FaceRecognition
Test Plan

Version <1.2>

Revision History

Date	Version	Description	Author
22.05.2016	1.0	First work	Pascal Treptow
13.06.2016	1.1	Added Metrics	Carolina Mehret
13.06.2016	1.2	Added missing links	Carolina Mehret

Table of Contents

Test Plan

1. Introduction

1.1 Purpose

The purpose of the Iteration Test Plan is to gather all of the information necessary to plan and control the test effort for a given iteration. It describes the approach to testing the software, and is the top-level plan generated and used by managers to direct the test effort.

This *Test Plan* for the FaceRecognition supports the following objectives:

- The tests are aligned with the interface and functionality.

- The motivation is to develop an application with the fewest number of bugs.

1.2 Scope

This document addresses the following types and levels of testing:
- Unit Tests
- Functional Tests
- User Interface and Usability Tests
- Stress Test

1.3 Intended Audience

n/a

1.4 Document Terminology and Acronyms

tba

1.5 References

Title	Date
Overall Use Case Diagram	07.04.2016
Software Requirements Specification	07.04.2016
Software Architecture Documentation	11.05.2016
Use Case: Browse Image	04.05.2016
Use Case: Delete Image	04.05.2016
Use Case: Detect Face	04.05.2016
Use Case: Label Image	04.05.2016
Use Case: Upload Image	04.05.2016
Use Case (S2): Attributes	04.05.2016
Use Case (S2): Eye Detection	04.05.2016
Use Case (S2): Gender Classification	04.05.2016
Use Case (S2): Statistics	04.05.2016
SRS	12.06.2016
Codacy	-
Coveralls	-
SAD	-

SonarQube	-

1.6 Document Structure

n/a

2. Evaluation Mission and Test Motivation

2.1 Background

Testing supports all project members with feedback to their work. It ensures that added functionalities do not change the behavior of the application in a bad way.

2.2 Evaluation Mission

In general our mission is to improve our design and code quality.
This contains to find as many bugs as possible, find quality risks and so forth.

2.3 Test Motivators

Tests reduce bugs in new features and in existing features. Also tests are good documentation and reduce the cost of work if something needs to be changed.

3. Target Test Items

The listing below identifies those test items□software, hardware, and supporting product elements □that have been identified as targets for testing. This list represents what items will be tested.

- Operation System
- WebInterface
- Logic of application

4. Outline of Planned Tests

4.1/4.2/4.3 Missing

5. Test Approach

5.1 Initial Test-Idea Catalogs and Other Reference Sources
n/a

5.2 Testing Techniques and Types

5.2.1 Function Testing

Technique Objective:	This tests should ensure that the functions of our project are working correctly
Technique:	The program starts functions and compares the results with an expected condition
Oracles:	Test run via Travis CI
Required Tools:	Junit
Success Criteria:	If the result and the condition matches the test was successful, Unit tests don't fail
Special Considerations:	The Attribute Use Case can not be tested with Junit tests. With every commit the output changes due to the implementation of a machine learning algorithm which improves itself.

5.2.2 User Interface Testing

Technique Objective:	*Functionalities of the userintface got emulated*
Technique:	User interaction, like clicks, got emulated. The tool compared therefore the condition before and after the emulation.

Oracles:	The test are successful, if the userinterface acts like the emulation predicted it.
Required Tools:	Selenium/Gherkin
Success Criteria:	All tests need to run through successfully
Special Considerations:	

5.2.3 Load Testing

Technique Objective:	This test should provide that our project runs even when there is a lot of load
Technique:	Therefor the program generates a high number of thread, like 5000. Then there are definded calls which this threads can use, for calling a page, login in or uploading an image. When the program starts all this threads work at the same time and cause so a lot of load
Oracles:	After the test finished there a lot of different possibilities to view the result. One opportunity is to show the response time as a graph for every definded call.
Required Tools:	JMeter
Success Criteria:	In the result you can look up all call and can see if a call failed or not. You can also see for every defined call how much of the calls fails as a percentage.
Special Considerations:	tbd

