

A thick red vertical bar is positioned on the left side of the slide, extending from the top to the bottom.

Webbasierte Anwendungen

Qualitätssicherung

Prof. Dr. Ludger Martin

Gliederung

- ◆ Softwarequalität
- ◆ Testen
- ◆ Testfälle
- ◆ Qunit
 - ★ Web-Browser
 - ★ Node.js
- ◆ Selenium
- ◆ Zusammenfassung

Einleitung

- ◆ Qualität kommt nicht von selbst
- ◆ Modelle, die Qualitätssicherung erleichtern:
 - ★ **EN ISO 9001**: Anforderungen an ein Qualitätsmanagementsystem
 - ★ **CMMI**: Prozessmodell zur Beurteilung und Verbesserung der Qualität von Produkt-Entwicklungsprozessen
 - ★ **ISO/IEC 9126**: ein Modell, um Softwarequalität sicherzustellen

Softwarequalität

Merkmale für Softwarequalität (ISO/IEC 9126):

- ◆ Funktionalität
- ◆ Zuverlässigkeit
- ◆ Benutzbarkeit
- ◆ Effizienz
- ◆ Änderbarkeit
- ◆ Übertragbarkeit




Ausschließlich
für
Produktqualität

Softwarequalität

Merkmale:


- ◆ **Funktionalität**
- ◆ Zuverlässigkeit
- ◆ Benutzbarkeit
- ◆ Effizienz
- ◆ Änderbarkeit
- ◆ Übertragbarkeit

- 
- ★ Angemessenheit
 - ★ Richtigkeit
 - ★ Interoperabilität
 - ★ Sicherheit
 - ★ Konformität

Softwarequalität

Merkmale:


- ◆ Funktionalität
- ◆ **Zuverlässigkeit**
- ◆ Benutzbarkeit
- ◆ Effizienz
- ◆ Änderbarkeit
- ◆ Übertragbarkeit

- 
- ★ Reife
 - ★ Fehlertoleranz
 - ★ Robustheit
 - ★ Wiederherstellbarkeit
 - ★ Konformität

Softwarequalität

Merkmale:

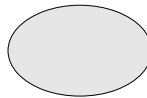
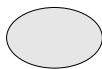
- ◆ Funktionalität
- ◆ Zuverlässigkeit
- ◆ **Benutzbarkeit**
- ◆ Effizienz
- ◆ Änderbarkeit
- ◆ Übertragbarkeit

- 
- ★ Verständlichkeit
 - ★ Erlernbarkeit
 - ★ Bedienbarkeit
 - ★ Attraktivität
 - ★ Konformität

Softwarequalität

Merkmale:


- ◆ Funktionalität
- ◆ Zuverlässigkeit
- ◆ Benutzbarkeit
- ◆ **Effizienz**
- ◆ Änderbarkeit
- ◆ Übertragbarkeit



Softwarequalität

Merkmale:

- ◆ Funktionalität
- ◆ Zuverlässigkeit
- ◆ Benutzbarkeit
- ◆ Effizienz
- ◆ Änderbarkeit
- ◆ Übertragbarkeit

