



SONARQUBE AND MAVEN

A GUIDE

Contents

SonarQube	2
Maven	2
Installing SonarQube.....	3
Using SonarCloud.....	3
Installing Maven.....	4
Creating a Maven Java Project.....	5
Converting to a Java project to Maven.....	6
Analysing a Project with SonarQube.....	7
Analysing a Project with SonarCloud	8
Looking at the report	9
Creating a Custom Quality Profile.....	11
Creating a Custom Quality Gates.....	12
References	13

SonarQube

SonarQube is an open-source platform for continuous inspection of code quality. Using static code analysis, it tries to detect bugs, code smells and security vulnerabilities. SonarQube supports many languages through built-in rulesets and can also be extended with various plugins.

SonarQube is an extremely useful tool for developers because of its functionality it reports on bad code smells and if there is any duplicate code. It can be deployed on a local machine, it can be put on a server or you have the option to use the online version which is free for open source projects but for individual use you must pay a small fee per month.

SonarQube can be used with many frameworks such as Maven, Gradle and Jenkins. It also has GitHub integration, so a project can be analysed with ease when there are any commits.

Upon analysis SonarQube will tell what problem(s) are in the system whether it be a bug, vulnerability or code smell. You will then be told how many issues there are, or you are given a percentage value whichever is applicable. Issues can have different severity values such as critical, major and minor. You will be told whether the project has passed or failed the analysis. SonarQube will give you an estimated time telling you how long the problem will take to rectify. Clicking on a specific issue will tell you more about it. The project will pass or fail based on the “quality gate” this tells the developer whether the code can be pushed to production. For a project to pass the quality gate it must meet a series of requirements which can be defined by the developer.

Whilst SonarQube will tell you what is wrong with the code it will not fix it for you. SonarQube is designed to help improve code quality so that no more issues get into the codebase. Effort must still be put into writing good clean code so that issues don’t appear in the first place.

In addition to using SonarQube to improve your code you can get [SonarLint](#). SonarLint is an IDE extension that gives instantaneous feedback on the code you are writing, this will help you to avoid any issues before you commit. SonarLint is a stand-alone plugin that works with IntelliJ, Eclipse and Visual Studio.

SonarQube is an incredibly useful and powerful tool that can be used by developers to ensure that code is of a desired quality and the correct standards and practises are met.

[Link](#) to SonarQube documentation.

Maven

Maven is an open-source tool designed to standardise the build and management of any Java-based project. It aims to take care of the frustrations of building a java project such as dependency management or special build requirements. Maven is supported by all Major IDEs like IntelliJ, Eclipse and NetBeans.

Maven is based off the POM (Project Object Model). The POM is an XML file that contains the details of the project like the project name, version, dependencies, plugins etc. Maven builds a project by using the POM and a set of plugins that are shared by all projects that use Maven, this provides a uniform build system so once you know how one project is built you know how all of them are built.

Dealing with dependencies in Maven is easy because Maven connects to remote repositories and then downloads all of the dependencies that is needed to build the project. Adding dependencies is easy too, you just have to write the dependency in the pom file. You can search for dependencies and add them to your pom file [here](#).

Mavens functionality comes from plugins these plugins can be used to clean the project, compile the project, test the code, create java docs, create a project report and many more, a list can be found [here](#). In Maven a plugin provides a set off goals to be executed. There syntax is as follows, *mvn [plugin-name]:[goal]* for example *mvn clean:clean*. These plugins can be added in the pom file.

Maven is built around the concept of build life cycles, inside these there are build phrases and inside these there are build goals. You can execute a build phrase or a build goal. When executing a build phrase, you will also execute all the build goals in that phrase too, or you can just execute the goal. There are three in-built lifecycles these are *default*, *clean* and *site*. In the default lifecycle the phrases are *validate*, *compile*, *test*, *package*, *verify*, *install* and *deploy*. Build lifecycles are executed sequentially, so calling *mvn install* will execute *validate*, *compile*, *test* etc. before executing the *install* build phrase.

Maven to wants to encourage best development practices, unit testing is very common when working with Maven, the project directory structure is common across projects as well, so once again if you know one you know them all.

[Link](#) to Maven documentation.

Installing SonarQube

1. Download SonarQube [here](#).
2. Unzip it in C:\sonarqube
3. Start the SonarQube server, "C:\sonarqube\bin\windows-x86-64\StartSonar.bat". This may take a few minutes.
4. You can now open SonarQube by going to <http://localhost:9000>.

