

FULL STACK



CI/CD Pipeline with Jenkins

FULL STACK

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



FULL STACK

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



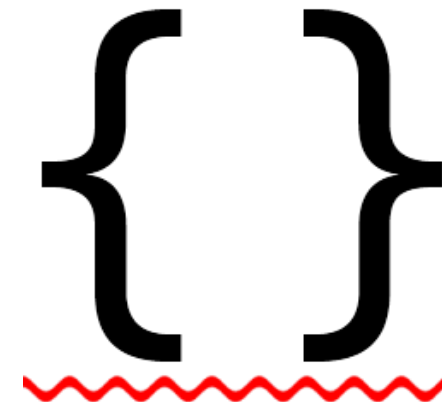
SonarQube



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

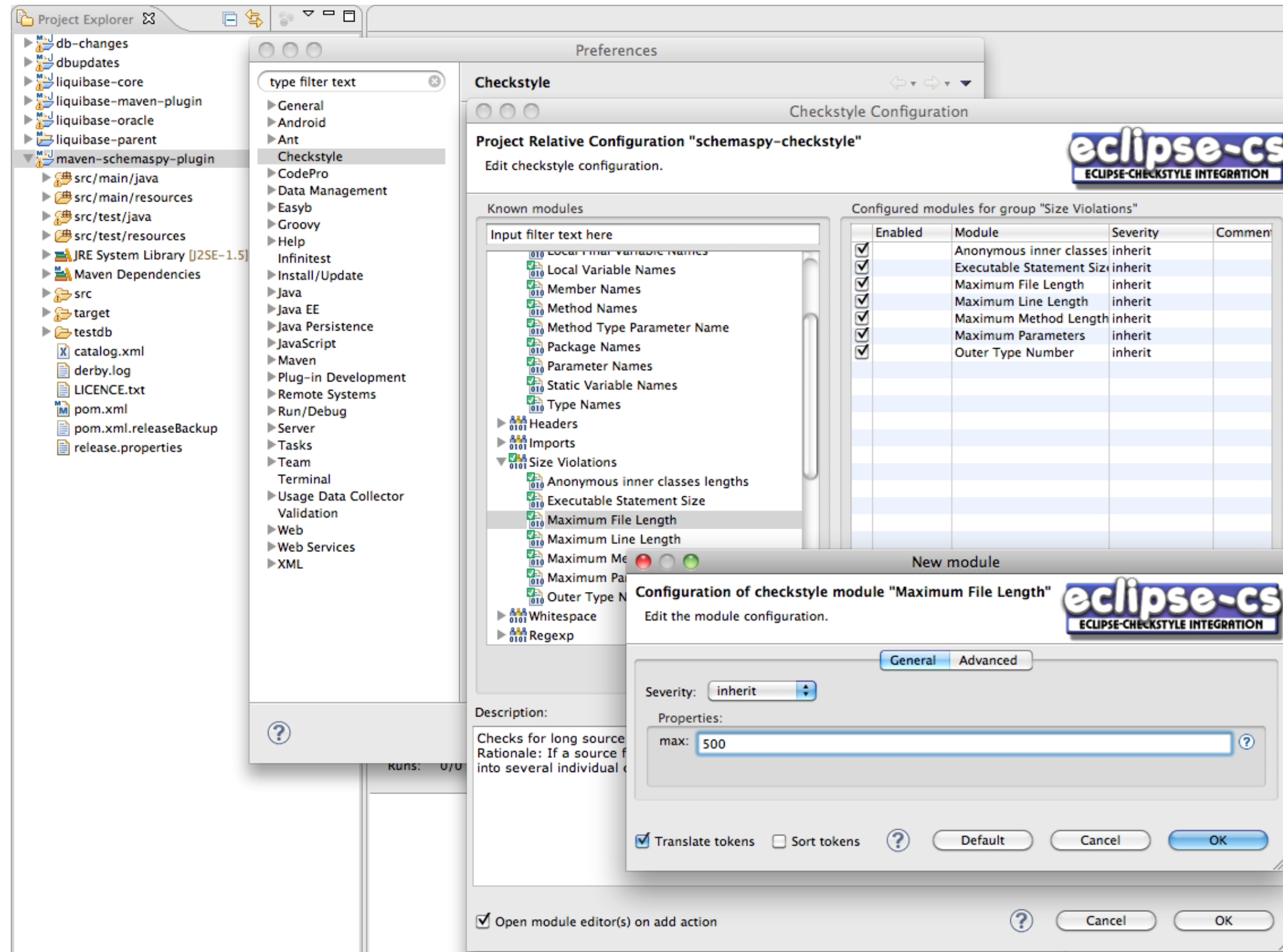
Checkstyle is a static analysis tool for Java. It was designed to define a set of coding standards, however, now it also checks for poor coding practices, and complex and duplicated code.

- 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

Checkstyle Eclipse Plugin



Checkstyle with Maven

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

Maven2:

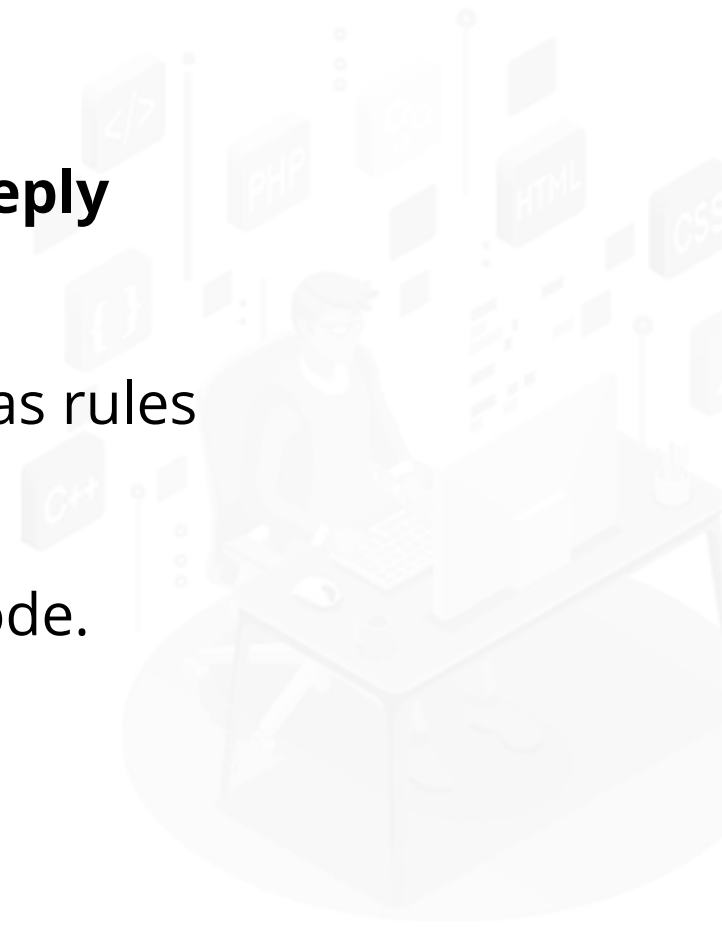
```
<reporting>
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>2.4</version>
    <configuration>
      <configLocation> src/main/config/company-checks.xml </configLocation>
    </configuration>
  </plugin>
</plugins>
</reporting>
```

