university of groningen // PASTAQ

Testing Document

Software Engineering Project Feb 2022
Students: Tudor Dragan, Teresa Ferreira, Cristian Iacob, Mohammed Nacer Lazrak, Björn Schönrock, Dominic Therattil & Kaitlin Vos
First Supervisor: Lars Andringa
Second Supervisor: Prof Dr A. Capiluppi

Contents

1 User Acceptance Test

For the user acceptance testing, we have created a table with all of the tests that need to be fulfilled. Each acceptance requirement is uniquely identifiable in the `ID` column. The next column, `Acceptance requirements`, holds the requirements that need to be tested which are taken from our requirements document. The `Critical` column indicates whether the acceptance requirement is critical or not. All the must-haves are critical. The `Test result` column contains color-coded checkmarks which are associated to each of our testers and they are ticked when the tester has verified the acceptance requirement. The `Comments` column contains comments made by the testers about each acceptance requirement.

Each of our testers are active in the target field and have been carefully selected. In our requirements document, we have specified our general users as researchers, data scientists, chemists and biologists. Nowadays, pure data scientists, chemists and biologists are rare since interdisciplinary expertise are strongly encouraged in these fields. Our testers can been seen as a subset of the aforementioned general users, who cover all of these fields. Our testers are the following:

Dr. J.C. (Karin) Wolters
Occupation: Principal investigator
Fields of expertise: Proteomics/Mass Spectrometry/Biochemistry

Prof Dr P.L. (Peter) Horvatovich
Occupation: Full Professor in Computational Mass Spectrometry, Head of Department of Analytical Biochemistry
Fields of expertise: Computational mass spectrometry, bioinformatics, proteogenomics data integration, biomarker discovery, trace analysis, mass spectrometry imaging, detection of irradiated food, computational proteomics, analytical chemistry

Dr. H.P. (Hjalmar) Permentier
Occupation: Head of the Interfaculty Mass Spectrometry Center
Fields of expertise: Biochemistry & Molecular Biology and Analytical Chemistry

A. Sánchez Brotons
Fields of expertise: Analytical Biochemistry

✓ Dr. J.C. (Karin) Wolters
✓ Prof Dr P.L. (Peter) Horvatovich
✓ dr. H.P. (Hjalmar) Permentier
✓ A. Sánchez Brotons

Table 1: User Acceptance Tests

ID	Acceptance requirement	Critical	Test result	Comments
Tested by three tester(s)				
UAT1.1	When the reading and loading of the configuration file fails, the user must be informed.	Yes	✓ ✓ ✓	

Continued on next page

Table 1: User Acceptance Tests (Continued)

ID	Acceptance requirement	Critical	Test result	Comments
UAT1.2	The GUI must allow the user to browse for the path to the MSFragger .jar file.	Yes	✓ ✓ ✓	Needs to be clear that you can only browse and not input the path manually
UAT1.3	The GUI must allow the user to browse for the path to the idconvert .exe file.	Yes	✓ ✓ ✓	
UAT1.4	The GUI must allow the user to browse for the path to the .params file.	Yes	✓ ✓ ✓	
UAT1.5	The GUI must accept .mzID files.	Yes	✓ ✓ ✓	
UAT1.6	The GUI must accept .mgf files.	Yes	✓ ✓ ✓	
UAT1.7	The GUI must have tool-tips for the parameters to explain its functionalities to the user.	Yes	✓ ✓ ✓	Good when students need to use the research tool and need a reminder for certain parameter terms but the tooltips should be consistent
UAT1.8	The GUI must offer a drop down navigation menu where the user can reset parameters to their default values.	Yes	✓ ✓ ✓	Depending on the instrument type, the default parameters need to be defined
UAT1.9	The GUI must have a button to select all the given files.	Yes	✓ ✓ ✓	Good feature because the GUI is mostly used with a long list of files
UAT1.10	The run button must be disabled if there are no input files or if at least one file path is invalid.	Yes	✓ ✓ ✓	
UAT1.11	The GUI should have a shortcut to create a new project.	No	✓ ✓ ✓	Good that the industry standardized shortcuts are provided
UAT1.12	The GUI should have a shortcut to open a project.	No	✓ ✓ ✓	
UAT1.13	The GUI should have a shortcut to save the opened project.	No	✓ ✓ ✓	
UAT1.14	The GUI should have a shortcut to remove the selected files.	No	✓ ✓ ✓	
UAT1.15	The GUI should have a shortcut to select all the given files.	No	✓ ✓ ✓	
UAT1.16	The GUI should have a means of inputting files through a drag and drop feature.	No	✓ ✓ ✓	Feedback when something has been dropped desired