Using SonarCloud

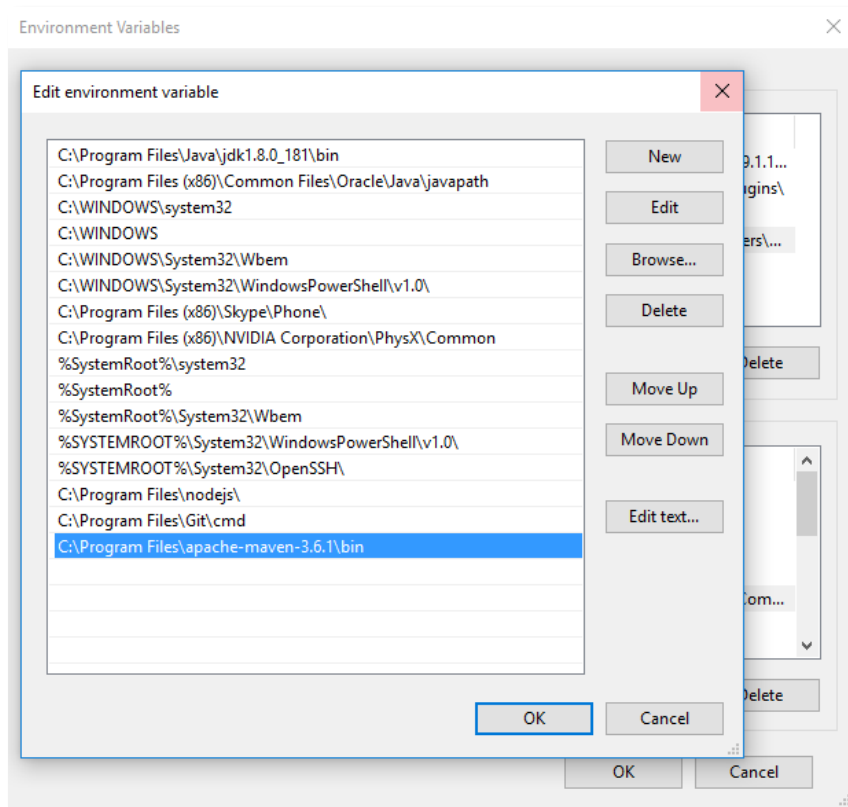
The alternative to SonarQube is to use SonarCloud it is the same service, but it is online, and you just need to install maven for it to work. It should be noted though that for private use it costs €10 a month although at the time of writing you do get a 14-day free trail.

1. To get started with SonarCloud go [here](#).
2. You can login using your GitHub account.

Analysing projects either locally or using SonarCloud is similar although they do differ at points, but we will go through how to use both.

Installing Maven

1. First ensure that you have the Java JDK download and the correct PATH variable set.
2. Download Maven [here](#), select the latest Binary zip archive.
3. Unzip the files and place them anywhere.
4. Now add the bin directory to the PATH environment variable.



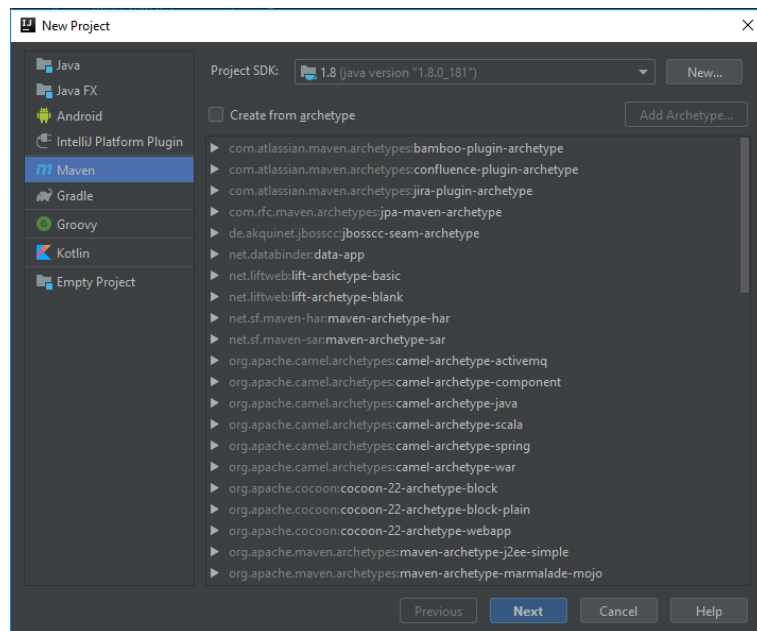
5. You can verify the installation by opening a terminal and typing `mvn -v` if successful you should see something like this.

```
C:\>mvn -v
C:\>
Apache Maven 3.6.1 (d66c9c0b3152b2e69ee9bac180bb8fcc8e6af555; 2019-04-04T20:00:29+01:00)
Maven home: C:\Program Files\apache-maven-3.6.1\bin\..
Java version: 1.8.0_181, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_181\jre
Default locale: en_GB, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
C:\>
C:/>
```

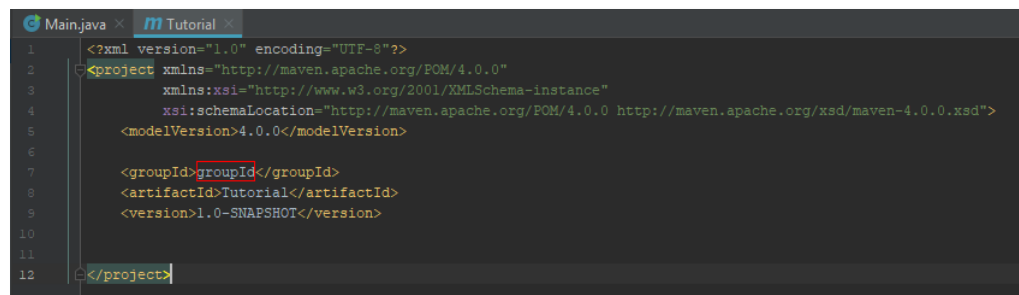
Creating a Maven Java Project

For this tutorial I will be using the IntelliJ IDE.