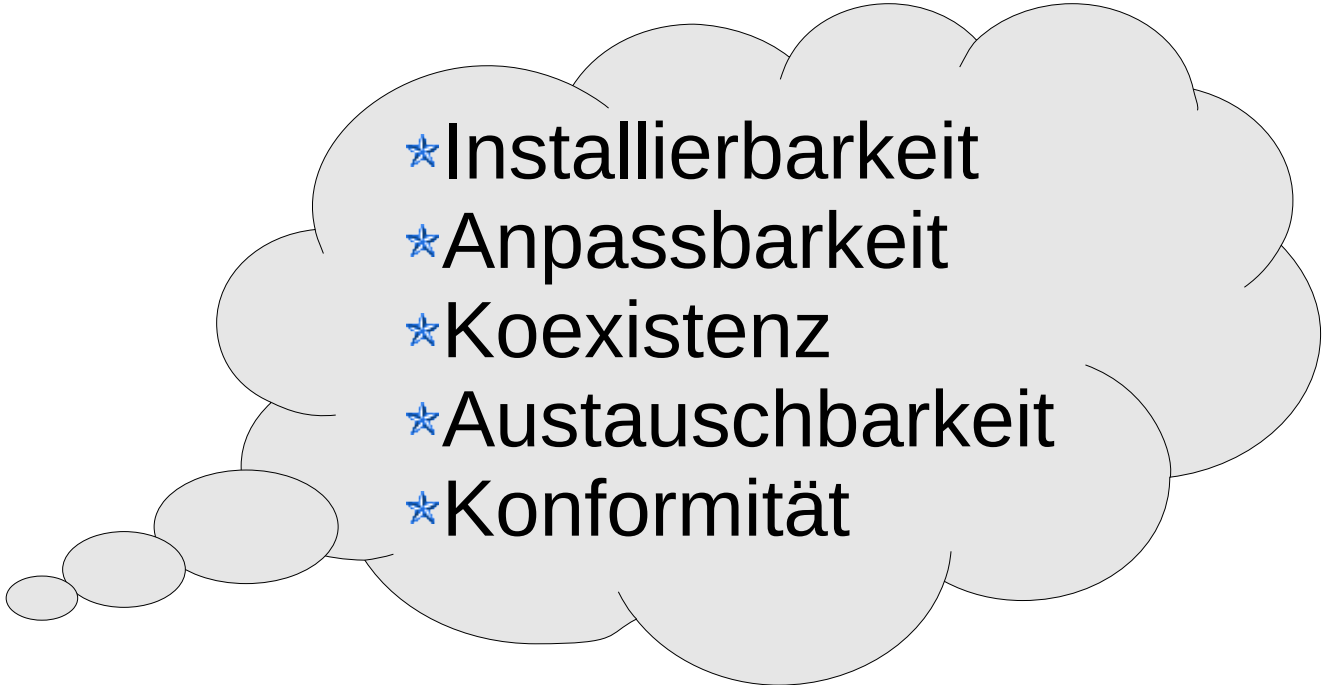


- ★ Analysierbarkeit
- ★ Modifizierbarkeit
- ★ Stabilität
- ★ Prüfbarkeit
- ★ Konformität

Softwarequalität

Merkmale:

- ◆ Funktionalität
- ◆ Zuverlässigkeit
- ◆ Benutzbarkeit
- ◆ Effizienz
- ◆ Änderbarkeit
- ◆ Übertragbarkeit



- ★ Installierbarkeit
- ★ Anpassbarkeit
- ★ Koexistenz
- ★ Austauschbarkeit
- ★ Konformität

Softwarequalität

◆ Funktionalität:

- ★ Richtigkeit

◆ Zuverlässigkeit:

- ★ Fehlertoleranz
- ★ Robustheit

◆ Änderbarkeit:

- ★ Prüfbarkeit

Klassen müssen
ausführlich
getestet werden!

Tests in verschiedenen
Einsatzgebieten!

Testfälle müssen
mitgeliefert werden!

Testen

- ◆ Testen (Beck 1994):
 - ★ Finden von Mängeln (engl. **failure**). Testen, ob ein erwartetes Ergebnis erscheint
 - ★ Ein Fehler (engl. **error**) ist gravierender. Ist nicht testbar.
- ◆ Testen der kleinsten Einheit (Unit Test)
- ◆ Verschiedene Vorgehensweisen beim Testen

Testen

Black-Box-Test

- ◆ Übereinstimmung eines Softwaresystems mit seiner Spezifikation überprüfen
- ◆ Das zu testende System wird als Ganzes betrachtet
- ◆ Nur nach außen sichtbares Verhalten fließt in Test ein
- ◆ Test ohne Kenntnisse über die interne Funktionsweise
- ◆ Separates Team zur Entwicklung der Tests

Testen

Black-Box-Test

◆ Vorteile

- ★ Fehler gegenüber der Spezifikation aufdecken
- ★ Überprüfung der Spezifikation

◆ Nachteile

- ★ Nicht geeignet, um fehlerhafte Stelle zu finden
- ★ Testfälle einer unzureichenden Spezifikation sind unbrauchbar
- ★ Bei Implementierung zusätzlich eingeführte Funktionen werden nur durch Zufall getestet
- ★ Soziale Aspekte (Test-Team)
- ★ Großer organisatorischer Aufwand

Testen

Black-Box-Test

◆ Finden der Testfälle

★ **Grenzwerte und spezielle Werte**

★ **Entscheidungstabellen**

Beim Entwurf von Programmen eingesetzt, um komplexe Abhängigkeiten zwischen mehreren Bedingungen übersichtlich und vollständig darzustellen

★ **Zustandsbezogene Tests**

Konform zu UML Zustandsdiagrammen

★ **Use Case Test**

Tests anhand von Anwendungsfällen

Testen

White-Box-Test

- ◆ Tests mit Kenntnisse über die innere Funktionsweise des zu testenden Systems
- ◆ Betrachtung des Quelltextes gestattet
- ◆ Große Anzahl von Testfällen nötig
- ◆ Besonders geeignet um Fehler zu lokalisieren
- ◆ Tests durch dieselben Entwickler

Testen

White-Box-Test

♦ Vorteile

- ★ Testen von Teilkomponenten und der internen Struktur
- ★ Lokalisieren von Fehlern
- ★ Einfacher in der Durchführung
- ★ Geringer organisatorischer Aufwand

♦ Nachteile

- ★ Erfüllung der Spezifikation nicht überprüfbar
- ★ Eventuell testen „um den Fehler herum“

