

LocalSharing

Test Plan

Version <1.1>

Revision History

Date	Version	Description	Author
13.05.2015	0.9	Anlegen des Dokumentes	Johannes
14.05.2015	1.0	Einfügen der Tests	Corinna
21.05.2015	1.1	Performance Tests	Julia

Table of Contents

1.	Introduction	2
1.1	Purpose	2
1.2	Scope	2
2.	Evaluation Mission and Test Motivation	2
2.1	Background	2
2.2	Evaluation Mission	3
3.	Test Approach	3
3.1	Testing Techniques and Types	4
3.1.1	Function Testing	4
3.1.2	Performance Profiling	5
4.	Deliverables	6
4.1	Test Evaluation Summaries	6
4.2	Reporting on Test Coverage	9
5.	Metrics	9
5.1	Duplizierter Code	9
5.2	Cyclomatic Complexity	13
6.	Test-Workflow	15
6.1	1.Entwickler	15
6.2	2. GitLab	16
6.3	3. Jenkins	17
6.4	4. SonarQube	17

Test Plan

1. Introduction

1.1 Purpose

Der Zweck dieses Test Plans liegt darin alle für das Testen notwendigen Informationen in einem Dokument zusammenzustellen. Dabei wird auf die verwendeten Testmethoden eingegangen und erläutert, wie diese angewendet werden.

1.2 Scope

Getestet wird durch Function-Tests, Unit-Tests und Performance-Tests. Durch die Unit-Tests werden die Controller und das Model des implementierten Architectural Patterns Model View Controller umfangreich getestet. Die Function-Tests dienen dem automatisierten Prüfen der Benutzeroberfläche und somit der View.

2. Evaluation Mission and Test Motivation

Motiviert sind die Tests durch das Bemühen eine möglichst gut funktionierende Webanwendung aufzubauen, bei deren Nutzung die User auf keine Probleme stoßen sollten, da dies ihren Willen, LocalSharing weiterzuverwenden, schmälern könnte.

Demnach soll die Anwendung immer weiter verbessert werden. Das Ändern und Erweitern des Quellcodes kann aber dazu führen, dass Fehler entstehen, die es zuvor nicht gab. Viele davon werden ohne Tests erst bemerkt, wenn sie vom User gemeldet werden. Um das zu vermeiden, ist es sinnvoll Tests schon bei jedem Commit der Anwendung automatisiert durchführen zu lassen, sodass diese schnell festgestellt und behoben werden können.

2.1 Background

Die verschiedenen Testmethoden werden aus unterschiedlichen Gründen verwendet und bringen mehrere Vorteile mit sich. Zu Beginn des Projekts wurden die Function Tests eingeführt. Zu Beginn der Construction Phase wurden Unit-Tests hinzugefügt. Im Laufe der Construction Phase soll zudem mit den Performance-Tests begonnen werden.

Für die Function-Tests wurden Feature Files erstellt, worin vom Kunden beschrieben ist, wie sich die Anwendung verhalten soll. Pro Use Case wurde ein solches Feature File geschrieben. Damit kann in den Function-Tests geprüft werden, ob die Interaktion zwischen Benutzer und Anwendung den Anforderungen des Kunden entspricht.

Die Logik des Controllers und Models sowie der Services und weiterer Util-Klassen kann durch Unit-Tests getestet werden. Dadurch wird sichergestellt, dass das Refactoring nicht unerwünschterweiser Fehler hervorruft.

Da die Benutzung der Webanwendung für den Benutzer möglichst angenehm und reibungslos verlaufen sollte, ist es wichtig, dass diese auch mit vielen Zugriffen klarkommen kann. Mögliche Flaschenhälse sollten dementsprechend frühzeitig gefunden werden. Zudem ist es wichtig, dass es zu keinem Fehlverhalten in der Anwendung kommt, wenn sich diese im Mehrbenutzerbetrieb befindet. Das Risiko für zuvor genannte Probleme kann durch Performance Tests minimiert werden. Durch diese kann getestet werden, wie die Reaktionszeit der Anwendung bei einer zuvor bestimmten Anzahl an Zugriffen ist.

2.2 Evaluation Mission

Das Testen soll dazu führen, möglichst viele Bugs zu finden, sodass ein gewisser Qualitätsstandard der Webanwendung immer sichergestellt ist.

3. Test Approach

Alle Tests sollen automatisiert durchgeführt werden. Dies soll bei jedem Commit passieren, sodass die Auswirkungen der Änderungen sogleich festgestellt werden können. Da aber die Function Tests sowie die Performance Tests mehr Zeit in Anspruch nehmen als gewünscht, gibt es zwei Jenkins Jobs. Der erste führt nur die Unit Tests durch, sodass innerhalb von zwei Minuten ein Feedback zu erwarten ist. Der zweite Job führt die Function Tests und die Performance Tests durch. Dieser braucht dementsprechend deutlich länger und braucht mehr als acht Minuten, um ein Feedback zu senden.

