UNIVERSITY OF CASTILLA-LA MANCHA

ESI CIUDAD REAL

SOFTWARE ENGINEERING

GRADE ON COMPUTER SCIENCE

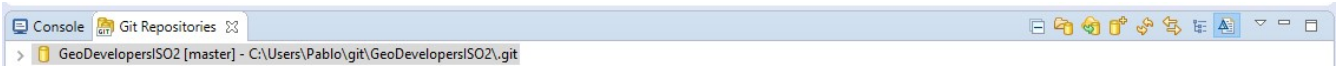
Testing Lab

Enrique Garrido Pozo
Pablo Mora Herreros
Adrián Ollero Jiménez
Beka Bekeri
Daniele Acquaviva

Contents

1 Setup

First of all, we clone the repository in eclipse. We copy the url, and with the plugin of *Egit* we can get it.



Then, we import it as a maven project to update the project dependencies.

1- We add in the *pom* file outside the build element a reporting element, then a plugins element and the certain plugin. Later, the *Surefire Plugin*, which is used during the test phase of the build lifecycle to execute the unit tests of an application. We are going to use for testing *Jacocoç* too.

Spotif/pom.xml

```
115
116    <reporting>
117        <plugins>
118            <plugin>
119                <groupId>org.apache.maven.plugins</groupId>
120                <artifactId>maven-project-info-reports-plugin</artifactId>
121            </plugin>
122            <plugin>
123                <groupId>org.apache.maven.plugins</groupId>
124                <artifactId>maven-surefire-report-plugin</artifactId>
125                <configuration>
126                    <skipEmptyReport>false</skipEmptyReport>
127                </configuration>
128            </plugin>
129            <plugin>
130                <groupId>org.jacoco</groupId>
131                <artifactId>jacoco-maven-plugin</artifactId>
132                <reportSets>
133                    <reportSet>
134                        <reports>
135                            <report>report</report>
136                        </reports>
137                    </reportSet>
138                </reportSets>
139            </plugin>
```

We tried to join all the information of jacoco in one file but we couldn't achieve this. So, to see the information of the testing, you have to enter in each module of the project.

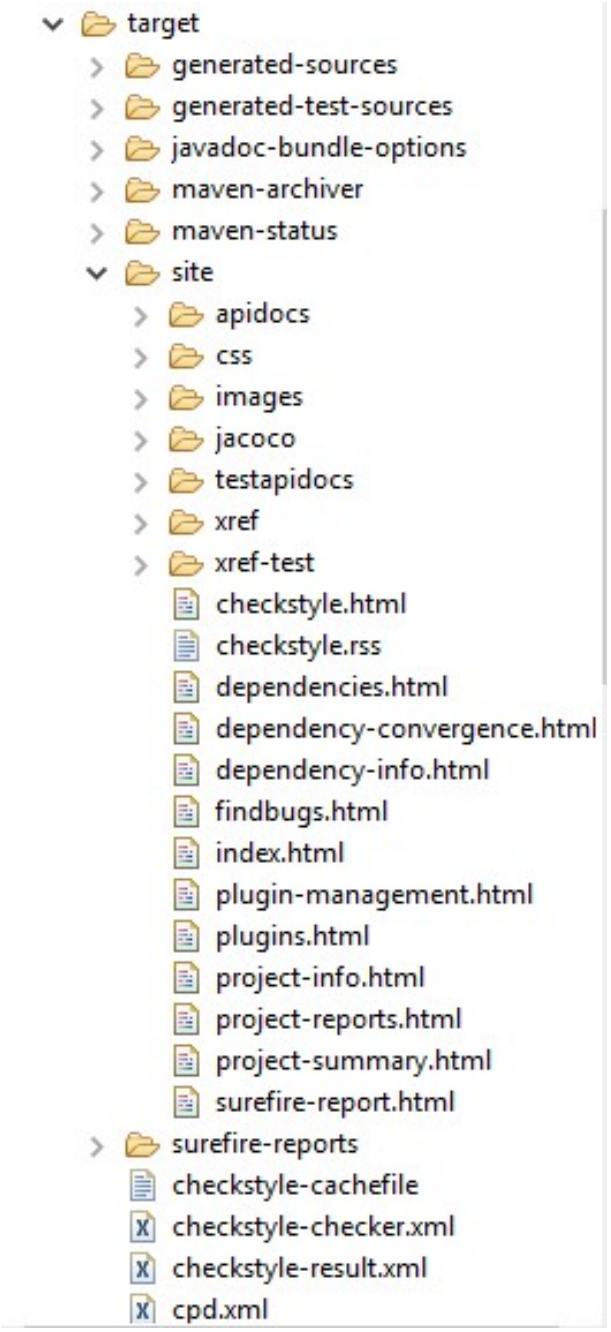
The *pom.xml* will be more or less like this:

```
<build>
    ...

    <plugins>
        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.7.9</version>
            <executions>
                <execution>
                    <id>default-prepare-agent</id>
                    <goals>
                        <goal>prepare-agent</goal>
                    </goals>
                </execution>
                <execution>
                    <id>default-prepare-agent-integration</id>
                    <goals>
                        <goal>prepare-agent-integration</goal>
                    </goals>
                </execution>
                <execution>
                    <id>default-report</id>
                    <goals>
                        <goal>report</goal>
                    </goals>
                </execution>
                <execution>
                    <id>default-report-integration</id>
                    <goals>
                        <goal>report-integration</goal>
                    </goals>
                </execution>
                <execution>
                    <id>default-check</id>
                    <goals>
                        <goal>check</goal>
                    </goals>
                    <configuration>
                        <rules>
                            <!-- implementation is needed only for Maven 2 -->
                            <rule implementation="org.jacoco.maven.RuleConfiguration">
                                <element>BUNDLE</element>
                                <limits>
                                    <!-- implementation is needed only for Maven 2 -->
                                    <limit implementation="org.jacoco.report.check.Limit">
                                        <counter>COMPLEXITY</counter>
                                        <value>COVEREDRATIO</value>
                                        <minimum>0.60</minimum>
                                    </limit>
                                </limits>
                            </rule>
                        </rules>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
```