Testen

White-Box-Test

◆ Finden der Testfälle

- ★ **Anweisungsüberdeckung**

Jede Anweisungen einmal durchlaufen

- ★ **Kanten- /Zweigüberdeckung**

Durchlauf aller möglichen Kanten von Verzweigungen des Kontrollflusses

- ★ **Bedingungsüberdeckung**

Testen jeder Bedingung und deren Auswertung

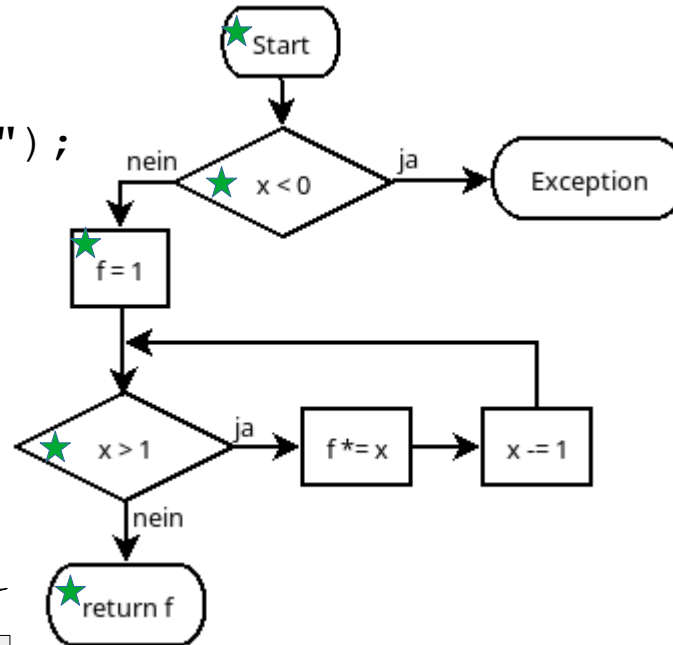
- ★ **Pfadüberdeckung**

Testen aller Pfade, die genommen werden können

Testen

White-Box-Test

```
function fakultaet(x) {  
  if (x < 0) {  
    throw new Error("x negativ");  
  }  
  let f = 1;  
  while (x > 1) {  
    f *= x;  
    x -= 1;  
  }  
  return(f);  
}
```

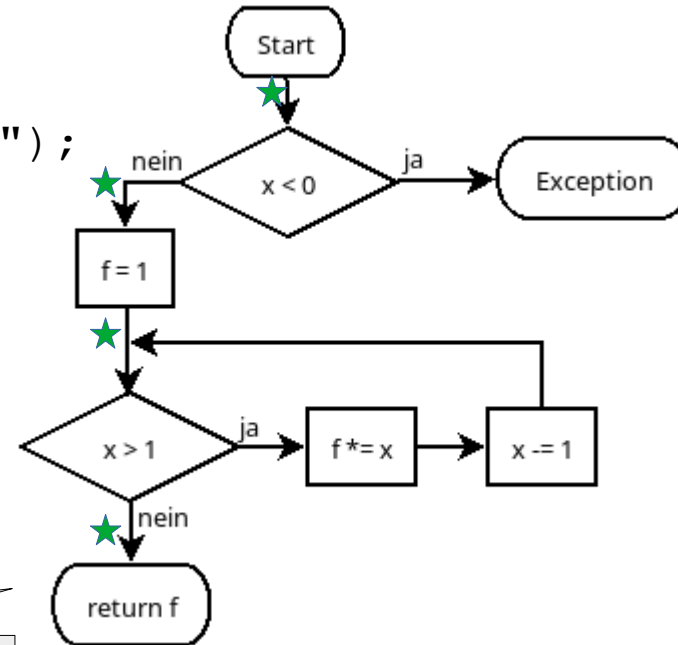


Anweisungsüberdeckung
fakultaet(1) – 3 von 5 Anweisungen

Testen

White-Box-Test

```
function fakultaet(x) {  
  if (x < 0) {  
    throw new Error("x negativ");  
  }  
  let f = 1;  
  while (x > 1) {  
    f *= x;  
    x -= 1;  
  }  
  return f;  
}
```

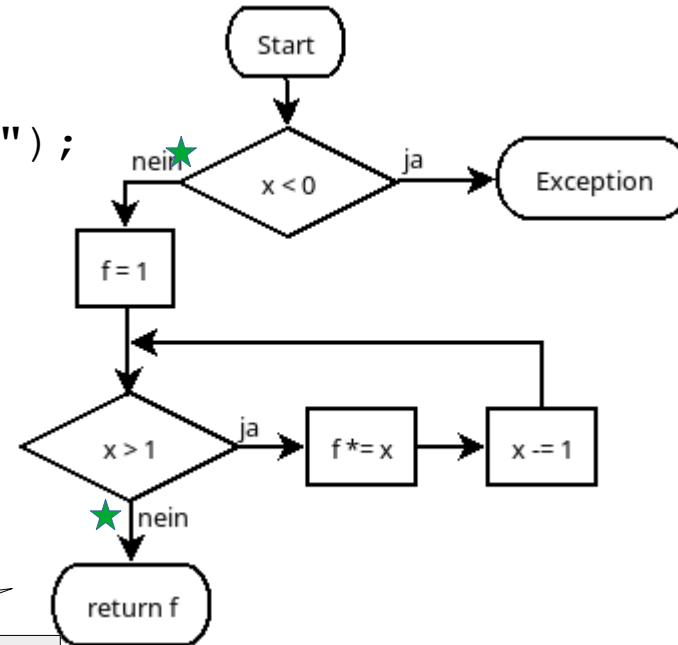


Kanten- /Zweigüberdeckung
fakultaet(1) – 4 von 8 Kanten

Testen

White-Box-Test

```
function fakultaet(x) {  
  if (x < 0) {  
    throw new Error("x negativ");  
  }  
  let f = 1;  
  while (x > 1) {  
    f *= x;  
    x -= 1;  
  }  
  return f;  
}
```