Guten Tag hier ist Bob der Baumeister am Werke!

 [Beschreibung hinzufügen](#)

All					
S	W	Name ↓	Letzter Erfolg	Letzter Fehlschlag	Letzte Dauer
		LocalSharing	37 Minuten - #127	3 Tage 15 Stunden - #111	1 Minute 9 Sekunden 
		LocalSharing_Function_Performance_Tests	19 Minuten - #20	35 Minuten - #18	8 Minuten 9 Sekunden 

Symbol: [S](#) [M](#) [L](#)

[Legende](#)



[RSS Alle Builds](#)



[RSS Nur Fehlschläge](#)



[RSS Nur jeweils letzter Build](#)

3.1 Testing Techniques and Types

3.1.1 Function Testing

Function Tests werden verwendet, um zu prüfen, ob sich eine Anwendung den Anforderungen des Kunden entsprechend verhält. Dabei wird nicht der Code selbst geprüft, sondern nur das Verhalten der Anwendung, wobei diese als Black Box betrachtet wird, mit der nur über die Benutzeroberfläche interagiert wird. Anhand der Ergebnisse kann die Anwendung geprüft werden.

Technique Objective:	Ziel ist das Überprüfen der korrekten Umsetzung der Use Cases.
Technique:	Der für jeden Use Case festgelegte Flow of Events stellt die übliche Abfolge von Ereignissen dieses Use Cases dar. Darin wird auch festgelegt, was die Vor- und Nachbedingungen hierfür sind. Zudem sind weitere alternative Abläufe definiert, die zur Ausführung kommen, wenn vom Standardablauf abgewichen wird.
Oracles:	Für jeden User Case wurde ein Feature File erstellt. Diese werden als Function Tests automatisiert ausgeführt. Darin definiert ist das Verhalten für den Fall, dass der Use Case wie erwartet durchgeführt wird sowie alternative Szenarien, bei denen die Durchführung des Use Cases vom Benutzer gewünscht oder versehentlich durch eine fehlerhafte Eingabe abgebrochen wird.
Required Tools:	<ul style="list-style-type: none">• GitLab• Jenkins CI• Cucumber• Selenium• X Window Virtual Framebuffer (Xvfb)• Mozilla Firefox
Success Criteria:	Für jeden Use Case existiert ein Feature File, das jeweils einen Standardablauf sowie wenn nötig alternative Abläufe definiert. Deren Durchführung erfolgt ohne Fehler.
Special Considerations:	Die neuen Versionen von Mozilla Firefox sowie von Selenium ergeben an einigen Stellen Probleme.

3.1.2 Performance Profiling

Bei den Performance-Tests geht es darum zeitkritische Funktionalitäten der Anwendung zu überprüfen. Dabei wird sie durch einen Stresstest auf ihre Belastbarkeit hin geprüft, um herauszufinden, was passiert, wenn unerwartet viele Benutzer auf sie zugreifen.

