Caltech | Center for Technology & Management Education

Post Graduate Program In DevOps

simplilearn

Code Quality Improvement Using Jenkins

Learning Objectives

By the end of this lesson, you will be able to:

- Explain the significance of code quality and its metrics
- Implement different code quality analysis tools
- Generate XML reports on code complexity
- Implement SonarQube with Jenkins

Code Quality and Jenkins

Introduction

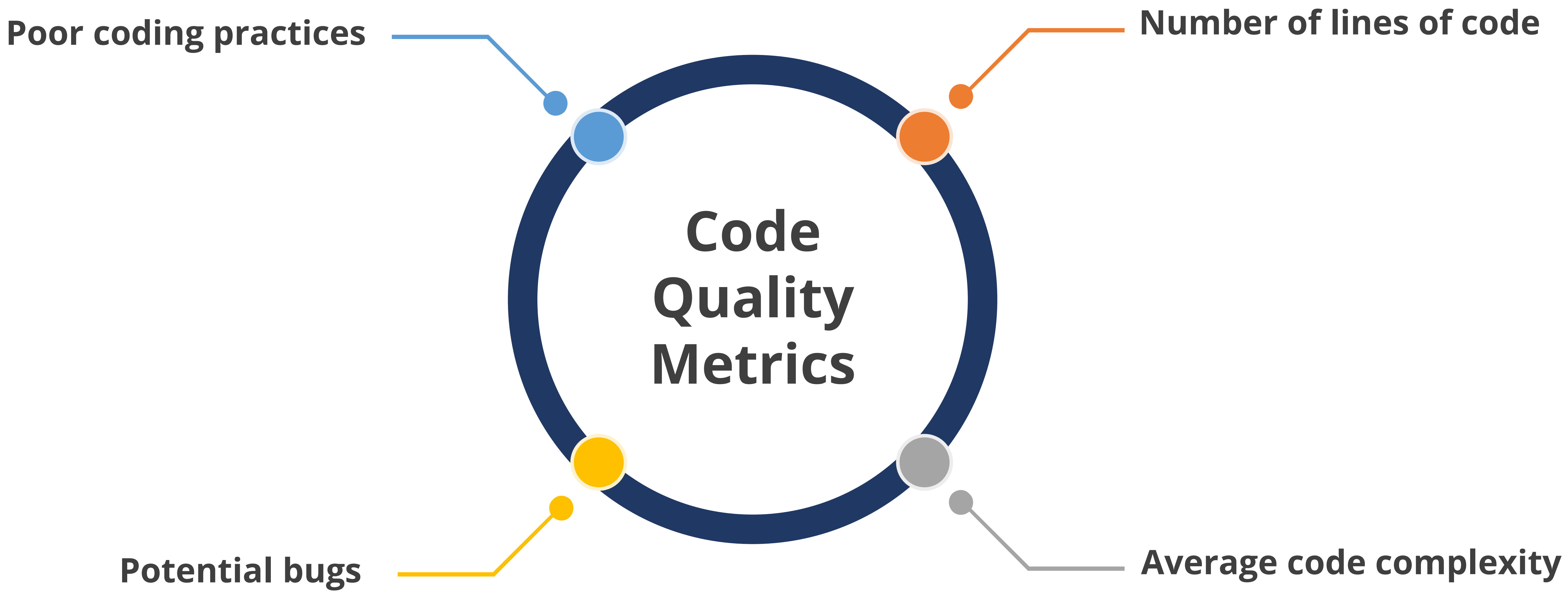
Coding standards are rules that define the coding styles and conventions that are common across an organization for better understanding of the product development.

- Coding standards include both aesthetic aspects such as formatting, and bad practice such as missing brackets after a condition in Java
- A consistent coding style makes it easy to work on the code written by other developers by making it clean and readable

QUALITY CODING

Code Quality Metrics

Code quality metrics include different aspects of code, such as coding standards and best practices to code coverage, along with compiler warnings to TODO comments.



Code Quality Tools

CheckStyle

SonarQube

FindBugs

Cobertura

StyleCop

Code Quality in Build Process

Before working with Jenkins, it is important to understand code quality metrics as an isolated entity with one framework or server and make it part of a broader process.

The first level of code quality integration is the IDE

- Modern IDEs support most of the code quality tools, such as Checkstyle, PMD, and FindBugs have plugins for Eclipse, NetBeans, and IntelliJ
- It is a fast and reliable way to provide feedback to individual developers

The second level is your build server

- In addition to a single unit and integration test build jobs, dedicated code quality build is a useful entity, which runs after the build and test
- It aims to produce project-wide code quality metrics

Code Quality in Build Process

As code coverage analysis and other analysis tools can be quite slow to run, it is recommended to run code quality analysis job separately.

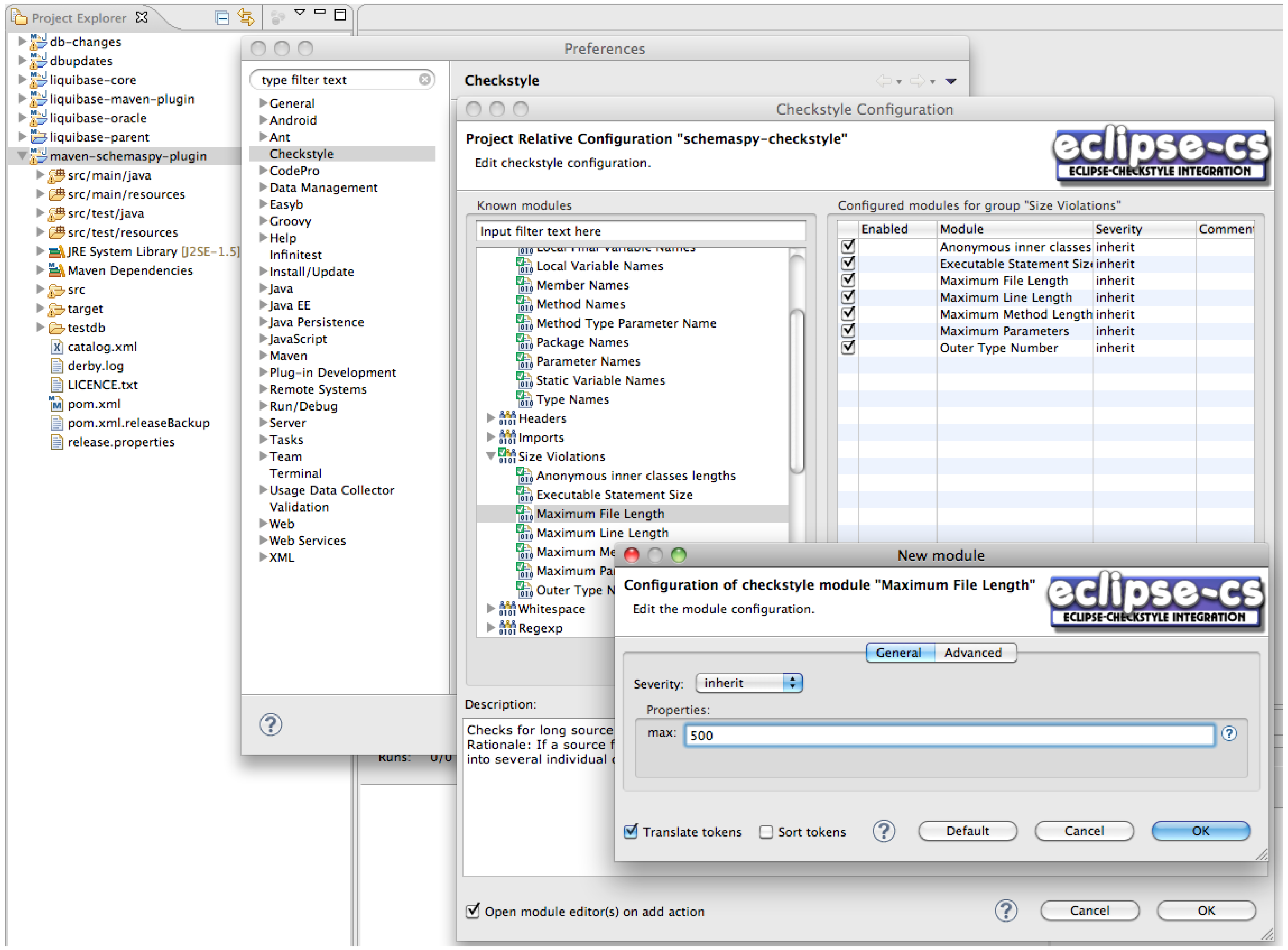
- Code quality reporting is a relatively passive process
- One has to search for the information on the build server, rather than simply reporting on code quality
- It is important to set up a dedicated code quality build that runs after the build and test, and configure the build to fail
- Jenkins provides the facility to do code quality reporting or one can also do it in build script
- The advantage of configuring code quality outside the build script is that one can change code quality build failing criteria more easily without hindering the source code

Checkstyle

Checkstyle is a static analysis tool for Java. It was designed to define a set of coding standards, however, now it also checks for poor coding practices, and complex and duplicated code.