Continued on next page

Table 1: User Acceptance Tests (Continued)

ID	Acceptance requirement	Critical	Test result	Comments
UAT1.17	MacOS users should be informed that the automatic file processing is not supported on their operating system.	No	✓ ✓ ✓	
UAT1.18	Attempting to close the GUI must display a pop up dialog box if there are any unsaved changes.	Yes	✓ ✓ ✓	Preferred over automatic saving (because some changes are intentionally not saved)
UAT1.19	Attempting to close the GUI must display a pop up dialog box to clarify, whether to save the project and close.	Yes	✓ ✓ ✓	
UAT1.20	Attempting to close the GUI must display a pop up dialog box to clarify, whether to discard any changes and close.	Yes	✓ ✓ ✓	
UAT1.21	Attempting to close the GUI must display a pop up dialog box to clarify, whether cancel the action and not close the GUI.	Yes	✓ ✓ ✓	
UAT1.22	The parameters tab must be reorganized in a way that only one parameter category is shown at a time, including the parameter category description.	Yes	✓ ✓ ✓	Cleaner, selectable steps with a clear structure; description of each category at the top is really useful
UAT1.23	The user must be able to navigate through the parameter categories.	Yes	✓ ✓ ✓	
UAT1.24	The user must be sufficiently informed about all the components of the GUI.	Yes	✓ ✓ ✓	Information flow could still be increased, rated 7 out of 10; with better information flow 8-9 out of 10
UAT1.25	Running the GUI must not display the cancel button after completion.	Yes	✓ ✓ ✓	
UAT1.26	The GUI window must be resizable.	Yes	✓ ✓ ✓	Possibility greatly appreciated
UAT1.27	The GUI must have the top row of buttons (New project, Open project, Save and Save as) represented by a drop down navigation menu instead.	Yes	✓ ✓ ✓	
UAT1.28	The GUI should provide the possibility to access the PASTAQ-GUI guide found under the following link [5].	No	✓ ✓ ✓	Guide not the first thing one would look at but definitely useful
UAT1.29	The GUI window should support a fullscreen option.	No	✓ ✓ ✓	

Continued on next page

Table 1: User Acceptance Tests (Continued)

ID	Acceptance requirement	Critical	Test result	Comments
UAT1.30	Support for Windows should be maintained	No	✓ ✓ ✓	
UAT1.31	The GUI could have a logo.	No	✓ ✓ ✓	Logo is liked and perceived as fun
UAT1.32	The GUI could have a splash screen, which appears before the GUI.	No	✓ ✓ ✓	Chemistry related scribbles and colors made a good impression and is overall really liked
UAT1.33	The GUI could have the logo as its icon.	No	✓ ✓ ✓	
UAT1.34	The GUI could display the logo in the main window.	No	✓ ✓ ✓	
UAT1.35	The GUI's elements could have their color changed to become more intuitive.	No	✓ ✓ ✓	
UAT1.36	The GUI could have a dark mode theme which offers a darker color palette.	No	✓ ✓ ✓	
Tested by two tester(s)				
UAT1.37	An executable file for Windows should automatically be generated using Github CI.	No	✓ ✓	Easily downloadable applications are very much appreciated
UAT1.38	The user should be able to keep track of their own progress while setting the parameters by the use of checkboxes.	No	✓ ✓	No additional value, misleading
Tested by one tester(s)				
UAT1.39	The configuration file must be accessible form any project.	Yes	✓	
UAT1.40	The configuration file must be loaded when the user opens a project or creates a new project.	Yes	✓	
UAT1.41	Support for Linux should be maintained	No	✓	
UAT1.42	The Github CI pipeline could run a test suite for every new release to ensure that the user does not get incorrect code.	No	✓	
UAT1.43	When a path of the identification or quantification file is not valid, the user must be informed.	No	✓	