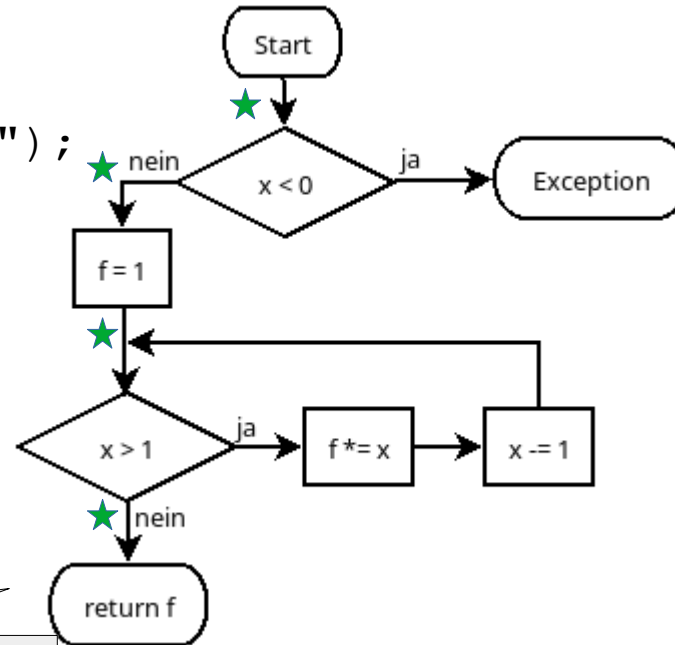


Bedingungsüberdeckung
fakultaet(1) – 2 von 4 Bedingungsauswertungen

Testen

White-Box-Test

```
function fakultaet(x) {  
  if (x < 0) {  
    throw new Error("x negativ");  
  }  
  let f = 1;  
  while (x > 1) {  
    f *= x;  
    x -= 1;  
  }  
  return f;  
}
```



Pfadüberdeckung

fakultaet(1) – 2 von 4 Bedingungsauwertungen

Testen

Grey-Box-Test

- ◆ Vorteile von Black-Box-Test und White-Box-Test miteinander verbinden
- ◆ Test durch dieselben Entwickler
- ◆ Anfänglich Unkenntnis über Interna des zu testenden System („test first“)
- ◆ Dann Testen von Teilsystemen
- ◆ Hohe Disziplin erforderlich
- ◆ Kein vollwertiger Ersatz für Black-Box-Test

Testen

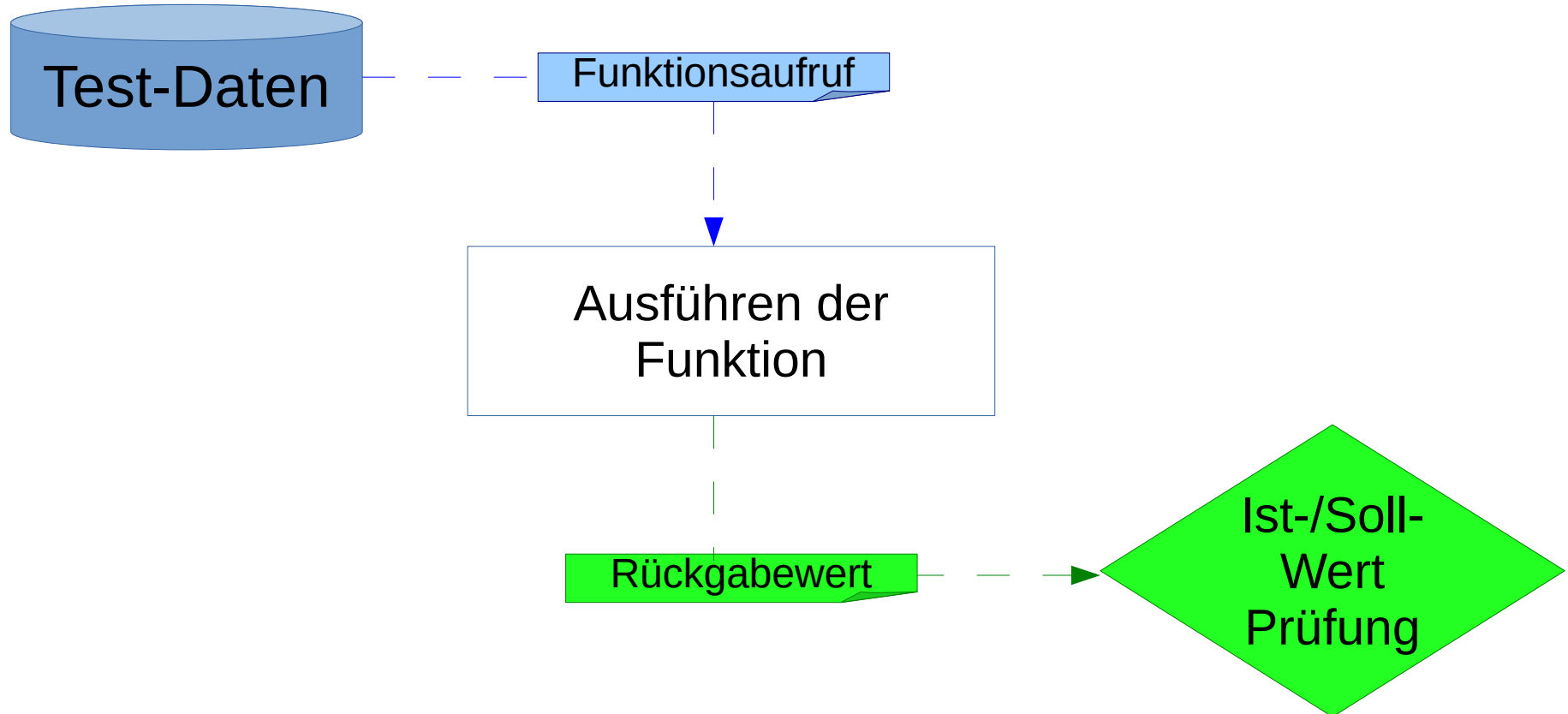
Regressionstest

- ◆ Wiederholen aller oder einer Teilmenge aller Testfälle
- ◆ Aufspüren von Nebenwirkungen von Modifikationen
 - ★ Software-Pflege
 - ★ Software-Änderung
 - ★ Software-Korrektur
- ◆ Ein Wiederauftreten eines Fehlers zu vermeiden
- ◆ Automatische Ausführung

