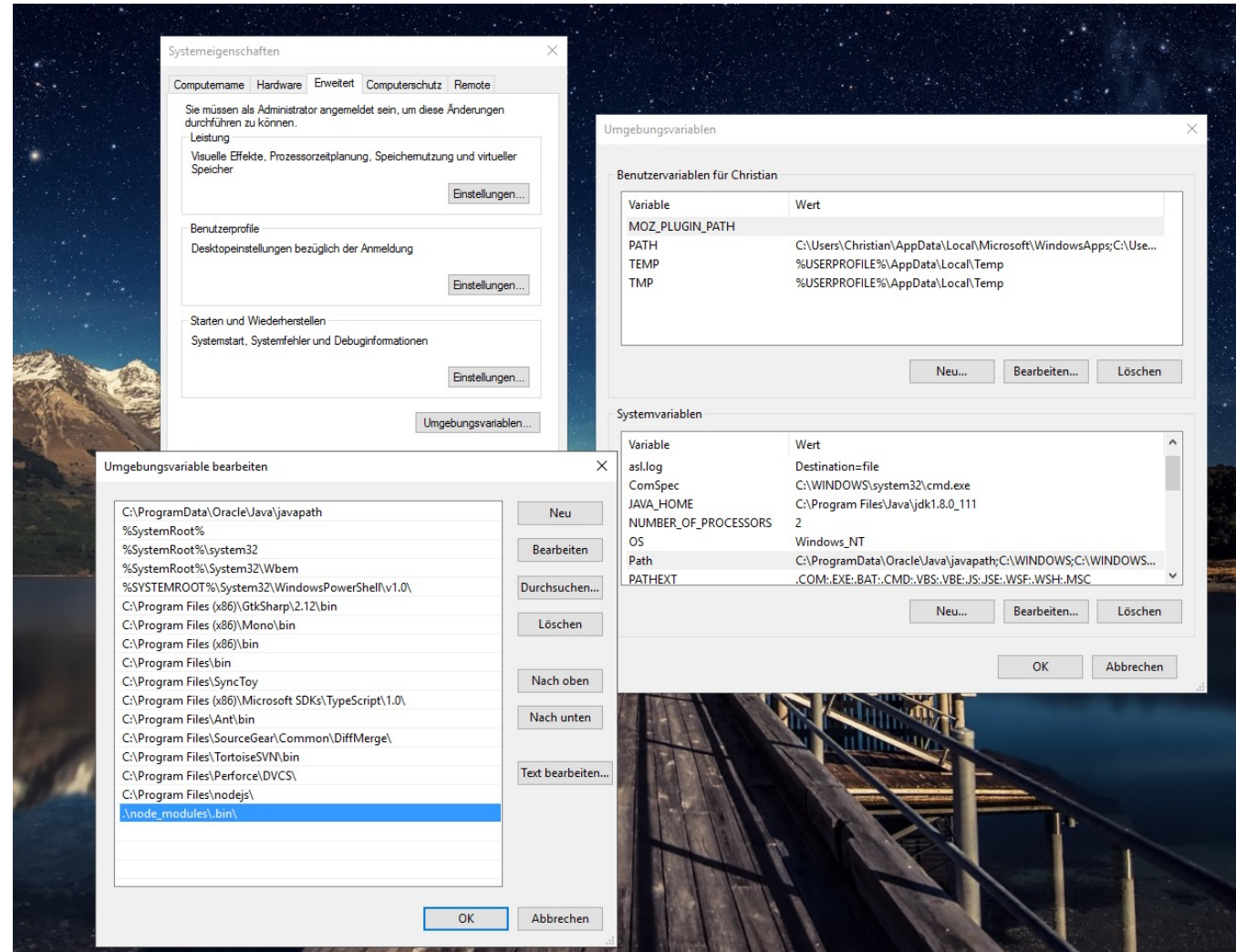
- Eigenschaftsnamen des `scripts`-Objekts sind Namen der Build-Tasks
  - Gängige Konvention: Varianten des gleichen Build-Tasks enthalten einen durch Doppelpunkt getrennten Namenssuffix
- Eigenschaftswerte sind einfache Kommandozeilenbefehle
  - Verkettung mithilfe von `&&`
  - Umleitungen mithilfe von `<`, `>` und `|`
  - Kommandos der entsprechenden Shell: `cd`, `mkdir`, `cp`, `mv`, `rm`, ...
- Aufruf der Build-Tasks via `npm run <scriptname>`, z.B. `npm run lint`

```
"scripts": {  
  "clean": "npm run clean:dist && rm -rf node_modules",  
  "clean:dist": "rm -rf dist",  
  "lint": "semistandard --verbose src/js/**/*.mjs | snazzy && lesshint --reporter lesshint-reporter-stylish src/styles",  
  "format": "semistandard --verbose --fix src/js/**/*.mjs | snazzy",  
  "test": "mocha",  
  ...  
}
```

