Wartbare Javascript-lastige Webanwendungen

W3L AG
info@W3L.de

2014

Agenda

- **Motivation / Problemstellung**
- **Wartbarkeit**
  - ISO 25010 (vormals ISO 9126-1)
- **Wartbares Javascript**
  - Strukturierung und Modularisierung
  - Programmierrichtlinien
  - Quellcodeanalyse und weitere Werkzeuge
  - Verknüpfung von Javascript und HTML
- **Alternatives Vorgehen: „Kompilierung von JS“**
- **Testbarkeit von Javascript**
- **Schlussbetrachtung**

Motivation

- **HTML5 und Javascript: Webbrowser als Zielplattform**
  - Mobile Web-Apps / Offline-Unterstützung von Webseiten
  - Javascript-basiert „Rich Internet Applications“
    - Paradigma der „Single-page Webanwendung“
- **Ähnliche Funktionsvielfalt und Bedienung wie Desktopanwendungen**
  - Direkte Reaktion auf Benutzereingaben / Reduzierte Wartezeiten
  - Inhalte werden asynchron aktualisiert, ohne die komplette Webseite neu zu laden
- **Resultat: Ein Wandel der Webarchitekturen findet statt**
  - Vermehrte clientseitige Logik zur Realisierung der Anwendung
  - Höhere Interaktivität und Selbstständigkeit des Webclients
  - Thin-Client-Architektur → Rich-Client-Architektur → Fat-Client
- **Anwendungsbeispiele: Google Maps; Google Docs; Twitter; Facebook**

Problemstellung

- **Webanwendungen werden immer Javascript-lastiger**
  - Dynamische Programmiersprache (keine feste Typisierung)
  - „Die typisierte Programmierung gehört zu den mächtigsten Hilfsmitteln der konstruktiven Qualitätssicherung“ (vgl. [Hof08, S. 76])
  - Wenige sprachliche Mittel zur Strukturierung des Quellcodes
- **Umgang mit der zunehmender Selbstständigkeit des Webclients**
  - Zeitweiser Betrieb ohne einen Web-Server (Offline-WebApps)
  - HTML5-Web-Storage-Technologie
  - Techniken, wie Flash oder Silverlight werden verdrängt
- **Problem: Steigender Aufwand in der Entwicklung, der Qualitätssicherung und in der Wartung**
  - Wie können möglichst „wartbare“ Webclients entwickelt werden?
  - Wie ist mit Javascript umzugehen?
  - Welche Optionen stehen neben einer manuellen Qualitätssicherung zur Verfügung?

Wartbarkeit

- **Das Qualitätsmerkmal „Wartbarkeit“ wird in der ISO/IEC 9126-1 bzw. ISO/IEC 25010 definiert**
  - „Fähigkeit des Softwareprodukts änderungsfähig zu sein. [..]“
  - Im „Kleinen“: Korrekturen, Verbesserungen
  - Im „Großen“: Änderungen der Anforderungen und der funktionalen Spezifikationen
- **Teilmerkmale zur Konkretisierung [ISO/IEC 25010]**
  - Modularität, Wiederverwendbarkeit, Analysierbarkeit/Verständlichkeit, Änderbarkeit/Modifizierbarkeit, Änderungsstabilität, Testbarkeit/Prüfbarkeit
  - **Eine Erreichung ist nur in konstruktiver Form möglich**
- **Software-Entwicklungsprozess aus der Sicht eines Entwicklers**
  - Softwareinfrastruktur/Tool-Set: Werkzeuge zur Unterstützung bei der Entwicklung einer Javascript-lastigen Webanwendung
  - Entwicklungsumgebung; Debugger & Profiler; Quellcodeanalyse & Metrikberechnung