1. File > New > Project.
2. Select Maven from the left-hand side and make sure the correct JDK is selected.



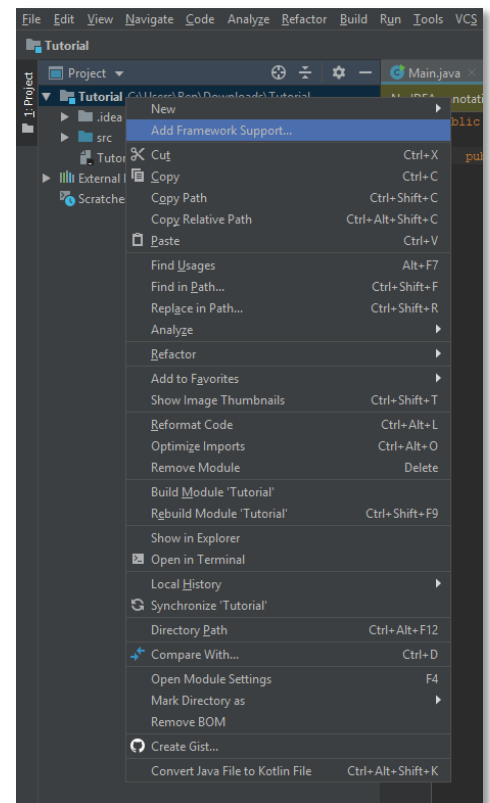
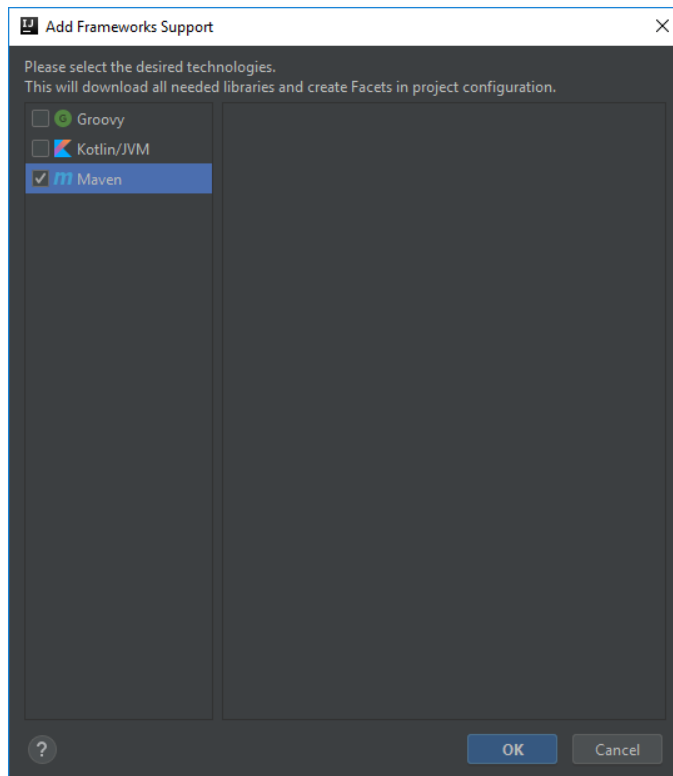
3. Insert a groupId and ArtifactId click *Next*.
4. Insert the Project name and select the Project location.
5. Click *Finish*. You will see the POM file and you can continue as you normally would.



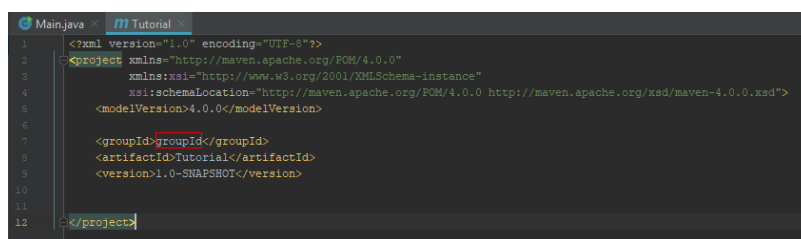
Converting to a Java project to Maven

For this tutorial I will be using the IntelliJ IDE.

1. Open the Java project you wish to convert to Maven.
2. Right click the project name and click *Add Framework Support*.
3. Select *Maven*.



4. You will now see the newly created POM.xml file.



Analysing a Project with SonarQube

You can download a sample Maven project from [here](#).

