# SonarQube

**What is Sonar?**

Sonar is a software quality management platform primarily for Java programming language, enabling developers to access and track code analysis data ranging from styling errors, potential bugs, and code defects to design inefficiencies, code duplication, lack of test coverage, and excess complexity. Everything that affects our code base, from minor styling details to critical design errors, is inspected and evaluated by Sonar.

**Sonar defines the following technical axes:**

* Coding standards—respect coding standards and follow best practices
* Potential bugs—eliminate code violations to prevent vulnerabilities
* Documentation and comments—provide documentation especially for the Public API, the source code
* Duplicated code—isolates and refines duplications, Don't Repeat Yourself Complexity—equalizes disproportionate distributed complexity among components; eliminates complexity if possible
* Test coverage—writes unit tests, especially for complex parts of the software
* Design and architecture—minimize dependencies

**Source Code analyzers**

Sonar utilizes some of the most popular and proven tools available in the open source community. These tools pass through source performing standard checks reviewing errors and possible bugs, each from their own perspective.

Sonar includes following five analyzers:

1. Squid
2. Checkstyle
3. PMD
4. Findbugs
5. Cobertura & Clover

SonarQube, which will help you manage your code quality, instead of letting your code quality manage you.

SonarQube is a free and open source “code quality platform.” It gives you a moment-in-time snapshot of your code quality today, as well as trending of lagging (what’s already gone wrong) and leading (what’s likely to go wrong in the future) quality indicators. For test coverage (a leading indicator), a score of 50% may not look great, but what was it last month? If you’re up from 35%, it’s high-fives all around.

SonarQube doesn’t just show you what’s wrong. It also offers quality-management tools to actively help you put it right: IDE integration, integration for Jenkins, a popular Continuous Integration server, and code-review tools.

SonarQube’s commercial competitors in the code-quality space offer some of those things too (depending on which one you’re looking at); but they seem to focus their definition of quality mainly on bugs and complexity, whereas SonarQube’s offerings span what its creators call the Seven Axes of Quality.

1. Architecture & Design
2. Duplicates
3. Comments
4. Coding rules
5. Potential bugs
6. Complexity
7. Unit tests

Code quality tools for Java: FindBugs, PMD, and JaCoCo

Installation and setup

1. Download and install sonarqube from <http://www.sonarqube.org/downloads/>
2. From the bin folder run sonar.bat /sonar.sh file
3. Install sonar scanner from (<http://docs.sonarqube.org/display/SONARQUBE52/Installing+and+Configuring+SonarQube+Scanner>) link.
4. Create **sonar.properties** file under your project.
5. And run the sonar-runner command from your base project.

**Analyzing with SonarQube Runner**

SonarQube Runner is a Java application you fire from the command line. You feed it your project and a simple set of properties, and with those two things it can run the analysis for any language SonarQube handles (Java, C, C++, C#, ABAP, COBOL and so on).

**NOTE**: You also need to make sure you have Java installed—because the SonarQube Runner was written in Java, you need to have Java on your machine to run it.

Set Environmental varialble for sonarRunner so that sonar\_runner is avaialable everywhere in the system.

You can set some common properties for the entire project can be set at $SONAR\_HOME directory/conf/sonar-runner.properties file.

**Analyzing**

Once the sonarQube runner setup is done create sonar.properties under your project main folder.

#Required metadata

sonar.projectKey=test:webroot

sonar.projectName=Webroot Connector

sonar.projectVersion=1.0

#Comma-separated list of library directories

sonar.libraries=lb/\*.jar

#Comma-separated list of source directories

sonar.sources=src

#comma-delimited list of paths to thest sources

sonar.tests=test1,test2

sonar.binaries=bin

#Below are global properties you can configure in sonar-runner.propertis

sonar.host.url=http://localhost:9000

sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar

sonar.jdbc.driver=com.mysql.jdbc.Driver

sonar.jdbc.username=sonar

sonar.jdbc.password=sonar

**Multi-module projects**

If you are dealing with multi-module project you need to setup the properties for each module. Or you can setup common ones at the parent level and override what you need to at the module level.