- It is a flexible tool
- It can be used in any Java-based code quality analysis strategy
- It supports a number of rules, such as naming conventions, Java file comments, class and method size, annotations, code complexity metrics, and bad coding practices
- Duplicated code is a critical code quality issue as it is harder to maintain and debug
- Though Checkstyle provides support for detecting a duplicated code, one can use tools, such as CPD for better results in detection of duplicated code

Checkstyle Eclipse Plugin



Project Explorer

- db-changes
- dbupdates
- liquibase-core
- liquibase-maven-plugin
- liquibase-oracle
- liquibase-parent
- maven-schemaspy-plugin
  - src/main/java
  - src/main/resources
  - src/test/java
  - src/test/resources
  - JRE System Library [J2SE-1.5]
  - Maven Dependencies
  - src
  - target
  - testdb
  - catalog.xml
  - derby.log
  - LICENCE.txt
  - pom.xml
  - pom.xml.releaseBackup
  - release.properties

Preferences

type filter text

General, Android, Ant, Checkstyle, CodePro, Data Management, Easyb, Groovy, Help, Infinitest, Install/Update, Java, Java EE, Java Persistence, JavaScript, Maven, Plug-in Development, Remote Systems, Run/Debug, Server, Tasks, Team, Terminal, Usage Data Collector, Validation, Web, Web Services, XML

Checkstyle

Checkstyle Configuration

Project Relative Configuration "schemaspy-checkstyle"

Edit checkstyle configuration.

eclipse-cs ECLIPSE-CHECKSTYLE INTEGRATION

Known modules

Input filter text here

- Local Final Variable Names
- Local Variable Names
- Member Names
- Method Names
- Method Type Parameter Name
- Package Names
- Parameter Names
- Static Variable Names
- Type Names
- Headers
- Imports
- Size Violations
  - Anonymous inner classes lengths
  - Executable Statement Size
  - Maximum File Length
  - Maximum Line Length
  - Maximum Me
  - Maximum Pa
  - Outer Type N
- Whitespace
- Regexp

Configured modules for group "Size Violations"

Enabled	Module	Severity	Comment
✓	Anonymous inner classes	inherit	
✓	Executable Statement Size	inherit	
✓	Maximum File Length	inherit	
✓	Maximum Line Length	inherit	
✓	Maximum Method Length	inherit	
✓	Maximum Parameters	inherit	
✓	Outer Type Number	inherit	

Description:

Checks for long source
Rationale: If a source f
into several individual

Runs: 0/0

New module

Configuration of checkstyle module "Maximum File Length"

Edit the module configuration.

eclipse-cs ECLIPSE-CHECKSTYLE INTEGRATION

General | Advanced

Severity: inherit

Properties:

max: 500

☑ Translate tokens ☐ Sort tokens

Default Cancel OK

☑ Open module editor(s) on add action

Cancel OK

Checkstyle with Maven

To integrate Checkstyle with Maven 3, enter the code given below in **pom.xml:**

```
<reporting>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-checkstyle-plugin</artifactId>
          <version>3.1.2</version>
          <reportSets>
            <reportSet>
              <reports>
                <report>checkstyle</report>
              </reports>
            </reportSet>
          </reportSets>
        </plugin>
      </plugins>
    </reporting>
```

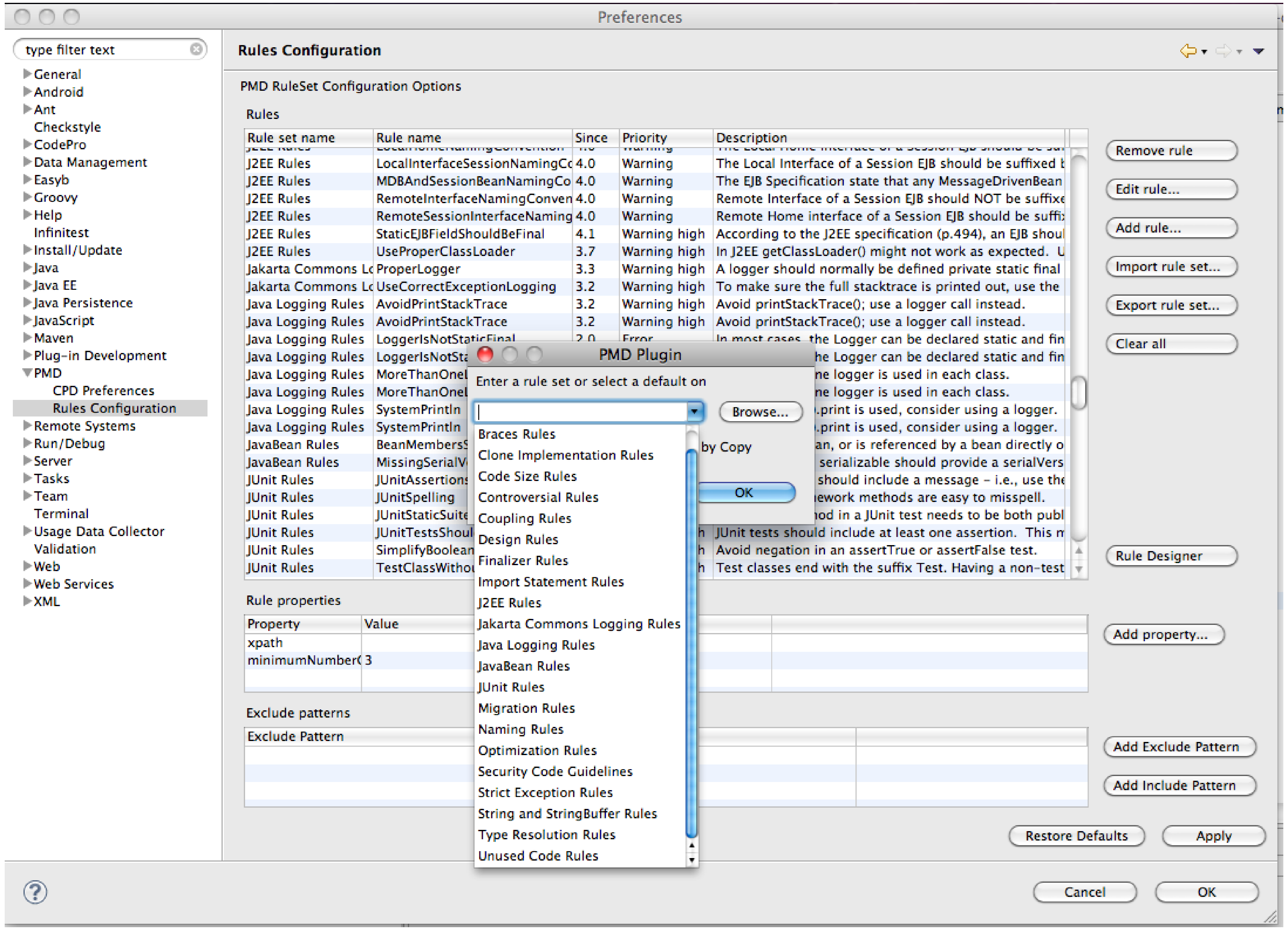
Running **mvn checkstyle:checkstyle** or **mvn site** will analyze your source code and generate XML reports.

PMD

PMD is another popular static analysis tool, which focuses on potential coding problems such as unused code, code size and complexity, and coding practices.

- PMD contains rules like Empty If Statement, Broken Null Check, Avoid Deeply Nested If Statements, and Logger Is Not Static Final
- PMD does have some more technical rules, and more specialized ones such as rules related to JSF and Android, when compared to Checkstyle
- PMD also comes with CPD, which is an open source detector of duplicated code

PMD Rules in Eclipse



Preferences

type filter text

- General
- Android
- Ant
- Checkstyle
- CodePro
- Data Management
- Easyb
- Groovy
- Help
- Infinitest
- Install/Update
- Java
- Java EE
- Java Persistence
- JavaScript
- Maven
- Plug-in Development
- PMD
  - CPD Preferences
  - Rules Configuration
- Remote Systems
- Run/Debug
- Server
- Tasks
- Team
- Terminal
- Usage Data Collector
- Validation
- Web
- Web Services
- XML

Rules Configuration

PMD RuleSet Configuration Options

Rules

Rule set name	Rule name	Since	Priority	Description
J2EE Rules	LocalInterfaceSessionNamingCo	4.0	Warning	The Local Interface of a Session EJB should be suffixed b
J2EE Rules	MDBAndSessionBeanNamingCo	4.0	Warning	The EJB Specification state that any MessageDrivenBean
J2EE Rules	RemoteInterfaceNamingConven	4.0	Warning	Remote Interface of a Session EJB should NOT be suffixe
J2EE Rules	RemoteSessionInterfaceNaming	4.0	Warning	Remote Home interface of a Session EJB should be suffi:
J2EE Rules	StaticEJBFieldShouldBeFinal	4.1	Warning high	According to the J2EE specification (p.494), an EJB shou
J2EE Rules	UseProperClassLoader	3.7	Warning high	In J2EE getClassLoader() might not work as expected. U
Jakarta Commons Lo	ProperLogger	3.3	Warning high	A logger should normally be defined private static final
Jakarta Commons Lo	UseCorrectExceptionLogging	3.2	Warning high	To make sure the full stacktrace is printed out, use the
Java Logging Rules	AvoidPrintStackTrace	3.2	Warning high	Avoid printStackTrace(); use a logger call instead.
Java Logging Rules	AvoidPrintStackTrace	3.2	Warning high	Avoid printStackTrace(); use a logger call instead.
Java Logging Rules	LoggerIsNotStaticFinal	2.0	Error	In most cases, the Logger can be declared static and fin
Java Logging Rules	LoggerIsNotSta			he Logger can be declared static and fin
Java Logging Rules	MoreThanOne			ne logger is used in each class.
Java Logging Rules	MoreThanOne			ne logger is used in each class.
Java Logging Rules	SystemPrintln			.print is used, consider using a logger.
Java Logging Rules	SystemPrintln			.print is used, consider using a logger.
JavaBean Rules	BeanMembersS			an, or is referenced by a bean directly o
JavaBean Rules	MissingSerialV			serializable should provide a serialVers
JUnit Rules	JUnitAssertions			should include a message – i.e., use the
JUnit Rules	JUnitSpelling			ework methods are easy to misspell.
JUnit Rules	JUnitStaticSuite			od in a JUnit test needs to be both publ
JUnit Rules	JUnitTestsShoul		h	JUnit tests should include at least one assertion. This m
JUnit Rules	SimplifyBoolean		h	Avoid negation in an assertTrue or assertFalse test.
JUnit Rules	TestClassWitho		h	Test classes end with the suffix Test. Having a non-test

Remove rule | Edit rule... | Add rule... | Import rule set... | Export rule set... | Clear all | Rule Designer

Rule properties

Property	Value
xpath	
minimumNumber(	3

Add property...