## npm als Build-Tool (PATH)

- Die npm-Module in `devDependencies` sind typischerweise CLI-Tools und liegen im Unterordner `./node_modules/.bin`
  - Keine global installierten npm-Module
- Damit der Aufruf dieser Tools aus den Build-Tasks heraus ohne Angabe von Pfaden funktioniert, muss dieser relative Pfad in die Umgebungsvariable `PATH` des Betriebssystems aufgenommen werden
  - Unixoider Shell:  
`export PATH=$PATH:./node_modules/.bin`
  - Windows:  
Systemeigenschaften > Umgebungsvariablen > Systemvariablen

# npm als Build-Tool (PATH)



## npm als Build-Tool (scripts)

```
"scripts": {  
  ...  
  "debug": "npm run html && npm run css && npm run js",  
  "build": "npm run debug && npm run minify",  
  "html": "mkdir -p dist && cp src/index.html dist/index.html",  
  "css": "mkdir -p dist && lessc src/styles/style.less dist/style.css",  
  "js": "mkdir -p dist && esbuild src/js/Main.mjs --log-level=warning --bundle --outfile=dist/bundle.js",  
  "minify": "npm run minify:css && npm run minify:js",  
  "minify:css": "lessc --clean-css dist/style.css dist/style.css",  
  "minify:js": "terser dist/bundle.js --compress --mangle --comments=false -o dist/bundle.js",  
  "start": "http-server dist"  
}
```