Wartbares Javascript

- **Es ist hohe Disziplin bei der Programmierung zu wahren**
  - Javascript wurde ursprünglich als Skriptsprache für einen überschaubaren Einsatz im Rahmen von Webseiten entwickelt
  - Dynamisch und keine Deklarationspflicht
  - Keine Klassen, keine Schnittstellen, keine Packages, keine Module
- **Die Dynamik von Javascript kann zu fehlerhaften und schlecht wartbaren Anwendungen führen**
  - Ein und dieselbe Variable kann zu verschiedenen Zeitpunkten zwei völlig unterschiedliche Objekttypen referenzieren
  - Keine Strukturierung des Programmcodes
  - → **Prozedurale Programmierung mit globalen Variablen**
- **Mögliche konstruktive Maßnahmen**
  - Strukturierung und Modularisierung der Javascript-Anwendung
  - Programmierrichtlinien festgelegen

Strukturierung & Modularisierung

- **Klassische Strukturierungsmittel in objektorientierten Programmiersprachen**
  - Klassen- und Vererbungsstrukturen, Paketstrukturen, Modularisierung
- **Ziele: „Wartbarkeit“ & „Beherrschbarkeit“**
  - Zerlegung des Gesamtproblems einer Anwendung in einzelne Teile
  - Erreichung einer besseren Analysierbarkeit des Quellcodes
  - Wiederverwendbarkeit von Teilkomponenten in anderen Systemen
  - Bildung von unabhängigen Komponenten (*Kopplung minimieren)*
  - Gruppierung von ähnlichen Belangen (*Bindung maximieren)*
- **→ Auch mit den eingeschränkten sprachlichen Mitteln von Javascript können Strukturen und Module gebildet werden**

Vererbungsstrukturen in Javascript

Vererbung + Geheimhaltung + Konstruktor-Kette

```
function Mitarbeiter (name) {                          // Klassendefinition “Mitarbeiter”
        var name = name || “”;;                        // private-member
        this.durchwahl = 122;                          // public-member
        this.getName = function(){                     // Getter für Name
            return(name);
        };
}
function Manager (name, abteilung) {                   // Klassendefinition “Manager”
        Mitarbeiter.call( this, name);                 // Konstruktor der Klasse „Mitarbeiter“ auf
                                                          das aktuelle Objekt aufrufen.
        this.abteilung = abteilung;                    // Neue public Variable
}

Manager.prototype = new Mitarbeiter;                   // Prototype auf Mitarbeiter-Instanz setzen
var vorstand = new Manager(“Müller”, “Vorstand”);      // Manager erzeugen
vorstand.durchwahl = 121;
//Objektzustand: Object { durchwahl: 121, getName: Mitarbeiter/this.getName(), abteilung: "Vorstand" }
```

Paketstrukturen in Javascript

- Das Konzept der Pakete bzw. „Namespaces“ ist nicht vorhanden
- Eine Nachbildung ist über Javascript-Objekte möglich

```
//Globales Objekt für die Anwendung definieren, fall nicht schon vorhanden
var SPA_APPLICATION = SPA_APPLICATION || {};
SPA_APPLICATION.MODEL = {                                   //Untergeordneter Namespace 'Model'
        getProduct: function (id) {
                return "TBD";
        }
};
SPA_APPLICATION.LOGIC = {                                   //Untergeordneter Namespace 'Logic'
        addToCart: function (productID) {
                return "TBD";
        },
        doCalculations: function (productID) {
                var p = new SPA_APPLICATION.MODEL.getProduct(productID);
                return "TBD";
        }
};
```

Modularisierung in Javascript

- **Javascript beinhaltet kein Konzept zur Modularisierung**
  - Bausteinartige Aufteilung des Gesamtsystems in einzelne Komponenten
  - Durch die Modularität und die Festlegung von Schnittstellen können die Komponenten losgelöst voneinander entwickelt werden
  - Austauschbar, wiederverwendbar und separat testbar
- **Nachbildung mit Javascript**
  - Variante 1: Definition von globalen Objekten als Modulschnittstellen
    - Jede Komponente exportiert ihre Funktionalität in Form eines globalen Objektes
    - Per Namenskonvention können andere Komponenten zugriff nehmen
    - **Schwierigkeiten: Abhängigkeiten & Lade-Reihenfolgen; doppelte Bezeichnungen**
  - Variante 2: Einsatz des AMD-Formats („Asynchronous module definition“)
    - → Sieht eine spezielle Moduldefinition bzw. Komponentendefinition vor