1.1 Feedback implementations

Based on the acceptance testing, the following things have been changed:

- UAT1.2 - UAT1.4: If there is no path to any of the paths required for the `paths` tab, there is a placeholder text, which informs the user that the paths can only be inputted through browsing.
- UAT1.16: The drag and drop area in the `QDialog` changes to a green color when something has been successfully dropped, and additionally, the `QDialog` contains a `QListWidget` which displays all the files that have either been dropped or selected during the browsing alternative. This way the user is able to see which files will be loaded into the GUI.
- UAT1.38: This has been received in a negative way by all our testers and the clients. The main parameters that need to be set are the `Instrument settings` and the rest of the parameters are solely for fine-tuning purposes. On top of that, the checkboxes were confusing and not interpreted in the way we have intended. This is why we have taken all this feedback into consideration and decided to remove this feature.

2 Traceability Matrix

Our testing has been conducted through unit testing and acceptance testing. Our unit tests can be recognized through our unique identifiers that range from $T1$ to $T5$. The identifiers correspond to our files as follows, T1=`app.py`, T2/T4=`parameters.py/buttons.py`, T3=`files.py` and T5=`pipeline.py`. Our acceptance tests have the same identifiers as in the previous section, namely `UAT`.
For testing, 103 unit tests have been developed and 43 acceptance tests. We have also used SonarQube to evaluate one of our requirements, which can be identified by the `SQT` identifier

Legend:

- Requirement: Refers to the requirements from the Requirements document.
- Requirement status: Specifies whether said requirement was implemented or not.
- Module used: Refers to whether said requirement is part of the MVC pattern the GUI follows, the Continuous Integration requirements or the Operating System requirements. (M - Model, V - View, C - Controller, CI - Continuous Integration, OS - Operating System)
- Files affected: Refers to the code file associated with said requirement.
- Tests: Refers to the tests which correlate with said requirement.
- Test Status: Specifies whether said requirement was tested or not.

Table 2:

Requirement	Requirement status	Module used	Files affected	Tests	Test Status
Functional requirements					
R1.1	Implemented	M	app.py, files.py, parameter.py, pipeline.py	T5.1, T5.2, T5.4	Tested
R1.2	Implemented	M	files.py, parameters.py	T3.1-T3.5, T3.15-T3.22, T3.34-T3.40	Tested
R1.3	Implemented	M	files.py	T3.16, T3.23-3.28, T3.42-T3.43, T3.45-T3.46	Tested
R1.4	Implemented	M	files.py, parameters.py	T3.14, T3.17-T3.22	Tested
R1.5	Implemented	M	app.py, files.py, parameter.py, pipeline.py	T5.3	Tested
R1.6	Implemented	M	app.py, files.py	T1.29-T1.31, T1.37-T1.41, T3.6-T3.14	Tested
R1.7	Implemented	M	app.py, files.py, parameter.py	T1.29, UAT1.39	Tested
R1.8	Implemented	C	app.py, files.py, parameter.py	T1.30, T1.31,T1.35, T1.39, UAT1.40	Tested
R1.9	Implemented	M	app.py, files.py, parameter.py	T3.14	Tested
R1.10	Implemented	M	app.py, files.py, parameter.py, pipeline.py	T5.1-T5.3	Tested
R1.11	Implemented	M	app.py, files.py	-	Not Tested
R1.12	Implemented	M	files.py	T3.33, T3.44	Tested
R1.13	Implemented	MV	app.py, files.py	UAT1.43	Not Tested
R1.14	Implemented	MV	app.py, files.py, parameter.py, pipeline.py	T3.18, T3.21, T3.24, T3.27, T3.32, T3.41, T3.46	Tested
R1.15	Implemented	MV	app.py	T1.32, T1.33, UAT1.1	Tested
R1.16	Implemented	V	app.py, parameter.py	UAT1.2	Tested
R1.17	Implemented	V	app.py, parameter.py	UAT1.3	Tested