- For a Maven 3 project, you need to add the plugin to the **<reportPlugins>** element of the **<configuration>** section of the **maven-site-plugin**.
- Running `mvn checkstyle:checkstyle` or `mvn site` will analyse your source code and generate XML reports

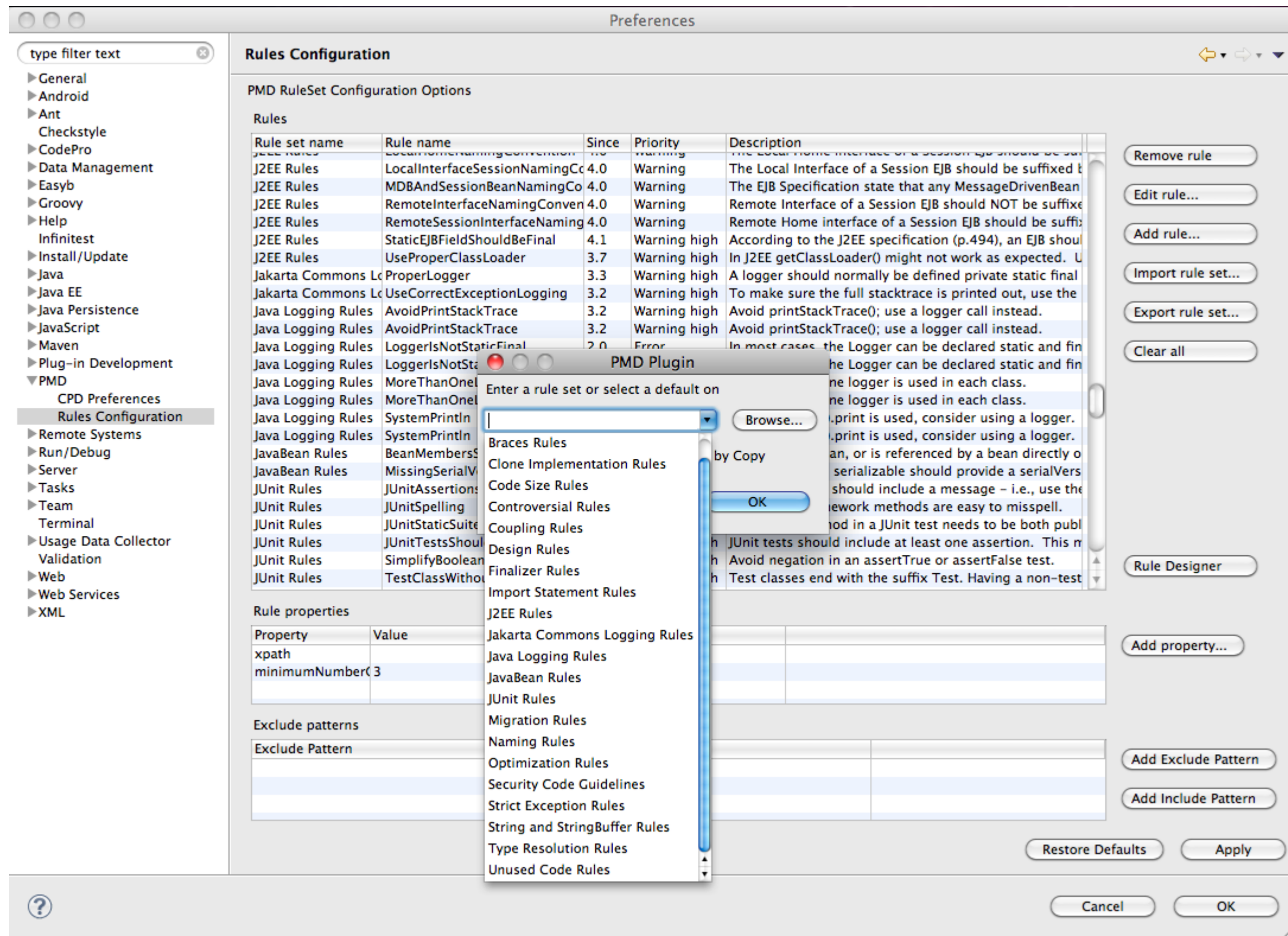
PMD

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



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 for PMD and CPD reports with an exported XML ruleset, follow the code snippet as shown here:

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <version>2.5</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

In Maven 3, place the plugin definition in the **<maven-siteplugin>** configuration section. Run either **mvn site** or **mvn pmd:pmd pmd:cpd** to generate the PMD and CPD reports.

```
<project>
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-site-plugin</artifactId>
<version>3.0-beta-2</version>
<configuration>
<reportPlugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-pmd-plugin</artifactId>
<version>2.5</version>
<configuration>
<!-- PMD options -->
<targetJdk>1.6</targetJdk>
<aggregate>true</aggregate>
<format>xml</format>
<rulesets>
<ruleset>/pmd-rules.xml</ruleset>
</rulesets>
```

```
<!-- CPD options -->
<minimumTokens>50</minimumTokens>
<ignoreIdentifiers>true</ignoreIdentifiers>
</configuration>
</plugin>
</reportPlugins>
</configuration>
<dependencies>
<dependency>
<groupId>com.wakaleo.code-
quality</groupId>
<artifactId>pmd-rules</artifactId>
<version>1.0.1</version>
</dependency>
</dependencies>
</plugin>
</plugins>
</build>
</project>
```

