How you could tune your code quality in the development process

Artem Sokovets

February 2024

Agenda

1. About me
2. Context
3. Three brothers
4. CheckStyle Customization
5. PMD Customization
6. Methods for Customizing SonarQube Rules
7. Our experience (EK Payments)
8. Conclusion
9. QA

About me

Artem Sokovets

Ex.Engineering Manager/Tech lead in Sber
Senior Backend Developer in Emirates (Payments)
14+ years in IT

Java/Kotlin/Dart
SpringBoot/Android/FlutterSDK

System design
Hackathons
Writer
Courses
Startup experience

linkedin.com/in/artem-sokovets-24a9845b/

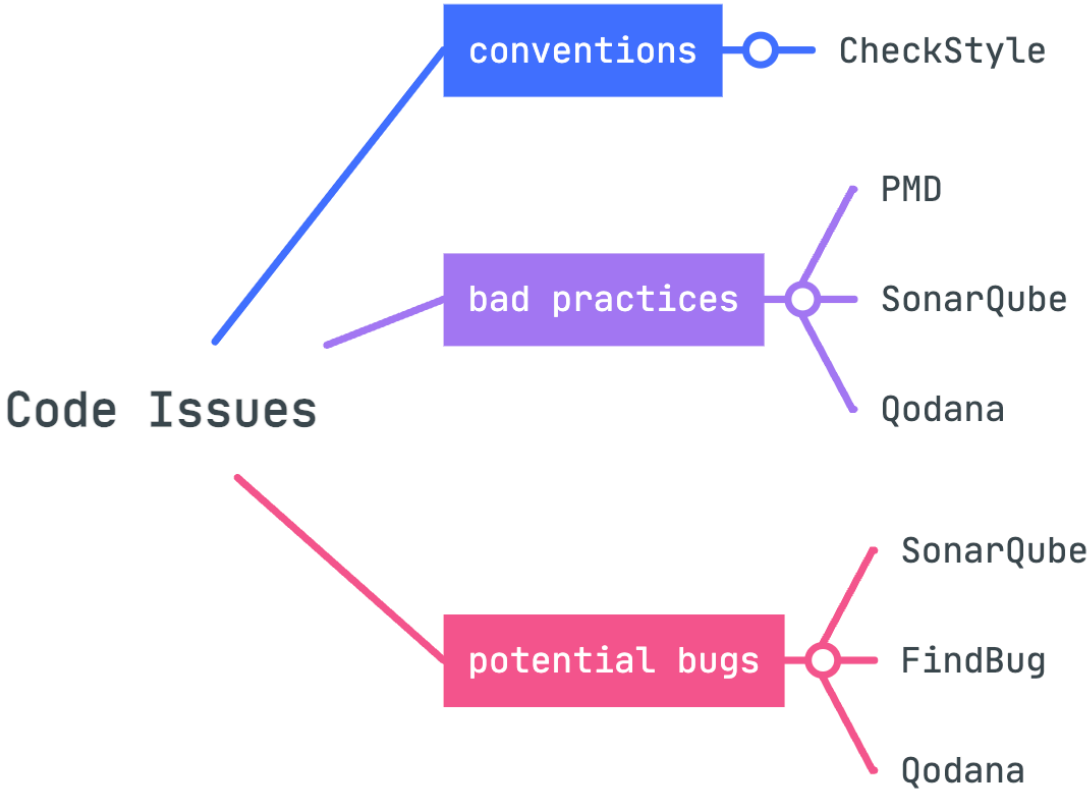
Context - No problem

It’s as easy as pie to manage your code

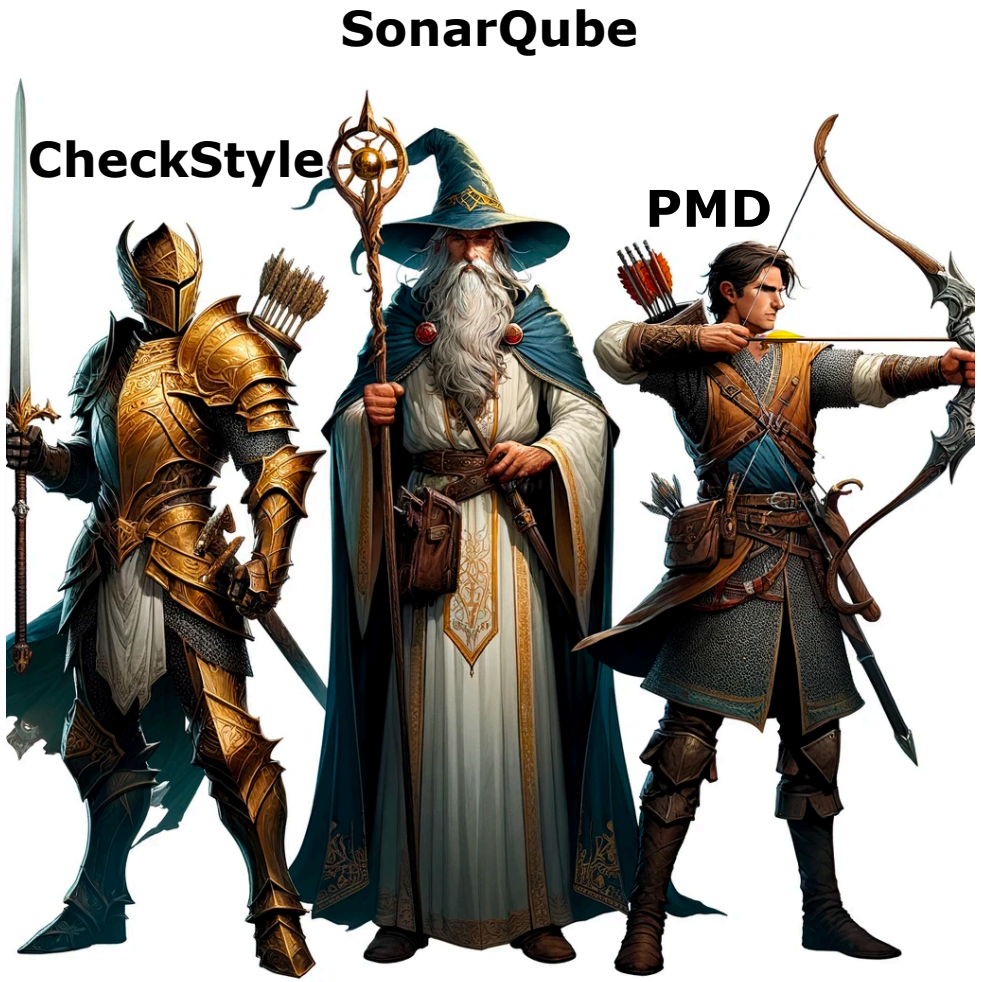
when you alone

Context - We have a problem

But it becomes harder when several teams are working on one project



How to solve these issues?



SonarQube

SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs and code smells.