Continued on next page

Table 2: (Continued)

Requirement	Requirement status	Module used	Files affected	Tests	Test Status
R1.18	Implemented	V	app.py, parameter.py	UAT1.4	Tested
R1.19	Implemented	M	app.py, parameter.py	UAT1.5	Tested
R1.20	Implemented	M	app.py, parameter.py	UAT1.6	Tested
R1.21	Implemented	V	app.py, buttons.py, parameter.py	UAT1.7	Tested
R1.22	Implemented	VC	app.py	T1.3, T1.5, T1.13, T1.16, T1.17, UAT1.8	Tested
R1.23	Implemented	VC	app.py, files.py, parameter.py	UAT1.9, 1.12	Tested
R1.24	Implemented	MV	app.py, files.py	UAT1.10	Tested
R2.1	Not Implemented	MVC	app.py, files.py, parameter.py, pipeline.py	-	Not Tested
R2.2	Implemented	VC	app.py, files.py, parameter.py	UAT1.11	Tested
R2.3	Implemented	VC	app.py, files.py, parameter.py	UAT1.12	Tested
R2.4	Implemented	MVC	app.py, files.py, parameter.py	UAT1.13	Tested
R2.5	Implemented	C	app.py, files.py, parameter.py	UAT1.14	Tested
R2.6	Implemented	C	app.py, files.py, parameter.py	UAT1.15	Tested
R2.7	Implemented	VC	app.py, files.py, parameter.py	UAT1.16	Tested
R2.8	Implemented	V	-	UAT1.17	Tested
R2.9	Implemented	CI	build.yml	UAT1.37	Tested
R3.1	Not implemented	CI	build.yml	-	Not Tested
R3.2	Implemented	CI	build.yml	-	Not Tested
R3.3	Implemented	CI	test.yml	UAT1.42	Tested
R3.4	Not implemented	M	app.py, files.py, parameter.py, pipeline.py	-	Not Tested
R3.5	Not implemented	V	app.py, files.py, parameter.py	-	Not Tested
Non-Functional requirements					
NFR4.1	Implemented	MV	app.py, parameter.py	UAT1.18	Tested

Continued on next page

Table 2: (Continued)

Requirement	Requirement status	Module used	Files affected	Tests	Test Status
NFR4.2	Implemented	MV	app.py, parameter.py	UAT1.19	Tested
NFR4.3	Implemented	V	app.py, parameter.py	UAT1.20	Tested
NFR4.4	Implemented	V	app.py, parameter.py	UAT1.21	Tested
NFR4.5	Implemented	V	app.py, buttons.py, parameter.py	UAT1.22	Tested
NFR4.6	Implemented	V	app.py, parameter.py	UAT1.23	Tested
NFR4.7	Implemented	V	-	-	Not Tested
NFR4.8	Implemented	V	app.py, buttons.py, files.py, parameter.py	UAT1.24	Tested
NFR4.9	Implemented	V	app.py, files.py, parameter.py, pipeline.py	UAT1.25	Tested
NFR4.10	Implemented	V	app.py	UAT1.26	Tested
NFR4.11	Implemented	V	app.py	UAT1.27	Tested
NFR4.12	Implemented	V	app.py, buttons.py, parameter.py	UAT1.38	Tested
NFR4.13	Implemented	V	app.py	UAT1.28	Tested
NFR4.14	Implemented	MVC	app.py, files.py, parameter.py, pipeline.py	-	Not Tested
NFR4.15	Implemented	V	app.py	UAT1.29	Tested
NFR4.16	Implemented	OS	build.yml test.yml	UAT1.30	Tested
NFR4.17	Implemented	OS	build.yml test.yml	UAT1.41	Tested
NFR4.18	Implemented	OS	build.yml test.yml	-	Not Tested
NFR4.19	Implemented	V	app.py	UAT1.31	Tested
NFR4.20	Implemented	V	app.py	UAT1.32	Tested
NFR4.21	Implemented	V	app.py	UAT1.33	Tested
NFR4.22	Implemented	V	app.py	UAT1.34	Tested
NFR4.23	Implemented	MVC	app.py, files.py, parameter.py, pipeline.py	SQT1.1	Tested
NFR4.24	Implemented	C	app.py, buttons.py, files.py, parameter.py	-	Not Tested
NFR4.25	Implemented	V	app.py	UAT1.35	Tested
NFR4.26	Implemented	V	app.py	UAT1.36	Tested