6. Entry and Exit Criteria

7. Deliverables

7.1 Test Evaluation Summaries

Our summary of our tests are always up-to-date in Travis CI:
https://travis-ci.org/sapacaFaceRecognition/FaceRecognition

7.2 Reporting on Test Coverage

Our Test Coverage can be looked up at Coverall.io Our
Also on SonarQube: http://193.196.7.25/overview?id=sapaca

7.3 Perceived Quality Reports

 Codacy: https://www.codacy.com/app/chi340/FaceRecognition/dashboard
SonarQube: http://193.196.7.25/overview?id=sapaca

7.4 Incident Logs and Change Requests

We used SonarQube and Travis-Ci to trace and log our Incident Logs and Change Request. If a

Build fails on Travis CI it will automatically create a ticket in Jira.

7.5 Smoke Test Suite and Supporting Test Scripts

 n/a

7.6 Additional Work Products

n/a

    7.6.1   Detailed Test Results

n/a

    7.6.2   Additional Automated Functional Test Scripts

Our Repository can be found under GitHub

    7.6.3   Test Guidelines

n/a

    7.6.4   Traceability Matrices

 n/a

8. Testing Workflow

After a commit on github Travis-Ci tries to builds our project, after that Coveralls runs our definded tests and Codacy analysis it.
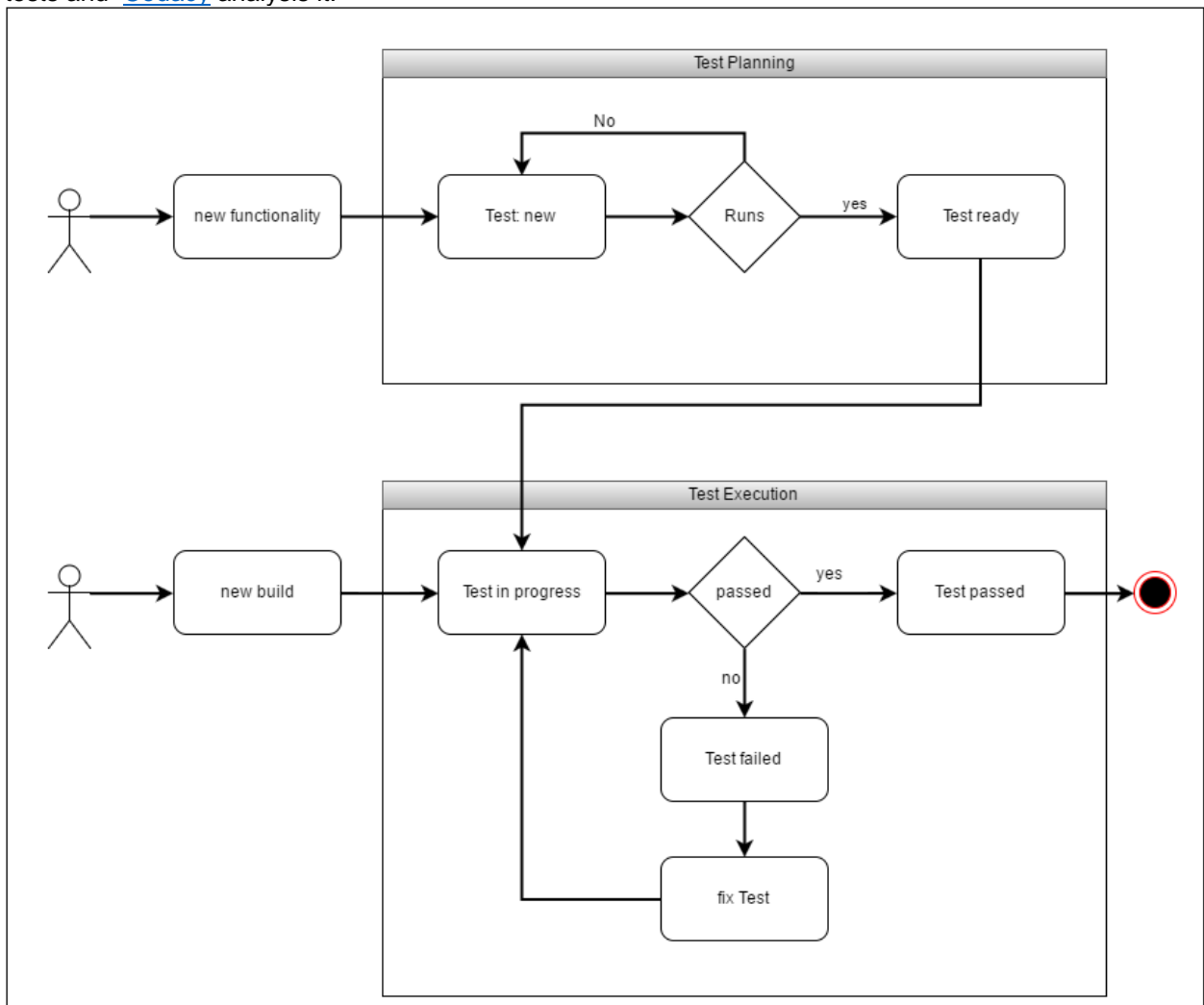