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**:
- 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:

```
<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>
```

```
<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

For Maven 2, keep a local copy of the FindBugs installation. 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>2.3.1</version>
      <configuration>
        <effort>Max</effort>
        <xmlOutput>true</xmlOutput>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```



FindBugs with Maven

For Maven 3:

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-site-plugin</artifactId>
    <version>3.0-beta-2</version>
    <configuration>
      <reportPlugins>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>findbugs-maven-plugin</artifactId>
          <version>2.3.1</version>
          <configuration>
            <effort>Max</effort>
            <xmlOutput>true</xmlOutput>
          </configuration>
        </plugin>
      </reportPlugins>
    </configuration>
  </plugin>
</plugins>
```

Reporting on Code Quality

Jenkins provides a useful plugin for code quality called the Violations plugin. Although it will not analyze your project source code, it does a great job reporting on the code quality metrics generated for individual builds and trends over time.

- Violation 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 Violations plugin does not generate the code quality metrics data. It is added in the build configuration as shown below:

Build

Maven Version	<input type="text" value="Maven 2.2.1"/>	
Root POM	<input type="text" value="pom.xml"/>	
Goals and options	<input type="text" value="clean checkstyle:checkstyle pmd:pmd findbugs:findbugs cobertura:cobertura javadoc:javadoc"/>	
		<input type="button" value="Advanced..."/>

- In the above example, plugins are called directly, which results in faster builds.
- After the set up, configure the violations plugin to generate reports and, if required, trigger notifications, based on the report results.
- Go to the **Post build Actions** and check the **Report Violations** checkbox.

FULL STACK




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 violations 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.**

☒ Report Violations



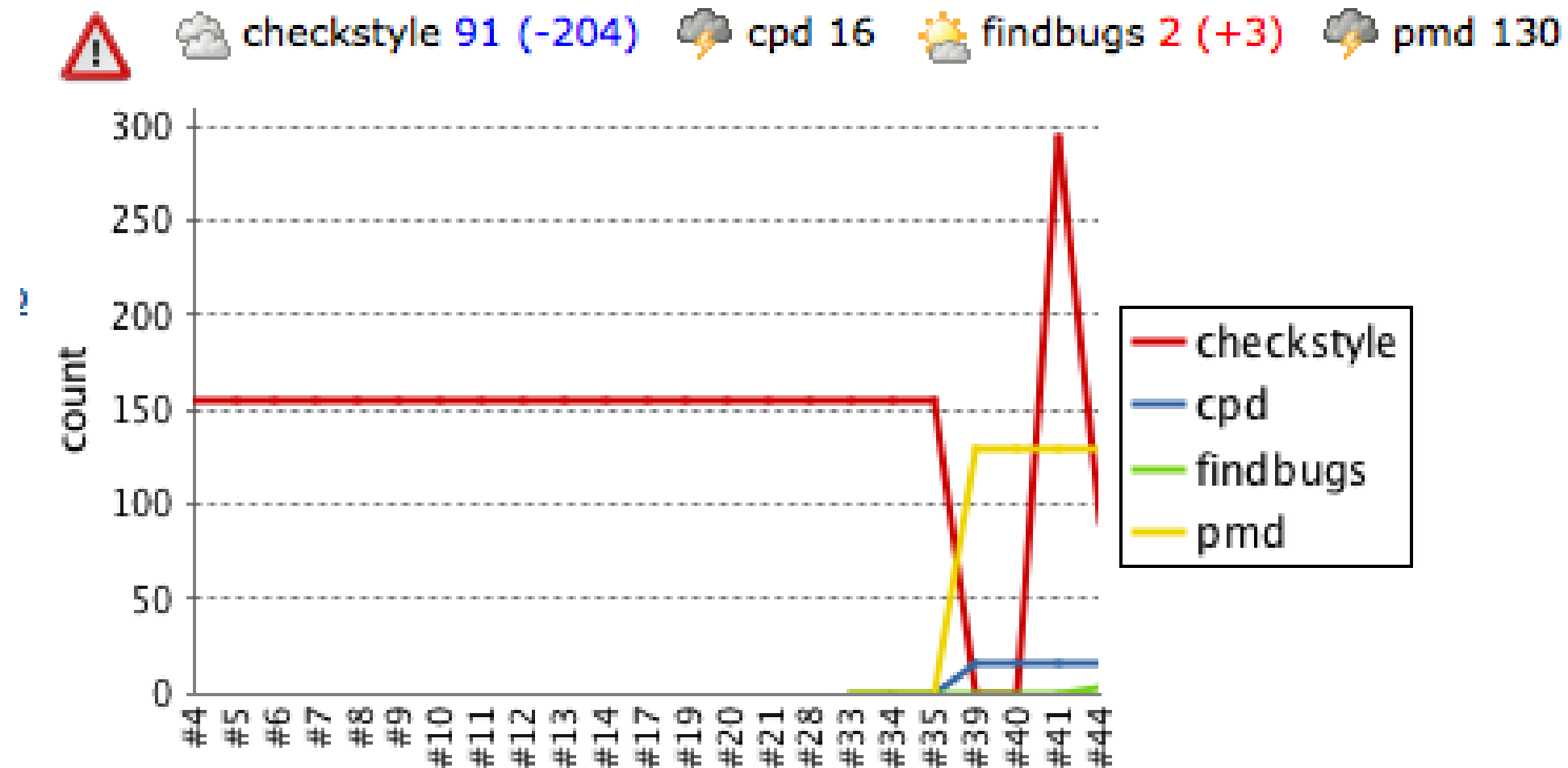
XML filename pattern

checkstyle	10	200	500	**/target/checkstyle-result.xml
codenarc	10	999	999	
cpd	0	10	15	**/target/cpd.xml
findbugs	0	50	50	**/target/findbugs.xml
fxcop	10	999	999	
gendarme	10	999	999	
jcreport	10	999	999	
jslint	10	999	999	
pmd	0	100	200	**/target/pmd.xml
pylint	10	999	999	
simian	10	999	999	
stylecop	10	999	999	
Per file limit	100			
Source Path Pattern				
Faux Project Path				
Source encoding	default			



Working with Freestyle Jobs

- The Violations plugin generates a graph which tracks the total number of issues over time.
- The graph illustrates different colored lines for each violation tracked.
- It also displays a summary of the latest results.