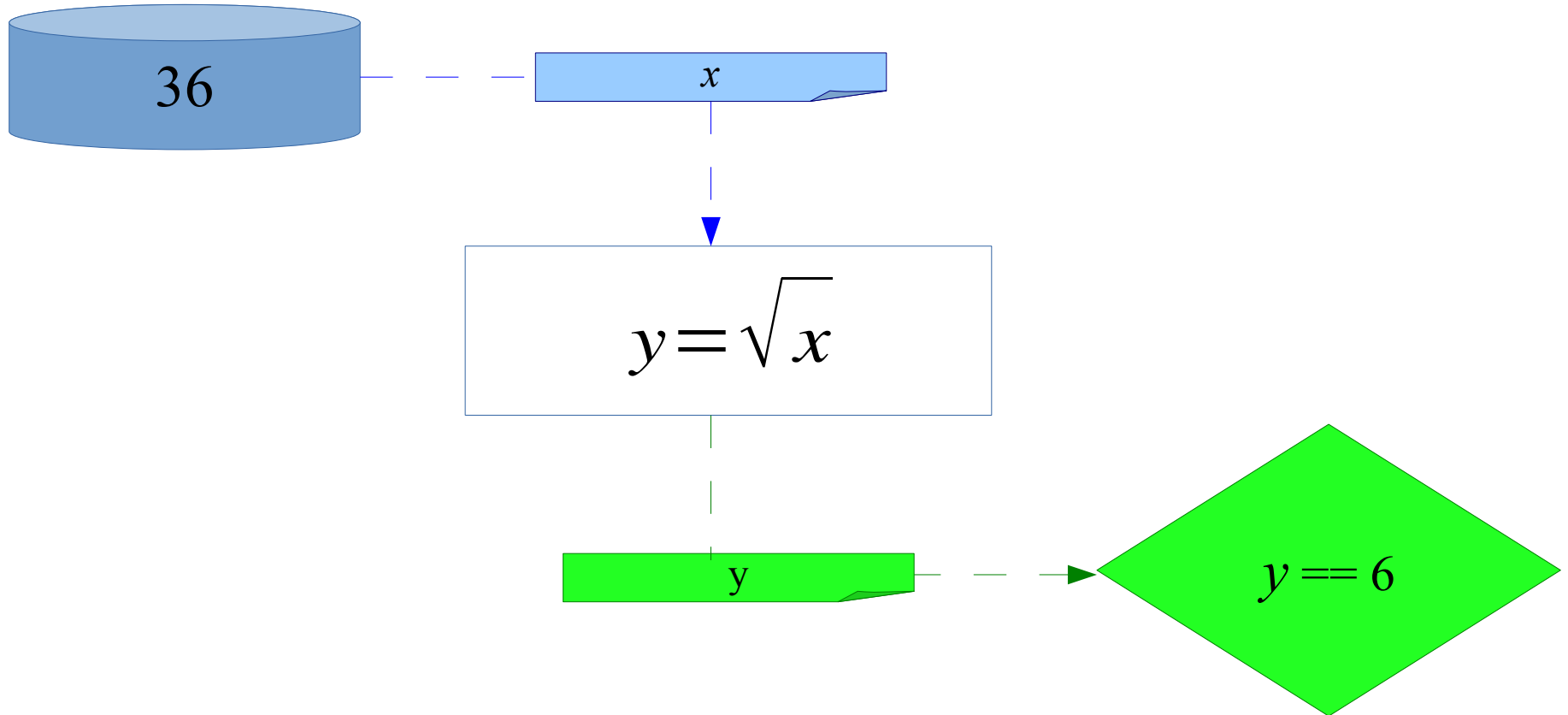
Testfälle

- ◆ Ermitteln der Funktion, die getestet werden soll
 - ★ Black-Box-Test
 - ★ White-Box-Test
 - ★ Grey-Box-Test
- ◆ Ggf. Test-Zustand für Funktion herstellen
- ◆ Ermitteln der Eingabedaten für diese Funktion
- ◆ Definition eines Soll-Ergebnisses
- ◆ Ein Mangel tritt auf, wenn **Ist-Ergebnis** nicht gleich **Soll-Ergebnis** ist.