Abbildung 1: Testing Workflow
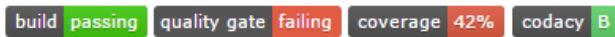
FaceRecognition

Abbildung 2: Badges from our GitHub

9. Environmental Needs

9.1 Base System Hardware

The following table sets forth the system resources for the test effort presented in this *Test Plan*.

System Resources		
Resource	**Quantity**	**Name and Type**
Database Server	1	MySQL Database
Database Name	1	sapaca
Client Test PCs		
Software		Windows with Java JDK 8, XAMPP

9.2 Base Software Elements in the Test Environment

The following base software elements are required in the test environment for this *Test Plan*.

Software Element Name	Version	Type and Other Notes
Windows	10	Operating System
XAMPP for Windows 5.6.14	5.6.14	Virtuell Server and Database
IntelliJ	Community Editon 2016.1.1	Editor
Spring Tool Suite	3.6.4	Editor
Internet Explorer		Internet Browser
Mozilla Firefox		Internet Browser

9.3 Productivity and Support Tools

The following tools will be employed to support the test process for this *Test Plan*.

Tool Category or Type	Tool Brand Name	Vendor or In-house	Version
Test Coverage Monitor	Coverall.io	Lemur Heavy Industries	
Code Climate, Metrics	Sonarqube	SonarSource S.A	4.5.7
Build Tool	Travis.ci	Travis CI GmbH	
Metrics	MetricsReloaded	MetricsReloaded (Plugin for IntelliJ)	1.7
Testing	JUnit		4.12
IDE	IntelliJ IDEA		2016.1.1
Project Management	JIRA		7.0.0

Tool Category or Type	Tool Brand Name	Vendor or In-house	Version
Automatic Deployment, auto-matically create jar-File	Travis.ci	Travis CI GmbH	

9.4 Test Environment Configurations

The following Test Environment Configurations needs to be provided and supported for this project.

Configuration Name	Description	Implemented in Physical Configuration
Average user configuration	Number of users who are accessing the application at the same time	50 Useres
Minimal configuration supported	Speed and power of the internet connection provided by the server host	
Test Computer	Specs of a test computer	Processor: AMD A10-8700P Radeon R6 1,8 GHz RAM: 8GB HDD: 1 TB OS: Windows 10

10. Responsibilities, Staffing, and Training Needs

10.1 People and Roles

This table shows the staffing assumptions for the test effort.