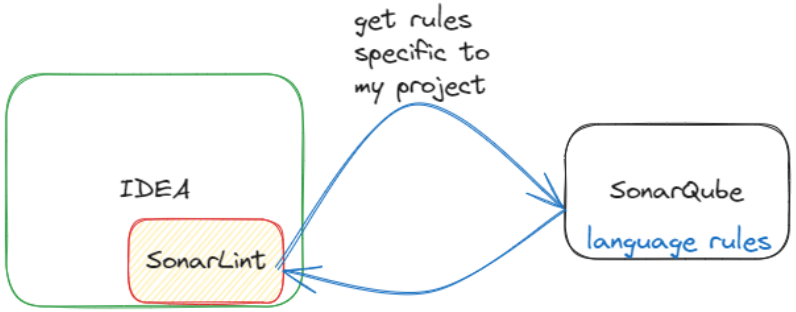
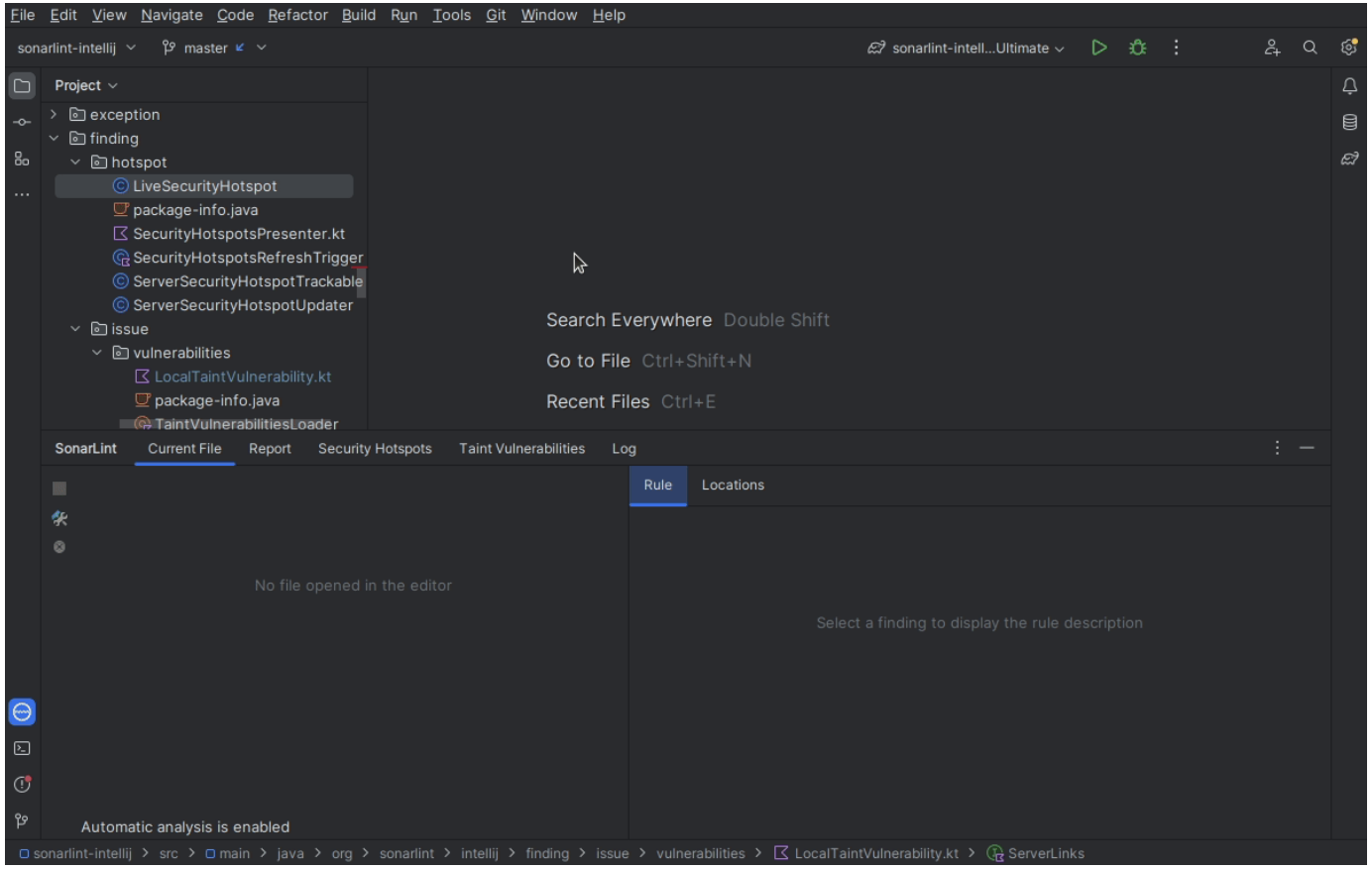
