

Multimedia Engineering II

05 Debugging und Testen

Johannes Konert



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences



Agenda

- **Wiederholung und 1,5h Aufgabe**
- **Was ist ein Bug?**
- **Rollen bei der Softwareentwicklung**
- **Beschreiben eines Bugs**
- **Debug-Vorgehen: 6 Stufenplan**
 - 3. Minimierung
 - 4. Hypothesen
 - 5. Analysewerkzeuge
Code editieren
 - 6. Testen und Testwerkzeuge
- **Abschließende kurze Übung**
- **Ihre zusammenfassenden Fragen**
- **Ausblick**

Agenda

- **Wiederholung und 1,5h Aufgabe**
- **Was ist ein Bug?**
- **Rollen bei der Softwareentwicklung**
- **Beschreiben eines Bugs**
- **Debug-Vorgehen: 6 Stufenplan**
 - 3. Minimierung
 - 4. Hypothesen
 - 5. Analysewerkzeuge
Code editieren
 - 6. Testen und Testwerkzeuge
- **Abschließende kurze Übung**
- **Ihre zusammenfassenden Fragen**
- **Ausblick**

Wiederholung

- Für welche zwei unterschiedlichen Programmier-Ziele werden `app.route(...)` und `express.Router()` verwendet?

`app.route()`

HTTP Methoden für gleiche Route

Konzept des Fluent Interfaces

```
app.route('/:id')
  .get(function(req, res) {
    res.send('...');
  })
  .put(function(req, res) {
    res.send('...');
  })
  .delete(function(req, res) {
    res.send('...');
  });
```

`express.Router()`

Neue Instanz mit Middleware und Router

Konzept der Komposition

```
var retweets = express.Router();

retweets.use(function (req, res, next) {
  // do..
  next();
});

retweets.get('/', function(req, res) {
  res.send('list of retweets');
});

retweets.delete('/:id', function(req, res) {
  res.send('deleted retweet');
});
```

```
app.use('/:id/retweets', retweets);
```

Wiederholung

- Wann bietet es sich an, **nodemon** statt **node** zu verwenden?
- **Modul nodemon bietet**
 - Verwendung wie node zum Ausführen von bspw. app.js Code
 - Überwacht alle geladenen Ressourcen
 - Bei Änderung erfolgt automatischer Neustart
- **Installation**
 - **npm install --g nodemon**
 - Parameter **--g** für globale Installation außerhalb des Projektordners

Wiederholung

- Was ist der Unterschied zwischen den Methoden `app.use(...)` und `app.all(...)` ?

.use(prefix, handler)

- trifft auf alle HTTP Methoden zu
- Handler ist Function
- **prefix ist ein (optionaler) URL-Route Präfix**
- **Typischerweise für Middlewares benutzt**

.all(pattern, handler)

- trifft auf alle HTTP Methoden zu
- Handler ist Function
- **pattern ist ein Muster (String oder RegExp)**
- **Typischerweise für Request/Response-Bearbeitung aller HTTP-Anfragen benutzt**

Zusammenfassende Fragen und Wiederholung heute und nächste Woche



Aufgabe:

1. Sie schreiben die zusammenfassenden Fragen + Antworten selbst auf
2. Nutzen Sie dazu die Moderationskarten
 - eine Seite Frage
 - Andere Seite Antwort(en) + ggf. Foliennummer v. heute
3. Von jedem am Ende mindestens eine Karte bei mir abgeben



4. Nächste Woche ziehen Sie 3x aus diesem Stapel

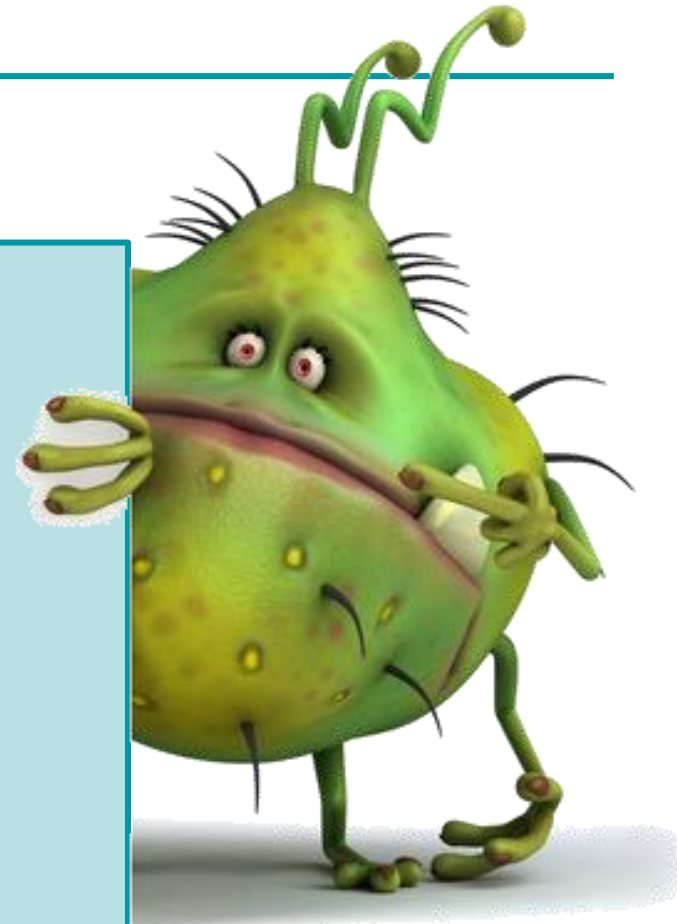
Agenda

- Wiederholung und 1,5h Aufgabe
- Was ist ein Bug?
- Rollen bei der Softwareentwicklung
- Beschreiben eines Bugs
- Debug-Vorgehen: 6 Stufenplan
 3. Minimierung
 4. Hypothesen
 5. Analysewerkzeuge
Code editieren
 6. Testen und Testwerkzeuge
- Abschließende kurze Übung
- Ihre zusammenfassenden Fragen
- Ausblick

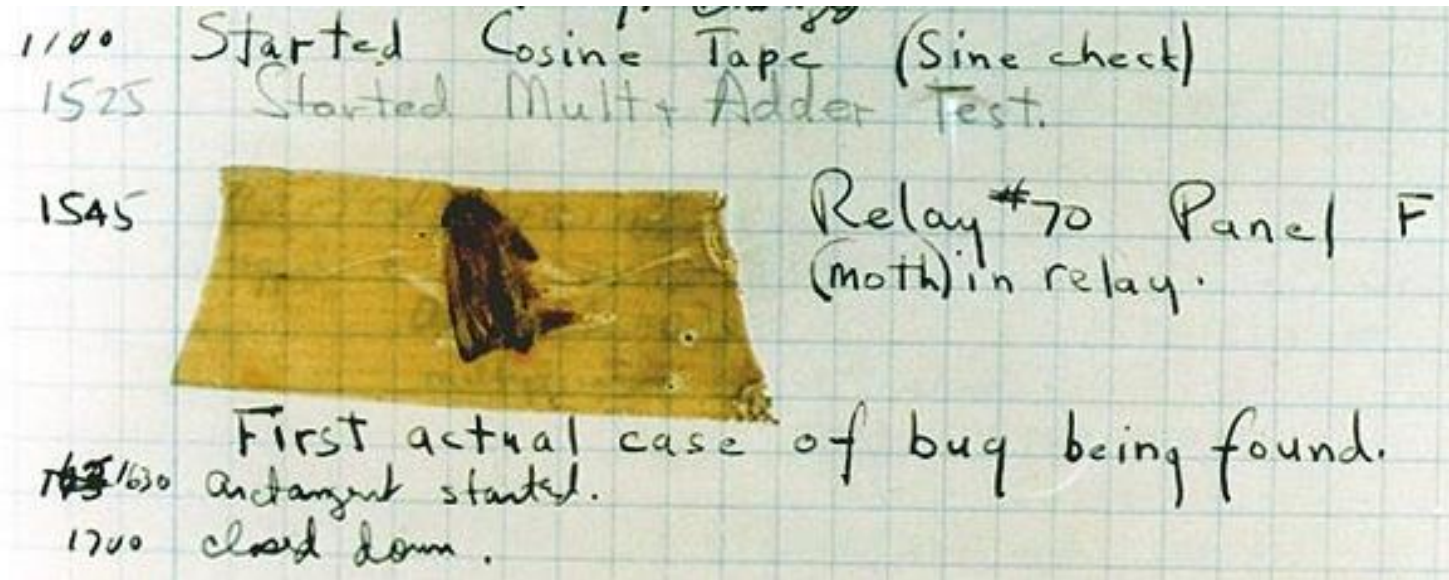
Was ist ein Bug?

Bug

- **Nichterfüllung einer Anforderung**
(EN ISO 9000:2005)
- **Abweichung** eines tatsächlichen Verhaltens (**IST**) vom erwarteten (**SOLL**)



Der erste Bug?