1. Start the SonarQube server
2. Login in using *admin* and *admin* this is the default username and password.
3. Locate the project file and run a terminal window.
4. Enter: `mvn clean install sonar:sonar -Dsonar.host.url=http://localhost:9000`

```
C:\Work\sonarqube-scanner-maven>mvn clean install sonar:sonar -Dsonar.host.url=http://localhost:9000
```

5. The analysis should begin, this may take a few minutes
 - a. During analysis you may have error reported that *"lambda expressions are not supported in -source 1.5"* or similar. This is because maven it not recognising JDK 1.8, so you need to add a plugin to your POM file get the latest version [here](#).

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.0</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

6. You should then be greeted with a build success message.

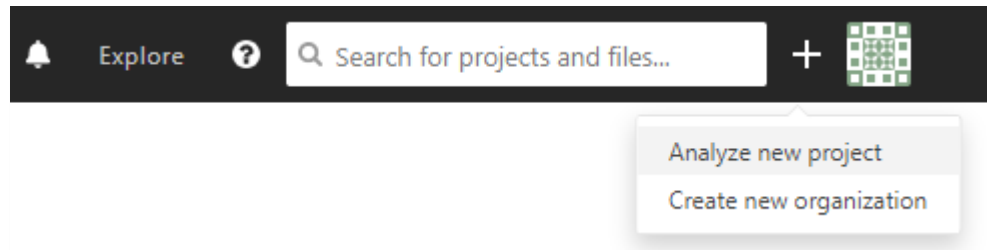
```
[INFO] Sensor Zero Coverage Sensor (done) | time=3ms
[INFO] Sensor Java CPD Block Indexer
[INFO] Sensor Java CPD Block Indexer (done) | time=20ms
[INFO] No SCM system was detected. You can use the 'sonar.scm.provider' property to explicitly specify it.
[INFO] 2 files had no CPD blocks
[INFO] Calculating CPD for 0 files
[INFO] CPD calculation finished
[INFO] Analysis report generated in 251ms, dir size=94 KB
[INFO] Analysis report compressed in 55ms, zip size=23 KB
[INFO] Analysis report uploaded in 849ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard?id=org.sonarqube%3Aparent
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://localhost:9000/api/ce/task?id=AWpy-PiV5dS08IyDLk9K
[INFO] Analysis total time: 10.183 s
[INFO] -----
[INFO] Reactor Summary for Example of SonarQube Scanner for Maven + Code Coverage by UT and IT 1.0-SNAPSHOT:
[INFO]
[INFO] Example of SonarQube Scanner for Maven + Code Coverage by UT and IT SUCCESS [ 16.514 s]
[INFO] Java :: JaCoco Multi Modules :: App ..... SUCCESS [ 12.515 s]
[INFO] Groovy :: JaCoco Multi Modules :: App ..... SUCCESS [ 36.408 s]
[INFO] JaCoco Multi Modules :: App IT ..... SUCCESS [ 1.409 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:34 min
[INFO] Finished at: 2019-05-01T11:38:00+01:00
[INFO] -----
C:\Work\sonarqube-scanner-maven>
```

7. Go to sonarqube and refresh the page and you should see the project

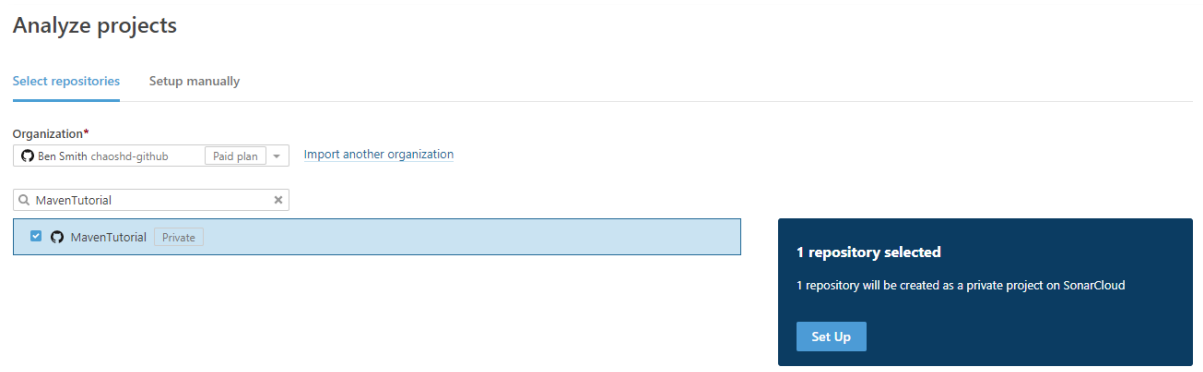
Analysing a Project with SonarCloud

The process for using SonarCloud is largely the same just with some small differences.