AMD-Format

Modulspezifikation

- Das AMD-Format legt die Form der Modulspezifikation fest
  - define(“Name des Moduls”, [“Abhängigkeit1”, “Abhängigkeit2”], factory);
- Die Factory ist für die Instanziierung der Komponente zuständig
- Die Abhängigkeiten 1 und 2 werden als Übergabeparameter an die Factory-Methode übergeben

AMD-Loader

- Der AMD-Loader implementiert dieses Format und verwaltet sämtliche Komponenten (→ „Dependency Injection“)
- AMD-Loader-Bibliotheken: „RequireJS“, „The Dojo Loader“ oder „curl.js“

Code-Beispiel

Programmierrichtlinien

- **Annahme: „Programmtexte werden häufiger analysiert als geschrieben.“**
- **Ziele von Programmierrichtlinien**
  - Die Definition und die Einhaltung von Programmierrichtlinien helfen, das Erzeugnis des Programmierprozesses zu verbessern
  - Insbesondere individuelle Programmierstile können abgefangen werden
  - Standardisierung des Quellcodes → Steigerung der Wartbarkeit
- **Programmierrichtlinien geben in der Regel folgende Dinge vor:**
  - Formatierung des Quellcodes (Klammerungsregeln, Umgang mit Whitespaces)
  - Umgang mit Kommentaren; Form der Variablen- und Funktionsdeklaration
  - Namenskonventionen für Bezeichner
  - Umgang mit Sprachkonstrukten (syntaktischen und semantischen Restriktionen)
- **Für Programmiersprachen wie C# oder Java geben Microsoft bzw. Sun/Oracle bereits Programmierrichtlinien vor. *Was ist mit Javascript?***

Programmierrichtlinien

- **Programmierrichtlinien für Javascript**
  - Standardwerke
    - Crockford, Douglas (2008). *JavaScript: The Good Parts* [O'Reilly Media]
    - Zakas, Nicholas C. (2012). *Maintainable JavaScript* [O'Reilly Media]
  - Online-Richtlinien
    - Google JavaScript Style Guide
    - jQuery JavaScript Style Guide
- **Alle vier genannte Quellen für Richtlinien besitzen Überschneidungen**
  - Crockford befasst sich ausgiebiger mit den sprachlichen Mitteln von Javascript
    - „Good Parts and Bad Parts“
  - Zakas gibt intensiver verschiedene „Best Practices“ an
  - Die Onlinequellen sind komprimiert und zusammengefasst
- **→ Im Folgenden werden einzelne ausgewählte Richtlinien aufgeführt**

Programmierrichtlinien

Aktivierung des Strict Mode nach [Cro08] und [Zak12]

- „Opt-In-Feature“ von Javascript in der Version ES5
- Definiert ein „bereinigtes“ Subset von Javascript und aktiviert eine stärkere Quellcode-Validierung durch die Ausführungsengine

```
“use strict”;
undefined = 42;                    // TypeError - “undefined” kann nur undefined sein
delete Object.prototype;           // TypeError - Object.prototype ist schreibgeschützt
var myObj = { wert: “x”, wert: “y” }; // SyntaxError – ‘wert’ kann nicht  zweimal definiert werden
function test(x, x, z){}           // SyntaxError – Keine doppelten Parameternamen
if(true){
  function FehlerhafteDefinition(){} // SyntaxError
}
```

Ziele:

- Javascript vereinfachen & Eigenarten beheben → Programmierfehlererkennung
- Quellcodequalität verbessern

Programmierrichtlinien

- *„Don't Modify Objects You Don't Own"* nach [Zak12]
  - In Javascript kann jedes Objekt beliebig bearbeitet und erweitert werden
  - Regel: Programmierschnittstellen von Objekten aus externen Komponenten dürfen nicht manipuliert werden
  - Extremfall: Manipulation von nativen Javascript-Objekten