In addition, we have to change the version of *junit* in order to be able to use *Jacoco*.

For executing it, we run *jacoco:prepare-agent site:site post-integration-test* and all the html code for the site will be generated after all tests are executed. The outgoing site is in *target/site/index.html*.

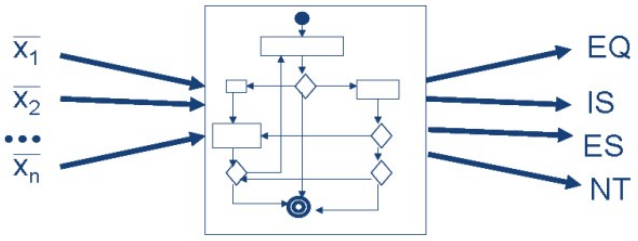


2 Testing Plan

We are going to concentrate in check the project with unitary test. Each one is a method with an alternative result. At the beginning we put the getters but in a meeting we decide to comment this part of the tests.

We use the concept of coverage, usually with a percentage, to express the range with a set of proofs test suite, that covers the structure or architecture in question.
It's normal using support tools to calculate the covert of the code in the case of Components proofs or integration of components proofs (for example, when we draw the hierarchy of calls between elements). When we look for the internal behaviour, these proofs we denominate white-box testing which are we are going to implement.

- **White box or *Structural* tests**
  - Internal structure is known during testing
  - Parts of the program that are executed are taken into account during testing → *COVERAGE*
- **Triangle program example:**



3 Test values

Here, we can find some test values using appropiates values with MINIMAX technique.Test cases should be built to generate values for every: equivalence partitioning, boundary values and error-guessing. All this information is in the Excel called *Testing values*. This is an example:

iteration	class	method	parameters	equivalence classes	values
1	Song	Song	idSong	(-inf,100000000aaa)	5704hbc
				[100000000aaa,999999999zzz]	306578981qwe
				(999999999zzz,+inf)	28469863526opt
					99999999zzz
					100000000aaa
					100000000aab
					999999999zzy
					999999999zzz
					1000000000aaa
			title	Song's name	Despacito
					null
					String with more than 255 characters
			singer	Song's singer	Luis Fonsi ft. Daddy Yankee
					null
					String with more than 255 characters
			price	(-inf,0)	-8.00€
				[0, +inf)	2.99€
					-0.01€
					0.00€
					0.01€
			date	dd--mm-yyyy	12-01-2017