Exclude patterns

Exclude Pattern

Add Exclude Pattern | Add Include Pattern

Restore Defaults | Apply

Cancel | OK

PMD Plugin

Enter a rule set or select a default on

Browse...

by Copy

OK

- Braces Rules
- Clone Implementation Rules
- Code Size Rules
- Controversial Rules
- Coupling Rules
- Design Rules
- Finalizer Rules
- Import Statement Rules
- J2EE Rules
- Jakarta Commons Logging Rules
- Java Logging Rules
- JavaBean Rules
- JUnit Rules
- Migration Rules
- Naming Rules
- Optimization Rules
- Security Code Guidelines
- Strict Exception Rules
- String and StringBuffer Rules
- Type Resolution Rules
- Unused Code Rules

PMD with Ant

PMD comes with an Ant task that can be used to generate PMD and CPD reports.

First step is to define the tasks, as shown below:

```
<path id="pmd.classpath">
        <pathelement location="${build}"/>
        <fileset dir="lib/pmd">
                <include name="*.jar"/>
        </fileset>
</path>
<taskdef name="pmd" classname="net.sourceforge.pmd.ant.PMDTask“ classpathref="pmd.classpath"/>
<taskdef name="cpd" classname="net.sourceforge.pmd.cpd.CPDTask“ classpathref="pmd.classpath"/>
```

PMD with Ant

PMD comes with an Ant task that can be used to generate PMD and CPD reports.

- Next, generate the PMD XML report by calling the PMD task, as shown below:

```
<target name="pmd">
<taskdef name="pmd" classname="net.sourceforge.pmd.ant.PMDTask“ classpathref="pmd.classpath"/>
<pmd rulesetfiles="basic" shortFilenames="true">
<formatter type="xml" toFile="target/pmd.xml" />
<fileset dir="src/main/java" includes="**/*.java"/>
</pmd>
</target>
```

To generate the CPD XML report:

```
<target name="cpd">
<cpd minimumTokenCount="100" format="xml" outputFile="/target/cpd.xml">
<fileset dir="src/main/java" includes="**/*.java"/>
</cpd>
</target>
```

PMD with Maven

To configure Maven 2 and 3 for PMD and CPD reports with an exported XML ruleset, use the code snippet given below:

```
<reporting>
        <plugins>
                <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-pmd-plugin</artifactId>
                <version>3.14.0</version>
                <configuration>
                        <!-- PMD options -->
                        <targetJdk>1.6</targetJdk>
                        <aggregate>true</aggregate>
                        <format>xml</format>
                        <rulesets>
                                <ruleset>/pmd-rules.xml</ruleset>
                        </rulesets>
                        <!-- CPD options -->
                        <minimumTokens>20</minimumTokens>
                        <ignoreIdentifiers>true</ignoreIdentifiers>
                </configuration>
                </plugin>
        </plugins>
</reporting>
```

PMD with Maven

Run either **mvn site** or **mvn pmd:pmd pmd:cpd** to generate the PMD and CPD reports. You can also run PMD automatically when building your project using the snippet below:

```
<project>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-pmd-plugin</artifactId>
        <version>3.14.0</version>
        <configuration>
                      <failOnViolation>true</failOnViolation>
                        <printFailingErrors>true</printFailingErrors>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>check</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

This will run PMD automatically during the **verify** phase of the build. You can additionally run CPD, if you add cpd-check as a goal.

FindBugs

FindBugs is a code quality analysis tool that checks application byte code for potential bugs, performance problems, and poor coding habits.

- FindBugs can detect critical issues such as null pointer exceptions, and infinite loops
- Although FindBugs detects less number of issues, the ones detected are critical
- FindBugs is less configurable than the other tools
- In FindBugs, one can't configure a shared XML file between Maven builds and the IDE

FindBugs with Ant

FindBugs comes bundled with an Ant task. Define the FindBugs task in Ant as shown below.

- FindBugs needs to point to the FindBugs home directory, where the binary distribution has been unzipped

- To make the build more portable, store the FindBugs installation in the project directory structure, in the **tools/findbugs directory:**

```
<property name="findbugs.home"
value="tools/findbugs" />
<taskdef name="findbugs"
classname="edu.umd.cs.findbugs.anttask.FindB
ugsTask" >
<classpath>
<fileset dir="${findbugs.home}/lib"
includes="**/*.jar"/>
</classpath>
</taskdef>
```

- To run FindBugs, set up a target as shown below:

- Note that it runs against the application byte-code, and not source code, so you need to compile your source code first:

```
<target name="findbugs" depends="compile">
<findbugs home="${findbugs.home}"
output="xml" outputFile="target/findbugs.xml">
<class location="${classes.dir}" />
<auxClasspath refId="dependency.classpath" />
<sourcePath path="src/main/java" />
</findbugs>
</target>
```

FindBugs with Maven

Configure FindBugs in the reporting section as shown in the **pom.xml** file:

```
<reporting>
        <plugins>
                <plugin>
                        <groupId>org.codehaus.mojo</groupId>
                        <artifactId>findbugs-maven-plugin</artifactId>
                        <version>3.0.4-SNAPSHOT</version>
                        <configuration>
                                <effort>Max</effort>
                                <xmlOutput>true</xmlOutput>
                        </configuration>
                </plugin>
        </plugins>
</reporting>
```

Reporting on Code Quality

Jenkins provides a useful plugin for code quality called the Warnings Next Generation plugin. It collects compiler warnings or issues reported by static analysis tools and visualizes the results.

***Warnings Next Generation* plugin caters to reports on code quality metrics received from the static analysis tools, including:**

For Java:
Checkstyle, CPD, PMD, FindBugs, and jcreport

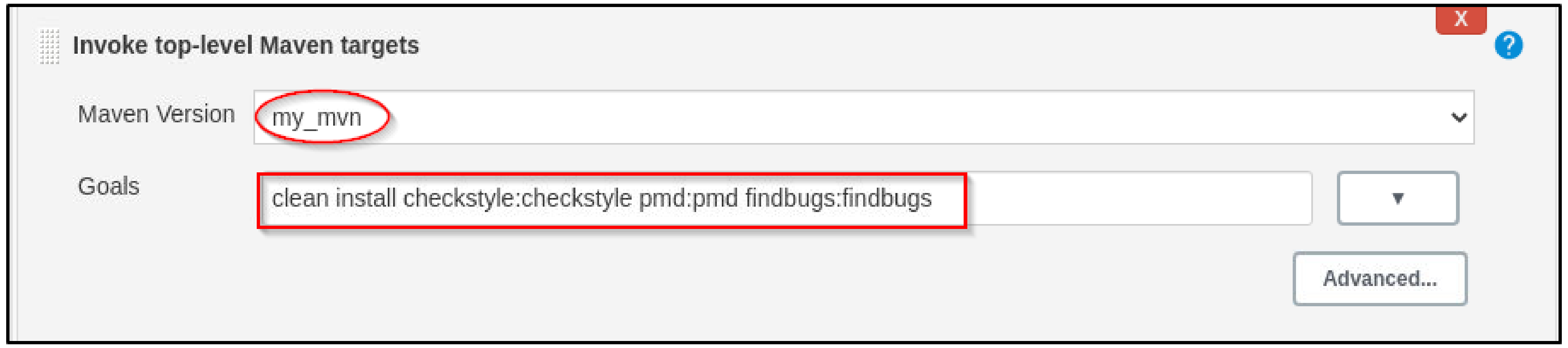
For Groovy:
codenarc

For JavaScript:
jslint

For .Net:
gendarme and stylecop

Reporting on Code Quality

The Warnings Next Generation plugin publishes a report of the issues found in the build. You can find the details of the issues in the summary report.


Invoke top-level Maven targets

Maven Version: my_mvn

Goals: clean install checkstyle:checkstyle pmd:pmd findbugs:findbugs

In the above example, static analysis tools are called directly, which results in faster builds.

After the setup, configure the Warnings Next Generation plugin to generate reports and, if required, trigger notifications based on the report results.

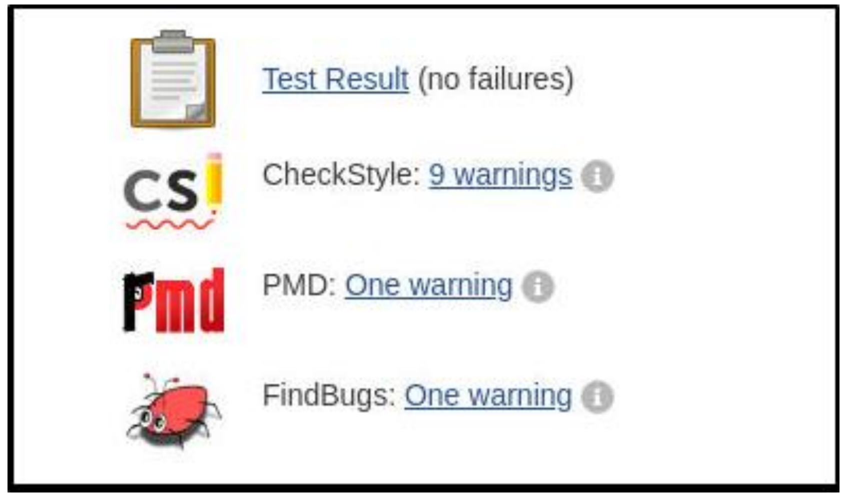
Go to the Add Post Build Actions, select record compiler warnings and static analysis results, and add all the static analysis tools you want to use

Internals of Jenkins Jobs

Working with Freestyle Jobs

Freestyle build jobs provide maximum configuration flexibility and are the best option for non-Java projects.

- While setting up Warnings Next Generation plugin with a Freestyle build job, specify the locations to all XML reports generated by the static analysis tools
- Use a wildcard expression to identify the reports you want. For example: ****/target/checkstyle.xml**
- The Warnings Next Generation plugin generates a report which tracks the total number of issues over time



Test Result (no failures)

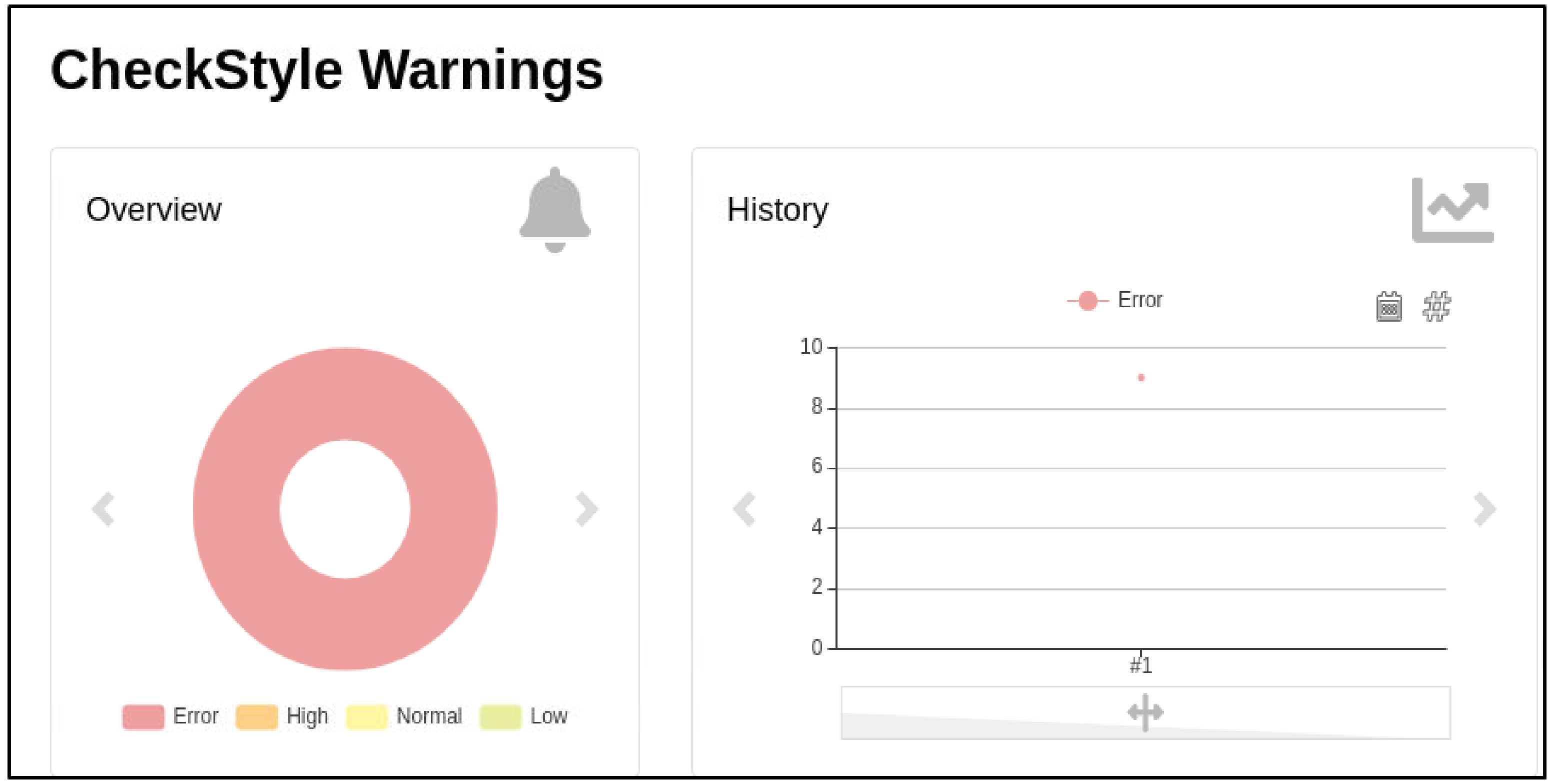
CheckStyle: 9 warnings

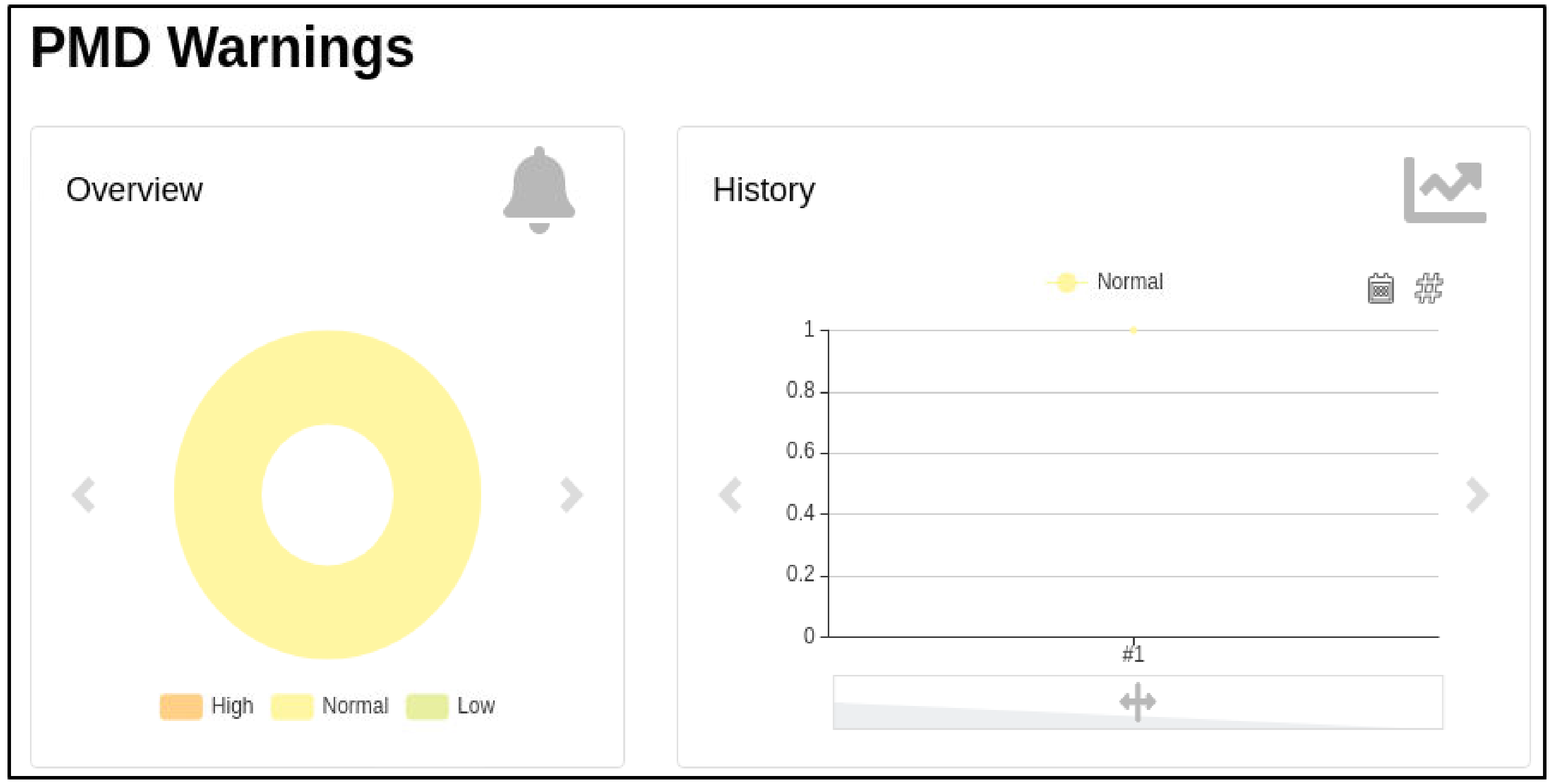
PMD: One warning

FindBugs: One warning

Working with Freestyle Jobs

By clicking on the report, one can expand and analyze the details of every issue, as shown below:





Code Quality and Jenkins

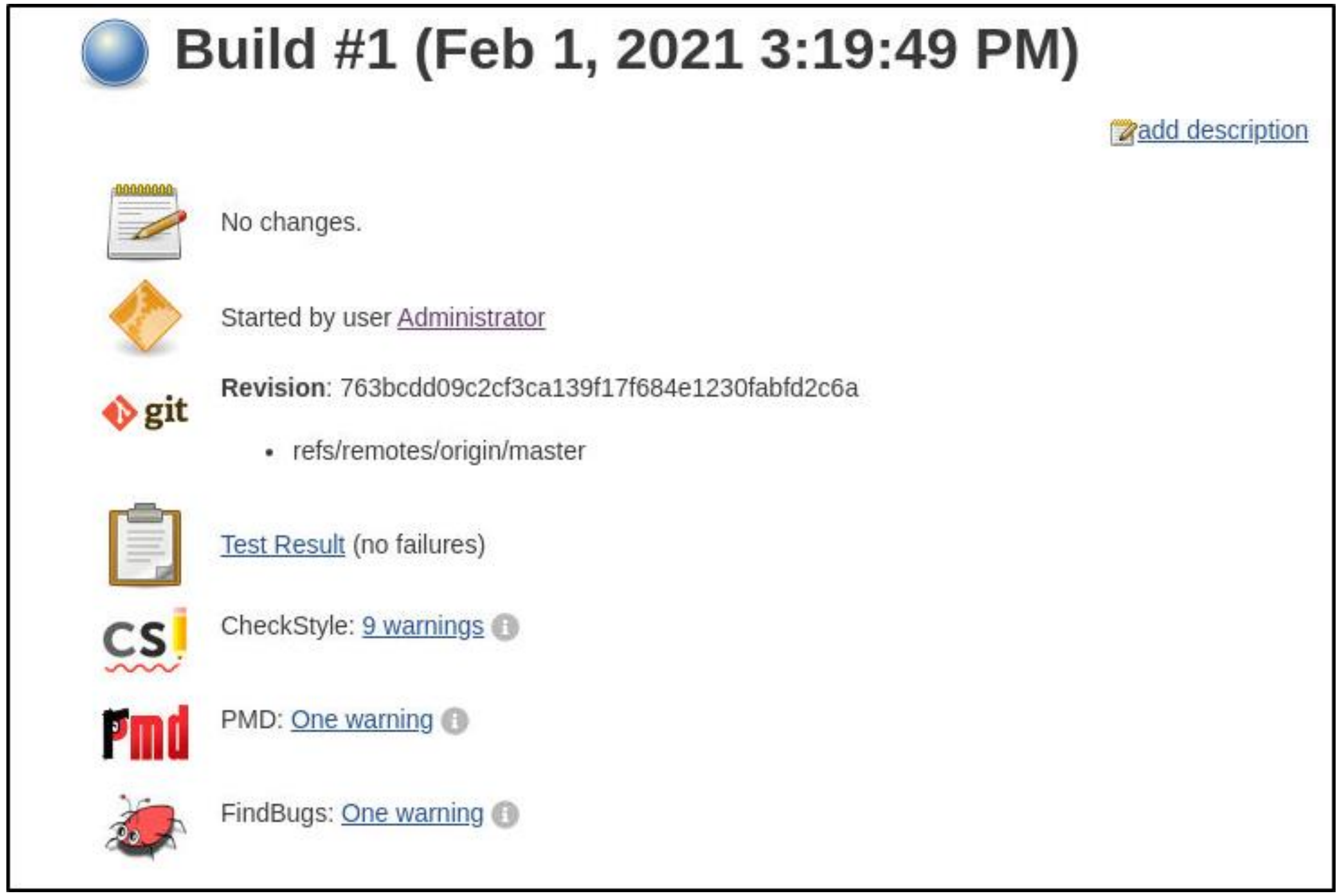
Working with Maven Build Jobs

Maven build jobs in Jenkins use the Maven conventions and data present in the project's **pom.xml** file to make configuration easier and execution faster.