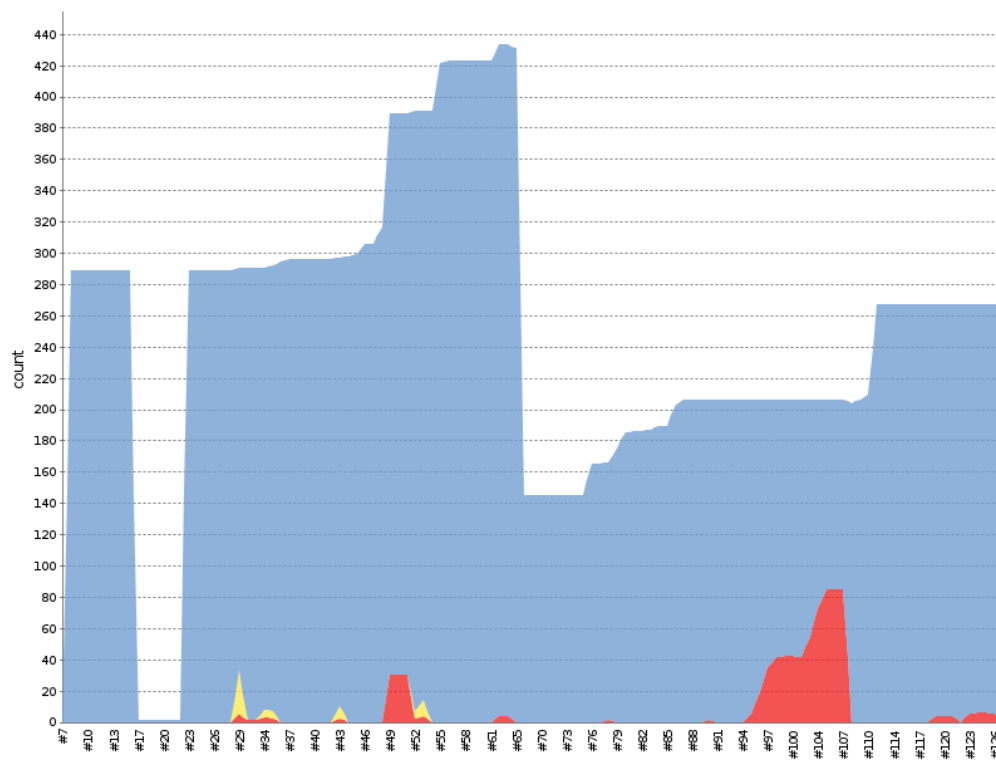
Technique Objective:	Ziel ist das Prüfen des Anwendungsverhalten im Hinblick auf die Performance. Dazu wird diese unter normalen sowie unter besonders schwierigen Bedingungen getestet.
Technique:	<p>Verschiedene Funktionen werden auf ihre Performance hin getestet. Dabei stehen im Speziellen die zeitaufwendigen Funktionen im Vordergrund. Häufig sind das solche, die mit der Datenbank interagieren.</p> <p>Hierfür kommen zwei Methoden zum Einsatz:</p> <p>Bei der Ersten wird von einer zuvor definierten Anzahl an Benutzern eine Funktion mit einer zuvor bestimmten Häufigkeit aufgerufen. Das Ergebnis ist die Reaktionszeit der Anwendung.</p> <p>Die zweite Methode stellt eine Art Stresstest dar, wobei auf die Funktion solange zugegriffen wird bis auf die Anfrage nicht mehr reagiert wird.</p>
Oracles:	Für beide Methoden ist es sinnvoll die getesteten Funktionen entlang der Use Cases auszuwählen. So sollen die durch die Zugriffe der in den Use Cases definierten Abläufe nachgestellt werden. Dabei kann getestet werden, wie die Anwendung auf mehrere gleichzeitige Benutzeranfragen reagiert, sodass auch bestimmt werden kann, welche Garantie bei einem Service-Level-Agreement (SLA) gegeben werden kann. Für die Festlegung eines solchen Testplans wird das Tool JMeter verwendet.
Required Tools:	<ul style="list-style-type: none"> • Git • Jenkins CI • Apache Maven • Apache JMeter
Success Criteria:	Auslieferung der angefragten Seiten in annehmbarer Zeit. Diese wird für das Projekt auf durchschnittlich 50ms festgelegt.
Special Considerations:	<p>JMeter erstellt virtuelle User.</p> <p>Server-Ausleistung im Ruhezustand:</p> <pre> 1 [] 7 [] 13 [] 19 [] 2 [] 8 [] 14 [] 20 [] 3 [] 9 [] 15 [] 21 [] 4 [] 10 [] 16 [] 22 [] 5 [] 11 [] 17 [] 23 [] 6 [] 12 [] 18 [] 24 [] Mem[] Tasks: 322, 675 thr: 1 running Swp[] Load average: 0.06 0.51 Uptime: 115 days(!), 18:19:34 </pre> <p>Server-Auslastung bei 500.000 simulierten Zugriffen:</p> <pre> 1 [] 7 [] 13 [] 19 [] 2 [] 8 [] 14 [] 20 [] 3 [] 9 [] 15 [] 21 [] 4 [] 10 [] 16 [] 22 [] 5 [] 11 [] 17 [] 23 [] 6 [] 12 [] 18 [] 24 [] Mem[] Tasks: 354, 3156 thr: 33 running Swp[] Load average: 2.92 1.41 Uptime: 115 days(!), 18:28:18 </pre> <p>Die simulierten User sowie der LocalSharing-Server befinden sich auf demselben Server. Das ist wichtig, da sonst zusätzlich zu der Performance der Anwendung auch die Netzwerk-Performance gemessen werden würde, was nicht Ziel des Tests ist.</p>