## npm als Build-Tool (scripts)

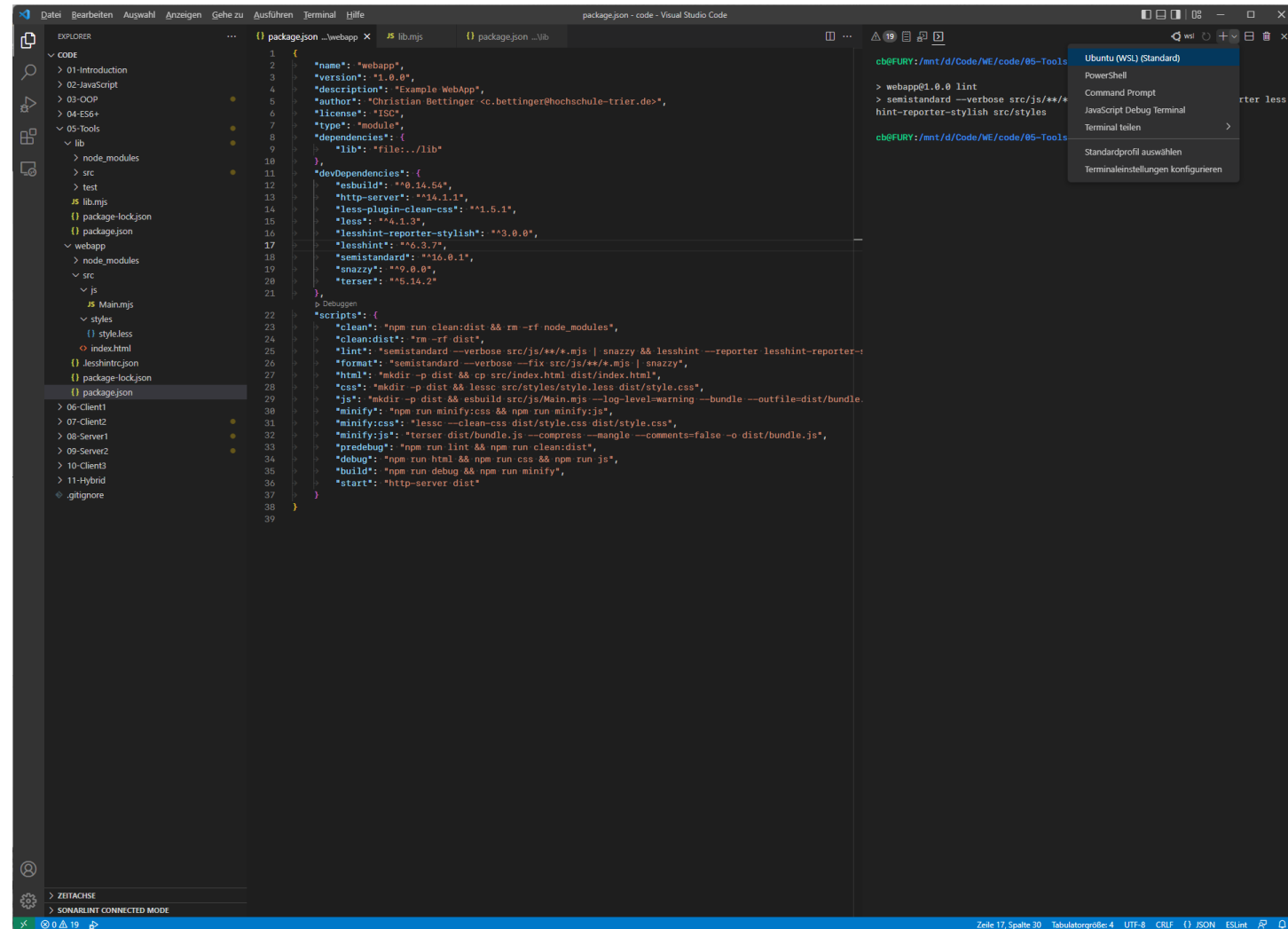
- Eigenschaftsnamen, die mit `pre` bzw. `post` beginnen, werden automatisch vor bzw. nach dem entsprechend benannten Build-Task ausgeführt
- `pre-Hook`: Sicherstellung von Vorbedingungen
  - z.B Linting, Unit-Tests
  - Eigentlicher Build-Task wird nicht ausgeführt, falls `pre-Hook` nicht mit Exit-Code 0 endet
- `post-Hook`: "Aufräumarbeiten"

```
"scripts": {  
  ...  
  "predebug": "npm run lint && npm run clean:dist"  
}
```

# npm als Build-Tool

- npm-Build-Skripte werden in der betriebssystemspezifischen Shell ausgeführt
  - Windows: `cmd.exe`, `powershell.exe`
  - Unixoides Betriebssystem: `bash`, `ash`, `dash`, `ksh`, `pdksh`, `mksh`, `csh`, `tcsh`, `zsh`, `fish`, ...
- Problem
  - `&&`, `<`, `>` und `|` werden sowohl von Windows als auch von Unix-Shells unterstützt
  - Einzelne Unix-Kommandos (z.B. `cp`, `mv` oder `rm`) haben in Windows-Shells andere Namen (`cmd.exe`: `COPY`, `MOVE`, `DEL`) oder andere Argumente (`powershell.exe`: `rm`)
- Lösungsansätze
  - Falls nur auf einer Plattform entwickelt wird, können die Kommandos dieser Plattform verwendet werden
  - Verwendung von npm-Modulen mit entsprechender Funktionalität (z.B. `cp-cli`, `mv`, `rimraf`)
    - Problem: Löschen von `node_modules`
  - Empfehlung: Verwendung der Unix-Kommandos und Nutzung einer Unix-artigen Kommandozeile unter Windows (z.B. Windows-Subsystem für Linux, Cygwin, Git BASH, ...)
    - Hausarbeit: sh-Kompatibilität wird vorausgesetzt

# WSL-Integration in Visual Studio Code



Fragen?

© 2015 Christian Bettinger

Nur zur Verwendung im Rahmen des Studiums an der Hochschule Trier.

Diese Präsentation einschließlich aller Abbildungen ist urheberrechtlich geschützt. Jede Verwertung außerhalb der Grenzen des Urheberrechts ist ohne Zustimmung des Autors unzulässig.

Die Quellen der Abbildungen sind entsprechend angegeben. Alle Marken sind das Eigentum ihrer jeweiligen Inhaber, wobei alle Rechte vorbehalten sind.

Die Haftung für sachliche Fehler ist ausgeschlossen.