Human Resources		
Role	**Minimum Resources Recommended** **(number of full-time roles allocated)**	**Specific Responsibilities or Comments**
Test Manager	Sascha Kühne	Provides management oversight. Responsibilities include: • planning and logistics • agree mission • identify motivators • acquire appropriate resources • present management reporting • advocate the interests of test • evaluate effectiveness of test effort
Test Analyst	Carolina Mehret	Identifies and defines the specific tests to be conducted. Responsibilities include: • identify test ideas

Human Resources		
Role	**Minimum Resources Recommended** **(number of full-time roles allocated)**	**Specific Responsibilities or Comments**
		• define test details • determine test results • document change requests • evaluate product quality
Test Designer	Sascha Kühne	Defines the technical approach to the implementation of the test effort. Responsibilities include: • define test approach • define test automation architecture • verify test techniques • define testability elements • structure test implementation
Tester	Pascal Treptow	Implements and executes the tests. Responsibilities include: • implement tests and test suites • execute test suites • log results • analyze and recover from test failures • document incidents
Test System Administrator	Sascha Kühne	Ensures test environment and assets are managed and maintained. Responsibilities include: • administer test management system • install and support access to, and recovery of, test environment configurations and test labs
Database Administrator, Database Manager	Sascha Kühne	Ensures test data (database) environment and assets are managed and maintained. Responsibilities include: • support the administration of test data and test beds (database).
Designer	Pascal Treptow	Identifies and defines the operations, attributes, and associations of the test classes. Responsibilities include:

Human Resources		
Role	**Minimum Resources Recommended** **(number of full-time roles allocated)**	**Specific Responsibilities or Comments**
		• defines the test classes required to support testability requirements as defined by the test team
Implementer	Carolina Mehret	Implements and unit tests the test classes and test packages. Responsibilities include: • creates the test components required to support testability requirements as defined by the designer

10.2   Staffing and Training Needs

n/a

11. Iteration Milestones

n/a

12. Risks, Dependencies, Assumptions, and Constraints

Risk	**Mitigation Strategy**	**Contingency (Risk is realized)**
Project crashes and can´t be started	Bevor every commit it must be sure the project is working	• Testing bevor committing
The port for the server is occupied	Make sure no program is running on the port the server is using	• Give the server a special port • Restart your comupter
Changes influence existing functionalities	Chances, especially functionally chances, have to be tested bevor	• Reverse a commit • Testing bevor committing

13. Management Process and Procedures

n/a

## 14. Metrics

We used Sonar for Complexity and the IntelliJ Plugin MetricsReloaded for RFC. Blogpost metrics: https://sapacablog.wordpress.com/2016/05/29/metrics/ Blogpost

14.1   Metric 1: Complexity

We used Sonar for the Complexity metric and changed the class MainController.java. The following screenshot shows the before state with a complexity of 83 and complexity /functions of 3.8.
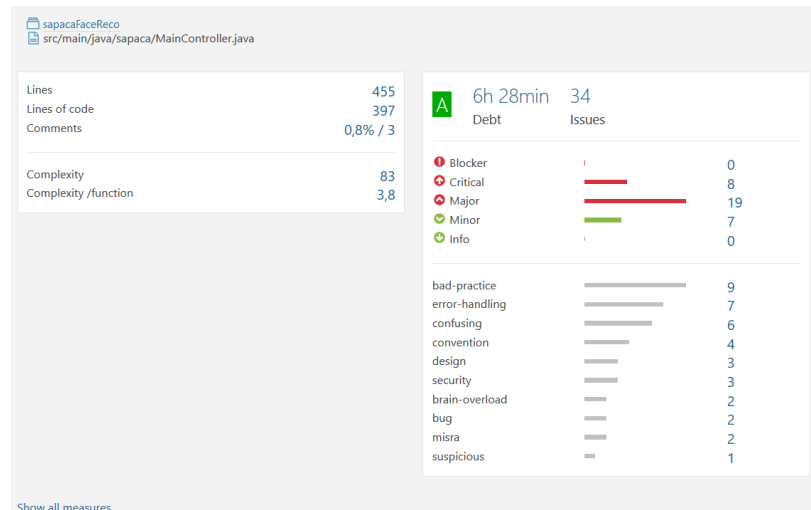
So we decided to reduce the complexity of the method (and the class) and improve the readability at the same time. Therefor we exported the calculation of the statistics from the MainController class into the class **Statistics** and created one new method in the MainController class. MainController source before improvement: https://github.com/sapacaFaceRecognition/FaceRecognition/blob/1f3b2420e7f126d715eac1f4dc7d51ace05674dc/complete/src/main/java/sapaca/MainController.java