- 1947 eine Motte im Relay einer Rechenmaschine
- Wird Grace Hopper zugeschrieben

Tatsächlich jedoch

- u.a. Thomas Edison (1878) schreibt Knackgeräusche im Telefon einem Käferknabbern zu

Agenda

- Wiederholung und 1,5h Aufgabe
- Was ist ein Bug?
- Rollen bei der Softwareentwicklung
- Beschreiben eines Bugs
- Debug-Vorgehen: 6 Stufenplan
 - 3. Minimierung
 - 4. Hypothesen
 - 5. Analysewerkzeuge
Code editieren
 - 6. Testen und Testwerkzeuge
- Abschließende kurze Übung
- Ihre zusammenfassenden Fragen
- Ausblick

Rollen bei der Softwareentwicklung als Informatiker/in

Aufgabe	Rolle
Design	Architekt/in
Coding	Ingenieur/in
Testen	Vandale/in
Debugging	Detektiv/in

Zeitaufwand bei Erfüllen der Rollen

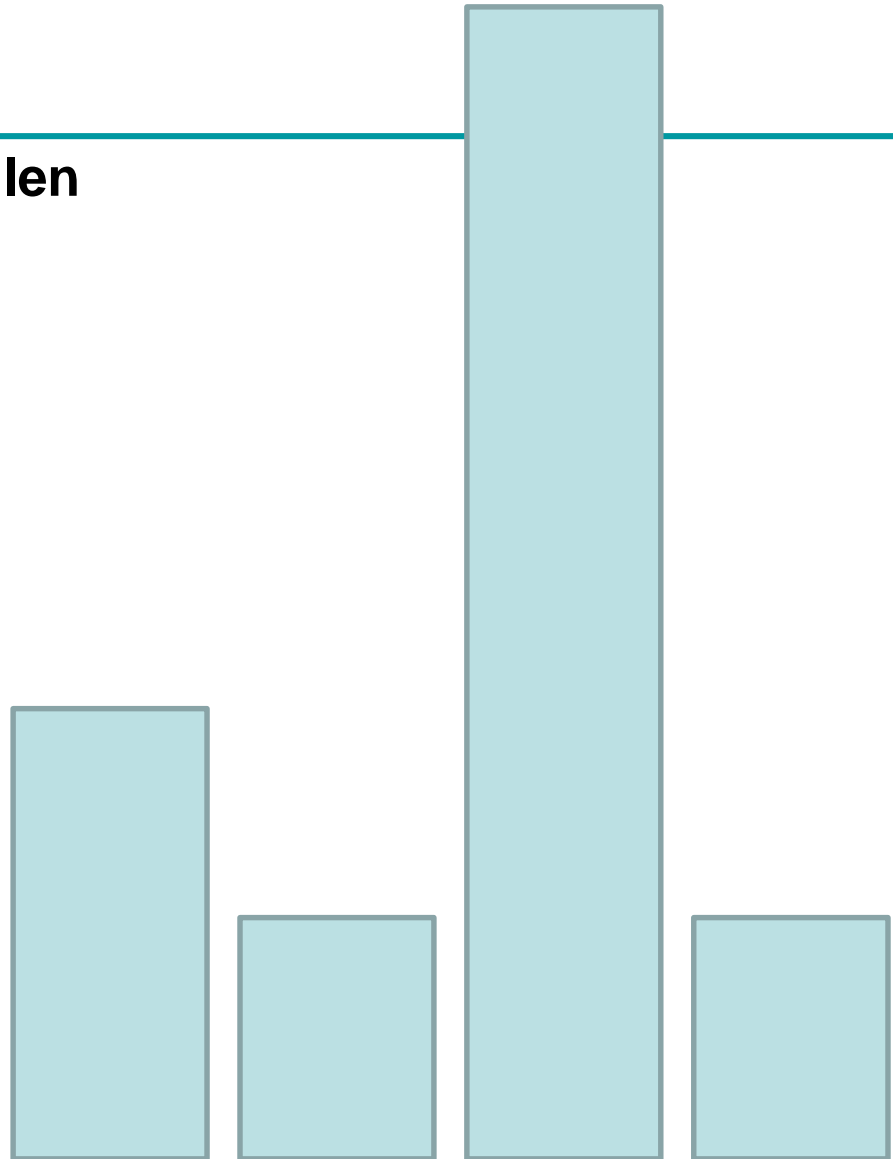


Design

Coding

Testen

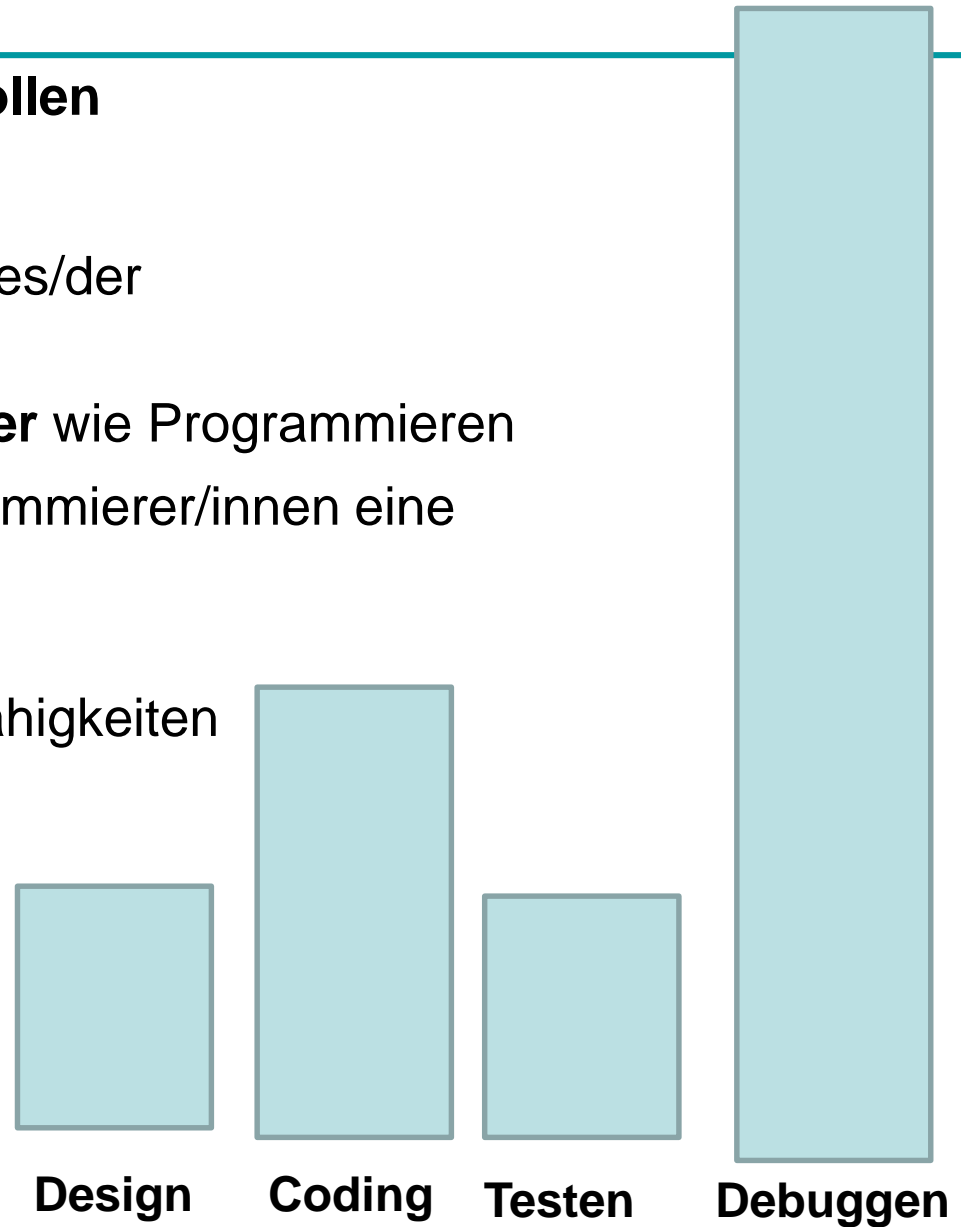
Debuggen



Zeitaufwand bei Erfüllen der Rollen

Debuggen ist

- eine **wesentliche Fähigkeit** des/der Softwareentwicklers/in
- mindestens **doppelt so schwer** wie Programmieren
- Auch für professionelle Programmierer/innen eine **wesentliche Aufgabe**
- also kein Zeichen von schlechten Programmierfähigkeiten



Agenda

- **Wiederholung und 1,5h Aufgabe**
- **Was ist ein Bug?**
- **Rollen bei der Softwareentwicklung**
- **Beschreiben eines Bugs**
- **Debug-Vorgehen: 6 Stufenplan**
 - 3. Minimierung
 - 4. Hypothesen
 - 5. Analysewerkzeuge
Code editieren
 - 6. Testen und Testwerkzeuge
- **Abschließende kurze Übung**
- **Ihre zusammenfassenden Fragen**
- **Ausblick**