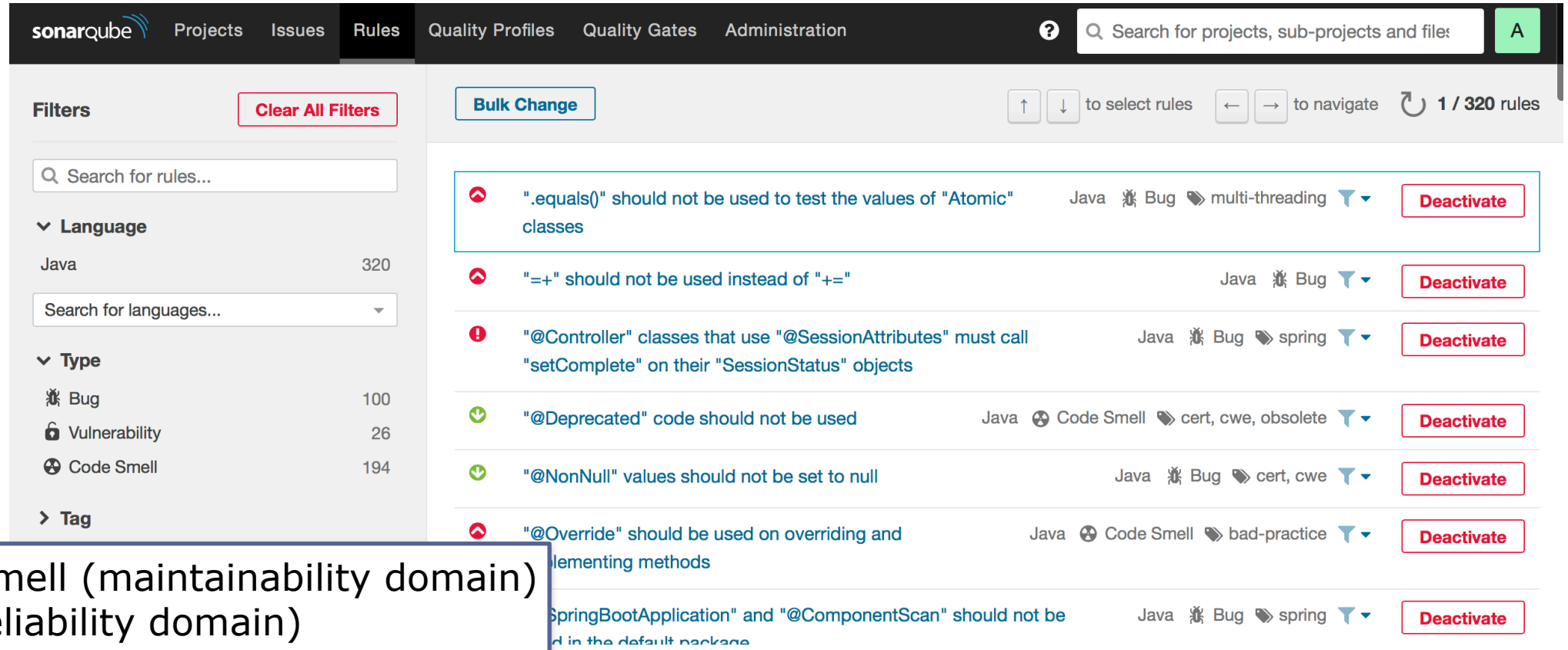