Working with Freestyle Jobs

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

Hudson

[Hudson](#) » [game-of-life-freestyle-metrics](#)

DISABLE AUTO REFRESH

[Back to Dashboard](#)[Status](#)[Changes](#)[Workspace](#)[Build Now](#)[Delete Project](#)[Configure](#)[Maven Report](#)[Violations](#)

Build History

#45

Dec 7, 2010 12:01:51 AM

200KB

#44

Dec 6, 2010 5:38:37 PM

200KB

#43

Dec 6, 2010 5:26:39 PM

179KB

#42

Dec 6, 2010 4:26:20 PM

157KB

#41

Dec 5, 2010 3:35:22 AM

273KB

#40

Dec 5, 2010 3:05:05 AM

132KB

#39

Dec 4, 2010 10:13:49 PM

132KB

#38

Dec 4, 2010 10:11:18 PM

133KB

#37

Dec 4, 2010 10:09:32 PM

250KB

#36

Dec 4, 2010 10:08:10 PM

250KB

#35

Dec 4, 2010 10:06:31 PM

156KB

#34

Dec 4, 2010 10:05:59 PM

156KB

#33

Dec 4, 2010 9:52:58 PM

156KB

#32

Dec 3, 2010 5:56:26 AM

158KB

#31

Dec 1, 2010 7:40:00 AM

158KB

Violations Report for build 45

Type	Violations	Files in violation
checkstyle	91	6
cpd	16	3
findbugs	2	2
pmd	130	6

checkstyle

filename	l	m	h	number ↑
gameoflife-core/src/main/java/com/wakaleo/gameoflife/domain/Grid.java	0	30	0	30
gameoflife-web/src/main/java/com/wakaleo/gameoflife/web/controllers/GameController.java	0	27	0	27
gameoflife-core/src/main/java/com/wakaleo/gameoflife/domain/Universe.java	0	18	0	18
gameoflife-core/src/main/java/com/wakaleo/gameoflife/domain/Cell.java	0	8	0	8
gameoflife-core/src/main/java/com/wakaleo/gameoflife/domain/GridReader.java	0	5	0	5
gameoflife-core/src/main/java/com/wakaleo/gameoflife/domain/GridWriter.java	0	3	0	3

©Simplilearn. All rights reserved.

simplilearn

FULL STACK

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 violations 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.

Build Settings

☐ E-mail Notification ?

☐ Publish documents ?

☒ violations ?

   XML filename pattern ?

checkstyle	10	999	999	auto ?
codenarc	10	999	999	auto ?
cpd	10	999	999	auto ?
findbugs	10	999	999	auto pattern ?
fxcop	10	999	999	
gendarme	10	999	999	
jcreport	10	999	999	auto ?
jslint	10	999	999	auto ?
pmd	10	999	999	auto ?
pylint	10	999	999	
simian	10	999	999	
stylecop	10	999	999	auto ?
Per file limit	100 ?			
Source encoding	default ?			

Working with Maven Build Jobs

The Violations plugin is primarily used in multi-module Maven projects. Below screenshot is an example of multi-module project:

Hudson

search

?

Hudson » [game-of-life-code-quality](#)

DISABLE AUTO REFRESH

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Workspace](#)

[Build Now](#)

[Delete Project](#)

[Configure](#)

[Modules](#)

[Violations](#)

Build History [\(trend\)](#)

#5 [Dec 7, 2010 11:33:37 AM](#) 470KB

#4 [Dec 7, 2010 11:30:38 AM](#) 624KB

Modules

S	W	Job ↓	Last Success	Last Failure	Last Duration	
		gameoflife	1 day 4 hr (#5)	N/A	11 sec	
		gameoflife-build	1 day 4 hr (#5)	N/A	1.6 sec	
		gameoflife-core	1 day 4 hr (#5)	N/A	33 sec	
		gameoflife-web	1 day 4 hr (#5)	N/A	19 sec	
		gameoflife-webservice	1 day 4 hr (#5)	N/A	1.5 sec	
		gameoflife-cli	1 day 4 hr (#5)	N/A	1.7 sec	