3 SonarQube

3.1 Introduction

SonarQube is a code quality assurance tool that collects and analyzes source code and generates reports on a project's code quality. It combines static and dynamic analytic technologies and allows for

continuous quality monitoring throughout time. SonarQube inspects and evaluates everything from small stylistic choices to design mistakes. This gives us a rich searchable history of the code, allowing us to figure out where the code is going wrong and whether it's due to style issues, code failures, code duplication, a lack of test coverage, or overly complex code. Each of these issues can have one of the five different severity levels: blocker, critical, major, minor or info. The software will examine source code from several angles and drill down layer by layer, from module to class level, with each level giving metric values and data that should show problematic regions in the source code that need to be improved.

By making your code base clean and maintainable, Sonarqube ensures code dependability, application security, and eliminates technical debt.

3.2 SonarQube Results

The following Figures contain the SonarQube results:

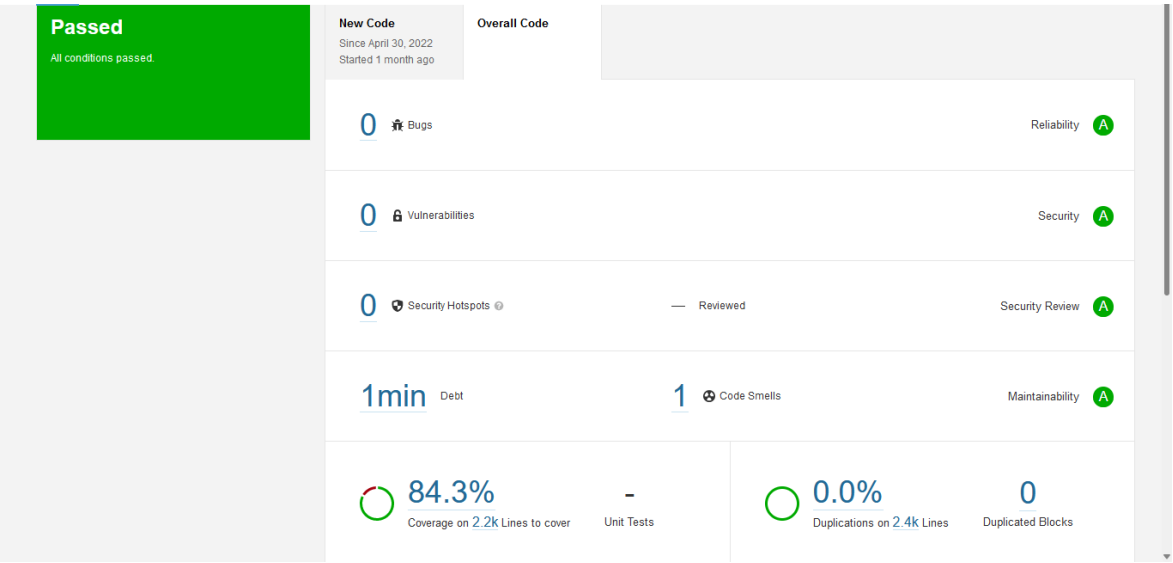

Figure 1: SonarQube Overview

Our project overview looks like the following:

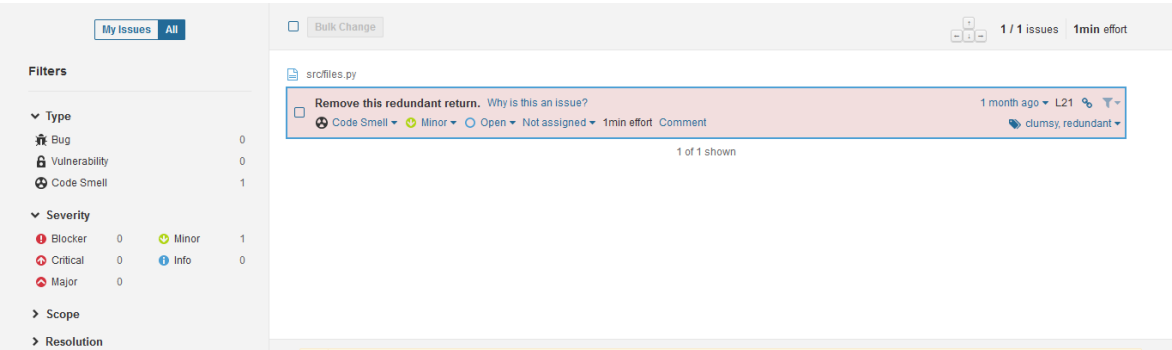

Figure 6: SonarQube Issues

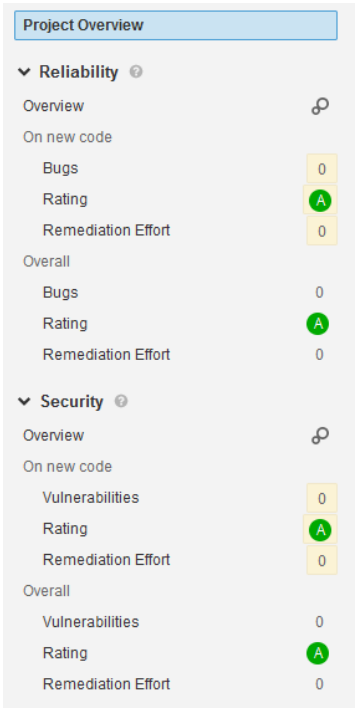

Figure 2: Reliability & Security

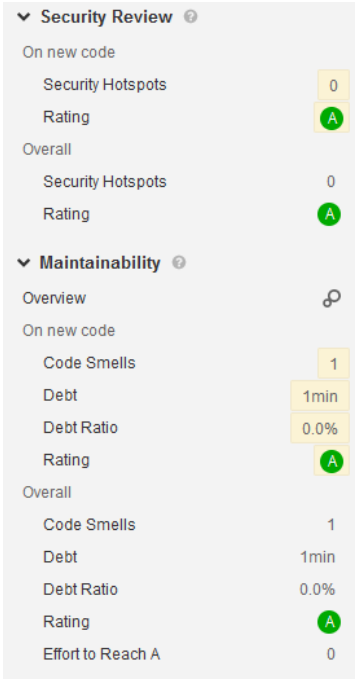

Figure 3: Security Review & Maintainability

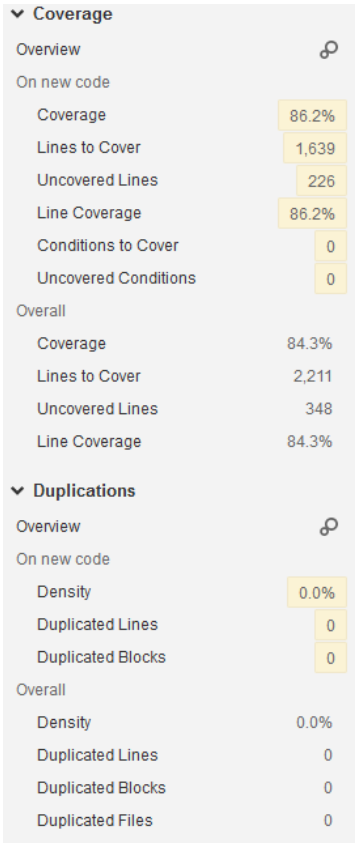

Figure 4: Coverage & Duplications

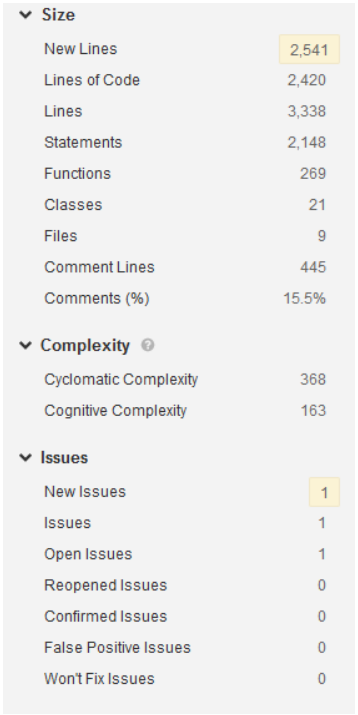

Figure 5: Size, Complexity & Issues

There is one aspect of the code that SonarQube classifies as a code smell. It concerns a `return` statement, which SonarQube categorizes as redundant. This is however not true and the statement is necessary.

Figure 7: SonarQube Measure

3.3 Coverage

The SonarQube coverage is 84.3%[7] because this project mainly concerns the GUI and includes the clients code as well. The coverage report has been generated with `Code coverage for Python, version 6.3.3 with C extension`. The rest of the GUI/view related aspects are covered by our acceptance testing.

3.4 SQT1.1