Statistics source before improvement: https://github.com/sapacaFaceRecognition/FaceRecogntion/blob/5500b15184afa84aa98c903e8a638a1540f6670e/coplete/src/main/java/sapaca/Statistics.java

MainController source after improvement: https://github.com/sapacaFaceRecognition/FaceRecognition/blob/master/complete/src/main/java/sapaca/MainController.java

Statistics source after improvement: https://github.com/sapacaFaceRecognition/FaceRecognition/blob/master/complete/src/main/java/sapaca/Statistics.java

The changes led to an improvement of the complexity of the MainController Class. The following screenshot shows the complexity dropping from 83 to 74 and the complexity /function dropping to 3.9.
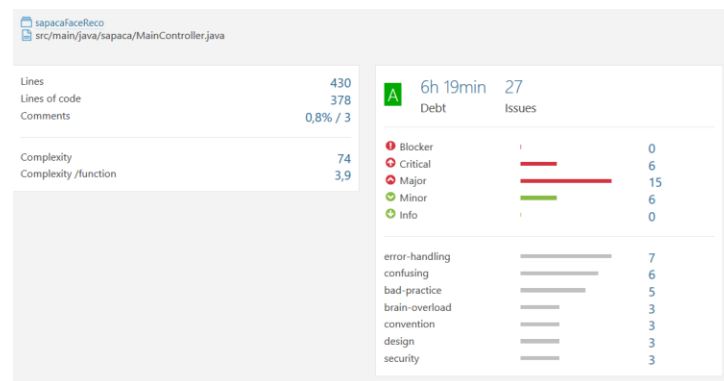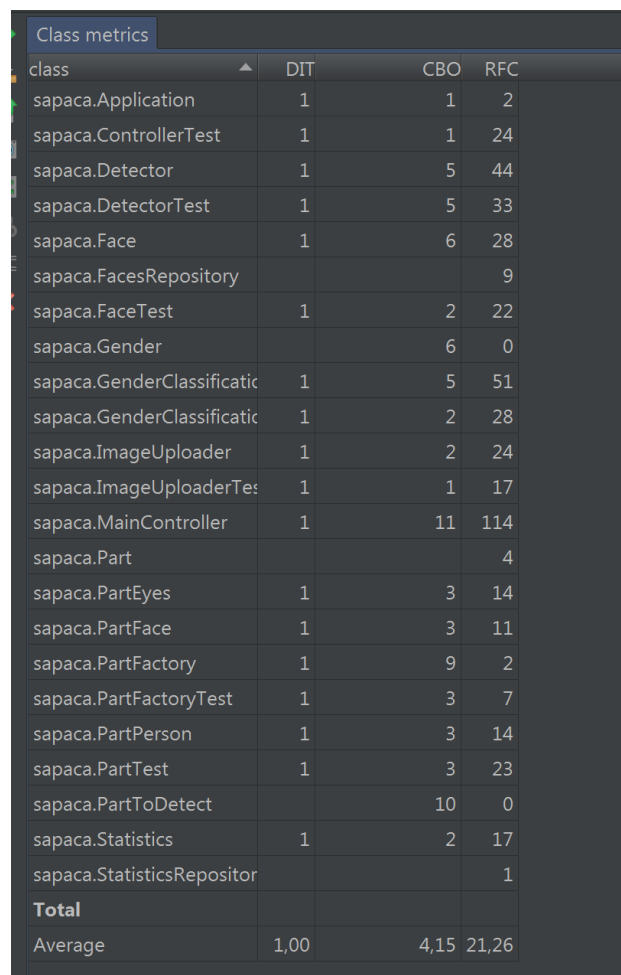
Metric 2: Response for Class (RFC)