- In Warnings Next Generation plugin with a Maven build job, Jenkins uses **pom.xml** file data to reduce configuration code for the plugin
- Jenkins gets the location of XML reports for many of the static analysis tools based on the Maven conventions and plugin configurations in **pom.xml**
- To override project conventions, choose the pattern option in the XML filename pattern drop-down list and enter a path similar to the freestyle build jobs

Working with Maven Build Jobs

By default, the Warnings Next Generation plugin will display an aggregated view of the code quality metrics as shown below. Click on each of the tool to view the detailed reports.



Build #1 (Feb 1, 2021 3:19:49 PM)

add description

No changes.

Started by user Administrator

Revision: 763bcdd09c2cf3ca139f17f684e1230fabfd2c6a

- refs/remotes/origin/master

Test Result (no failures)

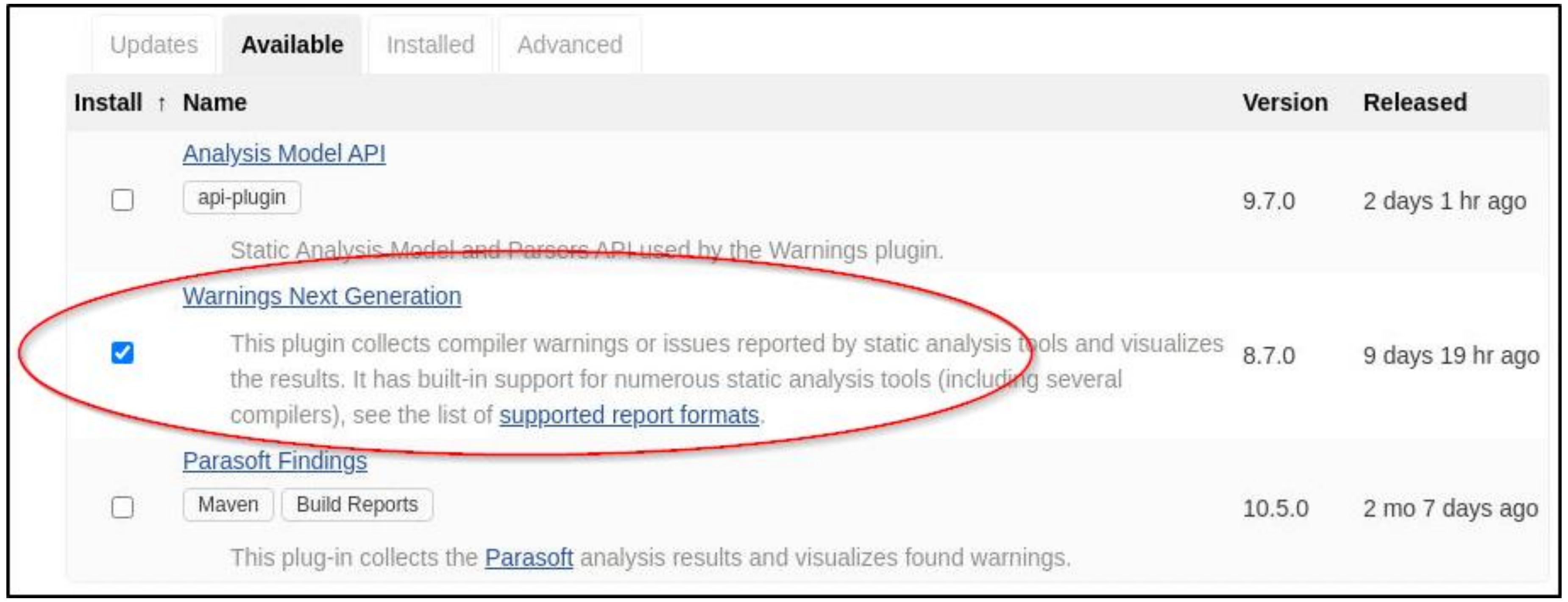
CheckStyle: 9 warnings

PMD: One warning

FindBugs: One warning

Using Checkstyle, PMD, and FindBugs Reports

The Jenkins Static Analysis Suite comprises plugins like *Android Lint, CheckStyle, Dry, FindBugs, PMD, Warnings, Static Analysis Utilities,* and *Static Analysis Collector*. It is now obsolete and replaced by a single plugin called as Warnings Next Generation plugin.


Updates | **Available** | Installed | Advanced

Install ↑	Name	Version	Released
☐	Analysis Model API api-plugin Static Analysis Model and Parsers API used by the Warnings plugin.	9.7.0	2 days 1 hr ago
☑	Warnings Next Generation This plugin collects compiler warnings or issues reported by static analysis tools and visualizes the results. It has built-in support for numerous static analysis tools (including several compilers), see the list of supported report formats.	8.7.0	9 days 19 hr ago
☐	Parasoft Findings Maven · Build Reports This plug-in collects the Parasoft analysis results and visualizes found warnings.	10.5.0	2 mo 7 days ago

- Once installed, set up the static analysis tools for which you want to generate the report by selecting record compiler warnings and static analysis results in the **add a post build** step of your jenkins job
- After configuring the required static analysis tools, run the jenkins job to generate the reports automatically for every build

Assisted Practice
Integration of Static Code Analysis Tools with Jenkins

Problem Statement: You are given a project to integrate and configure **Static Code Analysis tools** like *Checkstyle, Pmd,* and *FindBugs* with Jenkins.

Steps to perform:

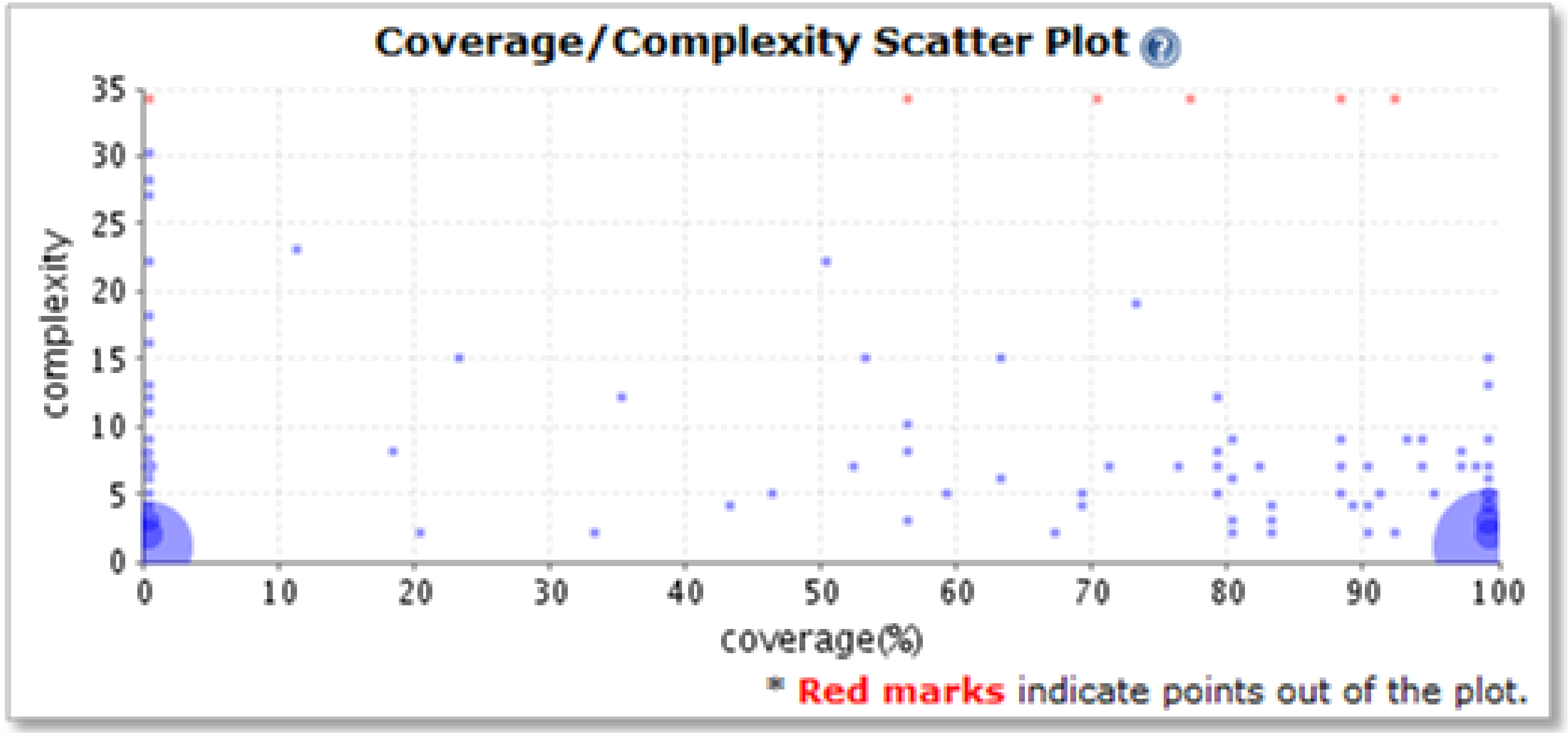
1. Log in to Jenkins
2. Install **Warnings Next Generation** plugin
3. Create a maven job

Reporting on Code Complexity

Reporting on Code Complexity

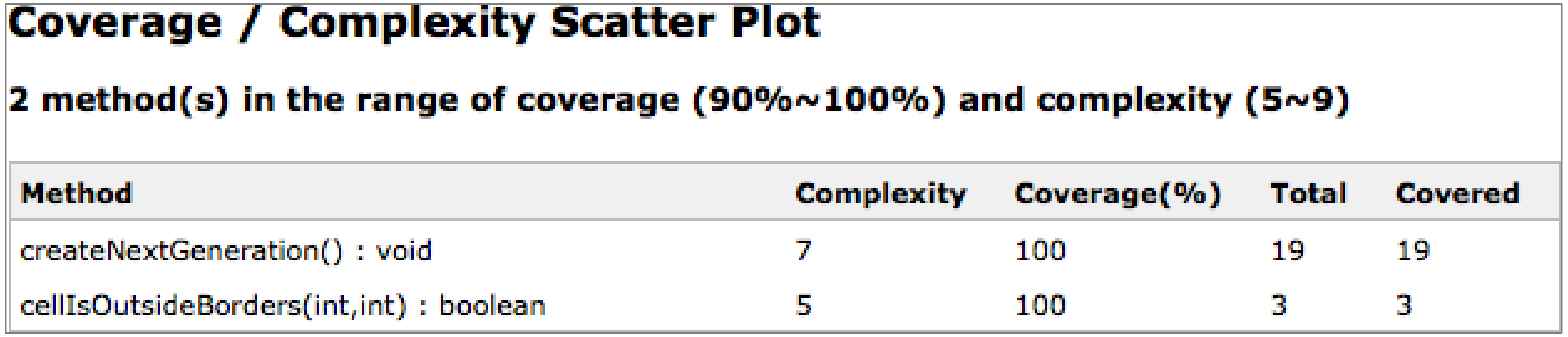
Code complexity can be measured in different ways and one of them is Cyclomatic Complexity. This involves measuring the number of different paths through a method.