4. Deliverables

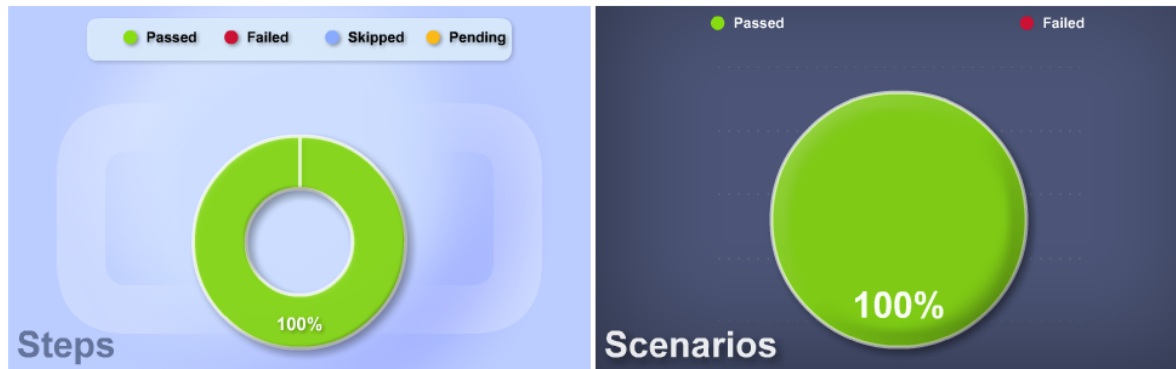
4.1 Test Evaluation Summaries

Nach jedem Commit werden zwei Jenkins Jobs gestartet, die Testreports zu allen drei Testmethoden erzeugen. Darin werden die Ergebnisse des jeweiligen Tests festgehalten.

Der erzeugte Report für die Unit-Tests sieht folgendermaßen aus und zeigt den Verlauf der Testergebnisse. In gelb zu sehen sind hierbei die übersprungenen Tests. Rot dargestellt werden fehlgeschlagene Tests.



Die Ergebnisse der Function-Tests werden durch nachfolgende Diagramme zusammengefasst. Darin zu sehen sind, wie aus der Legende geschlossen werden kann, die Anzahl der erfolgreich durchgeführten, übersprungenen bzw. fehlgeschlagenen Tests aufgeteilt nach Abläufen, Schritten und einzelnen Use Case.



Feature Statistics

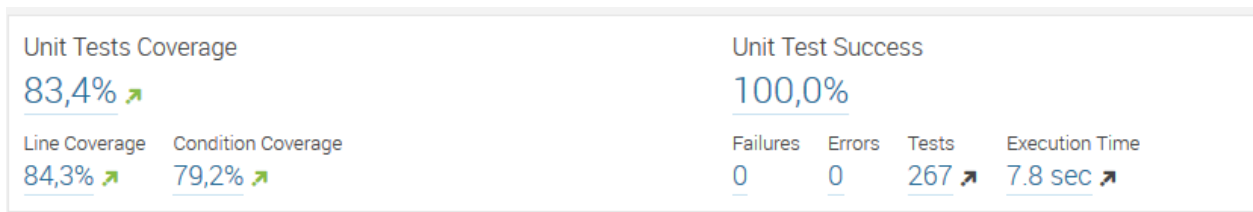
Feature	Scenarios			Steps					Duration	Status
	Total	Passed	Failed	Total	Passed	Failed	Skipped	Pending		
Register	3	3	0	41	41	0	0	0	16 secs and 323 ms	passed
Login	4	4	0	22	22	0	0	0	7 secs and 62 ms	passed
Edit profile information	3	3	0	54	54	0	0	0	20 secs and 689 ms	passed
Ausleih-Angebote verwalten	5	5	0	52	52	0	0	0	55 secs and 321 ms	passed
Hilfeleistungs-Angebote verwalten	5	5	0	48	48	0	0	0	56 secs and 108 ms	passed
Tausch-Angebote verwalten	5	5	0	46	46	0	0	0	53 secs and 328 ms	passed
6	25	25	0	263	263	0	0	0	3 mins and 28 secs and 834 ms	Totals

Ergebnisse der Performace-Tests

URI	Samples	Samples diff	Average (ms)	Average diff (ms)	Median (ms)	Median diff (ms)	Line90 (ms)	Minimum (ms)	Maximum (ms)	Http Code	Previous Http Code	Errors (%)	Errors diff (%)	Average (KB)	Total (KB)
	10	0	14	2	5	0	96	4	96	200		0.0 %	0.0 %		
	10	0	7	0	7	0	14	5	14	200		0.0 %	0.0 %		
	10	0	7	1	7	1	9	5	9	200		0.0 %	0.0 %		
	10	0	7	1	7	1	10	5	10	200		0.0 %	0.0 %		
	10	0	6	-1	7	0	8	5	8	200		0.0 %	0.0 %		
	10	0	6	-1	6	-2	9	5	9	200		0.0 %	0.0 %		
	10	0	8	-1	8	-2	13	7	13	200		0.0 %	0.0 %		
	10	0	6	-1	6	-1	9	5	9	200		0.0 %	0.0 %		
	10	0	6	-1	6	-1	9	5	9	200		0.0 %	0.0 %		
	10	0	6	0	6	0	7	5	7	200		0.0 %	0.0 %		
	10	0	119	1	115	3	153	109	153	200		0.0 %	0.0 %		
	10	0	5	-4	6	-1	8	5	8	200		0.0 %	0.0 %		
	10	0	7	1	6	0	18	5	18	200		0.0 %	0.0 %		
	10	0	7	1	8	0	9	7	9	200		0.0 %	0.0 %		
	10	0	6	0	6	-1	15	5	15	200		0.0 %	0.0 %		
	10	0	6	0	6	0	15	5	15	200		0.0 %	0.0 %		
	10	0	4	1	5	1	7	3	7	200		0.0 %	0.0 %		
	10	0	4	1	4	0	8	3	8	200		0.0 %	0.0 %		
	10	0	3	0	3	0	7	3	7	200		0.0 %	0.0 %		
	10	0	122	3	117	1	188	109	188	200		0.0 %	0.0 %		
	10	0	7	-1	7	0	10	5	10	200		0.0 %	0.0 %		
	10	0	7	0	7	0	10	5	10	200		0.0 %	0.0 %		
	10	0	13	0	12	0	20	9	20	200		0.0 %	0.0 %		
	10	0	6	-1	6	-1	10	5	10	200		0.0 %	0.0 %		
	10	0	6	-1	6	-1	9	5	9	200		0.0 %	0.0 %		
	10	0	6	0	6	0	9	5	9	200		0.0 %	0.0 %		
	10	0	10	-1	10	-1	15	8	15	200		0.0 %	0.0 %		
	10	0	6	-1	6	-1	10	5	10	200		0.0 %	0.0 %		
	10	0	6	-1	6	-1	10	5	10	200		0.0 %	0.0 %		
	10	0	9	3	6	0	32	5	32	200		0.0 %	0.0 %		
	10	0	8	0	7	0	29	5	29	200		0.0 %	0.0 %		
All URIs	310	0	14	0	6	-1	15	3	188			0.0 %	0.0 %		