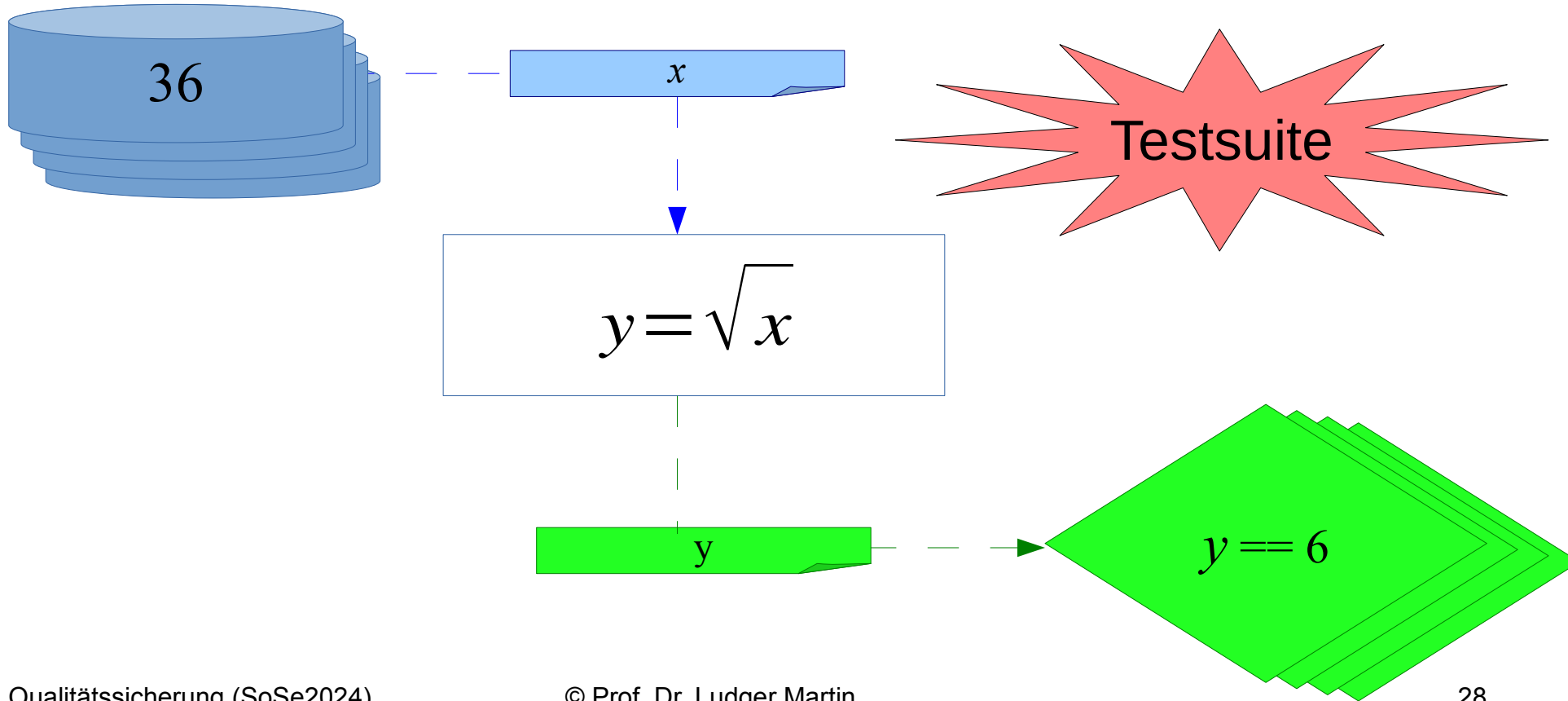
Testfälle



Testfälle



Testfälle



Testfälle

- ◆ **Richtigkeit:** Testfälle, um richtiges Arbeiten zu zeigen
 $6 = \sqrt{36}$
- ◆ **Fehlertoleranz:** Testfälle auch für Fehlerfälle schreiben
 $? = \sqrt{-1}$
- ◆ **Robustheit:** Testfälle für „unsinnige“ Eingaben
 $? = \sqrt{'a'}$
- ◆ **Prüfbarkeit:** Testfälle müssen als Test-Suite der Komponente mitgeliefert werden

QUnit


- ◆ QUnit is a powerful, easy-to-use JavaScript unit testing framework.
- ◆ Unter anderem genutzt bei jQuery, jQuery UI und jQuery Mobile
- ◆ Erlaubt das Testen von Klassen, Methoden und Funktionen.
- ◆ Strikte Trennung von Produktionscode und Testcode

Qunit

Web-Browser

◆ Erstellen von Testfällen:

```
<!DOCTYPE html><html lang="de">
<head>
  <meta charset="utf-8">
  <title>Qunit</title>
  <link rel="stylesheet" href="qunit.css"/>
</head>
<body>
  <div id="qunit"></div>
  <div id="qunit-fixture"></div>
  <script src="qunit.js"></script>
  <script src="zu_testende_Klasse.js"></script>
  <script>
    // Hier werden die Tests spezifiziert
  </script>
</body>
</html>
```



Bibliothek von
QUnit



Zu testende Klasse
oder Funktionen

QUnit

Web-Browser

◆ Aufruf der Tests durch die erstellte HTML-Datei

QUnit

☐ Hide passed tests ☐ Check for Globals ☐ No try-catch

Filter: Module:

QUnit 2.5.0; Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0

1 tests completed in 11 milliseconds, with 0 failed, 0 skipped, and 0 todo.
1 assertions of 1 passed, 0 failed.

1. Animal (1) Rerun 1 ms

1. getter name @ 0 ms

Source: @http://localhost/wba/qunitjs.html:17:3

Qunit

Node.js

- ◆ Datei `package.json` anlegen/bearbeiten:

```
{  
  "scripts": {  
    "test": "qunit"  
  }  
}
```



Module `qunit`
installieren!