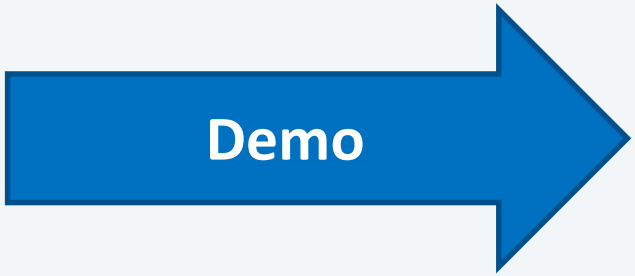
```
// Schlecht – Überschreiben von Funktionen
document._originalGetElementById = document.getElementById;
document.getElementById = function(id) {
        if (id == "window") {
            return window;
        } else {
            return document._originalGetElementById(id);
        }
};
// Schlecht – Entfernen von DOM-Operationen
delete document.getElementById;          //Hat keine Auswirkung
document.getElementById = null;          //Funktioniert, aber leider..
```

Weitere Programmierrichtlinien

- **Namenskonventionen nach dem „Google JavaScript Style Guide“**
  - functionNamesLikeThis = Camel-Case
  - ClassNamesLikeThis = Pascal-Case
- **Vergleichsoperatoren nach dem „jQuery JavaScript Style Guide“**
  - „==“ vs. „===“
- **Variablendeklaration nach [Cro08]**
  - Zu Beginn einer Funktion → Interpreter-Sprache: Fehleranfälligkeit
- **Globale Variablen nach dem „jQuery JavaScript Style Guide“**
  - Maximal eine globale Variable pro Komponente
- **Einsatz vom JSDoc nach dem „Google JavaScript Style Guide“**
- **Terminierung von Ausdrücken nach [Zak12]**
- **Klammerung von Abfragen und Schleifen nach [Zak12] und [Cro08]**
- **…**

Statische Quellcodeanalyse

- **Programmierrichtlinien helfen dabei konstruktive Vorgaben für die Entwicklung festzulegen**
- **Problem:** Wer prüft die Einhaltung dieser Regeln? [+ Aufwand]
- **Lösung:** Statische Quellcodeanalyse
  - Automatisierte Durchführung einer Konformitätsanalyse des Quellcodes
  - Prüfung von Syntaxrichtlinien / Einfache Kontroll- und Datenflussanalysen
  - Da Javascript erst zur Laufzeit interpretiert wird, ermöglicht die statische Quellcodeanalyse eine Fehleridentifizierung vor der eigentlichen Ausführung
- **Javascript-Werkzeuge**
  - JSLint von Douglas Crockford
  - JSHint als Weiterentwicklung von JSLint



Weitere Werkzeuge

- **Integrierte Entwicklungsumgebung (IDE)**
  - Javascript-Unterstützung ist in Eclipse, NetBeans und Visual Studio gegeben
  - Klassische sprachspezifische Funktion:
    - Syntaxhervorhebung, Autovervollständigung, Refactoring-Assistenten
  - Der Funktionsumfang ist schwächer im Vergleich zu Java oder C#
- **Debugger & Profiler**
  - Debugging- und Profiler-Werkzeuge unterstützen den Anwendungsentwickler dabei, eine Anwendung zur Laufzeit zu analysieren
  - Alle modernen Webbrowser beinhalten Werkzeuge zur Laufzeitanalyse
    - HTML & CSS-Inspektor, Javascript-Debugger, Netzwerkanalyse, Javascript-Profiling, Rendering-Analyse
  - Integration mit IDEs möglich
- **Metrik-Berechung mit „JSComplexity“ oder „JSMeter“**
  - Textuelle Komplexität nach Halstead, zyklomatische Komplexität nach McCabe
  - Lines of Code; Comment Lines of Code
  - Keine Komponentenmetriken / Keine objektorientierten Metriken („DIT“ / „CBO“)