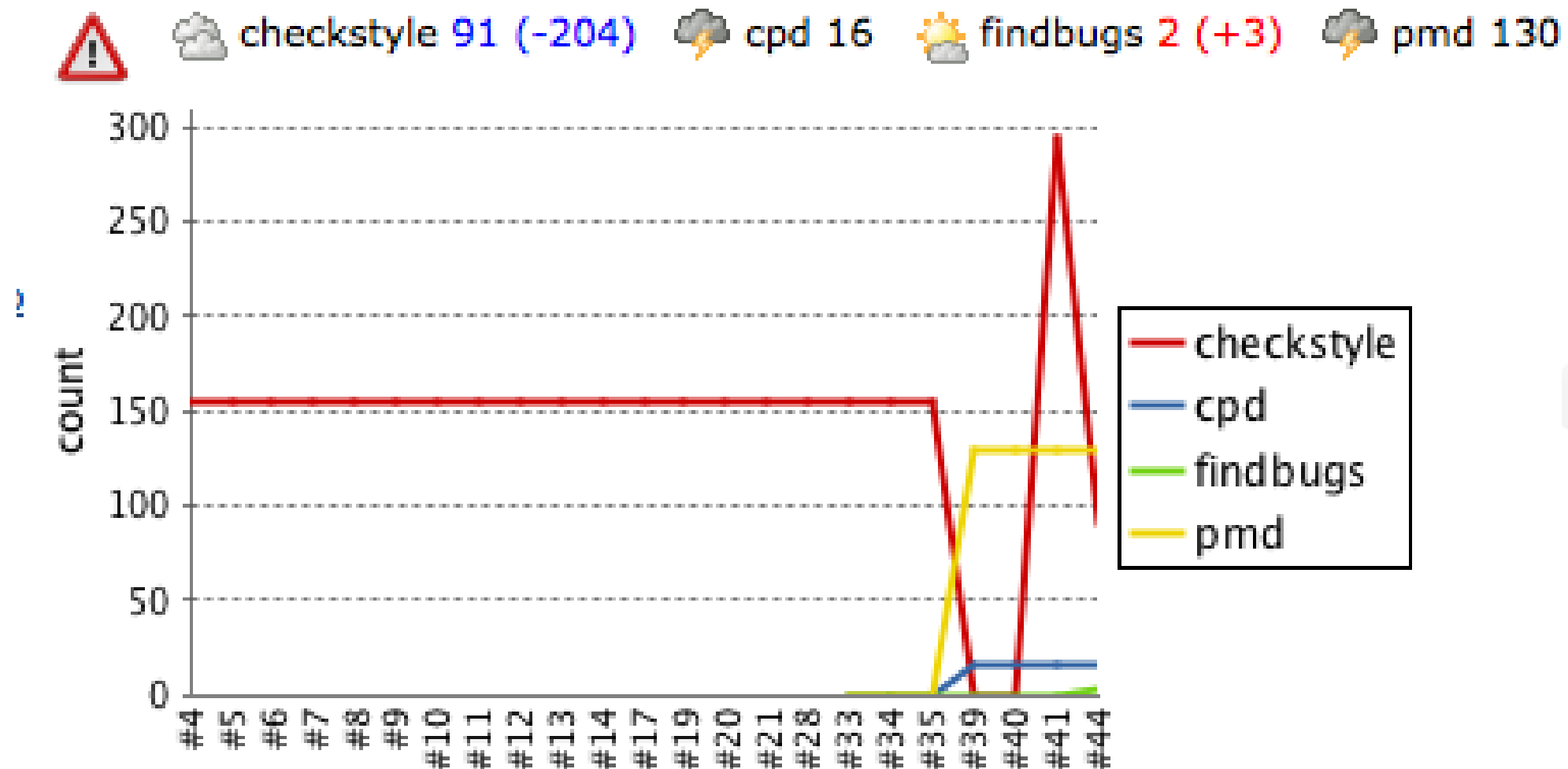
Icon: S M L

©Simplelearn. All rights reserved.

simplelearn

Working with Maven Build Jobs

By default, the violations plugin will display an aggregated view of the code quality metrics as shown below. Click on the violations graph to view the detailed reports for each module.



- For the efficient working of plugin, activate the violations plugin for each module in addition to the main project.

Exploring Maven Build Fields

- Click on the module you want to configure in the Modules screen, and click on the **Configure** menu.
- Activate the violations option and provide the thresholds if required.
- Click on the violations aggregate graph on the project build job home page to list the individual violations graphs for each module by Jenkins

Hudson search

Hudson » [game-of-life-code-quality](#) » [gameoflife-core](#)

[Back to Dashboard](#) [Status](#) [Changes](#) [Workspace](#) [Build Now](#) [Delete Module](#) [Configure](#) [Coverage Report](#) [Javadoc](#)

Build History (trend)

- #5 [Dec 7, 2010 11:33:41 AM](#) 151KB
- #4 [Dec 7, 2010 11:31:07 AM](#) 151KB
- #2 [Nov 28, 2010 5:45:55 PM](#) 91KB
- #1 [Nov 28, 2010 1:41:23 PM](#) 7KB

[for all](#) [for failures](#)

Description

☐ This build is parameterized

Github project

☐ Promote builds when...

Trac website

Build Triggers

☐ Build periodically

Build

Goals

Build Settings

☐ E-mail Notification

☐ Publish documents

☒ violations

XML filename pattern

checkstyle	10	999	999	auto
codenarc	10	999	999	auto
cpd	10	999	999	auto
findbugs	10	999	999	auto

Using Checkstyle, PMD, and FindBugs Reports

In addition to the violations plugin, Jenkins plugins also produce trend graphs and detailed reports for each tool. **Analysis Collector Plugin** is similar to violations plugin that displays the combined results in a graph.

Build Reports		
<input checked="" type="checkbox"/>	Analysis Collector Plugin This plug-in is an add-on for the plug-ins Checkstyle , Dry , FindBugs , PMD , Task Scanner , and Warnings : the plug-in collects the different analysis results and shows the results in a combined trend graph. Additionally, the plug-in provides health reporting and build stability based on these combined results.	1.8
<input checked="" type="checkbox"/>	Static Code Analysis Plug-ins This plug-in provides utilities for the static code analysis plug-ins.	1.14
<input type="checkbox"/>	CCCC Plugin This plugin generates the trend report for CCCC (C and C+\+ Code Counter).	0.3
<input type="checkbox"/>	CCM Plugin This plug-in generates reports on cyclomatic complexity for .NET code.	1.1
<input checked="" type="checkbox"/>	Checkstyle Plugin This plugin generates the trend report for Checkstyle , an open source static code analysis program.	3.10

- Once installed, set up the report in the project configuration by checking the **Publish Checkstyle analysis results** checkbox.
- In a freestyle build, specify a path to find the checkstyle XML reports.
- In a Maven 2 build, Jenkins will automatically analyze and find the location from the project configuration.

FindBugs Integration with Jenkins



Duration: 30 mins.