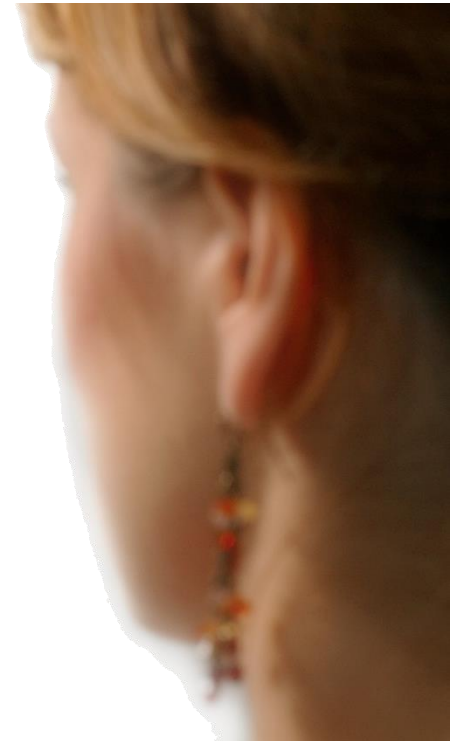
Strategien des Debugging Detektivs

Beschreiben:

- a.** Ziel
- b.** Annahmen (Eingabe, Kontext)
- c.** Code
- d.** Erwartetes Verhalten (SOLL)
- e.** Tatsächliches Verhalten (IST)

Effektiv für Lösungsfindung:


Bug (**a-e**) sich selbst
oder anderen beschreiben
(Mirror Talking)



Ziel heute

- Finden des Fehlers in der REST-API „miniTwitter“

http://localhost:300... + No environ

GET ▾	http://localhost:3000/tweets		Params	Send ▾
Authorization	Headers (1)	Body	Pre-request script	Tests
<input checked="" type="checkbox"/> Accept		application/json		
Header		Value		

Could not get any response

This seems to be like an error connecting to <http://localhost:3000/tweets>.

A. Zielbeschreibung

- **Der Node.js Server soll die falsche Angabe eines Accept-Version Headers erkennen und einen Fehler ausgeben**

B. Annahmen

- Der Request kommt korrekt bei `node.js` an
- Der neue Middlewarehandler wird beim Request immer aufgerufen (`app.use(...)`)
- Im Request-Objekt unter `req.get('Accept-Version')` steht der Wert des Headerfeldes
- Der Code prüft auf `!== 1.0` und setzt beim Response-Objekt mit `res.sendStatus(406)` einen Fehlercode
- Ansonsten wird über `andere Handler` der REST-Request bearbeitet
- Die gesetzten `Response-Werte` meines Codes sind `korrekt`
- `Node.js` liefert die Antwort aus
- ...

B. Annahmen

Typische fehlerhafte Annahmen beim Debuggen sind

- Ich habe keine Tippfehler im Code
- Bedingungen (if, ..) sind korrekt und tun, was sie sollen
- Eingabewerte (Parameter) sind wie erwartet
- Meine Ausgabewerte sind wie vom Rest des Codes erwartet
- Die von mir benutzen Methoden funktionieren wie erwartet
- Die eingesetzten Libraries, Frameworks, Betriebssysteme, Hardware funktioniert wie erwartet (Komponenten-Stack)

C. Code

```
"use strict";

// node module imports
var path = require('path');
var express = require('express');
var bodyParser = require('body-parser');

// own modules imports
var store = require('./blackbox/store.js');

// creating the server application
var app = express();

// Middleware *****
app.use(express.static(path.join(__dirname, 'public')));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

// logging
app.use(function(req, res, next) {
  console.log('Request of type ' + req.method + ' to URL ' + req.originalUrl);
  next();
});

// API-Version control. We use HTTP Header field Accept-Version instead of URL-part /v1/
app.use(function(req, res, next){
  // expect the Accept-Version 1.0
  var versionWanted = req.get('Accept-Version');
  if (versionWanted === undefined || versionWanted !== '1.0') {
    // 406 Accept-* header cannot be fulfilled.
    res.status(406);
  } else {
    next();
  }
});

// request type application/json check
app.use(function(req, res, next) {
  if (['POST', 'PUT'].indexOf(req.method) > -1 &&
    !( /application\/json/.test(req.get('Content-Type')) )) {
    // send error code 415: unsupported media type
    res.status(415).send('wrong Content-Type'); // user has SEND the wrong type
  } else if (!req.accepts('json')) {
    // send 406 that response will be application/json and request does not support it by 1
    // user has REQUESTED the wrong type
    res.status(406).send('response of application/json only supported, please accept this')
  } else {
    next(); // let this request pass through as it is OK
  }
});
```

```
// Routes *****

app.get('/tweets', function(req, res, next) {
  res.json(store.select('tweets'));
});

app.post('/tweets', function(req, res, next) {
  var id = store.insert('tweets', req.body); // TODO check that the element is really a tweet
  // set code 201 "created" and send the item back
  res.status(201).json(store.select('tweets', id));
});

app.get('/tweets/:id', function(req, res, next) {
  res.json(store.select('tweets', req.params.id));
});

app.delete('/tweets/:id', function(req, res, next) {
  store.remove('tweets', req.params.id);
  res.status(200).end();
});

app.put('/tweets/:id', function(req, res, next) {
  store.replace('tweets', req.params.id, req.body);
  res.status(200).end();
});

// CatchAll for the rest (unfound routes/resources *****

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handlers (express recognizes it by 4 parameters!)

// development error handler
// will print stacktrace as JSON response
if (app.get('env') === 'development') {
  app.use(function(err, req, res, next) {
    console.log('Internal Error: ', err.stack);
    res.status(err.status || 500);
    res.json({
      error: {
        message: err.message,
        error: err.stack
      }
    });
  });
}

// production error handler
...
```

D. Erwartetes Verhalten (SOLL)

- i. Wenn ich einen GET-Request mit gesetztem Header-Feld „Accept-Version: 1.0“ aus Postman heraus an localhost:3000/tweets absetze, dann kommen die tweets als JSON-Antwort zurück
- ii. Lasse ich das Feld aus oder sende eine andere Versionsnummer kommt kein BODY und im Response Header ist Status-Code 406 gesetzt.

E. Tatsächliches Verhalten (IST)

- i. Fall i.) funktioniert
- ii. Fall ii.) liefert keine Antwort. Der Browser wartet ewig auf die Antwort.

Ah, hmm. Moment „Der Browser wartet ewig auf die Antwort.“

...

Agenda

- **Wiederholung und 1,5h Aufgabe**
- **Was ist ein Bug?**
- **Rollen bei der Softwareentwicklung**
- **Beschreiben eines Bugs**
- **Debug-Vorgehen: 6 Stufenplan**
 - 3. Minimierung
 - 4. Hypothesen
 - 5. Analysewerkzeuge
Code editieren
 - 6. Testen und Testwerkzeuge
- **Abschließende kurze Übung**
- **Ihre zusammenfassenden Fragen**
- **Ausblick**

Von der Hypothese zum Nachweis: Den Bug finden

Ein 6 Stufenplan:

1. **Bug beschreiben** (u.a. SOLL und IST)
2. **Stabile Reproduzierbarkeit** herstellen
3. **Isolieren** und „Minimized Example“ erstellen
4. **Hypothesen** formulieren zur Frage:
„Wo könnte der Bug herkommen?“
 - a. Das wahrscheinlichste zuerst prüfen
 - b. Die einfache These bevorzugen vor der komplizierten
 - c. Mit jeder geprüften These 50% der Fehlerquellen ausschließen**
5. **Beheben durch Analysieren**
Nicht geklappt? 4. wiederholen.
Wenn nichts mehr übrig,
3. wiederholen, 4. detaillierter (Annahmen hinterfragen!)
6. **Doku und Tests:** 2. und 3. rückgängig machen und testen (ideal: Unit-Test)



** Konzept des Divide und Conquer (Teile und (Be)herrsche)

Stufenplan: 3. Isolieren und Minimieren

■ Code-Mini-Example (wo der Bug noch auftritt)

```
"use strict";
// module imports
var express = require('express');
var store = require('./blackbox/store.js');

var app = express();
// (..)
// API-Version control. We use HTTP Header field Accept-Version instead of URL-part /v1/
app.use(function(req, res, next){
    // expect the Accept-Version 1.0
    var versionWanted = req.get('Accept-Version');
    if (versionWanted === undefined || versionWanted !== '1.0') {
        // 406 Accept-* header cannot be fulfilled.
        res.status(406);
    } else {
        next();
    }
});
// Routes *****
app.get('/tweets', function(req, res, next) {
    res.json(store.select('tweets'));
});

// Start server *****
app.listen(3000); // no callback
```

Agenda

- **Wiederholung und 1,5h Aufgabe**
- **Was ist ein Bug?**
- **Rollen bei der Softwareentwicklung**
- **Beschreiben eines Bugs**
- **Debug-Vorgehen: 6 Stufenplan**
 - 3. Minimierung
 - Hypothesen
 - 5. Analysewerkzeuge
 - Code editieren
 - 6. Testen und Testwerkzeuge
- **Abschließende kurze Übung**
- **Ihre zusammenfassenden Fragen**
- **Ausblick**

Stufenplan: 4. Hypothese formulieren

■ Wo könnte der Bug liegen?