- ◆ Testfälle in Verzeichnis `test` ablegen



Zu testende Klasse
oder Funktionen als **Modul**

- ◆ Anfang eines Testfalls

```
const add = require('../zu_testende_Klasse.js');
```

Qunit

Node.js

- ◆ **Aufruf mit**
`npm test`
- ◆ **Ausgabe:**
`TAP version 13`
`ok 1 Animal`
`1..1`
`# pass 1`
`# skip 0`
`# todo 0`
`# fail 0`

JUnit

◆ Erstellen von Testfällen:

Für jeden Test
diese Methode aufrufen.

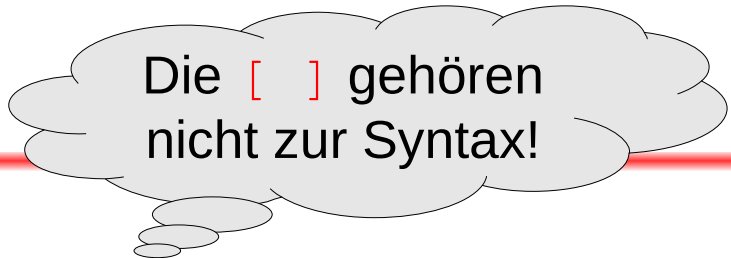
Name des Tests

```
JUnit.test('Animal', function (assert) {  
    let a = new Animal('Hydra', 4);  
    assert.equal(a.name, 'Hydra', 'getter name');  
});
```

Prüfung auf Gleichheit

Name der
Zusicherung

JUnit



Die [] gehören nicht zur Syntax!

◆ Zusicherungsmethoden

- ★ `equal(actual, expected [, message])`
Meldet einen durch `message` bezeichneten Fehler, falls die beiden Variablen `expected` und `actual` nicht identisch sind
- ★ `notEqual(actual, expected [, message])`
Prüfung auf Ungleichheit
- ★ `deepEqual(actual, expected [, message])`
Tiefer rekursiver Vergleich
- ★ `strictEqual(actual, expected [, message])`
Strikter Typ und Wertvergleich
- ★ ...

QUnit

Die [] gehören nicht zur Syntax!

- ◆ Prüfen, dass eine Anweisung eine Exception wirft
`throws(block [, expected] [, message])`

- ◆ Beispiel

```
QUnit.test('Animal with negative age.',  
  function (assert) {  
    assert.throws(function () {  
      new Animal('Kitty', -5);  
    }, Error, 'thrown exception');  
  }  
);
```

Wenn hier keine Exception
geworfen wird, wird ein
Fehler gemeldet!

Qunit

Testinventar

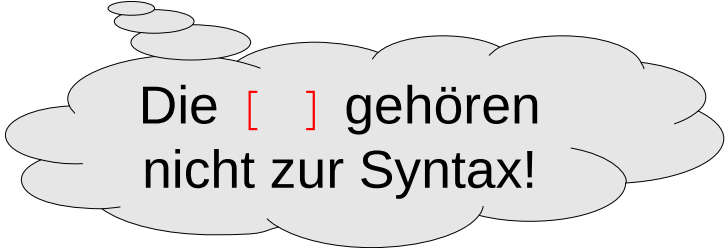
- ◆ Die Welt muss vor dem Test in einen definierten Zustand versetzt werden.
- ◆ Nach dem Test ist der ursprüngliche Zustand zurückzusetzen.
- ◆ Initialisierungscode kann mehrfach verwendet werden.
- ◆ Vor jeder Testmethode wird die Methode namens `beforeEach()` ausgeführt.
- ◆ Unabhängig davon, ob Test erfolgreich war oder nicht, wird die Methode `afterEach()` ausgeführt.

Qunit

Testinventar

- ◆ Es muss ein Modul definiert werden, in dem die Methoden `beforeEach()` und `afterEach()` definiert werden:

```
QUnit.module(name [, hooks ])
```




Die `[]` gehören nicht zur Syntax!

Qunit

Testinventar

◆ Beispiel:

```
QUnit.module('Module Animal', {  
    beforeEach: function () {  
        this.a = new Animal('Nemo', 2);  
    }  
});
```



Mit `this` können Attribute
im Modul gespeichert werden.