- The Coverage/Complexity Scatter Plot plugin is designed to display code complexity report in Jenkins
- Complex or untested methods will appear high on the graph, where as the well-written and tested methods will appear lower down



Reporting on Code Complexity

The Coverage/Complexity Scatter Plot also allows to expand and get more detailed information of the methods. You can click on any point in the graph to display the corresponding methods with their test coverage and complexity.



Coverage / Complexity Scatter Plot

2 method(s) in the range of coverage (90%~100%) and complexity (5~9)

Method	Complexity	Coverage(%)	Total	Covered
createNextGeneration() : void	7	100	19	19
cellIsOutsideBorders(int,int) : boolean	5	100	3	3

- Since this plugin requires Clover, the build would have already generated a Clover XML coverage report. Currently, it supports Clover and Cobertura plugins

SonarQube

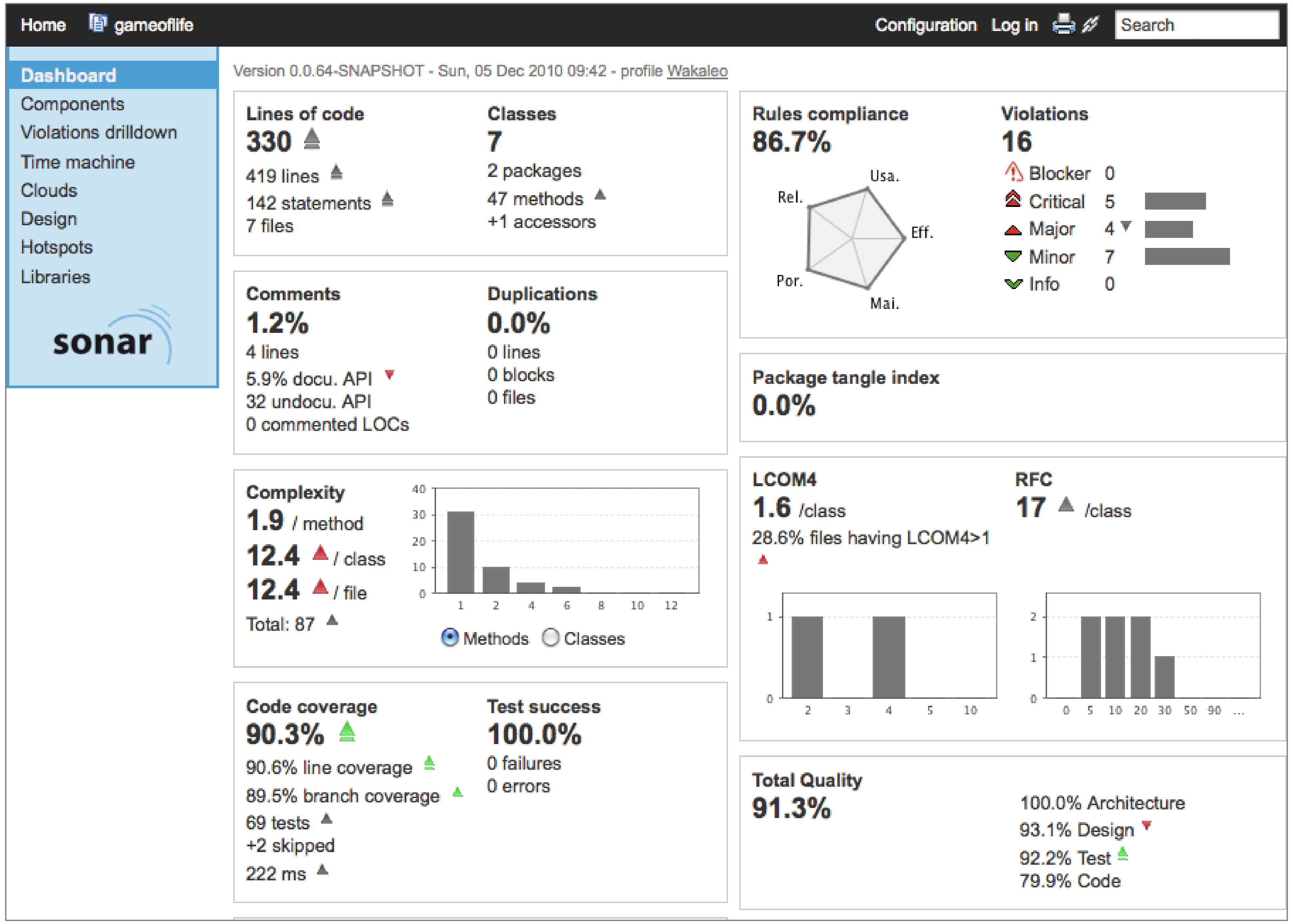
Introduction

Sonar is a tool that centralizes a range of code quality metrics into a single website. It uses Maven plugins to analyze Maven projects and generate a set of code quality metrics reports.

- SonarQube generates reports on code coverage, rule compliance and documentation, complexity, maintainability, and technical debt
- It can be extend to add support for other languages
- The rules configured centrally on the Sonar website and the Maven project don’t require any particular configuration
- This makes Sonar highly suitable for working on Maven projects

Sonar Report

The screenshot of the Sonar server that runs separately along with analyzing and displaying results is given below:



Home | gameoflife | Configuration | Log in | Search

Dashboard
Components
Violations drilldown
Time machine
Clouds
Design
Hotspots
Libraries

sonar

Version 0.0.64-SNAPSHOT - Sun, 05 Dec 2010 09:42 - profile Wakaleo

Lines of code
330
419 lines
142 statements
7 files

Classes
7
2 packages
47 methods
+1 accessors

Comments
1.2%
4 lines
5.9% docu. API
32 undocu. API
0 commented LOCs

Duplications
0.0%
0 lines
0 blocks
0 files

Complexity
1.9 / method
12.4 / class
12.4 / file
Total: 87

Code coverage
90.3%
90.6% line coverage
89.5% branch coverage
69 tests
+2 skipped
222 ms

Test success
100.0%
0 failures
0 errors

Rules compliance
86.7%

Violations
16

Blocker	0
Critical	5
Major	4
Minor	7
Info	0

Package tangle index
0.0%

LCOM4
1.6 /class
28.6% files having LCOM4>1

RFC
17 /class

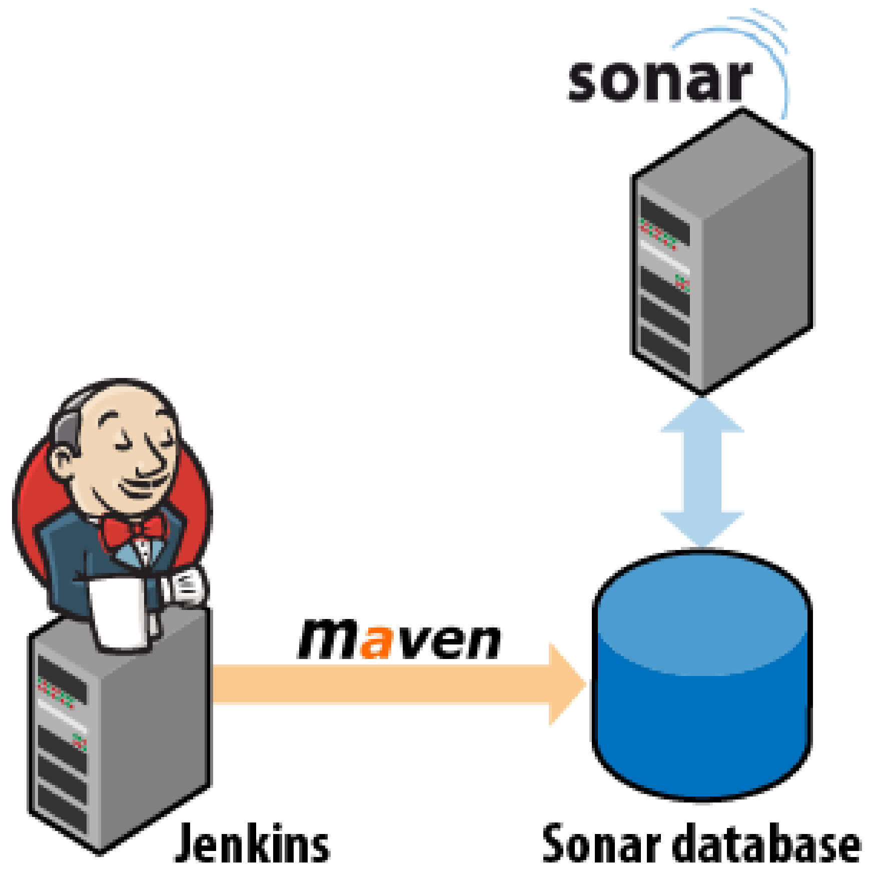
Total Quality
91.3%

100.0% Architecture
93.1% Design
92.2% Test
79.9% Code

SonarQube with Jenkins

The Jenkins **SonarQube Scanner** plugin provides the facility to define Sonar instances for all the projects and activates Sonar in particular builds.

- One can run Sonar server on a separate machine or on the same Jenkins instance
- Jenkins instance must have JDBC access to the Sonar database. This is the only constraint
- Sonar injects code quality metrics directly into the database