- Das wahrscheinlichste zuerst prüfen
- Die einfache These bevorzugen vor der komplizierten
- Mit jeder geprüften These 50% der Fehlerquellen ausschließen**



Aufgabe:

1. Formulieren Sie in Abstimmung mit Ihrem Banknachbarn **ZWEI** wahrscheinliche Hypothesen, weswegen der Bug wohl auftreten könnte. (2min)

2. Danach Tafelsammlung

■ Beispiele:

- Hohe Wahrscheinlichkeit: „*Der Server läuft gar nicht und die erste Antwort kommt aus einem Cache*“
- Geringe Wahrscheinlichkeit: „*Es wird eine Antwort gesendet, aber Postman 'kennt' den Statuscode 406 nicht und hängt daher*“

Stufenplan: 4. Hypothese formulieren // Code zur Aufgabe

```
"use strict";  
// module imports  
var express = require('express');  
var store = require('./blackbox/store.js');  
  
var app = express();  
// API-Version control. We use HTTP Header field Accept-Version  
app.use(function(req, res, next){  
  // expect the Accept-Version 1.0  
  var versionWanted = req.get('Accept-Version');  
  if (versionWanted === undefined || versionWanted !== '1.0') {  
    // 406 Accept-* header cannot be fulfilled.  
    res.status(406);  
  } else {  
    next();  
  }  
});  
// Routes *****  
app.get('/tweets', function(req, res, next) {  
  res.json(store.select('tweets'));  
});  
// Start server *****  
app.listen(3000); // no callback
```



Von der Hypothese zum Nachweis: Den Bug finden

Ein 6 Stufenplan:

1. **Bug beschreiben** (u.a. SOLL und IST)
2. **Stabile Reproduzierbarkeit** herstellen
3. **Isolieren** und „Minimized Example“ erstellen
4. **Hypothesen** formulieren zur Frage:
„Wo könnte der Bug herkommen?“
 - a. Das wahrscheinlichste zuerst prüfen
 - b. Die einfache These bevorzugen vor der komplizierten
 - c. Mit jeder geprüften These 50% der Fehlerquellen ausschließen**
5. **Beheben durch Analysieren**
Nicht geklappt? 4. wiederholen.
Wenn nichts mehr übrig,
3. wiederholen, 4. detaillierter (Annahmen hinterfragen!)
6. **Doku und Tests:** 2. und 3. rückgängig machen und testen (ideal: Unit-Test)



** Konzept des Divide und Conquer (Teile und (Be)herrsche)

Agenda

- **Wiederholung und 1,5h Aufgabe**
- **Was ist ein Bug?**
- **Rollen bei der Softwareentwicklung**
- **Beschreiben eines Bugs**
- **Debug-Vorgehen: 6 Stufenplan**
 - 3. Minimierung
 - 4. Hypothesen

Analysewerkzeuge

Code editieren

6. Testen und Testwerkzeuge

- **Abschließende kurze Übung**
- **Ihre zusammenfassenden Fragen**
- **Ausblick**

Stufenplan: 5. Beheben durch Analysieren

Tools für node-js/express

Inspektion

node-inspector
express-debug

Logging

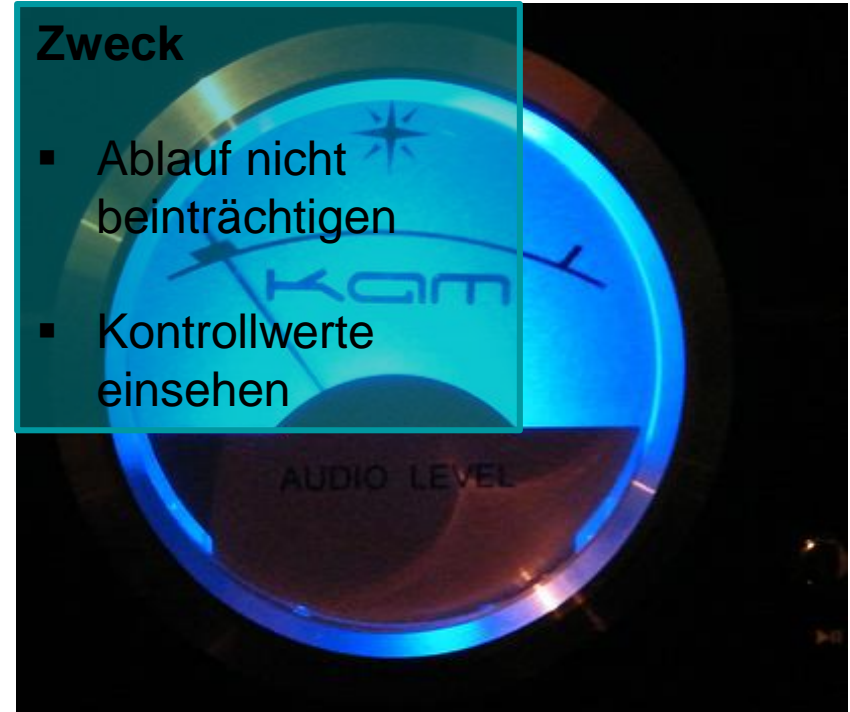
debug
morgan
log4js

Zweck

- Ablauf nachvollziehen
- Systemzustand erfassen

Zweck

- Ablauf nicht beeinträchtigen
- Kontrollwerte einsehen



Stufenplan: 5. Beheben durch Analysieren

■ npm install debug --save

```
// node module imports
var express = require('express');
var debug = require('debug');

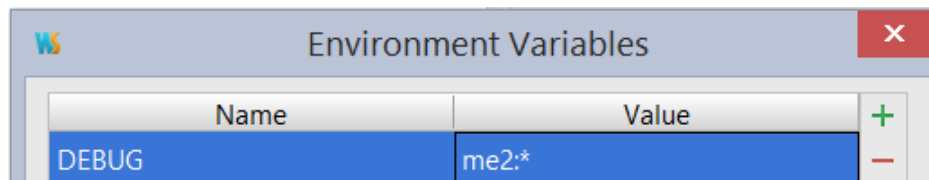
...
// creating the logger
var logger = debug('me2:su5app');

logger("Logger initialized");
```

Logging

debug
morgan
log4js

■ debug filtert Ausgaben anhand der Umgebungsvariable DEBUG. (fehlt die, kommt keine Ausgabe)



```
C:\>set DEBUG=me2:*
```


Stufenplan: 5. Beheben durch Analysieren

■ npm install debug --save

```
// node module imports
var express = require('express');
var debug = require('debug');

...
// creating the logger
var logger = debug('me2:su5app');

logger("Logger initialized");
```

Logging

debug
morgan
log4js

■ Express selbst nutzt debug!

- Setzen Sie z.B. `set DEBUG=me2:*, express:*`

```
...
Mon, 09 May 2016 08:37:20 GMT express:router:layer new /
Mon, 09 May 2016 08:37:20 GMT express:router:route new /tweets
Mon, 09 May 2016 08:37:20 GMT express:router:layer new /tweets
Mon, 09 May 2016 08:37:20 GMT express:router:route get /tweets
Mon, 09 May 2016 08:37:20 GMT express:router:layer new /
```

Stufenplan: 5. Beheben durch Analysieren

■ npm install debug --save

```
// node module imports
var express = require('express');
var debug = require('debug');

...
// creating the logger
var logger = debug('me2:su5app');

logger("Logger initialized");
```

Logging

debug
morgan
log4js

```
console.log("Das bitte nicht mehr!");
```

Stufenplan: 5. Beheben durch Analysieren

- **npm install morgan --save**
 - ein HTTP Request/Response Logger
 - als Middleware

```
// node module imports
...
var morgan = require('morgan');
...
// Middleware *****

app.use(morgan('common'));
```

Logging

debug
morgan
log4js

Konsolenausgabe

```
::1 - - [09/May/2016:09:04:04 +0000] "GET /tweets HTTP/1.1" 200 190
::1 - - [09/May/2016:09:04:28 +0000] "GET /tweets/101 HTTP/1.1" 404 23
```

Stufenplan: 5. Beheben durch Analysieren

- **npm install morgan --save**
 - ein HTTP Request/Response Logger
 - als Middleware

```
// node module imports
...
var morgan = require('morgan');
...
// Middleware *****

app.use(morgan('tiny'));
```