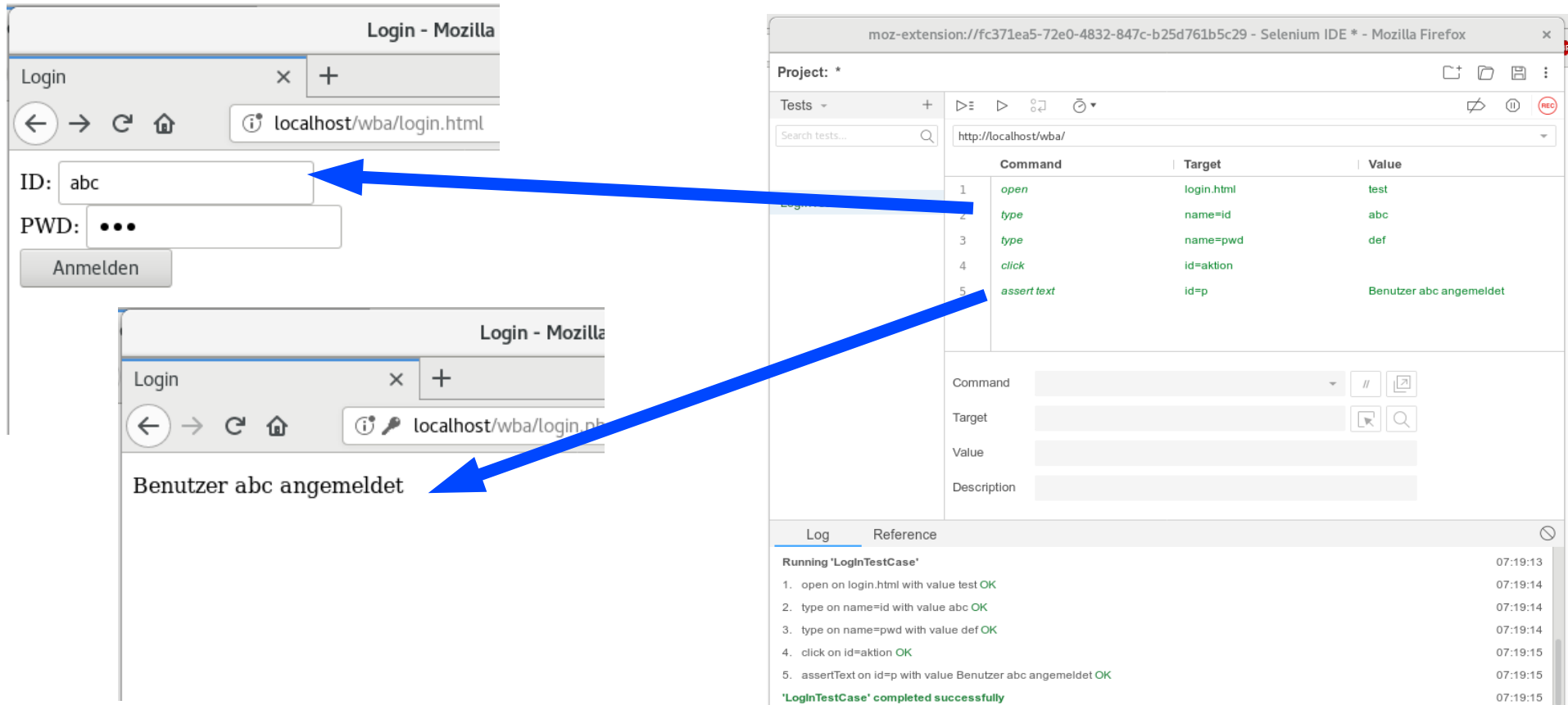
◆ Nun kann auf das zuvor instanzierte Objekt zugegriffen werden.

```
QUnit.test('Animal', function (assert) {  
    assert.equal(this.a.name, 'Nemo',  
        'getter name');  
});
```


Selenium IDE

- ◆ <https://www.selenium.dev/selenium-ide/>
- ◆ Testframework für Web-Anwendungen
- ◆ Interaktionen mit einer Web-Anwendung aufzeichnen
- ◆ Automatisiertes Ausführen der Interaktion
 - Erstellen von Testfällen
- ◆ Mit Selenium IDE können interaktiv Testfälle erstellt werden
- ◆ Test-Suites können angelegt werden

Selenium IDE



The image displays the Selenium IDE interface within a Mozilla Firefox browser window. The browser shows a login page at `localhost/wba/login.html` with input fields for ID (abc) and PWD (def), and a button labeled "Anmelden".

The Selenium IDE interface shows a test case named "LoginTest" with the following steps:

Command	Target	Value
open	login.html	test
type	name=id	abc
type	name=pwd	def
click	id=aktion	
assert text	id=p	Benutzer abc angemeldet

The "Log" pane at the bottom shows the execution progress:

- Running "LoginTest"
- 1. open on login.html with value test OK
- 2. type on name=id with value abc OK
- 3. type on name=pwd with value def OK
- 4. click on id=aktion OK
- 5. assertText on id=p with value Benutzer abc angemeldet OK
- "LoginTest" completed successfully

Zusammenfassung

- ◆ „test-first“-Methode (Grey-Box-Test)
- ◆ Bestandteil von extreme Programming
- ◆ Zuerst geschriebene Tests lassen sich zunächst nicht ausführen
- ◆ Während des Schreibens von Tests wird über die Schnittstelle des Objekts nachgedacht
- ◆ Beim Codeschreiben muss nur noch die Implementierung berücksichtigt werden
- ◆ Oft kürzerer Zyklus zwischen Testen und Implementieren

Zusammenfassung

◆ „test-first“-Vorgehen

- ★ Test schreiben, der den Produktionscode motiviert
 - Es existiert immer ein Test für jede Funktionalität
 - Jede Bedingung im Pflichtenheft muss getestet werden
 - 😊 Beim Formulieren eines Tests setzt sich Programmierer mit Anforderungen auseinander
- ★ Nur minimal benötigten Produktionscode schreiben, der vom Test verlangt wird
 - Jede zusätzliche Zeile kann unnötige Fehler enthalten

Zusammenfassung

- ◆ Tests erhöhen Vertrauen in Code
- ◆ Auswirkungen von Änderungen schnell erkennbar
- ◆ Testen schließt nicht aus, dass noch Fehler vorhanden sind.
- ◆ Nicht entscheidbar, wann genug getestet wurde

Literatur

- ◆ Beck, Kent: Extreme Programming - Das Manifest, Addison-Wesley, 2000
- ◆ Bergmann, Sebastian: Professionelle Softwareentwicklung mit PHP 5, dpunkt.verlag, 2005
- ◆ QUnit: <http://qunitjs.com/>
- ◆ Selenium: <https://www.selenium.dev/>