1. Login to sonarcloud.io.
2. Go to the top right of the page and click *Analyse new project*.



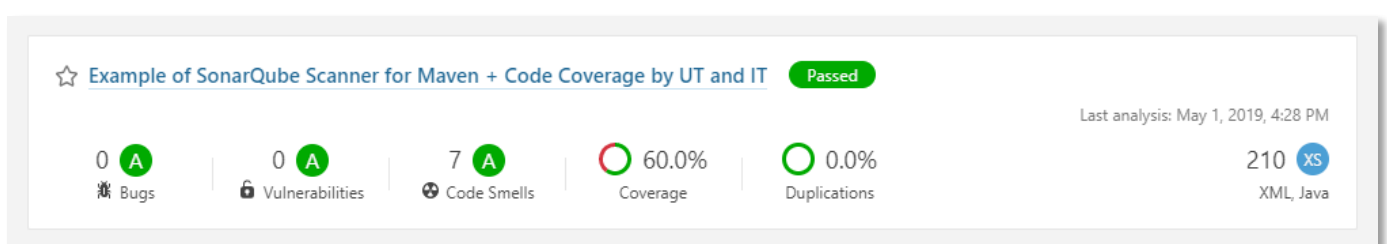
3. Select the project you want to analyse and click *Set Up*.



4. SonarCloud works off tokens, these tokens are for security and are unique. Select the name for the token and click *Generate* and copy the token key. Or you can use an existing token. Click *Continue*.
5. Answer the questions given to you and you will then be given the command line you will have to run in the project folder. Remove the back slashes and put `clean install` between `mvn` and `sonar` and you should have something that looks like this:

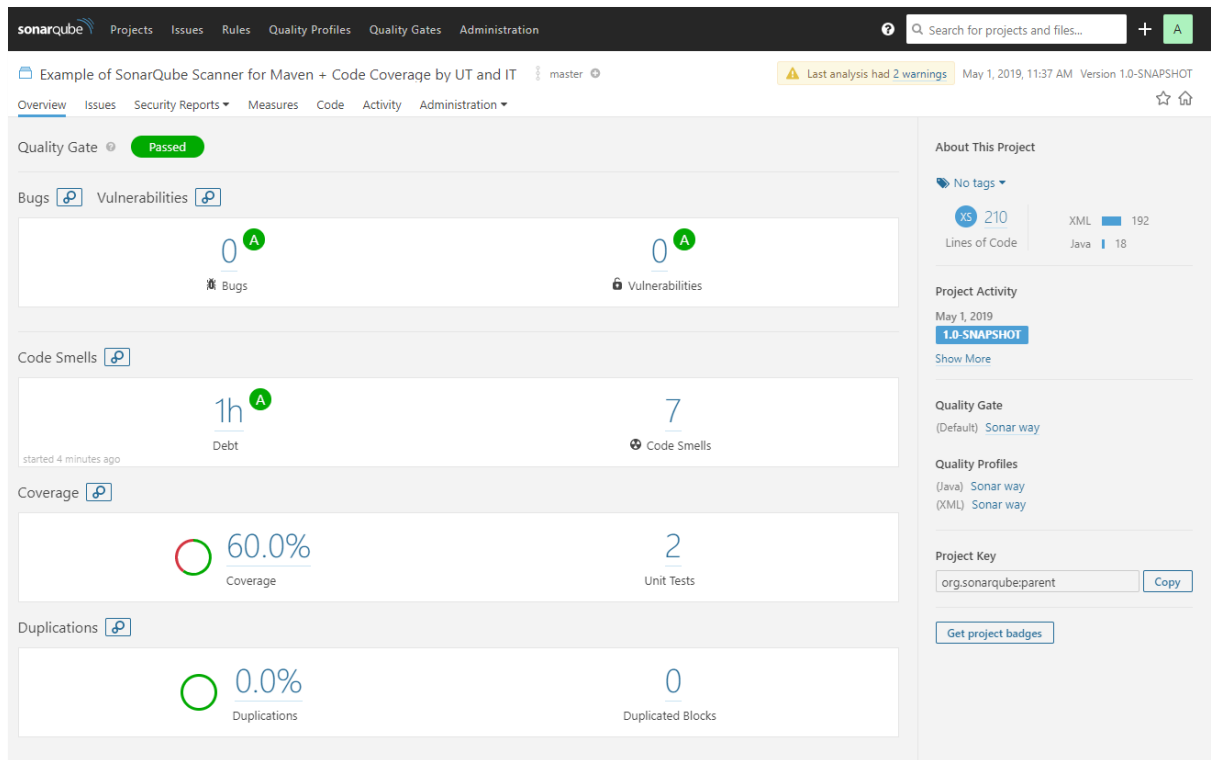
```
mvn clean install sonar:sonar -
Dsonar.projectKey=YOURGITHUB_MavenTutorial -
Dsonar.organization=YOURGITHUB-github -
Dsonar.host.url=https://sonarcloud.io -
Dsonar.login=YOURTOKEN
```

6. Go to your local project folder and open a terminal window and enter the above information, the same as Step 4 previously.
7. The analysis time will take significantly longer when using SonarCloud than doing it locally.
8. Once the analysis is complete you should be automatically redirected to the analysis page. If you are not, just navigate back to the projects page and it will be the first in the list.