Logging

debug
morgan
log4js

Ausgabe-Format

common
tiny
combined
dev
short

Konsolenausgabe

```
GET /tweets 200 190 - 3.738 ms
GET /tweets/101 404 23 - 2.546 ms
```

Stufenplan: 5. Beheben durch Analysieren

■ debug als Ausgabe von morgan

```
// node module imports
...
var debug = require('debug');
var morgan = require('morgan');

// creating the logger
var logger = debug('me2:su5app');

// Middleware *****
app.use(morgan('tiny', {
  "stream": { write: function(str) { logger(str); } }
}));
```

Logging

debug
morgan
log4js

Konsolenausgabe

```
Mon, 09 May 2016 09:12:23 GMT me2:su4app GET /tweets 200 190 - 3.341 ms
Mon, 09 May 2016 09:12:26 GMT me2:su4app GET /tweets/101 404 23 - 0.724 ms
```

Stufenplan: 5. Beheben durch Analysieren

- **npm install log4js --save**
 - Ein umfangreiches Logging Framework
 - Basiert auf browser-seitigem log4js
 - Portiert für node (version 0.6.27)

Logging

debug
morgan
log4js

```
var log4js = require('log4js');  
...  
// creating the logger  
var logger = log4js.getLogger('me2:su5app');  
logger.setLevel('TRACE');  
...  
logger.trace("Details about the sys state");  
logger.debug("Logger initialized");  
logger.info("System startup");  
logger.warn("Hmm, should not be");  
logger.error("An exceptional state");  
logger.fatal("Ups!");
```

Stufenplan: 5. Beheben durch Analysieren

- **npm install log4js --save**
 - Ein umfangreiches Logging Framework
 - Basiert auf browser-seitigem log4js
 - Portiert für node (version 0.6.27)

Logging

debug
morgan
log4js

...

```
logger.trace("Details about the sys state");  
logger.debug("Logger initialized");  
logger.info("System startup");  
logger.warn("Hmm, should not be");  
logger.error("An exceptional state");  
logger.fatal("Ups!");
```

Konsolenausgabe

```
[2015-11-09 10:29:40.869] [TRACE] me2:su4app - Details about the sys state  
[2015-11-09 10:29:40.873] [DEBUG] me2:su4app - Logger initialized  
[2015-11-09 10:29:40.874] [INFO] me2:su4app - System startup  
[2015-11-09 10:29:40.874] [WARN] me2:su4app - Hmm, should not be  
[2015-11-09 10:29:40.874] [ERROR] me2:su4app - An exceptional state  
[2015-11-09 10:29:40.874] [FATAL] me2:su4app - Ups!
```

Stufenplan: 5. Beheben durch Analysieren

- **npm install --g node-inspector**
 - Debugger der im Chrome-Browser / Opera läuft
 - Server-seitige Breakpoints
 - Code-Editierung im Browser → auf Server geändert
- Start des Servers dann mit **node-debug app.js --save-live-edit**
 - Ruft automatisch node mit --debug-brk auf (Anhalten nach Start)
 - Startet automatisch Chrome Browser mit node-inspector auf port 8080

Inspektion

node-inspector
express-debug

Stufenplan: 5. Beheben durch Analysieren

- **npm install --g node-inspector**

Inspektion

node-inspector

The screenshot shows the Node Inspector web interface in a browser. The address bar shows the URL `127.0.0.1:8080/?ws=127.0.0.1:8080&port=5858`. The interface has tabs for Network, Sources, Profiles, and Console. The Sources tab is active, showing a file explorer on the left with the following structure:

- (core modules)
- file://
 - C:/Users/Johannes/OneDrive/Beu
 - blackbox
 - node_modules
 - app-minimized.js (selected)
 - app-minimized2.js
 - app-minimized3.js
 - app.js

The main editor displays the content of `app-minimized.js` with the following code:

```
11 "use strict";
12
13 // node module imports
14 var express = require('express');
15 var debug = require('debug');
16
17 // own modules imports
18 var store = require('./blackbox/store.js');
19
20 // creating the server application
21 var app = express();
22
23 // creating the logger
24 var logger = debug('me2:su4app');
25 logger("Logger initialized");
26
27
28 // Middleware *****
29 //none
30
31 // API-Version control. We use HTTP Header field Accept-Version instead
32 app.use(function (req, res, next) {
33   // expect the Accept-Version 1.0
34   var versionWanted = req.get('Accept-Version');
35   if (versionWanted === undefined || versionWanted !== '1.0') {
36     // 406 Accept-* header cannot be fulfilled.
37     res.status(406);
38   } else {
```

The right sidebar contains the following sections:

- Watch Expressions: `versionWanted: undefined`
- Call Stack: ☐ Async
- Scope Variables
- Breakpoints:
 - ☒ app-minimized.js:35
if (versionWanted === undefined || versi...

Stufenplan: 5. Beheben durch Analysieren

- **npm install express-debug --save-dev**
 - Bindet Debug-Tabs in die ausgelieferten HTML-Seiten ein**

Inspektion

node-inspector
express-debug

```
var expressdebug = require('express-debug');  
...  
var app = express();  
...  
expressdebug(app);
```

hello_world

express-debug

GET /2743

locals

request

session

template

software_info

profile

CLOSE

req

name	value		(-)
params	index	value	(-)
	id	"2743"	
body	{ }		
rawBody	undefined		
query	{ }		
files	{ }		
ip	"127.0.0.1"		
route	name	value	(-)
	path	("/:id")	
	method	"get"	
	callbacks	index	value

Stufenplan: 5. Beheben durch Analysieren

■ WebStorm Debugger

- IDEs haben oft integrierte Debugger
- Starten node und Klinken eigenen Debugger ein

Inspektion

node-inspector
chrome-devtools

+ IDE Debugger
(WebStorm)

```
// API-Version control. We use HTTP Header fi
app.use(function (req, res, next) {
  // expect the Accept-Version 1.0
  var versionWanted = req.get('Accept-Version');
  if (versionWanted === undefined || versionWanted !== '1.0') {
    // 406 Accept-* header cannot be fulfilled.
    res.status(406);
  } else {
    next();
  }
});
```

Debug 'app-minimized2.js' (Umschalt+F9)

app-minimized2.js

Stufenplan: 5. Beheben durch Analysieren

Inspektion

The screenshot shows the Visual Studio Code interface during a debug session. The file being edited is `app-minimized2.js`. The code is as follows:

```
35 // API-Version control. We use HTTP Header field Accept-Version instead of UR
36 app.use(function (req, res, next) { res: ServerResponse req: IncomingMessage
37 // expect the Accept-Version 1.0
38 var versionWanted = req.get('Accept-Version'); versionWanted: undefined
39 if (versionWanted === undefined || versionWanted !== '1.0') {
40 // 406 Not Acceptable header cannot be fulfilled.
41 versionWanted = undefined
42 next();
43 }
44 }
45 });
46
```

Red boxes highlight the variable declarations: `res: ServerResponse req: IncomingMessage` and `versionWanted: undefined`. A yellow tooltip shows `versionWanted = undefined`. The bottom panel shows the 'Variables' view with the following values:

- Local
 - `req` = IncomingMessage
 - `res` = ServerResponse
 - `versionWanted` = undefined
- Closure
 - `store` = Object

A blue arrow points from the text box to the 'Evaluate Expression' button in the debug toolbar.

Schrittweise Weitergehen,
Ausdrücke auswerten usw.

Stufenplan: 5. Beheben durch Analysieren

Inspektion

The screenshot shows the VS Code editor with the file `app-minimized2.js` open. The code is as follows:

```
37 // expect the Accept-Version 1.0
38 var versionWanted = req.get('Accept-Version'); versionWanted: undefined
39 if (versionWanted === undefined || versionWanted !== '1.0') {
40 // 406 Accept-* header cannot be fulfilled.
41 res.status(406);
42 } else {
43 next();
44 }
45 });
46
47 // Routes *****
48
```

A red arrow points to line 41, with a text box containing the text `..hier fehlt doch was`. The console shows the error `versionWanted: undefined`. The debugger shows the current state of variables:

- Local
 - `req` = IncomingMessage
 - `res` = ServerResponse
 - `versionWanted` = **undefined**
- Closure
 - `store` = Object

Agenda

- **Wiederholung und 1,5h Aufgabe**
- **Was ist ein Bug?**
- **Rollen bei der Softwareentwicklung**
- **Beschreiben eines Bugs**
- **Debug-Vorgehen: 6 Stufenplan**
 - 3. Minimierung
 - 4. Hypothesen
 - 5. Analysewerkzeuge
 - Code editieren**
 - 6. Testen und Testwerkzeuge
- **Abschließende kurze Übung**
- **Ihre zusammenfassenden Fragen**
- **Ausblick**