Sonar.modules=module\_folderOne, module\_folderTwo (should match folder name)

If your modules name doesn’t match its directory, you also need to specify its directory

moduleB.sonar.projectBaseDir=path/tp/moduleB

**Configuring with Maven**

Add below plugin if you are using maven3.0

<pluginManagement>

<plugins>

<plugin>

<groupId>org.sonarsource.scanner.maven</groupId>

<artifactId>sonar-maven-plugin</artifactId>

<version>3.0.1</version>

</plugin>

</plugins>

</pluginManagement>

and create new profile in settings.xml or pom.xml

<profiles>

<profile>

<id>sonar</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

<properties>

<sonar.jdbc.url>jdbc:mysql://localhost:3306/sonar</sonar.jdbc.url>

<sonar.jdbc.driver>com.mysql.jdbc.Driver</sonar.jdbc.driver>

<sonar.jdbc.username>root</sonar.jdbc.username>

<sonar.jdbc.password>admin</sonar.jdbc.password>

<!-- SERVER ON A REMOTE HOST -->

<sonar.host.url>http://localhost:9000</sonar.host.url>

</properties>

</profile>

</profiles>

Finally run below command to run the sonarQUBE

**$/>mvn sonar:sonar**

**We need to mention a couple of points here:**

* The parameter -Dmaven.test.failure.ignore should always be used to instruct SonarQube to continue with analysis even if one or more tests fail.
* By default, your unit tests are executed twice: once for the install goal and once for the sonar goal. It’s possible to skip the second run and reuse the test results from the first run. For details, see sonar.dynamicAnalysis and its companion properties in section B.4.
* Avoid using the Maven parameters -Dtest=false and -DskipTests=true. They’ll prevent SonarQube from running your unit tests, and you won’t get any test metrics.

**sonar.dynamicAnalysis** : One of **true**, **false**, **reuseReports** Unit tests are executed by default, but you can choose to turn that off or to reuse previously generated reports. If you choose reuseReports, you need to use one of the companion properties to specify the report type and location.

* sonar.jacoco.reportPath:
* sonar.surefire.reportsPath
* sonar.coberatura.reportPath
* sonar.clover.reportPath

**Test Covarage with Jacoco Plugin**

JaCoCo is a free Java code coverage tool. This is essentially the successor to Emma, and the EclEmma team as an Eclipse project has developed it.

JaCoCo offers line and branch coverage.

STEP-1: Open pom.xml and add below plugin

**<plugin>**

**<groupId>org.jacoco</groupId>**

**<artifactId>jacoco-maven-plugin</artifactId>**

**<version>0.7.6.201602180812</version>**

**<executions>**

**<execution>**

**<id>default-prepare-agent</id>**

**<goals>**

**<goal>prepare-agent</goal>**

**</goals>**

**</execution>**

**<execution>**

**<id>default-report</id>**

**<phase>prepare-package</phase>**

**<goals>**

**<goal>report</goal>**

**</goals>**

**</execution>**

**</executions>**

**</plugin>**

STEP-2: run mvn package command (make sure test cases available in your project)

STEP-3: open target/site/jacoco/index.html page to view report.

How it works

In the pom file, we instruct Maven to run the following two goals of the Maven JaCoCo plugin:

1. Plugin **prepare-agent**: This is bound by default to the initialize phase of the Maven default lifecycle. The goal runs and prepares the agent that does the analysis.
2. Plugin **report**: This agent gathers test coverage information when the tests are run and create the report as part of the prepare-package phase (which we have explicitly specified). The report gives information about the test coverage. Green indicates lines that are covered by tests and red indicates lines that are not covered by tests.

**There is more…**

You could subject the project to code coverage and generate the same report without making any changes to the pom file. To do this, run the following command:

**mvn jacoco:prepare-agent test jacoco:report**

Now, you may get the error: no plugin found etc..

You can fix this error with below code

**mvn org.jacoco:jacoco-maven-plugin:prepare-agent test org.jacoco:jacoco-maven-plugin:report**

**You can make build failed if the code covarage below threashold value**

How about failing the build if the code coverage is below a threshold value? To do this, perform the following steps:

<execution>

<id>default-check</id>

<phase>prepare-package</phase>

<goals>

<goal>check</goal>

</goals>

<configuration>

<rules>

<rule>

<element>BUNDLE</element>

<limits>

<limit>

<counter>COMPLEXITY</counter>

<value>COVEREDRATIO</value>

<minimum>0.60</minimum>

</limit>

</limits>

</rule>

</rules>

</configuration>

</execution>

**Rules that needs to remember as a Java developer**

1. BigDecimal (double) constructor should not be used

Instead use **BigDecimal bg=BigDecimal.valueOf (double)**

1. CheckStyle off suppression should not be used.

**//CHECKSTYLE:OFF**

1. If you implement Clobeable you must override clone method from object.

Without overriding clone, any cloned instances will potentially share members with the source instance.