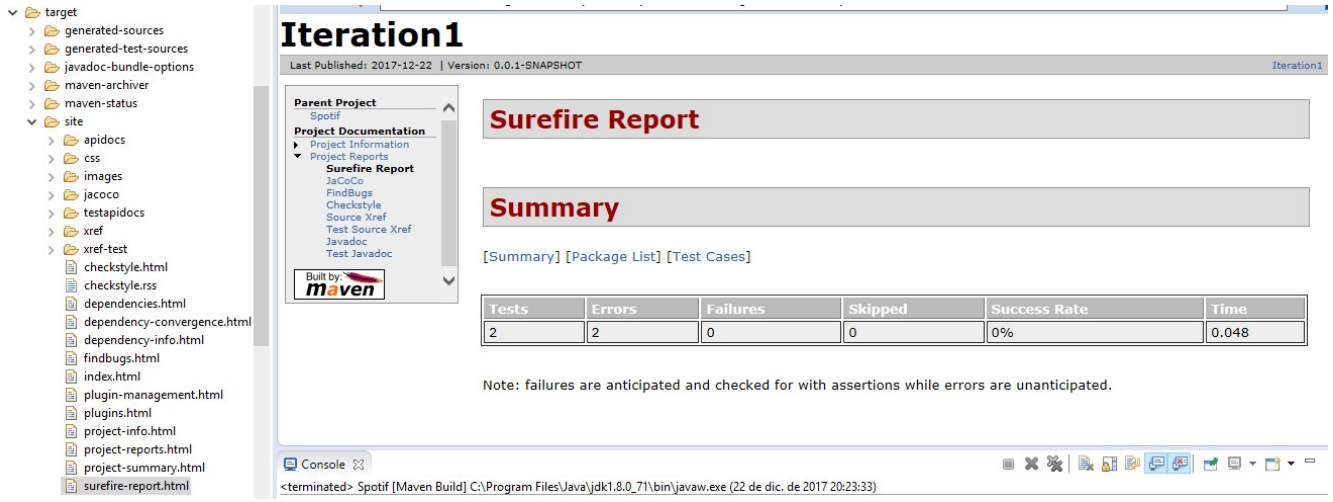
Due to we thought we will obtain a better code, we don't have more methods and we couldn't generate the decision or MC-MD parts.

Iteration	Num. Test classes	Num. Test methods
1	4	2
2	1	0
3	4	24
4	2	2
5	3	3
6	1	1
7	2	16
8	2	4

Table 1: Tests

To see this test classes you have to enter in the module, src, test and java. So, the total number of test classes is 19 and for methods is 52.

To sum up this part, generating the reports we can see all this information. This is an example:



This process is the same in each module or iteration. Some tests have errors because we haven't implemented the database yet.

4 Combination strategies

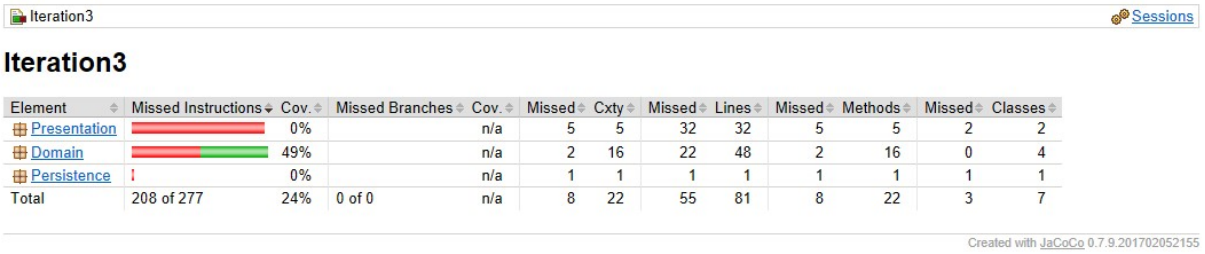
To generate all test cases we have defined all possible combinations that can be made with all values. This is done by combining each one of the values with the whole combination of the rest of values. This can be seen in the excel document named *All combinations*.
As this number of combination is huge, we have decided to assure that the use covert is used. This means that each value must be used at least once. This new combination have been defined in *Testing values*. This is an example:

Iteration 1				
Class: Song				
Method: Constructor				
idSong	Title	Singer	Price	Date
306578981qwe	Despacito	Luis Fonsi ft Daddy Yankee	2.99	12-01-2017
5704hbc	null	String with more than 255 characters	-8.00	null
28469863526opt	String with more than 255 characters	null	-0.01	12-01-2017
99999999zzz	Despacito	Luis Fonsi ft Daddy Yankee	0.00	12-01-2017
100000000aaa	Despacito	Luis Fonsi ft Daddy Yankee	0.01	12-01-2017
100000000aab	Despacito	Luis Fonsi ft Daddy Yankee	2.99	null
999999999zzy	Despacito	Luis Fonsi ft Daddy Yankee	2.99	12-01-2017
999999999zzz	Despacito	Luis Fonsi ft Daddy Yankee	2.99	12-01-2017
1000000000aaa	Despacito	Luis Fonsi ft Daddy Yankee	2.99	12-01-2017

This combinations are passed in the *setUp* method or as a parameter in the code of testing.

5 Coverage

In the *Jacoco folder* in the index.html, we can see the coverage of each iteration. We have been oriented only in the *domain* package as we can see in the reports. This is an example:


Iteration3

Sessions

Iteration3

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Presentation		0%		n/a	5	5	32	32	5	5	2	2
Domain		49%		n/a	2	16	22	48	2	16	0	4
Persistence		0%		n/a	1	1	1	1	1	1	1	1
Total	208 of 277	24%	0 of 0	n/a	8	22	55	81	8	22	3	7

Created with JaCoCo 0.7.9.201702052155

6 Conclusions

Once we have seen all the reports generated and our testing values are combined, we can start to make changes but, with the lack of code we could't do it. If you have any problem seeing the reports, download all the *testing* branch.