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

Steps to perform:

1. Login to Jenkins
2. Create a Maven job
3. Install FindBugs plugin
4. Add FindBugs dependencies in Maven project

ASSISTED PRACTICE

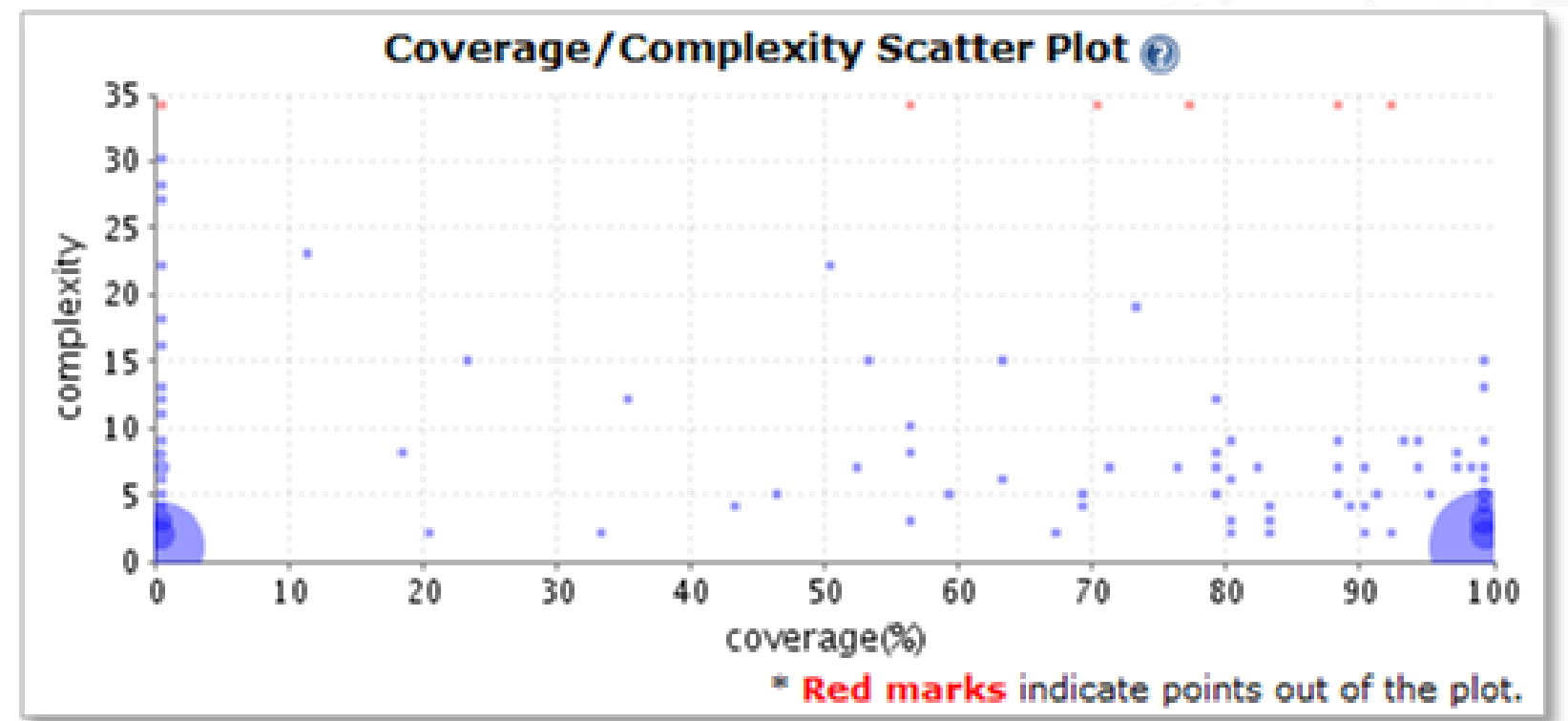
FULL STACK

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. Support for Cobertura and other tools is yet to be released.

Coverage/Complexity Scatter Plot with Jenkins



Duration: 30 mins.

Problem Statement: You are given a project to integrate **Coverage Complexity Scatter Plot plugin** with Jenkins.

Steps to perform:

1. Login to Jenkins
2. Install and set up Clover
3. Install **Coverage Complexity Scatter Plot** plugin

ASSISTED PRACTICE

FULL STACK

Reporting on Open Tasks

Reporting on Open Tasks

The Jenkins **Task Scanner** plugin keeps the track of different tags in the source code and represents a build with a bad weather icon on the dashboard if the number of open tasks are more.

- To set up, install both the **Static Analysis Utilities** plugin and the **Task Scanner** plugin.
- Activate the plugin in the project by checking the **Scan workspace for open tasks** checkbox in the **Build Settings** section of the project job configuration.
- Enter the tags for tracking with different priorities.
- By default, the plugin will scan all the Java source code in the project that can be limited by entering the files to the scan field.

☒ Scan workspace for open tasks

Files to scan

Files to exclude

Tasks tags

Run always

Health thresholds

Health priorities

[Fileset includes](#) setting that specifies the workspace files to scan for tasks, such as `**/*.java`. Basedir of the fileset is [the workspace root](#). If no value is set, then the default `**/*.java` is used.

[Fileset excludes](#) setting that specifies the workspace files to exclude scanning for tasks, such as library source files. Basedir of the fileset is [the workspace root](#).

High priority Normal priority Low priority Ignore case
 ☒

Configure the tags identifiers that should be looked for in the workspace files. For each priority a comma separated list of tags could be defined, e.g. TODO, FIXME, etc. Case of the the identifiers can be ignored, optionally.

☐

By default, this plug-in runs only for stable or unstable builds, but not for failed builds. If this plug-in should run even for failed builds then activate this check box.

 100%  0%

Configure the thresholds for the build health. If the actual number of warnings is between the provided thresholds, then the build health is interpolated.

☐ Only priority high ☐ Priorities high and normal ☒ All priorities

Determines which warning priorities should be considered when evaluating the build health.