The following Figure displays the SonarQube results of the original `PASTAQ-GUI` codebase.

[7]This coverage is the accurate coverage of our testing. If the coverage report were to be run on the final deliverable code, it will show a much lower value due to a certain clause in the code which prevents the Python code from being run when importing. To achieve the accurate coverage, we removed this clause.

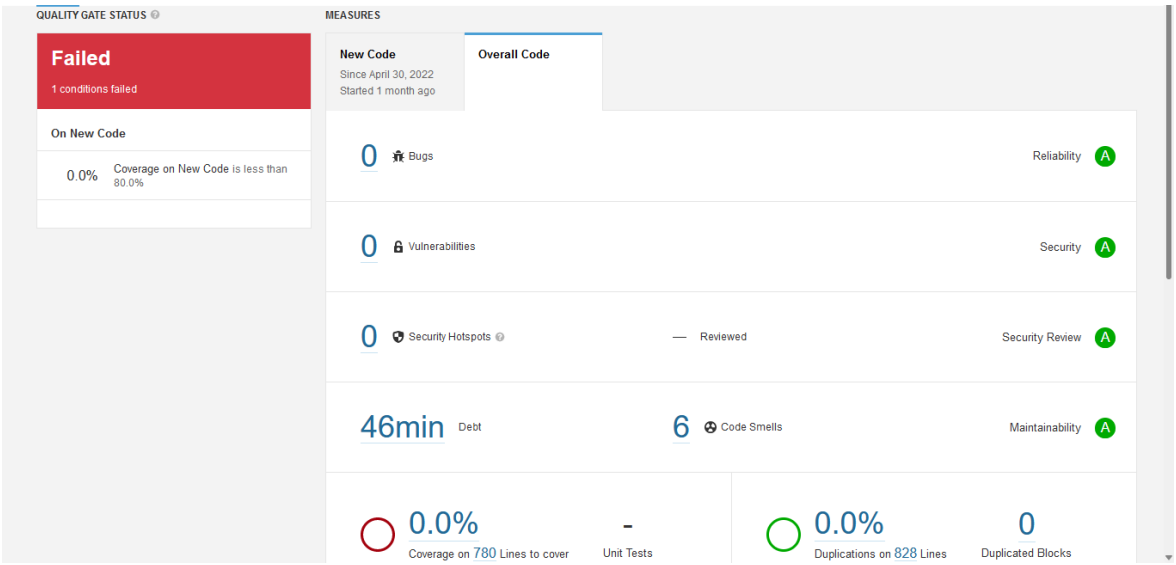

Figure 8: SonarQube on original code

Comparing the SonarQube results of the original code base, seen in Figure 8 and the code base after the project, it is noticeable how the Debt has been decreased from 46 minutes to only 1 minute. The code smells of the original code base have mostly been removed, from 6 to 1 code smell.
With SonarQube, we have been able to evaluate on of our requirements, which is why we consider this a test that it passed. This can be found in the previous traceability matrix as well.