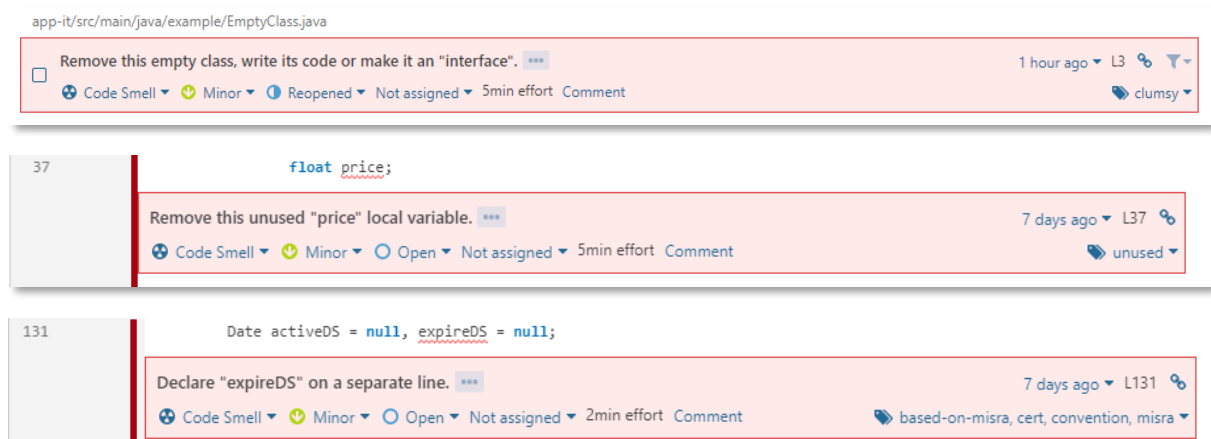


Looking at the report

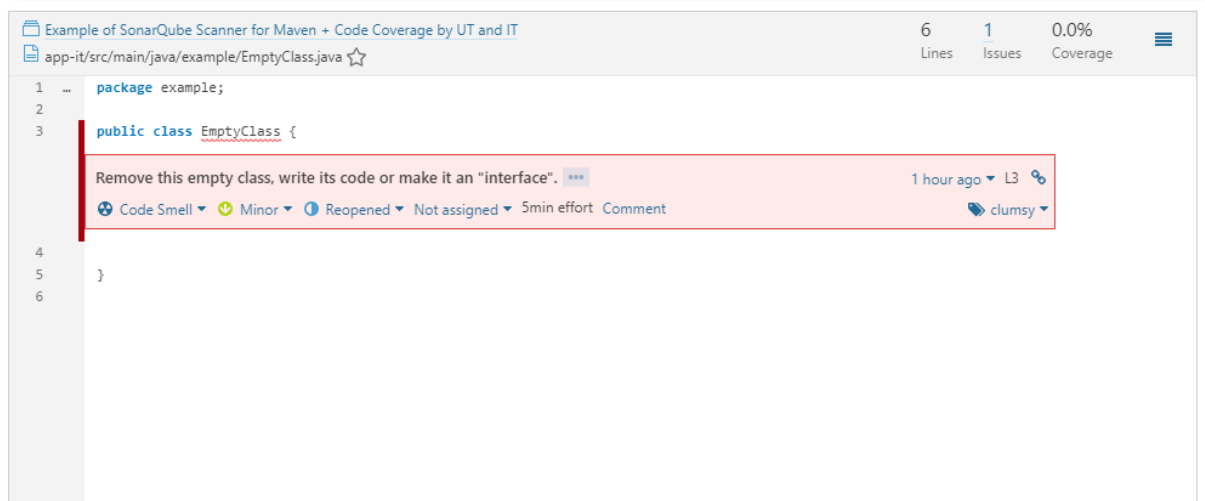
On the project overview you are presented with the number of bugs, vulnerabilities and code smells. You are told how much technical debt you are in, that is an estimate of how long SonarQube thinks it will take to fix the issues with the project. The code coverage, that is the code that was covered in the tests and amount duplications that are in the code. Performing a second analysis SonarQube will tell you if the program has improved or not.