Die weiteren Ergebnisse der Performance-Tests mit Antwortzeit-Graphen sind in dem Dokument ‘Trend report’ zu finden.

4.2 Reporting on Test Coverage

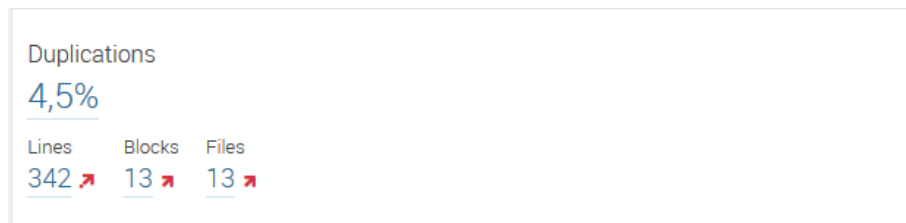


SonarQube wird zur statischen Code-Analyse verwendet. Dabei wird auch die Testabdeckung analysiert. Dies geschieht mit jedem Commit bei der Ausführung des ersten Jenkins Jobs. Darin enthalten ist die prozentuale Testabdeckung sowie die Ergebnisse des letzten Tests.

5. Metrics

Duplications (Duplizierter Code) & Complexity (Cyclomatic Complexity) wurden als Metrics ausgewählt.

5.1 Duplizierter Code



Durch die orangefarbene Markierung in der unteren Abbildung wird leicht ersichtlich, dass die Klassen sich nur sehr wenig voneinander unterscheiden. Der duplizierte Code kann durch eine gemeinsame Oberklasse reduziert werden.

```

public class GET_DeaktiviereEinAngebot extends BearbeitetWasAdmin {

    private static final String REQUEST_URL = "/disable/{id}/{type}";
    private static String SUCCESS_VIEW = "redirect:../../angebote/";
    private static String ERROR_VIEW = "redirect:angebote/";

    private Long angebotsId;
    private String type;

    @Autowired
    private B_DeaktiviereEinAngebot bDeaktiviereEinAngebot;

    @Autowired
    private LadeEinAngebot ladeEinAngebot;

    @RequestMapping(method = RequestMethod.GET, value = REQUEST_URL)
    protected String bearbeitetAnfrage(Principal principal,
        @PathVariable("id") String angebotsId,
        @PathVariable("type") String type) {
        this.angebotsId = Long.valueOf(angebotsId);
        this.type = type;
        SUCCESS_VIEW = "redirect:../../angebot/" + angebotsId + "/" + type;
        ERROR_VIEW = SUCCESS_VIEW;

        return bearbeitetAnfrageIntern(principal);
    }

    @Override
    protected LadenDaten attribute() {
        ladeEinAngebot.setId(angebotsId);
        ladeEinAngebot.setType(type);
        return ladeEinAngebot;
    }

    @Override
    protected String getSuccessView() {
        return SUCCESS_VIEW;
    }

    @Override
    protected String getErrorView() {
        return ERROR_VIEW;
    }

    @Override
    protected BearbeitenDaten getBereit() {
        return bDeaktiviereEinAngebot;
    }
}