- Sonar also works with Ant for non-maven users

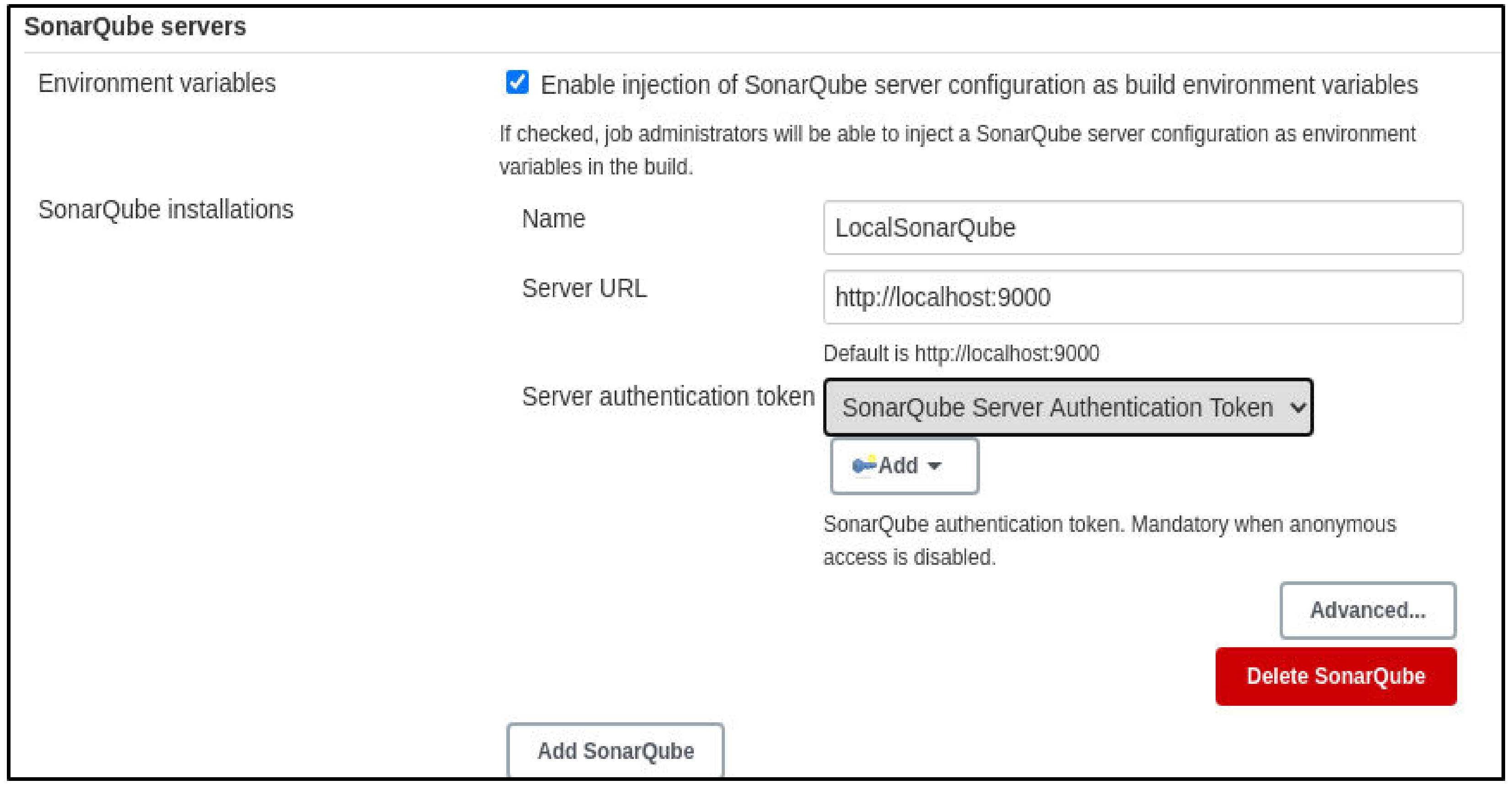


SonarQube with Jenkins

- Once the Sonar Scanner plugin is installed from the plugin manager, configure the SonarQube server in the Configure System screen, under the SonarQube server section

- Define the Sonar instances. By default, it assumes that Sonar is running locally with the default embedded database

- For a production environment, run Sonar on a real database such as MySQL



SonarQube servers

Environment variables

[✓] Enable injection of SonarQube server configuration as build environment variables

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

Name	LocalSonarQube
Server URL	http://localhost:9000

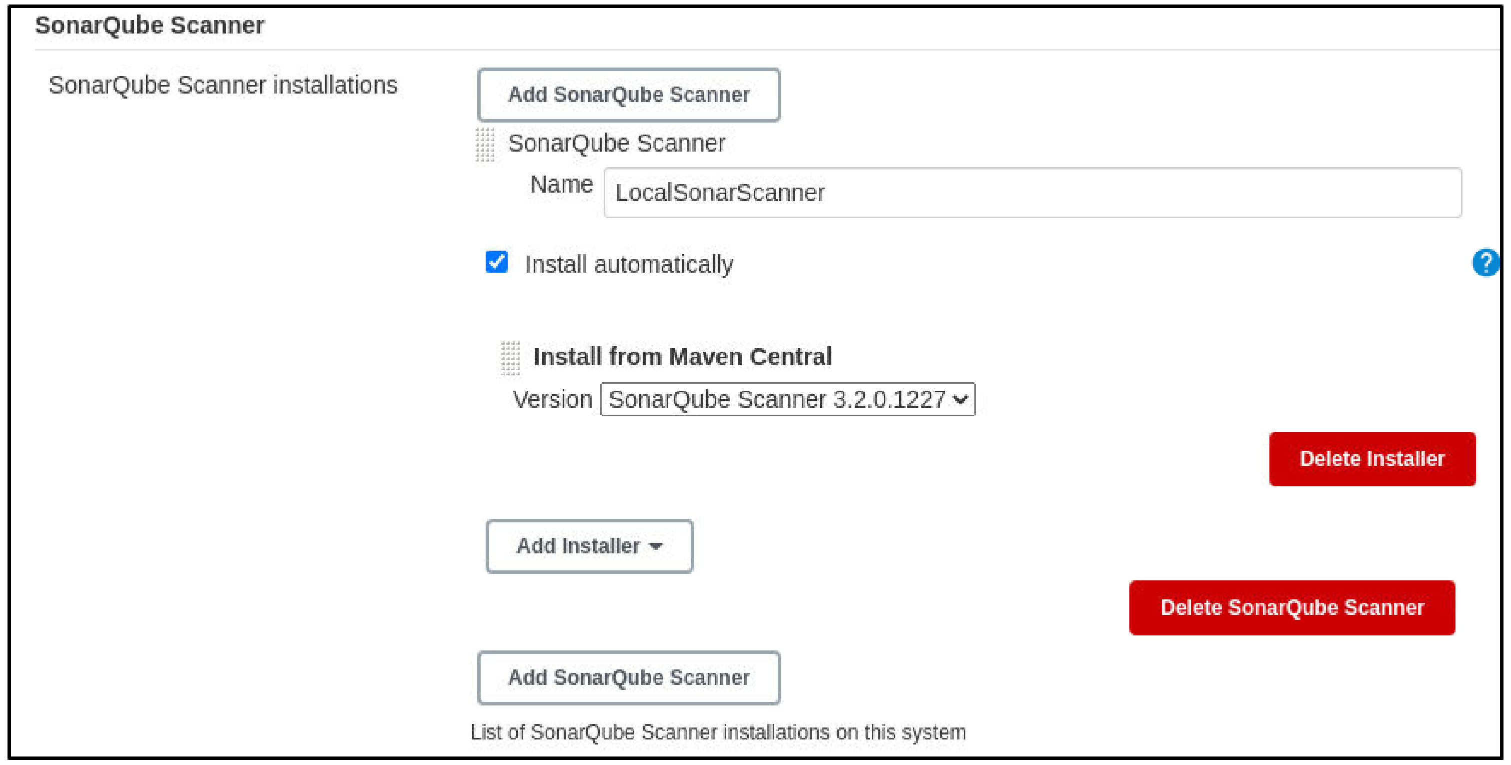
Default is http://localhost:9000

Server authentication token	SonarQube Server Authentication Token

SonarQube authentication token. Mandatory when anonymous access is disabled.

Sonar Configuration

Configure the SonarQube Scanner in Jenkins by clicking on the **Add SonarQube** Scanner in the Global tool configuration settings as shown below:



SonarQube Scanner

SonarQube Scanner installations

Add SonarQube Scanner

SonarQube Scanner

Name: LocalSonarScanner

☑ Install automatically

Install from Maven Central

Version: SonarQube Scanner 3.2.0.1227

Delete Installer

Add Installer

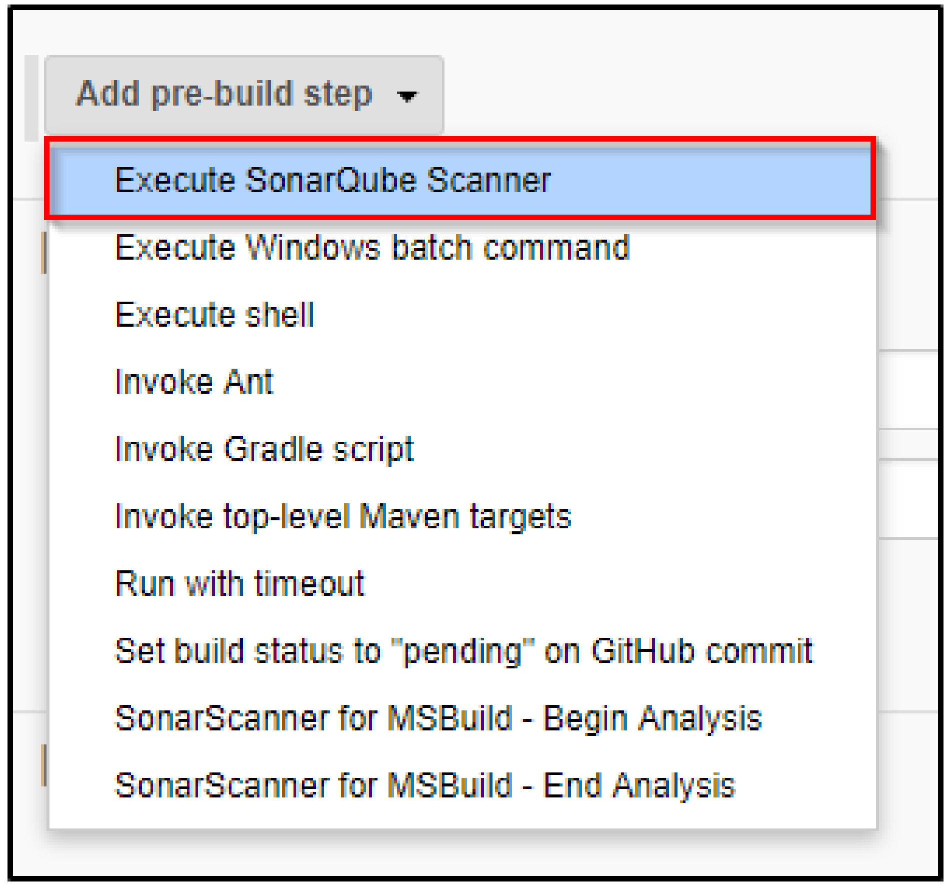
Delete SonarQube Scanner

Add SonarQube Scanner

List of SonarQube Scanner installations on this system

Sonar Configuration

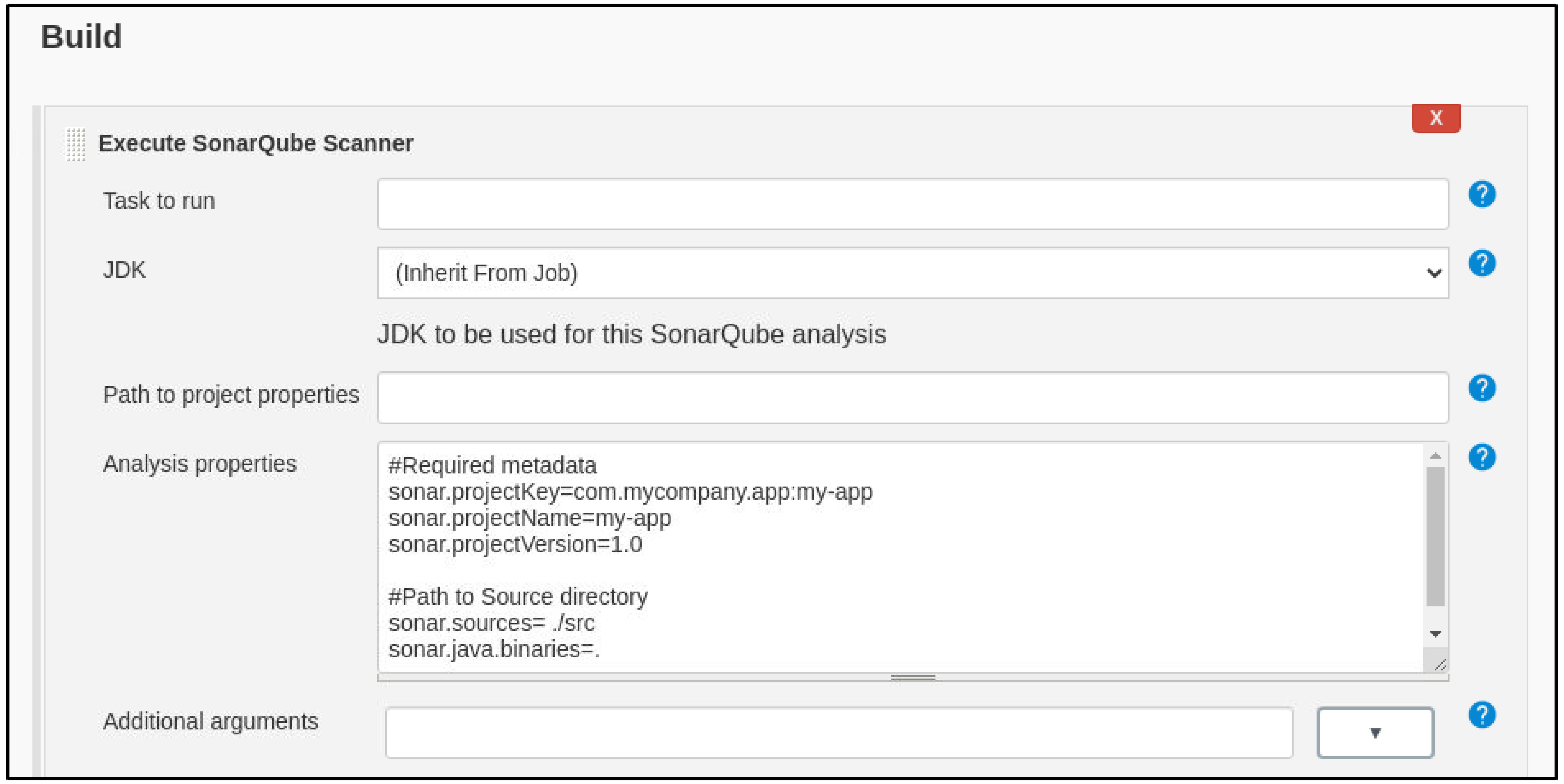
- Configure Sonar to run with one of the long-running Jenkins build jobs
- It is not recommended to run the Sonar build more than once in a day
- The default configuration executes a Sonar build in a Sonar-enabled build job when the job is triggered by a periodically scheduled build or by a manual build
- To activate Sonar in build job with the system-wide configuration, select the **Execute SonarQube Scanner** option in the **Build** step



Add pre-build step
Execute SonarQube Scanner
Execute Windows batch command
Execute shell
Invoke Ant
Invoke Gradle script
Invoke top-level Maven targets
Run with timeout
Set build status to "pending" on GitHub commit
SonarScanner for MSBuild - Begin Analysis
SonarScanner for MSBuild - End Analysis

Sonar Configuration

- Configure the project analysis settings
- There are multiple ways in which the analysis parameters may be configured
- You can configure the parameters as the command line arguments or you can create a .properties file containing all the analysis parameters and give the path to that file in the job configuration
- Sonar will run every time the build is started by one of the triggers


Build

Execute SonarQube Scanner

Task to run	
JDK	(Inherit From Job)
	JDK to be used for this SonarQube analysis
Path to project properties	
Analysis properties	see below
Additional arguments	