We used the RFC (Response for Class) as the second metric to improve the testability and reduce complexity. A high RFC means high complexity. It can be hard to test the behaviour of the class and to debug problems since comprehending class behaviour requires a deep understanding of the potential interactions that objects of the class can have with the rest of the system. We are going to improve the GenderClassification class with the RFC metric.

The following screenshot shows the RFC of the classes before the changes. The Genderclassification class has a RFC of 51. The screenshot shows two GenderClassification classes, ignore the second one since it's just the test class.

class	DIT	CBO	RFC
sapaca.Application	1	1	2
sapaca.ControllerTest	1	1	24
sapaca.Detector	1	5	44
sapaca.DetectorTest	1	5	33
sapaca.Face	1	6	28
sapaca.FacesRepository			9
sapaca.FaceTest	1	2	22
sapaca.Gender		6	0
sapaca.GenderClassificatic	1	5	51
sapaca.GenderClassificatic	1	2	28
sapaca.ImageUploader	1	2	24
sapaca.ImageUploaderTes	1	1	17
sapaca.MainController	1	11	114
sapaca.Part			4
sapaca.PartEyes	1	3	14
sapaca.PartFace	1	3	11
sapaca.PartFactory	1	9	2
sapaca.PartFactoryTest	1	3	7
sapaca.PartPerson	1	3	14
sapaca.PartTest	1	3	23
sapaca.PartToDetect		10	0
sapaca.Statistics	1	2	17
sapaca.StatisticsRepositor			1
Total			
Average	1,00	4,15	21,26

To improve the RFC we first removed methods which were used for the gender classification in our first try of implementing the gender classification. The methods are useless because our current implementation takes care of the steps the method executes. The following screenshots show the removed methods:

```
70   -        private static void stitcherArgs() {
71   -                String[] args = new String[2];
72   -                int usageLeft = 0;
73   -                if (args.length == 0) {
74   -                        printUsage();
75   -                        usageLeft = -1;
76   -                }
77   -                for (int i = 2; i < args.length; i++) {
78   -                        if ("--help".equals(args[i])) {
79   -                                printUsage();
80   -                                usageLeft = 1;
81   -                        } else if ("--output".equals(args[i])) {
82   -                                resultName = args[i + 1];
83   -                                i++;
84   -                        } else {
85   -                                Mat img = imread(args[i]);
86   -                        }
87   -                }
88   -        }
89   -
```

```
43   -        private static Gender classifyGender(int retval) {
44   -                if (retval != 0) {
45   -                        System.out.println("Something went wrong...");
46   -                }
47   -
48   -                Mat pano = new Mat();
49   -                Stitcher stitcher = Stitcher.createDefault(tryUseGpu);
50   -
51   -                int status = stitcher.stitch(imgs, pano);
52   -
53   -                if (status != Stitcher.OK) {
54   -                        System.out.println("Something went wrong...");
55   -                }
56   -                imwrite(resultName, pano);
57   -                stitcherVar++;
58   -                return gender;
59   -        }
60   -
```