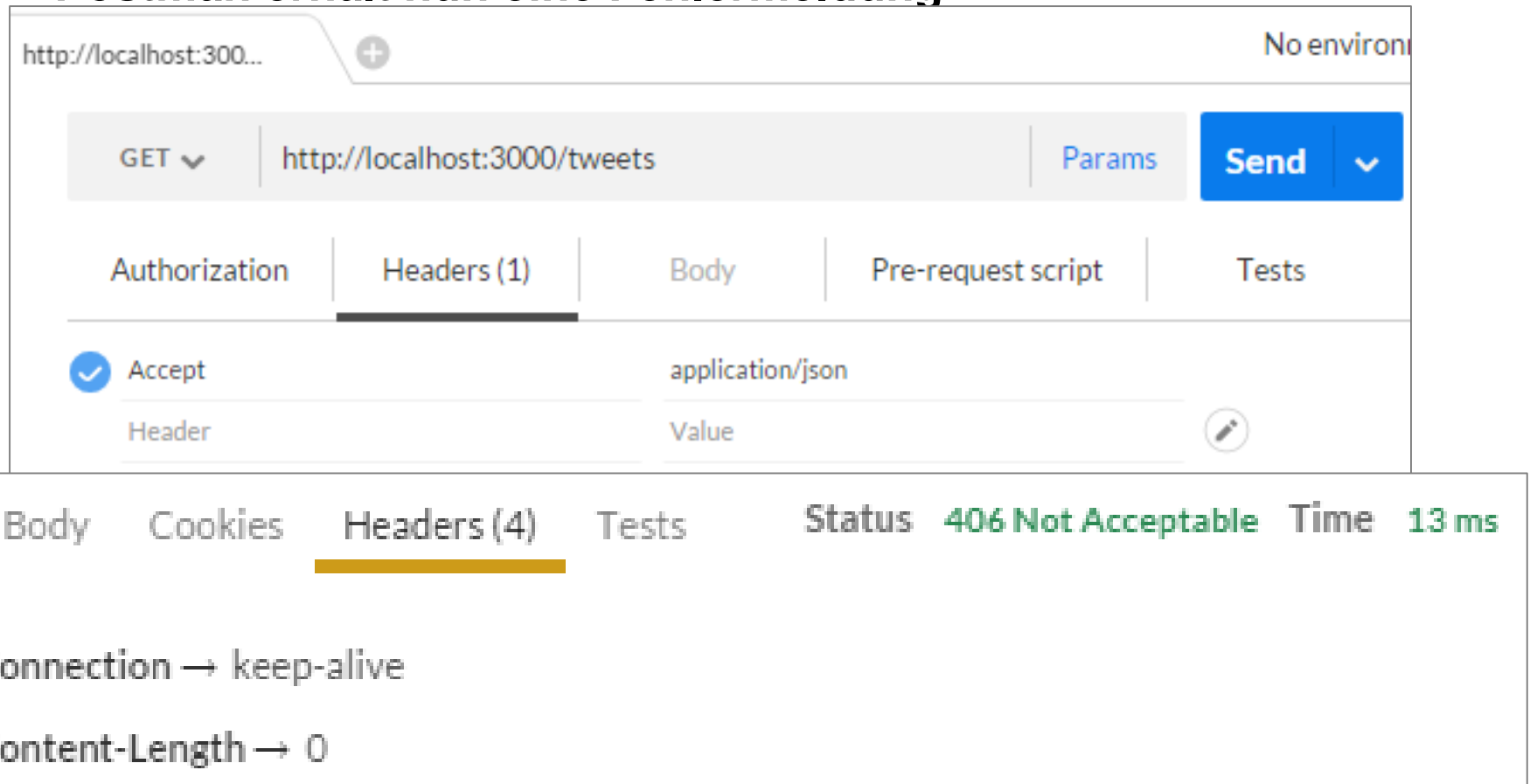
Stufenplan: 5. Beheben durch Analysieren

■ Code anpassen

```
// API-Version control. We use HTTP Header field Accept-Version  
app.use(function (req, res, next) {  
  // expect the Accept-Version 1.0  
  var versionWanted = req.get('Accept-Version');  
  if (versionWanted === undefined || versionWanted !== '1.0')  
    // 406 Accept-* header cannot be fulfilled.  
    res.status(406).end();  
  else {  
    next();  
  }  
});
```

Stufenplan: 5. Beheben durch Analysieren

- Erneuter Aufruf
- Postman erhält nun eine Fehlermeldung



The screenshot shows the Postman interface for a GET request to `http://localhost:3000/tweets`. The 'Headers' tab is selected, showing a single header: `Accept: application/json`. The response status is `406 Not Acceptable` with a response time of `13 ms`. The 'Body' tab shows the connection is `keep-alive` and the content length is `0`.

http://localhost:300... No environ

GET `http://localhost:3000/tweets` Params Send

Authorization Headers (1) Body Pre-request script Tests

✓ Accept `application/json`

Header Value

Body Cookies Headers (4) Tests Status **406 Not Acceptable** Time **13 ms**

Connection → keep-alive

Content-Length → 0

Bug erfolgreich behoben



Von der Hypothese zum Nachweis: Den Bug finden

Ein 6 Stufenplan:

1. **Bug beschreiben** (u.a. SOLL und IST)
2. **Stabile Reproduzierbarkeit** herstellen
3. **Isolieren** und „Minimized Example“ erstellen
4. **Hypothesen** formulieren zur Frage:
„Wo könnte der Bug herkommen?“
 - a. Das wahrscheinlichste zuerst prüfen
 - b. Die einfache These bevorzugen vor der komplizierten
 - c. Mit jeder geprüften These 50% der Fehlerquellen ausschließen**
5. **Beheben durch Analysieren**
Nicht geklappt? 4. wiederholen.
Wenn nichts mehr übrig,
3. wiederholen, 4. detaillierter (Annahmen hinterfragen!)
6. **Doku und Tests:** 2. und 3. rückgängig machen und testen (ideal: Unit-Test)



** Konzept des Divide und Conquer (Teile und (Be)herrsche)

Agenda

- **Wiederholung und 1,5h Aufgabe**
- **Was ist ein Bug?**
- **Rollen bei der Softwareentwicklung**
- **Beschreiben eines Bugs**
- **Debug-Vorgehen: 6 Stufenplan**
 - 3. **Minimierung**
 - 4. **Hypothesen**
 - 5. **Analysewerkzeuge**
 - Code editieren**

Testen und Testwerkzeuge

- **Abschließende kurze Übung**
- **Ihre zusammenfassenden Fragen**
- **Ausblick**

Stufenplan: 6. Testwerkzeuge

Testen

jasmine + frisby
Mocha + should-http

Achtung: Rollenwechsel

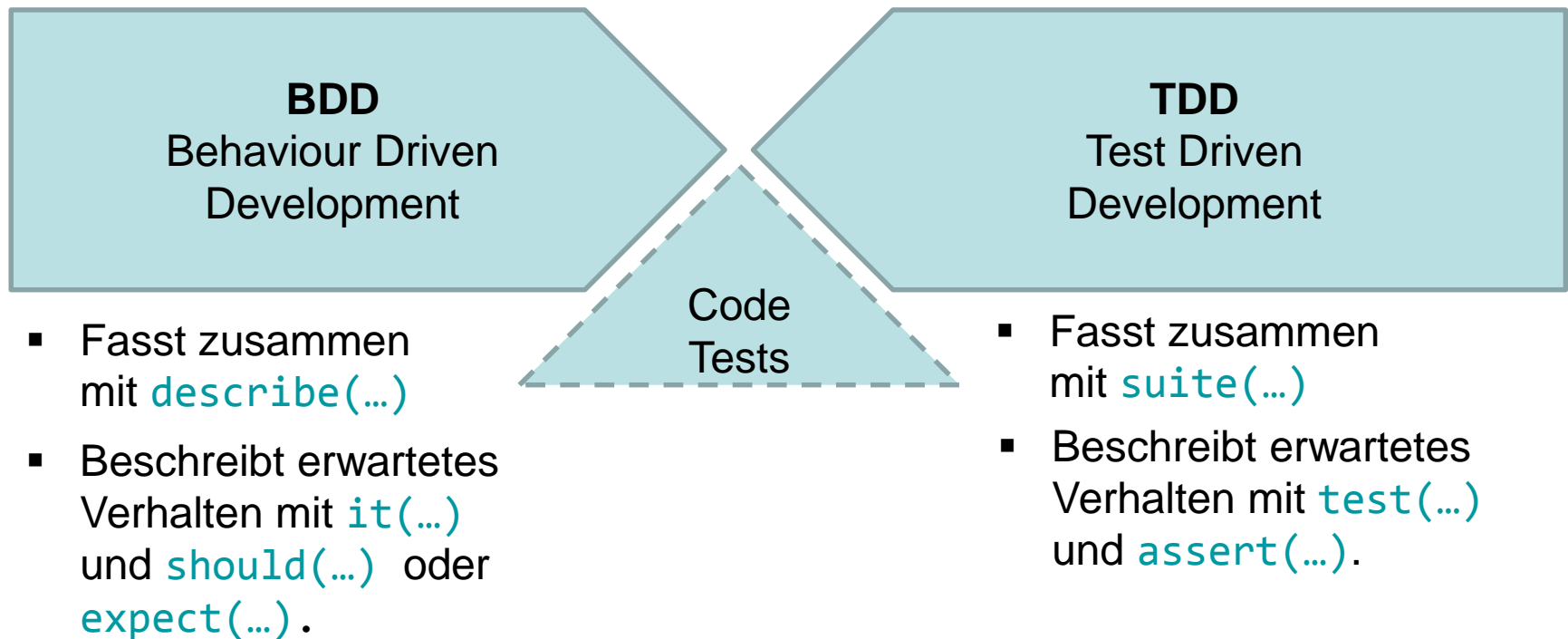
- Von Detektiv/in
- Zu Vandalen/in („think evil“)