Verknüpfung von Javascript und HTML

- **Technologie-Mix bei Javascript-lastigen Webanwendungen**
  - HTML + CSS + Javascript
  - Es existieren verschiedene Möglichkeiten diese Techniken zu kombinieren
  - Beispiele für Anti-Pattern:
    - Inline-Definition von Javascript-Code in ein HTML-Dokument
    - Nutzung der Style-Attributes
    - Einsatz von HTML-Eventhandler-Attributen im HTML-Dokument (onclick/ontouchstart)
- **Ziele: Trennung von Zuständigkeiten / Separierung der Techniken**
  - HTML-Dokument (Content):
    - Beinhaltet die darzustellenden Inhalte → Kein Javascript / Keine CSS-Styles
  - CSS (Visuelle Darstellung):
    - Die visuelle Darstellung wird in separaten CSS-Dateien definiert
  - Javascript (Verhalten):
    - Separate JS-Dateien / Event-Registrierung erfolgen im JS-Code

Verknüpfung von Javascript und HTML

```
<div class="container-fluid">
    <div class="row" id="page">
        <div class="col-md-4"
            id="sidebar"></div>
        <div class="col-md-8"
            id="main"></div>
    </div>
    <footer class="row">
        <p>M. Muster 2014</
p> </footer>
</div>
```

HTML

```
#sidebar {
 background-color: #F5F5F5;
 border-right: 1px solid #EEEEEE;
 overflow-x: hidden;
 overflow-y: auto;
 padding: 20px;
}
#main {
 padding: 5px 20px 5px 20px;
}
```

CSS

```
var Sidebar = ViewModel.extend({
 el: '#sidebar',
 template:
_.template(sidebarTemplate),
 events: {
  'submit #expense-form':'saveExpense'
 },
 saveExpense: function (ev) {
  //TBD
 }
});
```

Javascript

Vorteile:

- Reduzierung von Abhängigkeiten / Klar-definierte Abhängigkeiten
- Content, Darstellung und Verhalten können getrennt angepasst werden
- Leichtere Aufgabenverteilung bei der Durchführung eines Webprojektes

Kompilierung von JS

- **Alternative beim Umgang mit Javascript: „Kompilierung“**
  - Kompilierung von anderen Hochsprachen nach Javascript
  - Die „Probleme“ von Javascript werden so umgangen
  - Erweiterung der sprachlichen Mittel:
    - Typen, Schnittstellen, Klassen, Module und Generics
  - Varianten: CoffeeScript und TypeScript / ~ Google Web Toolkit (GWT)
- **Beispiel: Links = Input (TypeScript) / Rechts = Output (JS)**

```
class Student {
    fullname: string;
    constructor(public firstname, public middleinitial, public lastname) {
        this.fullname = firstname + " " + middleinitial + " " + lastname;
    }
}

interface Person {
    firstname: string;
    lastname: string;
}

function greeter(person: Person) {
    return "Hello, " + person.firstname + " " + person.lastname;
}

var user = new Student("Jane", "M.", "User");

document.body.innerHTML = greeter(user);
```

```
var Student = (function () {
    function Student(firstname, middleinitial, lastname) {
        this.firstname = firstname;
        this.middleinitial = middleinitial;
        this.lastname = lastname;
        this.fullname = firstname + " " + middleinitial + " " + lastname;
    }
    return Student;
})();

function greeter(person) {
    return "Hello, " + person.firstname + " " + person.lastname;
}

var user = new Student("Jane", "M.", "User");

document.body.innerHTML = greeter(user);
```