Then we removed the setAttributesForFalseClassification method, which sets all attributes to 0 in case of a failing classification and put the logic into the constructor. The following screenshots show these steps.

```
128  -        private void setAttributesForFalseClassification() {
129  -                genderConfidence = 0.0;
130  -                gender = Gender.UNKNOWN;
131  -                raceConfidence = 0.0;
132  -                race = "";
133  -                age = 0;
134  -                ageRange = 0;
135  -        }
```

```
  32     21                  public GenderClassification(IplImage image) {
  33     22                          this.image = image;
         23    +                     genderConfidence = 0.0;
         24    +                     gender = Gender.UNKNOWN;
         25    +                     raceConfidence = 0.0;
         26    +                     race = "";
         27    +                     age = 0;
         28    +                     ageRange = 0;
  34     29                          ImageUploader imageUploader = new ImageUploader(image);
  35     30                          String url = imageUploader.getUploadedUrl();
  36     31                          httpRequest(url);
  37     32                  }
```

Then we replaced the verifyResults() method with an if-statement. The whole method is small and just verifies that the classification didn't fail. The following screenshots show these steps.

```
-           private boolean verifyResults() {
-                   String json = response.getBody().toString();
-                   if (json.contains("Male") || json.contains("Female")) {
-                           return true;
-                   }
-                   return false;
-           }
```

```
if ((json.contains("Male") || json.contains("Female"))) {
        jsonGetGender();
        jsonGetRace();
        jsonGetAge();
```

We reduced the rfc to 39 with these steps as you can see in the following screenshot.. You can see all the changes in the Git Commit https://github.com/sapacaFaceRecognition/FaceRecognition/commit/fae3518183ccbc0333a032d0f099c2faf3e376a2

Class metrics

class	DIT	CBO	RFC
sapaca.Application	1	1	2
sapaca.ControllerTest	1	1	24
sapaca.Detector	1	5	44
sapaca.DetectorTest	1	5	33
sapaca.Face	1	6	28
sapaca.FacesRepository			9
sapaca.FaceTest	1	2	22
sapaca.Gender		6	0
sapaca.GenderClassification	1	4	39
sapaca.GenderClassificationTest	1	2	28
sapaca.ImageUploader	1	2	24
sapaca.ImageUploaderTest	1	1	17
sapaca.MainController	1	11	114
sapaca.Part			4
sapaca.PartEyes	1	3	14
sapaca.PartFace	1	3	11
sapaca.PartFactory	1	9	2
sapaca.PartFactoryTest	1	3	7
sapaca.PartPerson	1	3	14
sapaca.PartTest	1	3	23
sapaca.PartToDetect		10	0
sapaca.Statistics	1	2	17
sapaca.StatisticsRepository			1
Total			
Average	1,00	4,10	20,74

What we didn't change

We have not improved the following because it's wrong. SonarQube didn't recognized the Spring annotation **@RequestMapping** and therefor thinks the method is useless and can be replaced by a constant value:

```
@RequestMapping(value = "/home.html", method = RequestMethod.GET)
public String homeWithoutAuthentication() {
        return "home";
```

Remove this method and declare a constant for this value. ...   vor 22 Tagen ▾   L72 🔀 ▼▾
🚫 Major   ○ Open   Not assigned   Not planned   5min debt   🏷 confusing

```
}
```

We have also ignored the warning that I should declare the 4 int variables on separate lines, because we don't think that it's usefull. If the variables are completly different and maybe even have different values the warning is okay, but in this case we think our implementation is better:

```
int germany = 0, england = 0, usa = 0, france = 0;
```

Declare "england" on a separate line. ...   vor 5 Tagen ▾   L404 🔀 ▼▾
✅ Minor   ○ Open   Not assigned   Not planned   2min debt   🏷 convention

Declare "usa" on a separate line. ...   vor 5 Tagen ▾   L404 🔀 ▼▾
✅ Minor   ○ Open   Not assigned   Not planned   2min debt   🏷 convention

Declare "france" on a separate line. ...   vor 5 Tagen ▾   L404 🔀 ▼▾
✅ Minor   ○ Open   Not assigned   Not planned   2min debt   🏷 convention