```

```

public class GET_AktiviereEinAngebot extends BearbeitetWasAdmin {

    private static final String REQUEST_URL = "/enable/{id}/{type}";
    private static String SUCCESS_VIEW = "redirect:../../angebote/";
    private static String ERROR_VIEW = "redirect:angebote/";

    private Long angebotsId;
    private String type;

    @Autowired
    private B_AktiviereEinAngebot bAktiviereEinAngebot;

    @Autowired
    private LadeEinAngebot ladeEinAngebot;

    @RequestMapping(method = RequestMethod.GET, value = REQUEST_URL)
    protected String bearbeitetAnfrage(Principal principal,
        @PathVariable("id") String angebotsId,
        @PathVariable("type") String type) {
        this.angebotsId = Long.valueOf(angebotsId);
        this.type = type;
        SUCCESS_VIEW = "redirect:../../angebot/" + angebotsId + "/" + type;
        ERROR_VIEW = SUCCESS_VIEW;

        return bearbeitetAnfrageIntern(principal);
    }

    @Override
    protected LadenDaten attribute() {
        ladeEinAngebot.setId(angebotsId);
        ladeEinAngebot.setType(type);
        return ladeEinAngebot;
    }

    @Override
    protected String getSuccessView() {
        return SUCCESS_VIEW;
    }

    @Override
    protected String getErrorView() {
        return ERROR_VIEW;
    }

    @Override
    protected BearbeitenDaten getBereit() {
        return bAktiviereEinAngebot;
    }
}

```

LocalSharing

Nach Bildung der Oberklasse und somit Reduzierung des duplizierten Codes sehen die Klassen folgendermaßen aus:

```
@Controller
public class GET_DeaktiviereEinAngebot extends AktivierenDeaktivieren {

    private static final String REQUEST_URL = "/disable/{id}/{type}";

    @Autowired
    private B_DeaktiviereEinAngebot bDeaktiviereEinAngebot;

    @RequestMapping(method = RequestMethod.GET, value = REQUEST_URL)
    protected String bearbeiteAnfrage(Principal principal,
        @PathVariable("id") String angebotsId,
        @PathVariable("type") String type) {

        return bearbeiteAnfrageIntern(principal, angebotsId, type);
    }

    @Override
    protected BearbeiteDaten getBearbeiter() {
        return bDeaktiviereEinAngebot;
    }
}

@Controller
public class GET_AktiviereEinAngebot extends AktivierenDeaktivieren {

    private static final String REQUEST_URL = "/enable/{id}/{type}";

    @Autowired
    private B_AktiviereEinAngebot bAktiviereEinAngebot;

    @RequestMapping(method = RequestMethod.GET, value = REQUEST_URL)
    protected String bearbeiteAnfrage(Principal principal,
        @PathVariable("id") String angebotsId,
        @PathVariable("type") String type) {
        return bearbeiteAnfrageIntern(principal, angebotsId, type);
    }

    @Override
    protected BearbeiteDaten getBearbeiter() {
        return bAktiviereEinAngebot;
    }
}
```

LocalSharing

Die entstandene Oberklasse:

```
public abstract class AktivierenDeaktivieren extends BearbeiteEtwasAdmin {

    protected static String SUCCESS_VIEW = "redirect:../../angebote/";
    protected static String ERROR_VIEW = "redirect:angebote";

    protected Long angebotsId;
    protected String type;

    @Autowired
    private LadeEinAngebot ladeEinAngebot;

    protected String bearbeiteAnfrageIntern(Principal principal,
        String angebotsId, String type) {
        this.angebotsId = Long.valueOf(angebotsId);
        this.type = type;
        SUCCESS_VIEW = "redirect:../../angebot/" + angebotsId + "/" + type;

        return bearbeiteAnfrageIntern(principal);
    }

    @Override
    protected LadeDaten attribute() {
        ladeEinAngebot.setId(angebotsId);
        ladeEinAngebot.setType(type);
        return ladeEinAngebot;
    }

    @Override
    protected String getSuccessView() {
        return SUCCESS_VIEW;
    }

    @Override
    protected String getErrorView() {
        return ERROR_VIEW;
    }
}
```

Dadurch verringert sich der Wert der Duplications.