Quelle: http://www.typescriptlang.org/Playground/#tut=ex5

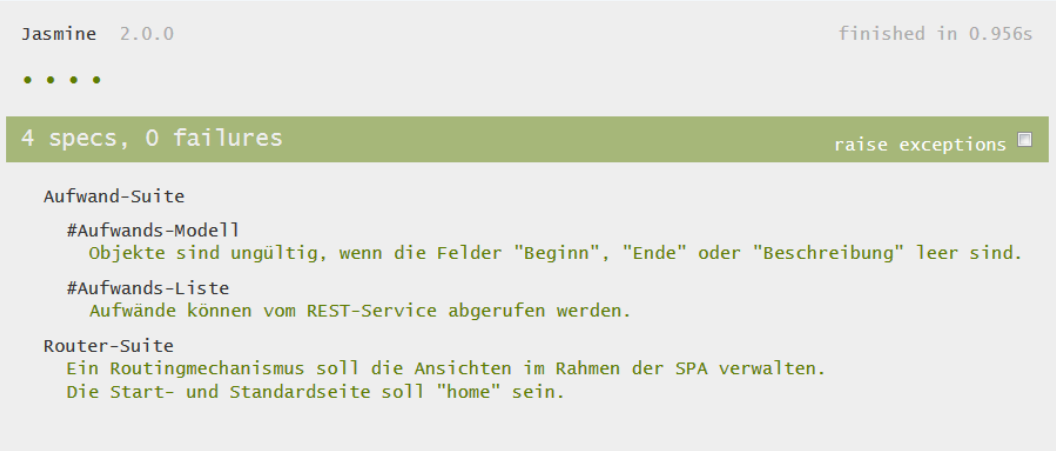
Testbarkeit von Javascript

- **„Testbarkeit bezeichnet die Fähigkeit, die Korrektheit des Softwareproduktes zu validieren“ (vgl. [Bal11, S. 116f])**
  - Die Realisierung einer testbaren JS-Anwendung beginnt bereits bei ihrem Architekturentwurf
    - Modulare Gestaltung & lose Kopplung → Separat testbare Einheiten
    - Umsetzung von Interaktionsmuster im Rahmen des Webclients (MVC/MVVM)
  - „Die Qualität der Architektur korreliert allerdings nicht notwendigerweise mit der Qualität des Endproduktes [...]. Architektur ist für die Qualität eines Systems notwendig, aber nicht hinreichend“ [Sta14, S. 38]
- **Testautomatisierung**
  - Im Rahmen der Wartbarkeit sind Regressionstests von besonderer Bedeutung
    - Weder die Zeit noch die Mittel zur manuellen Qualitätssicherstellung je Änderung
    - Die Automatisierung von Regressionstests hilft dabei Fehler aufgrund von Anpassungen zu identifizieren.
  - Continuous-Integration-Plattformen können diese Tests bei jeder Veränderung an der Codebasis automatisch ausführen

Testbarkeit von Javascript

Testautomatisierung bei Javascript-lastigen Webanwendungen

- Szenario:
  - Hoher Anteil an clientseitiger Präsentations- und Anwendungslogik
  - Im Rahmen von automatisierten Software-Tests werden häufig nur serverseitige Logiken validiert
- Unit-Testing-Frameworks, wie „Jasmine“ und „Qunit“, erlauben die Umsetzung von clientseitigen Tests
  - Ähnliche Vorgehensweise wie bei serverseitigen Unit -Testing-Frameworks
  - Die Ausführung erfolgt allerdings im Rahmen des Webbrowser als Zielplattform



Testbarkeit von Javascript

- **Testautomatisierung bei Javascript-lastigen Webanwendungen**
  - Problem: Keine Ausführung der Tests ohne den vollständigen Start eines Webbrowsers (→ „Continuous-Integration-Plattformen“)
  - Lösung: Verwendung von head-less Webbrowsern
    - PhantomJS ist beispielsweise eine Kommandozeilenversion der Webbrowserengine „WebKit“ (Apple Safari, Google Chrome und Opera-Browser)
    - Der vollständige Webbrowser-Stack samt DOM wird bereit gestellt
    - Die Ergebnisse können im Rahmen von Konsolenausgaben und Rückgabewerten ausgewertet werden
- **Testen der Oberfläche mit Selenium**
  - „Capture and Replay“-Werkzeug zur Aufzeichnungen von Interaktionen
  - Erlaubt die Validierung von DOM-Zuständen
  - Die „W3C WebDriver API“ erlaubt die Ausführung per Code