Testen

- Normale Anfragen
- Extremfälle
- Ungültige Fälle

Stufenplan: 6. Testwerkzeuge

■ Zunächst: 2 Ansätze



Stufenplan: 6. Testwerkzeuge

- **Frisby für REST-APIs**
- **Basierend auf allg. Test-Suite Jasmine**
 - **npm install jasmine-node --save-dev (oder -g)**

BDD Testen
jasmine + frisby
Mocha + should-http

Datei: mini_test_spec.js

```
// Tests
describe("a suite of tests", function() {
  it("delivers two tweets with id 101 and 102", function() {
    // do some GET
    //...
    expect(contentType).toContain('application/json');
    // and so on
  });
});
```

- **Starten mit jasmine-node**
(sucht alle Dateien mit *_spec.js, üblicher Order ist \spec)
 - **Achtung: node.js Server muss unabhängig vorab gestartet sein!**

Stufenplan: 6. Testwerkzeuge

- + frisby

- `npm install frisby --save-dev`

BDD Testen

jasmine + frisby

Mocha + should-http

Datei: `mini_test_spec.js`

```
var frisby = require('frisby');
```

```
// Tests
```

```
describe("a suite of tests", function() {  
    frisby.create('Test the GET for Status 200')  
        .get('http://localhost:3000/tweets')  
        .expectStatus(200)  
        .toss();  
    ...  
});
```

- Weiterhin Start mit `jasmine-node`

Stufenplan: 6. Testwerkzeuge

- + frisby
 - `npm install frisby --save-dev`

BDD Testen
jasmine + frisby
Mocha + should-http

- Konsolenausgabe

```
Finished in 0.045 seconds  
1 test, 1 assertion, 0 failures, 0 skipped
```

- Dokumentation
 - Jasmine expect(..) usw.
<http://jasmine.github.io/2.3/introduction.html#section-Expectations>
 - Frisby REST-Methoden
<http://frisbyjs.com/docs/api/>

Stufenplan: 6. Testwerkzeuge

■ Jasmine mit Frisby: Beispiel

```
describe („Tweet REST API“, function () {  
    frisby.create('will send status 200 on GET')  
        .get('http://localhost:3000/tweets')  
        .expectStatus(200)  
        .toss();  
  
    frisby.create('will send JSON data of two tweets 101,102')  
        .get('http://localhost:3000/tweets')  
        .expectHeaderContains('Content-Type', 'application/json')  
        .expectJSON('?', {  
            'id': 102  
        })  
        .afterJSON(function (jsonObj) {  
            expect(jsonObj[0].id).toMatch(101);  
            expect(jsonObj.length).toBe(2);  
        })  
        .toss();  
});
```

BDD Testen
jasmine + frisby
Mocha + should-http

Stufenplan: 6. Testwerkzeuge

■ Jasmine in node „integrieren“

BDD Testen
jasmine + frisby
Mocha + should-http

1. package.json anpassen

```
"scripts": {  
  "start": "node app.js",  
  "test": "jasmine-node ./spec/ "
```

2. Anschließend im Projektverzeichnis starten mit npm test

```
> me2-su5_tweet_bug@0.0.1 test C:\Users\Johannes\OneDrive\Beuth_Lehre\ME2\Code\S  
U5_Tweet_API_Bug  
> jasmine-node ./spec/  
  
..  
  
Finished in 0.061 seconds  
2 tests, 5 assertions, 0 failures, 0 skipped
```

Stufenplan: 6. Testwerkzeuge

BDD Testen

jasmine + frisby

Mocha + should-http



Stufenplan: 6. Testwerkzeuge

■ Mocha

`npm install --g mocha`

- Testsuit für Client oder Server
 - Kann mit CLI Option `--ui tdd` auf „Test Driven“ umgestellt werden

1. package.json anpassen

```
"scripts": {  
  "test": "mocha"
```

2. Anschließend im Projektverzeichnis wieder starten mit `npm test`

- Mocha sucht im Unterordner `.\test` nach `.js` Dateien

BDD Testen
jasmine + frisby
Mocha + should-http

Stufenplan: 6. Testwerkzeuge

■ Mocha-Code (mit should.js, supertest.js)

BDD Testen
jasmine + frisby
Mocha + should-http

```
var should = require('should');  
require('should-http');  
var request = require('supertest');  
  
describe("miniTwitter REST API", function() {  
  
    it('should sends status 200 on GET', function(done) {  
        ...  
        done();  
    });  
  
});
```

Stufenplan: 6. Testwerkzeuge

■ Mocha-Code (mit should.js, supertest.js)

BDD Testen
jasmine + frisby
Mocha + should-http

```
describe("miniTwitter REST API", function() {  
  var url = 'http://localhost:3000';  
  
  it('will send 2 tweets 101,102 as json', function(done) {  
    request(url)  
      .get('/tweets')  
      .set('Accept-Version', '1.0')  
      .set('Accept', 'application/json')  
      .expect('Content-Type', /json/)   
      .expect(200)  
      .end(function(err, res) {  
        should.not.exist(err);  
        res.should.be.json();  
        res.body.should.have.lengthOf(2);  
        res.body[0].should.have.keys(['id', 'message', 'creator'])  
        done();  
      })  
  })  
});
```

Stufenplan: 6. Testwerkzeuge

■ Mocha-Code (mit should.js, supertest.js)

BDD Testen
Mocha + should-http

```
C:\Users\Johannes\OneDrive\Beuth_Lehre\ME2\Code\SU5_Tweet_API_Bug>npm test

> me2-su5_tweet_bug@0.0.1 test C:\Users\Johannes\OneDrive\Beuth_Lehre\ME2\Code\SU5_Tweet_API_Bug
> mocha

miniTwitter REST API
  U should sends status 200 on GET
  U will send 2 tweets 101,102 with content type json

2 passing (56ms)
```

■ Dokumentationen:

- <https://github.com/shouldjs/should.js>
- <https://www.npmjs.com/package/should-http>
- <https://github.com/visionmedia/supertest>
- <https://mochajs.org/#getting-started>

Von der Hypothese zum Nachweis: Den Bug finden

Ein 6 Stufenplan:

1. **Bug beschreiben** (u.a. SOLL und IST)
2. **Stabile Reproduzierbarkeit** herstellen
3. **Isolieren** und „Minimized Example“ erstellen
4. **Hypothesen** formulieren zur Frage:
„Wo könnte der Bug herkommen?“
 - a. Das wahrscheinlichste zuerst prüfen
 - b. Die einfache These bevorzugen vor der komplizierten
 - c. Mit jeder geprüften These 50% der Fehlerquellen ausschließen**
5. **Beheben durch Analysieren**
Nicht geklappt? 4. wiederholen.
Wenn nichts mehr übrig,
3. wiederholen, 4. detaillierter (Annahmen hinterfragen!)
6. **Doku und Tests:** 2. und 3. rückgängig machen und testen (ideal: Unit-Test)

** Konzept des Divide und Conquer (Teile und (Be)herrsche)

Agenda

- **Wiederholung und 1,5h Aufgabe**
- **Was ist ein Bug?**
- **Rollen bei der Softwareentwicklung**
- **Beschreiben eines Bugs**
- **Debug-Vorgehen: 6 Stufenplan**
 - 3. Minimierung
 - 4. Hypothesen
 - 5. Analysewerkzeuge
Code editieren
 - 6. Testen und Testwerkzeuge
- **Abschließende kurze Übung**
- **Ihre zusammenfassenden Fragen**
- **Ausblick**

Ziele von Testing, Inspektion, Logging

(Unit)Tests

Debug-
Inspektion

Logging

- Aufgabe: Ordnen Sie die Ziele zu den drei vorgestellten Werkzeugen zu. **Einzelarbeit** (2-3min)