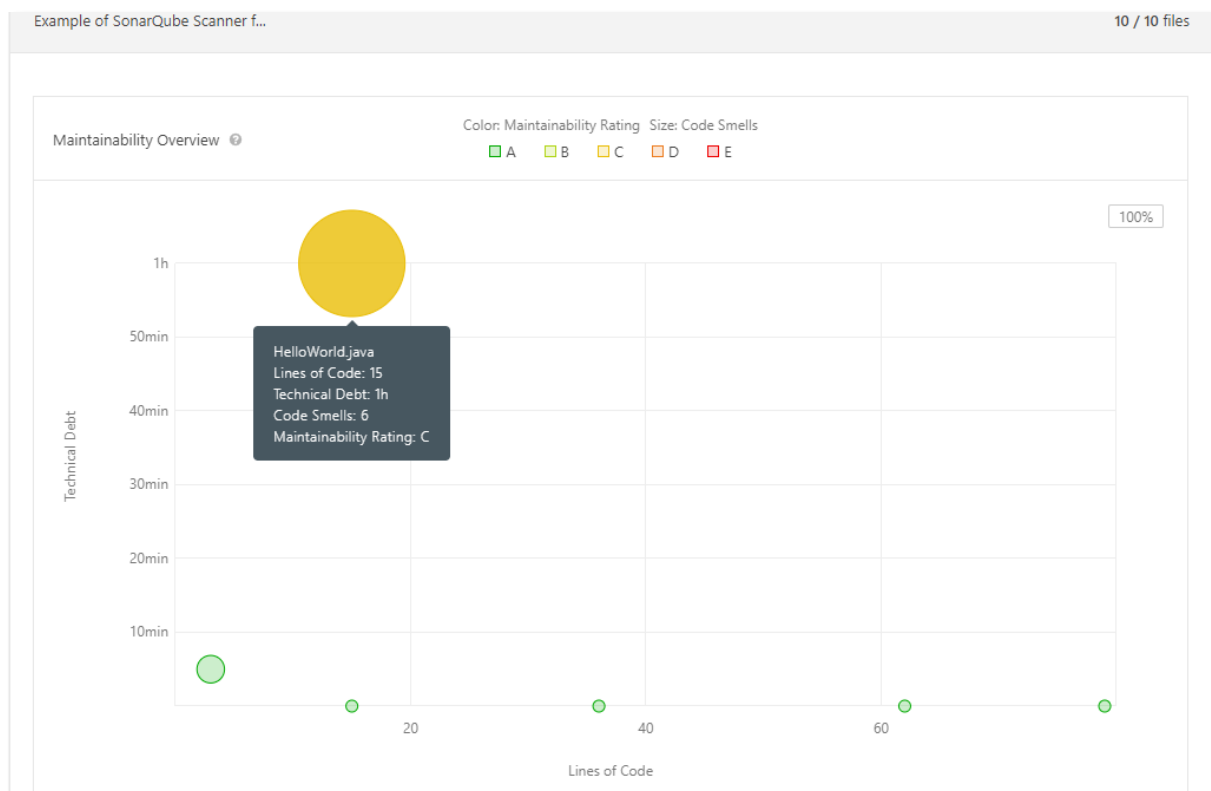
On the issues tab you can look at specific issue you can change the severity, declare it as resolved or a false positive, you can assign the issue to someone. SonarQube give you a rough time estimate of how long it will take to fix.



On the left-hand side you can use the filters to look for issues that are within certain criteria especially useful when going through big projects. Clicking on an issue you can see the related source code.



The measures tab allows you to view graphs that will visualise data relating to reliability, security, maintainability, coverage etc. Hovering over point on the graph will give you more details about the given measure and clicking on that point will take you to the related source code.



The code tab shows you each folder and file in the project, it will also show you the number of bugs vulnerabilities, code smells etc.

The activity page contains a graph which gives you information on the total number of issues over each scan.

The administration tab lets you configure general settings, quality profiles and the permissions about who can access the project report.

Creating a Custom Quality Profile

Why would you want to create a custom quality profile? Well during analysis, you might get a lot of false positives or the default profile might not meet your own needs, so we can create our own profile.

1. Go to the quality profiles section and create a new profile giving the name, language and Parent. Selecting the parent will change what rules are inherited in the profile.

The screenshot shows the 'Quality Profiles' section in SonarQube. A 'New Profile' dialog box is open in the center. The dialog has three main fields: 'Name*' with the value 'Test', 'Language*' with a dropdown menu set to 'Java', and 'Parent:' with a dropdown menu set to 'None'. At the bottom of the dialog are 'Create' and 'Cancel' buttons. To the left of the dialog, there's a list of existing profiles under the heading 'Filter profiles by:'. It shows two categories: 'C#, 1 profile(s)' and 'CSS, 1 profile(s)', each with a 'Sonar way' and a 'Built-in' button. To the right of the dialog, there's a 'Recently Added Rules' section listing several rules with their status (e.g., 'Failed unit tests should be fixed', 'C#, not yet activated').

2. You can now Activate or Disable rules that you may or may not want.

"==" and "!=" should not be used when "equals" is overridden	Java	Code Smell	cert, cwe, suspicious	Activate
"@EnableAutoConfiguration" should be fine-tuned	Java	Code Smell	performance, spring	Activate
"@Import"s should be preferred to "@ComponentScan"s	Java	Code Smell	performance, spring	Activate
"action" mappings should not have too many "forward" entries	Java	Code Smell	brain-overload, struts	Activate
"collect" should be used with "Streams" instead of "list::add"	Java	Code Smell	java8	Activate
"deleteOnExit" should not be used	Java	Code Smell	performance	Activate
"equals" methods should be symmetric and work for subclasses	Java	Bug	cert	Activate