CS

CS

CS

CS

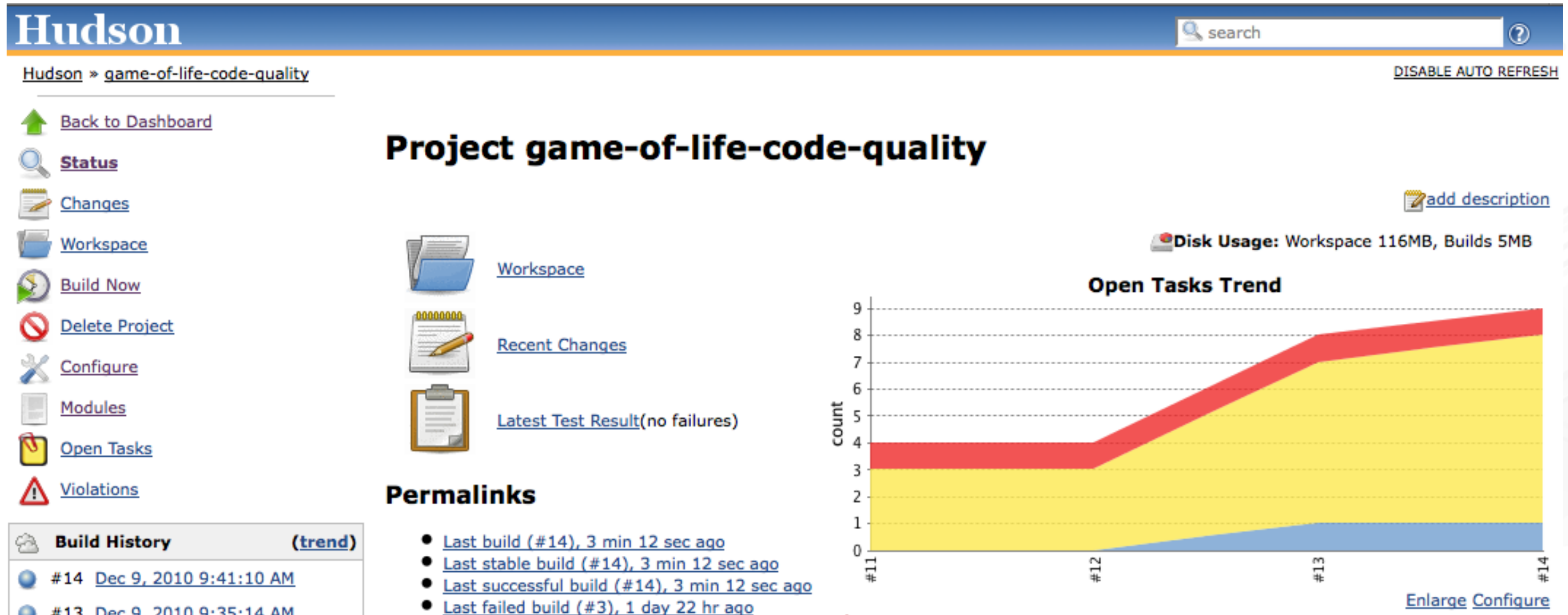
CS

CS

CS

Reporting on Open Tasks

The plugin generates a graph that shows tag trends by priority. Click on the **Open Tasks Report** to display the breakdown of tasks by Maven module, package or file, or open tasks. A sample graph is shown below:



FULL STACK

SonarQube

Introduction

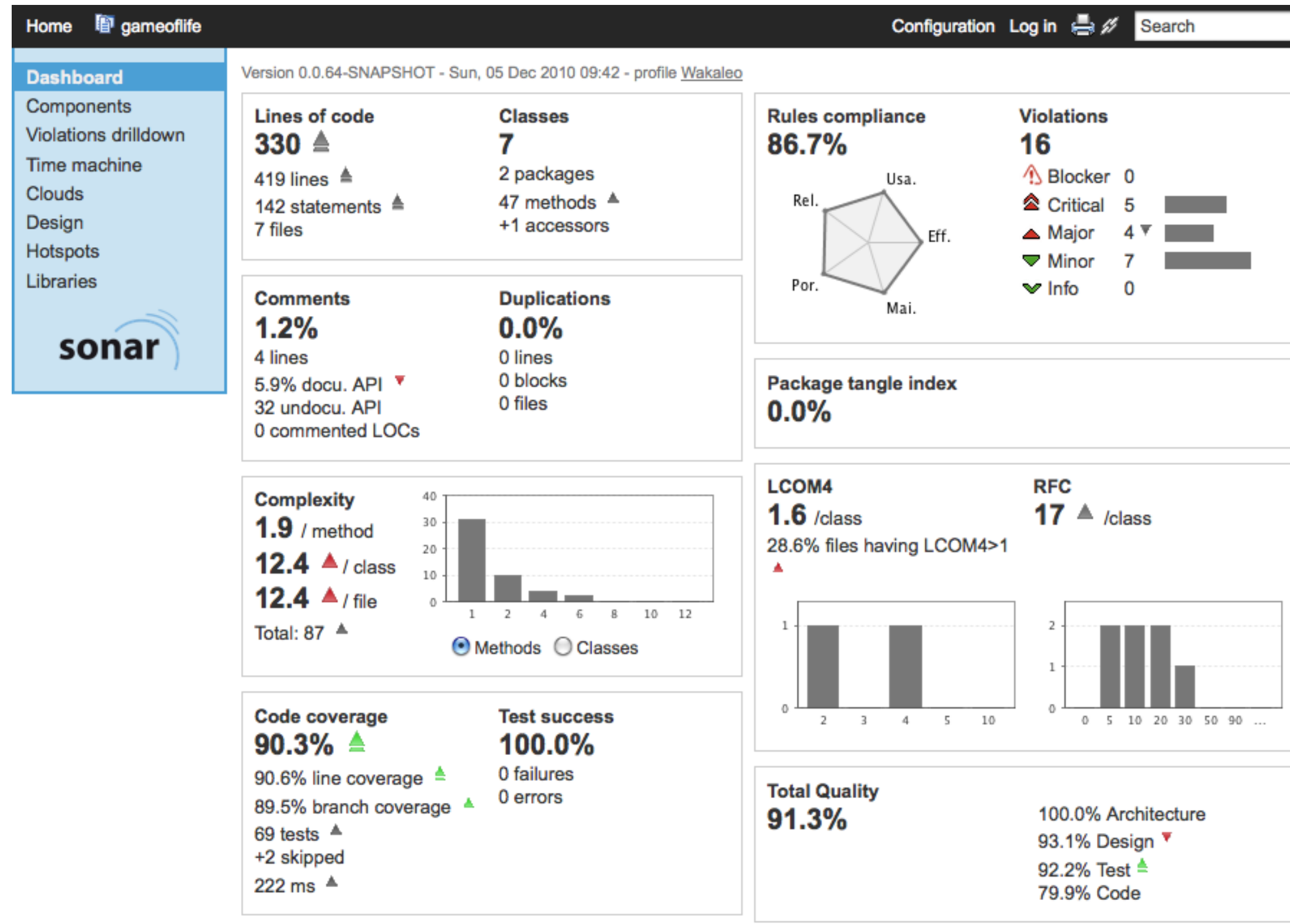
Sonar is a tool that centralizes a range of code quality metrics into a single website. It uses Maven plugins to analyze Maven projects and generate a set of code quality metrics reports.