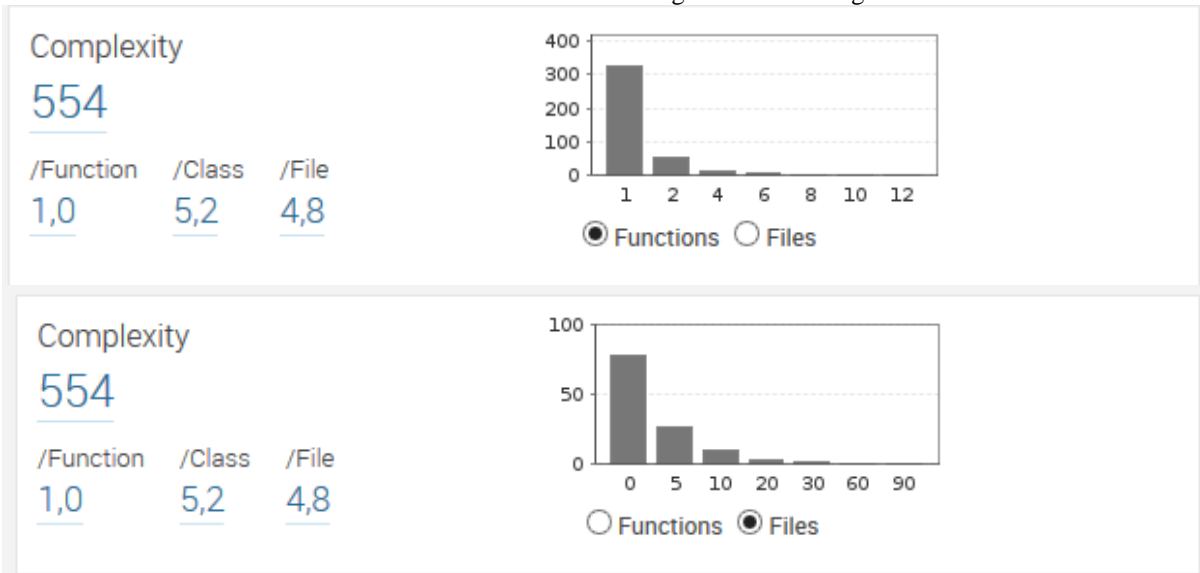
Duplications

3,6%

Lines	Blocks	Files
274 	11 	11 

5.2 Cyclomatic Complexity

Die Cyclomatic Complexity beschreibt die Komplexität eines Moduls, also beispielsweise einer Funktion oder einer Klasse. Dabei wird die Anzahl der linear unabhängigen Pfade eines Moduls berechnet. Diese Anzahl gibt damit also auch die maximale Anzahl der Testfälle an um einer vollständige Testabdeckung zu erreichen.



Durch obenstehende Abbildung der Complexity in Files konnte die Klasse *BenutzerServiceImpl* mit einer Complexity von über 30 identifiziert werden. Dies wurde folgendermaßen behoben.

Ursprünglicher Code:

```
@Override
public Benutzer findByAngebotsIdAndType(Long id, String type) {
    Benutzer benutzer = null;
    switch (type) {
        case "ausleihen":
            benutzer = ausleihartikelDao.findById(id).getBenutzer();
            break;

        case "tauschen":
            benutzer = tauschartikelDao.findById(id).getBenutzer();
            break;

        case "helfen":
            benutzer = hilfeleistungDao.findById(id).getBenutzer();
            break;
    }

    return benutzer;
}
```

LocalSharing

@Autowired

```
private Map<String, AngebotsDAO<?>> anbotDAOs;
```

```
public BenutzerServiceImpl() {  
    anbotDAOs = new HashMap<String, AngebotsDAO<?>>();  
    anbotDAOs.put("ausleihen", ausleihartikelDao);  
    anbotDAOs.put("tauschen", tauschartikelDao);  
    anbotDAOs.put("helfen", hilfeleistungDao);  
}
```

@Override

```
public Benutzer findById(long id) {  
    return benutzerDao.findById(id);  
}
```

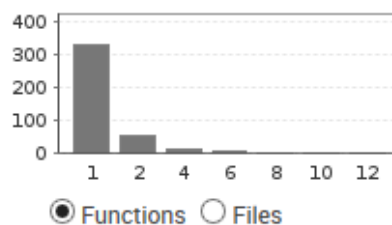
@Override

```
public Benutzer findByAngebotsIdAndType(Long id, String type) {  
    return anbotDAOs.get(type).findById(id).getBenutzer();  
}
```

Complexity

545 ↘

/Function	/Class	/File
1,1	5,0	4,6



Duplications

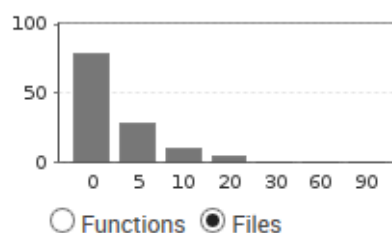
2,6% ↘

Lines	Blocks	Files
196 ↘	8 ↘	8 ↘

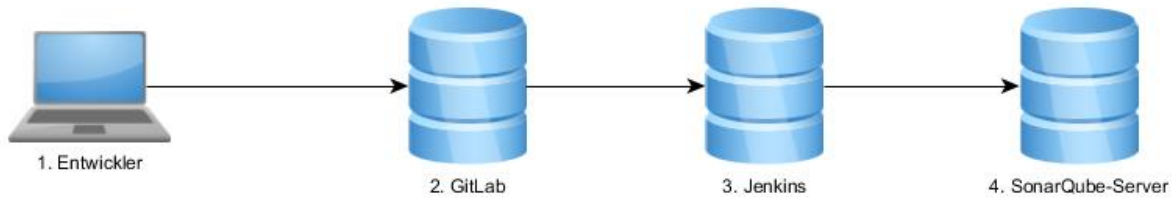
Complexity

545 ↘

/Function	/Class	/File
1,1	5,0	4,6



6. Test-Workflow



6.1 1.Entwickler

Commit Changes

Commit Changes to Git Repository

Commit message

SHARE-68: Test Coverage von 80% für Implementierung aus Sprint 1

Task-Url: <http://jira.it.dh-karlsruhe.de:8080/browse/SHARE-68>

Author: Johannes G <gruen.jojo.develop@gmail.com>

Committer: Johannes G <gruen.jojo.develop@gmail.com>


Files (1/1)