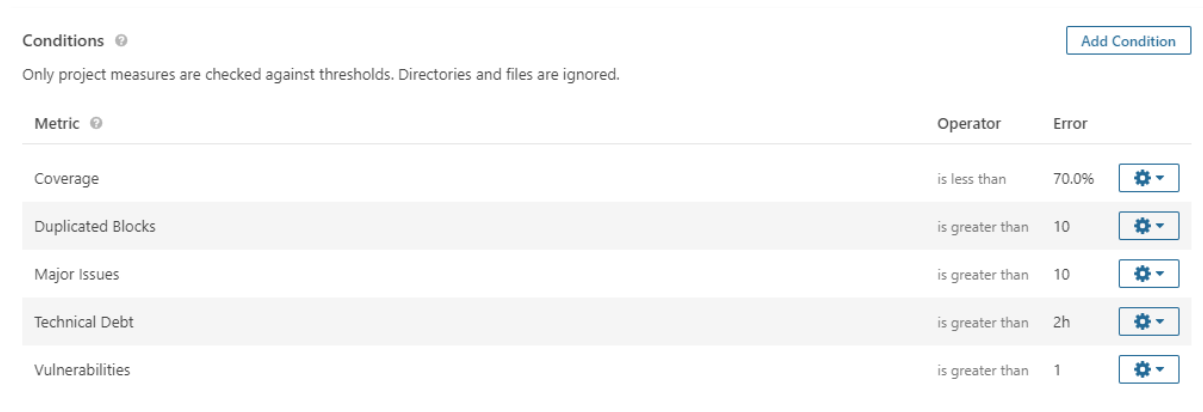
3. Once you have selected the rules you want be sure to assign the quality profile to a project.

The screenshot shows the 'Projects' section in SonarQube. At the top right is a 'Change Projects' button. Below it, there's a list of projects. The first project is 'Example of SonarQube Scanner for Maven + Code Coverage by UT and IT'. At the bottom right of the list, it says '1 of 1 shown'.

Creating a Custom Quality Gates

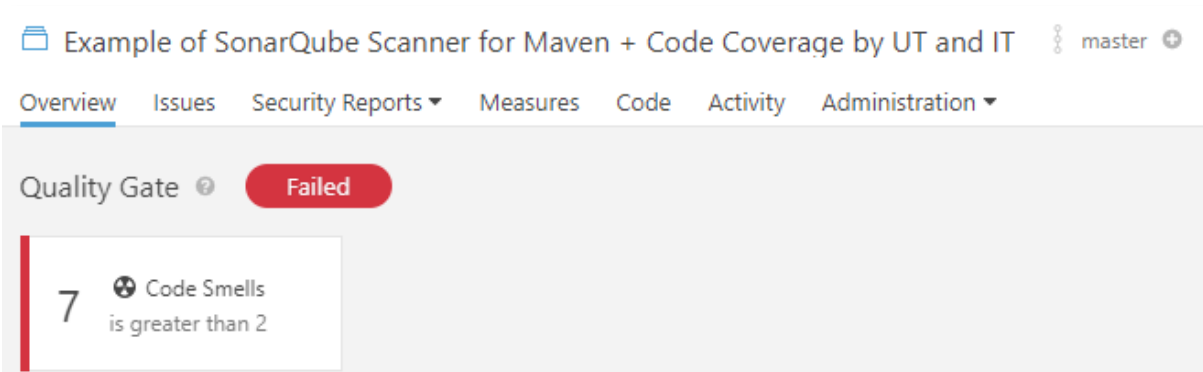
Quality gates in SonarQube are for you to determine whether a project has passed or failed the analysis based on a minimum threshold that the code must meet. You can create your own quality gate to determine whether your project has passed or failed on multiple criteria such as, the percentage of code covered with tests, the percentage of Cyclomatic Complexity, the percentage of code duplication etc. Quality gates are important to ensure the code is kept to a certain quality and is ready to be pushed to market.

1. Go to Quality Gates > Create and select a name.



Metric	Operator	Error
Coverage	is less than	70.0%
Duplicated Blocks	is greater than	10
Major Issues	is greater than	10
Technical Debt	is greater than	2h
Vulnerabilities	is greater than	1

2. You can then *Add Conditions* and chose what metrics the code must meet.
3. Under *Project* select the project(s) you want to analyse using this quality gate. The next time the code is analysed the new quality gate will be used.
4. Here you can see the sample project failed because it has too many code smells.



Example of SonarQube Scanner for Maven + Code Coverage by UT and IT master

Overview Issues Security Reports Measures Code Activity Administration

Quality Gate **Failed**

7 Code Smells is greater than 2

References

https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Build_Lifecycle_Basics

<https://maven.apache.org/plugins/index.html>

<https://maven.apache.org/what-is-maven.html>

<https://maven.apache.org/guides/introduction/introduction-to-plugins.html>

https://en.wikipedia.org/wiki/Apache_Maven

<https://www.theserverside.com/video/How-to-use-the-SonarQube-Maven-Plugin-for-continuous-inspection>

<https://www.sonarqube.org/features/clean-code/>

<https://sonarcloud.io/about>

<https://www.youtube.com/watch?v=BuT1JiOP9Ug>

<https://matthiasgeiger.wordpress.com/2014/02/19/sonarqube-what-is-it-how-to-get-started-why-do-i-use-it/>

<https://www.sonarlint.org/>