```
#Required metadata
sonar.projectKey=com.mycompany.app:my-app
sonar.projectName=my-app
sonar.projectVersion=1.0

#Path to Source directory
sonar.sources= ./src
sonar.java.binaries=.
```

Assisted Practice
SonarQube Integration with Jenkins

Problem Statement: You are given a project to integrate **SonarQube** with Jenkins.

Steps to Perform:

1. Install SonarQube 7.8 in the lab
2. Install and configure SonarQube Scanner plugin in Jenkins
3. Create a Jenkins job and run Sonarqube scanner

Key Takeaways

- Coding standards are rules that define the coding styles and conventions that are common across an organization
- It is important to set up a dedicated code quality build that runs after the build and test, and configure the build to fail
- Checkstyle provides support for the detection of duplicated code. One shall use tools such as CPD for better results in detection of duplicated code
- Warnings Next Generation plugin caters to reports on code quality metrics received from the static analysis tools
- Freestyle build jobs provide maximum configuration flexibility, and are the best option for non-Java projects

Lesson-End Project

Static Code Analysis

Problem Statement:

You're a DevOps engineer at ABC software, a service-based software development company. The company is trying to implement a new coding standard to help avoid code refactoring. As a pilot, the company wants to perform daily static code analysis on its new project which is a retail management system for their new clients FreshMart. You're tasked with setting up a Jenkins build job that polls the SCM for the FreshMart project nightly and performs static code analysis using SonarQube and publishes the result. The retail management system will be a Maven project and the developers will be committing their codes to a git repository daily before they clock out.

Requirements:

- The pipeline should be built with the SonarQube plugin
- The pipeline should poll the SCM every night and perform the analysis