type filter text


Status	Path
<input checked="" type="checkbox"/>	localSharing/src/test/java/pandha/swe/localsharing/controller/AngebotControllerTest.java


Commit and Push Commit Cancel

6.2 2. GitLab

 Johannes pushed to branch **master** at Pandha / LocalSharing
7a409d49 #SHARE-68: Test Coverage von 80% für Implementierung aus Sprint 1

 Johannes pushed to branch **master** at Pandha / LocalSharing
4f768bf3 #SHARE-68: Test Coverage von 80% für Implementierung aus Sprint 1

 Johannes pushed to branch **master** at Pandha / LocalSharing
fe99119b #SHARE-68: Test Coverage von 80% für Implementierung aus Sprint 1

 Johannes pushed to branch **master** at Pandha / LocalSharing
21a58100 #SHARE-68: Test Coverage von 80% für Implementierung aus Sprint 1

LocalSharing

6.3 3. Jenkins

Build #118 (12.05.2015 13:20:10)

Started by GitLab push by Johannes G



Changes

1. new UI draft for login ([detail](#))



[Started by GitLab push by Johannes G](#)



Revision: dd48edadfcf2b83b6ccb3d2b6b3b1188cb57def

- origin/newUI



[Testergebnis](#) (Kein Test fehlgeschlagen.)

Modul-Builds

 [localSharing](#) 32 Sekunden





Nachgelagerte Builds

[LocalSharing_Function_Performance_Tests](#) (Keine)

6.4 4. SonarQube

```
[LocalSharing] $ /opt/sonar-runner/bin/sonar-runner -e -Dsonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?autoReconnect=true&useUnicode=true&characterEncoding=utf8
-Dsonar.host.url=http://gruen-hd.de:9000 ***** -Dsonar.projectBaseDir=/var/lib/jenkins/workspace/LocalSharing -Dsonar.scm.user.secure=jira
-Dsonar.sources=LocalSharing/src/main -Dsonar.junit.reportsPath=LocalSharing/target/surefire-reports -Dsonar.jacoco.reportPath=LocalSharing/target/coverage-repor
-Dsonar.binaries=LocalSharing/target/classes -Dsonar.projectVersion=1.0 -Dsonar.projectKey=jjc -Dsonar.scm.url=scm:git:http://gruen-hd.de:8085/pandha/localsharin
-Dsonar.tests=LocalSharing/src/test/java -Dsonar.scm.password.secured=rTsPHSR4 -Dsonar.projectName=LocalSharing
SonarQube Runner 2.4
Java 1.7.0_79 Oracle Corporation (64-bit)
Linux 3.13.0-042stab094.8 amd64
INFO: Error stacktraces are turned on.
INFO: Runner configuration file: /opt/sonar-runner/conf/sonar-runner.properties
INFO: Project configuration file: NONE
INFO: Default locale: "en_US", source code encoding: "US-ASCII" (analysis is platform dependent)
INFO: Work directory: /var/lib/jenkins/workspace/LocalSharing/.sonar
INFO: SonarQube Server 5.1
13:20:47.744 INFO - Load global repositories
13:20:47.875 INFO - Load global repositories (done) | time=133ms
13:20:47.878 INFO - Server id: 20150502195014
13:20:47.880 INFO - User cache: /var/lib/jenkins/.sonar/cache
13:20:47.889 INFO - Install plugins
13:20:48.093 INFO - Install JDBC driver
13:20:48.099 INFO - Create JDBC datasource for jdbc:mysql://localhost:3306/sonar?autoReconnect=true&useUnicode=true&characterEncoding=utf8
13:20:49.396 INFO - Initializing Hibernate
13:20:50.838 INFO - Load project repositories
13:20:51.222 INFO - Load project repositories (done) | time=384ms
13:20:51.222 INFO - Load project settings
13:20:51.468 INFO - Load technical debt model
13:20:51.513 INFO - Apply project exclusions
13:20:51.874 INFO - Scan localSharing
13:20:51.879 INFO - Load module settings
13:20:52.061 INFO - Load rules
```

LocalSharing

Unit Tests Coverage		Unit Test Success	
83,4% 		100,0%	
Line Coverage	Condition Coverage	Failures	Errors
84,3% 	79,2% 	0	0
		Tests	Execution Time
		267 	7.8 sec 