- SonarQube generates reports on code coverage, rule compliance and documentation, complexity, maintainability, and technical debt.
- It can be extended 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

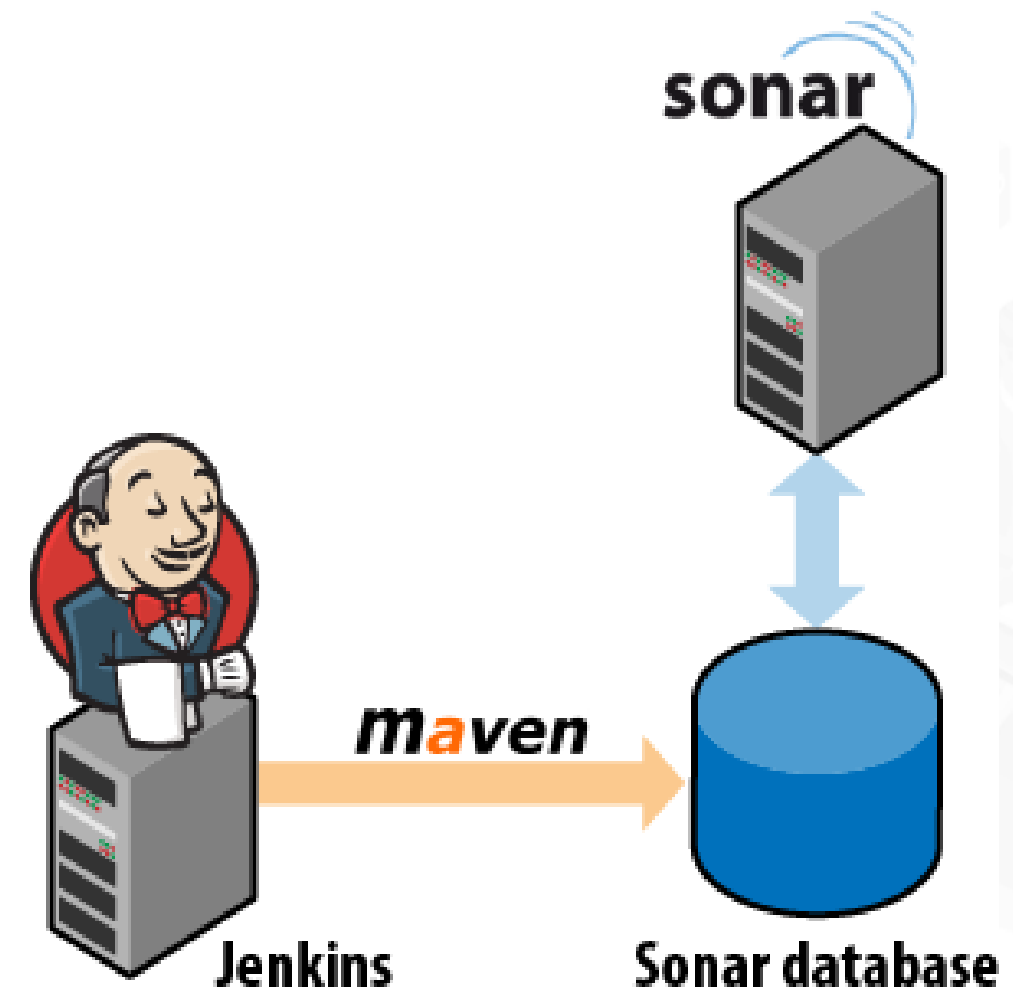
The screenshot of the Sonar server that runs separately along with analyzing and displaying results is given below:



SonarQube with Jenkins

The Jenkins **Sonar 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 installed from the plugin manager, configure the Jenkins Sonar plugin in the Configure System screen, under the Sonar 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.
- Configure the JDBC connection to the production Sonar database in Jenkins by clicking on the **Advanced** button in the configuration settings.



Sonar Configuration

Sonar

Sonar installations

Name	<input type="text" value="sonar-enterprise"/>
Disable	<input type="checkbox"/> <small>Check to quickly disable Sonar on all jobs.</small>
Server URL	<input type="text" value="http://www.acme.com/sonar"/> <small>Default is http://localhost:9000</small>
Server Public URL	<input type="text"/> <small>If not specified, then Server URL will be used</small>
Database URL	<input type="text" value="jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEr"/> <small>Do not set if default embedded database.</small>
Database login	<input type="text" value="sonar"/> <small>Default is sonar.</small>
Database password	<input type="text" value="secret"/> <small>Default is sonar.</small>
Database driver	<input type="text" value="com.mysql.jdbc.Driver"/> <small>Do not set if you use the default embedded database on localhost.</small>
Additional properties	<input type="text"/> <small>Additional properties to be passed to the mvn executable (example : -Dsome.property=some.value)</small>

Triggers

- ☐ Poll SCM
- ☒ Build periodically
- ☒ Manually started by user
- ☐ Build whenever a SNAPSHOT dependency is built
- ☐ Skip analysis on build failure

Delete Sonar

Add Sonar

List of Sonar installations



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, check the Sonar option in the **Post-Build Actions**.
- Sonar will run every time the build is started by one of the triggers.

☒ Sonar

Branch

MAVEN_OPTS

Additional properties

Optional sonar.branch property.

MAVEN_OPTS env var to provide, if not set the plugin will use the MAVEN_OPTS defined by the maven builder config.

Additional properties to be passed to the mvn executable (example: -Dsome.property=some.value).

☒ Dont use global triggers configuration

Triggers

☐ Poll SCM

☒ Build periodically

☒ Manually started by user

☐ Build whenever a SNAPSHOT dependency is built

☐ Skip analysis on build failure



SonarQube Integration with Jenkins



Duration: 30 mins.

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

Steps to Perform:

1. Install SonarQube on your local system.
2. Install and configure SonarQube plugin in Jenkins

ASSISTED PRACTICE

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 a better results in detection of duplicated code.
- Violation 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.



Static Code Analysis

Duration: 30 mins.

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