Schlussbetrachtung

- **Es wurden verschiedene Maßnahmen zur Entwicklung von wartbaren Javascript-lastigen Webanwendungen aufgeführt**
- **Ein wichtiger Aspekt ist die „Strukturierung“ einer JS-Anwendung**
  - Durch Pakete und Komponenten entstehen abgeschlossene analysierbare Abstraktionseinheiten
  - Vereinfacht die Wartung, da leichter ein mentales Modell der Anwendung aufgebaut werden kann
  - Komponenten können losgelöst qualitätsgesichert und gewartet werden
    - Aufgrund der Abgeschlossenheit solcher Komponenten verbessert sich die Änderungsstabilität und Testbarkeit der Anwendung
    - Nachteil: AMD-Format ist trotz der Verbreitung als proprietär zu bezeichnen
- **Programmierrichtlinien helfen dabei den Umgang mit Javascript zu normieren und die Qualität konstruktiv zu verbessern**
  - Überprüfung bei statischer Quellcodeanalyse

Schlussbetrachtung

- **Klassen, Paketstrukturen und Schnittstellen vermutlich erst nach ES6**
  - Die Schlüsselwörter wurden in der ES5 reserviert
  - Heutige Möglichkeit: Zwischensprachen wie „TypeScript“
- **Der Softwareentwicklungs- und der Wartungsprozess werden durch eine Vielzahl von Werkzeugen unterstützt**
  - Debugger; Profiler; Testing-Frameworks; Quellcodeanalyse
- **Alle Aspekte der Wartbarkeit gemäß ISO/IEC 25010 und 9126-1 können mit einer Javascript-lastigen Webanwendung erreicht werden**
  - Höherer Aufwand als bei klassischen Webprojekten
  - Eine klare Trennung der Zuständigkeiten beim Einsatz der Webtechniken ist zu berücksichtigen
  - → Keine Degeneration im Rahmen der Wartung zulassen

Quellen

- [Hof08] Hoffmann, Dirk W. (2008). *Software-Qualität.* Heidelberg, DE: Springer.
- [Cro08] Crockford, Douglas (2008). *JavaScript: The Good Parts: Working with the Shallow Grain of JavaScript.* Sebastopol, CA, USA: O'Reilly Media.
- [Bal11] Balzert, Helmut (2011). *Lehrbuch der Softwaretechnik - Entwurf, Implementierung, Installation und Betrieb.* Heidelberg, DE: Spektrum Akademischer Verlag.
- [Sta14] Starke, Gernot (2014). *Effektive Softwarearchitekturen: Ein praktischer Leitfaden.* München, DE: Carl Hanser Verlag.
- [Hal12] Hales, Wesley (2012). *HTML5 and JavaScript Web Apps.* Sebastopol, CA, USA: O'Reilly Media.
- [Zak12] Zakas, Nicholas C. (2012). *Maintainable JavaScript.* Sebastopol,

Vielen Dank!

Inhouse-Schulungen

Wir bieten Inhouse-Schulungen und Beratung durch unsere IT-Experten und –Berater.

Schulungsthemen

- Softwarearchitektur (OOD)
- Requirements Engineering (OOA)
- Nebenläufige & verteilte Programmierung

Gerne konzipieren wir auch eine individuelle Schulung zu Ihren Fragestellungen.

Sprechen Sie uns an!
Tel. 0231/61 804-0, info@W3L.de

W3L-Akademie

In Zusammenarbeit mit der Fachhochschule Dortmund bieten wir

zwei Online-Studiengänge

- B.Sc. Web- und Medieninformatik
- B.Sc. Wirtschaftsinformatik

und 7 Weiterbildungen im IT-Bereich an.

Besuchen Sie unsere Akademie!
http://Akademie.W3L.de