1. Don’t call size method on **ConcurrentLinkedQueue**: This Queue is thread safe and orders elements in FIFO Order.
2. **File.deleteOnExist ()** shouldn’t be used. It occures only in the case of normal JVM shutdown but not when the JVM crashes or is killed. For each file handler, the memory associated to the handler is released only at the end of the process.
3. Method hasCode and toString should n’t be called directly on Arrays instead use below

String arrayString= Arrays.toString (myArray);

Int arrayHashCode=Arrays.hashCode (myArray);

1. HttpServletRequest.getRequestedSessionId () shouldnot be used.
2. If you have written custom Iterator you must throw NoSuchElementException when hasNext() false
3. **java.lang.Error** and its subclasses represent abnormal conditions, such as **OutOfMemoryError**, which should only be encountered by the Java Virtual Machine. Don’t extend java.lang.Error
4. By contract, the **NullCipher** class provides an "identity cipher" -- one that does not transform or encrypt the plaintext in any way. As a consequence, the ciphertext is identical to the plaintext. So this class should be used for testing, and never in production code.
5. Important Difference between Lock and Sycnrhrnoized keyword:

* Synchronized keyword doesn’t provide fairness whereas we can set fairness to true while creating ReentrantLock object so that longest waiting thread gets the lock first.
* Synchronization blocks or methods can cover only one method whereas we can acquire the lock in one method and release it in another method with Lock API.
* Lock we are forced to have try-finally block to make sure Lock is released even if some exception is thrown between lock and unlock method calls.

<https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards>

1. If an NPE is being thrown to indicate that a parameter to the method should not have been null, use the **@NotNull** annotation instead.
2. Always invoke wait () and await () methods inside a (while) loop

final Lock lock = new ReentrantLock();

final Condition notFull = lock.newCondition();

...

notFull.**await**(); // dont

1. Use Runtime.addShutdownHook(new Runnable(){});

### "ScheduledThreadPoolExecutor" should not have 0 core threads

1. Serializable class should have serialVersionUID

A **serialVersionUID** field is required in all **Serializable** classes. If you do not provide one, the compiler will calculate one for you. The danger in not explicitly choosing the value is that when the class changes, the compiler will generate an entirely new id, and you will be suddenly unable to deserialize (read from file) objects that were serialized with the previous version of the class.

1. Serializable inner classes of “Serializable” classes should be static

Serializing a non-**static** inner class will result in an attempt at serializing the outer class as well. If the outer class is actually serializable, then the serialization will succeed but possibly write out far more data than was intended.

Making the inner class **static** (i.e. "nested") avoids this problem, therefore inner classes should be **static** if possible. However, you should be aware that there are semantic differences between an inner class and a nested one:

* An inner class can only be instantiated within the context of an instance of the outer class.
* A nested (**static**) class can be instantiated independently of the outer class.

### "Serializable" inner classes of non-serializable classes should be "static.

If main class is implements serializable, inner class must be static to make it serializable

1. **A static final arrays should be private and exposed cloned object instead of original one to the external world.**
2. StringBuilder or StringBuffer shouldn’t initialize with characters ‘x’ etc instead use “X”
3. Switch case should have at least 3 “case” classes otherwise use if else
4. Switch case should have default clause
5. Switch case shouldn’t have too many lines
6. **Wait should be used insteadof Thread.sleep () method**

If Thread.sleep (...) is called when the current thread holds a lock, it could lead to performance, and scalability issues, or even worse to deadlocks because the execution of the thread holding the lock is frozen. It's better to call wait(...) on the monitor object to temporarily release the lock and allow other threads to run.

1. Abstract class should comply with the a naming convention AbstractXXXX
2. Abstract class without fields shoule be converted to interfaces
3. Collections.emptyList (), emptyMap () and emptySet () should be used instead of Collections.EMPTY\_LIST, EMPTY\_MAP and EMPTY\_SET
4. Comparators should be serialized