Ablauf-Verfolgung

Code-Abdeckung

Einsicht in innere
Systemzustände

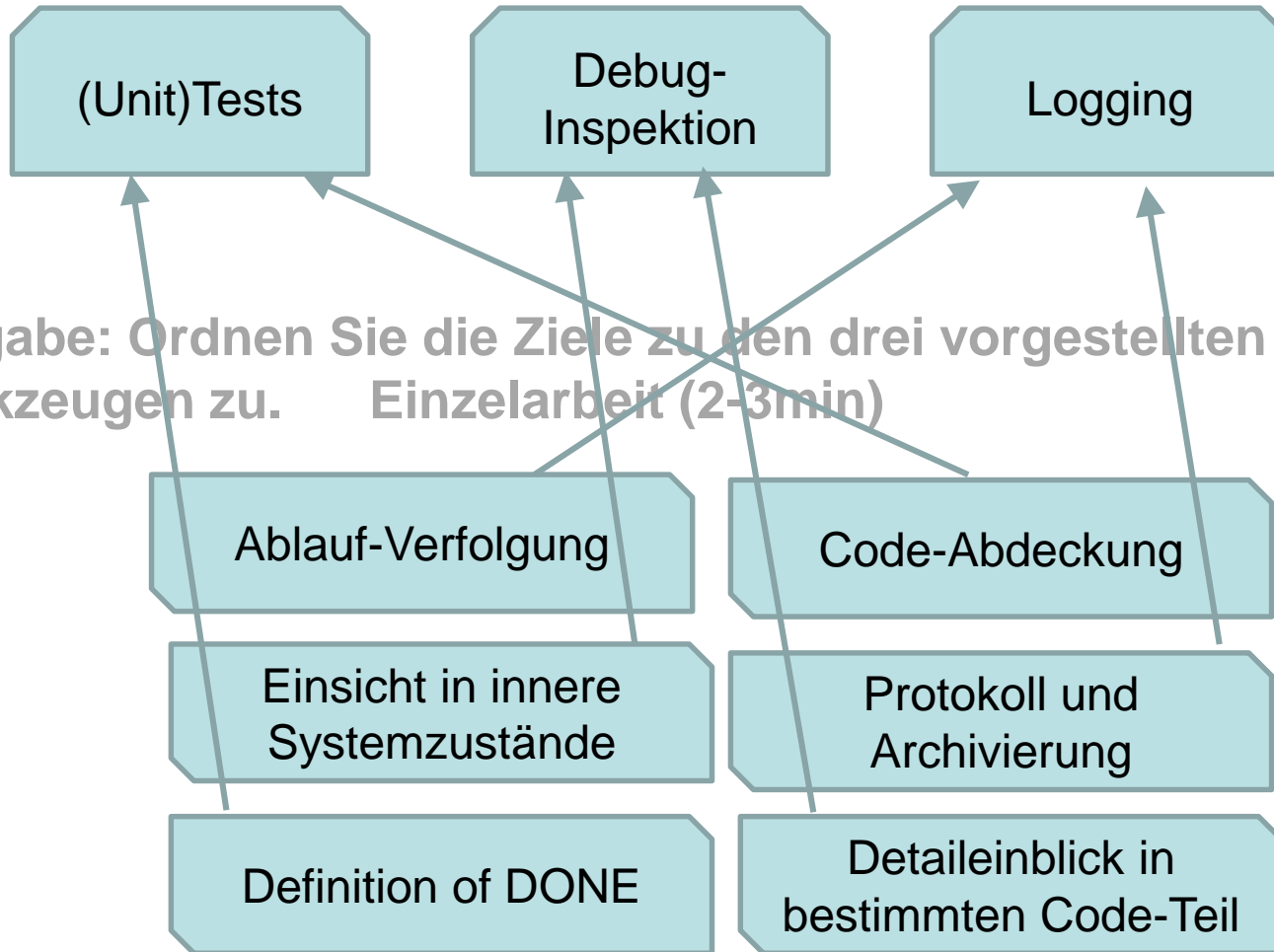
Protokoll und
Archivierung

Definition of DONE

Detaileinblick in
bestimmten Code-Teil

Ziele von Testing, Inspektion, Logging

- Aufgabe: Ordnen Sie die Ziele zu den drei vorgestellten Werkzeugen zu. Einzelarbeit (2-3min)



Ziele von Testing und Debugging

■ (Unit)Tests

- Code-Abdeckung
- Nutzbar für Definition of Done
- Standardisierte, reproduzierbare Prüfung der Funktionalität
- Nutzbar für agile Entwicklung und Test-driven Development
- Achtung: Trügerische Schein-Sicherheit

■ Debug-Inspektion

- Detaileinblick in bestimmten Code-Teil
- Einsicht in innere Systemzustände

■ Logging

- Ablauf-Verfolgung
- Protokoll und Archivierung
- (Einsicht in innere Systemzustände)

**Ziel für alle 3:
Erkennen von Fehlern!**

**→ Wirkung: Qualität
verbessern**

Agenda

- **Wiederholung und 1,5h Aufgabe**
- **Was ist ein Bug?**
- **Rollen bei der Softwareentwicklung**
- **Beschreiben eines Bugs**
- **Debug-Vorgehen: 6 Stufenplan**
 - 3. Minimierung
 - 4. Hypothesen
 - 5. Analysewerkzeuge
Code editieren
 - 6. Testen und Testwerkzeuge
- **Abschließende kurze Übung**
- **Ihre zusammenfassenden Fragen**
- **Ausblick**

Zusammenfassende Fragen

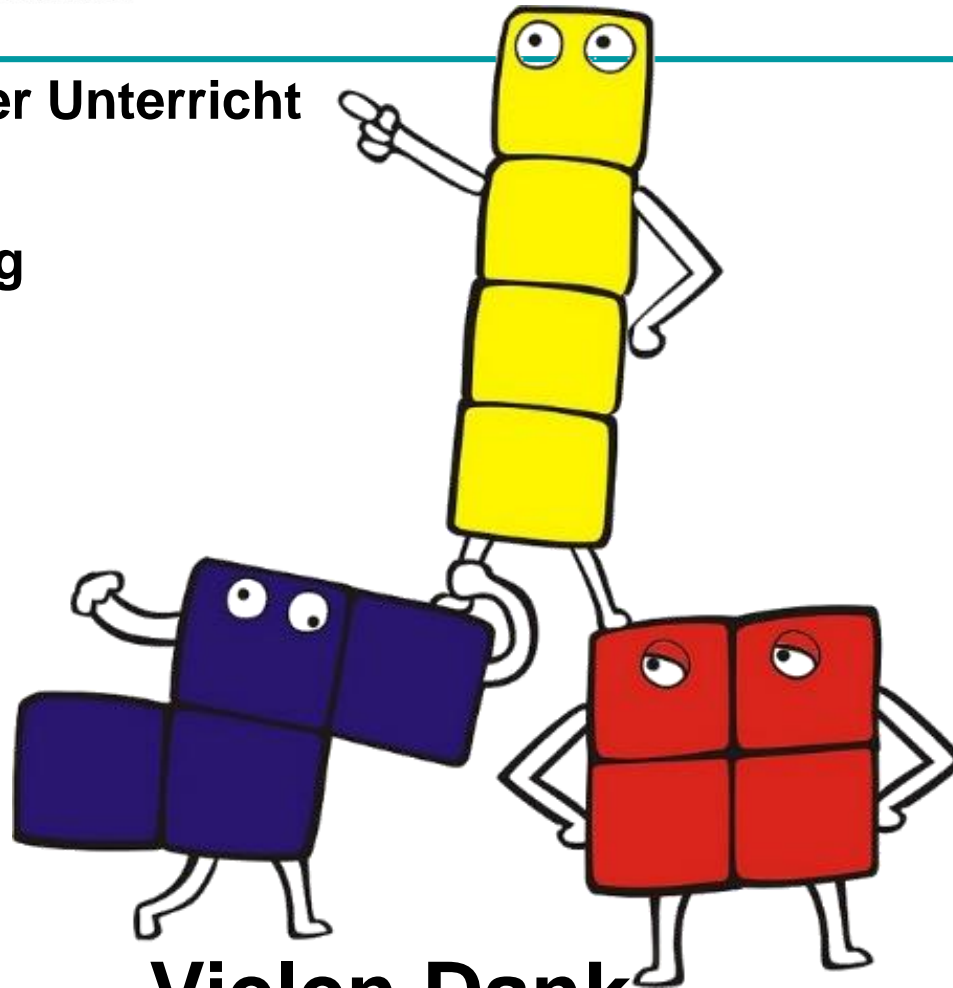
(1) Einsammeln

**(2) Drei Beispiele ziehen wir direkt
(und dann auch nächstes Mal)**



Ausblick / Nächster Unterricht

- **Strukturierung, Modularisierung**
 - **requireJS**
 - **AMD**



**Vielen Dank
und bis
zum nächsten Mal**