Figure 1: Getting latest language rules from server



SonarQube



* Code smell (maintainability domain)
* Bug (reliability domain)
* Vulnerability (security domain)
* Security hotspot (security domain)

Sonar Maven Plugin Setup

```
<plugin>
  <groupId>org.sonarsource.scanner.maven</groupId>
  <artifactId>sonar-maven-plugin</artifactId>
  <version>latest</version>
</plugin>
```

1

2

```
mvn clean verify sonar:sonar -Dsonar.projectKey=PROJECT_KEY
                             -Dsonar.projectName='PROJECT_NAME'
                             -Dsonar.host.url=http://localhost:9000
                             -Dsonar.token=THE_GENERATED_TOKEN
```

baeldung.com/sonar-qube

PMD

PMD is a source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It supports Java, JavaScript, Salesforce.com Apex and Visualforce, PLSQL, Apache Velocity, XML, XSL.

baeldung.com/pmd

PMD Setup

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-pmd-plugin</artifactId>
  <version>3.14.0</version>
  <executions>
    <execution>
      <goals>
        <goal>check</goal> <!-- Run in build phase -->
      </goals>
    </execution>
  </executions>
</plugin>
```

1

2

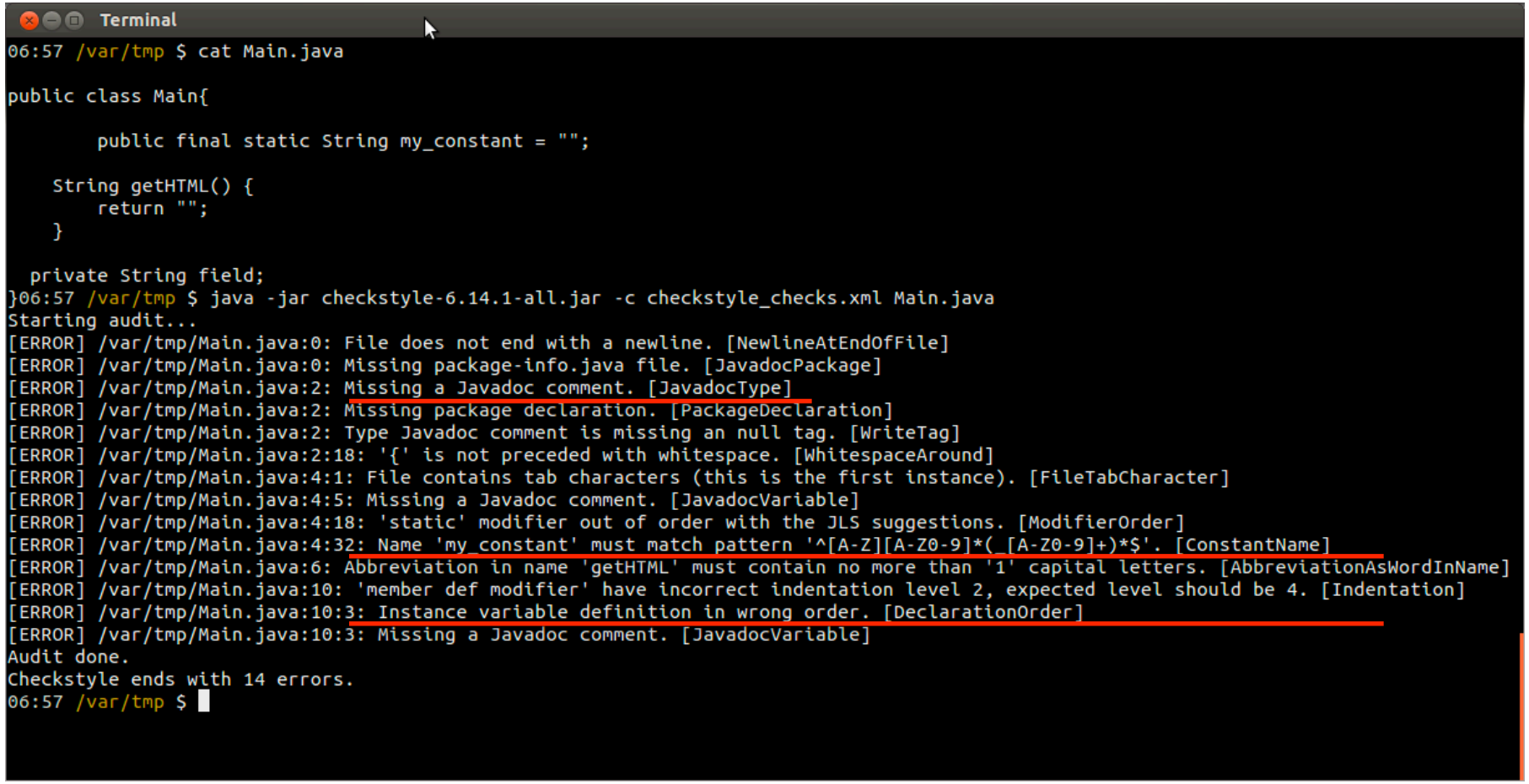
mvn pmd:check

3

```
[INFO] --- pmd:3.19.0:check (default) @ pricing-rewardconversion-service ---
[INFO] PMD version: 6.50.0
[INFO] PMD Failure: com.emirates.ocsl.pricing.*.domain.conversion.MilesDistributionConverter:49
Rule:ExcessiveMethodLength Priority:3 Avoid really long methods..
```

CheckStyle

Checkstyle is a static code analysis tool used in software development for checking if Java source code is compliant with specified coding rules.



CheckStyle Setup

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <version>3.2.0</version>
  <configuration>
    <configLocation>src/main/resources/checkstyle.xml</configLocation>
  </configuration>
  <executions>
    <execution>
      <id>validate</id>
      <phase>validate</phase>
      <goals>
        <goal>check</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

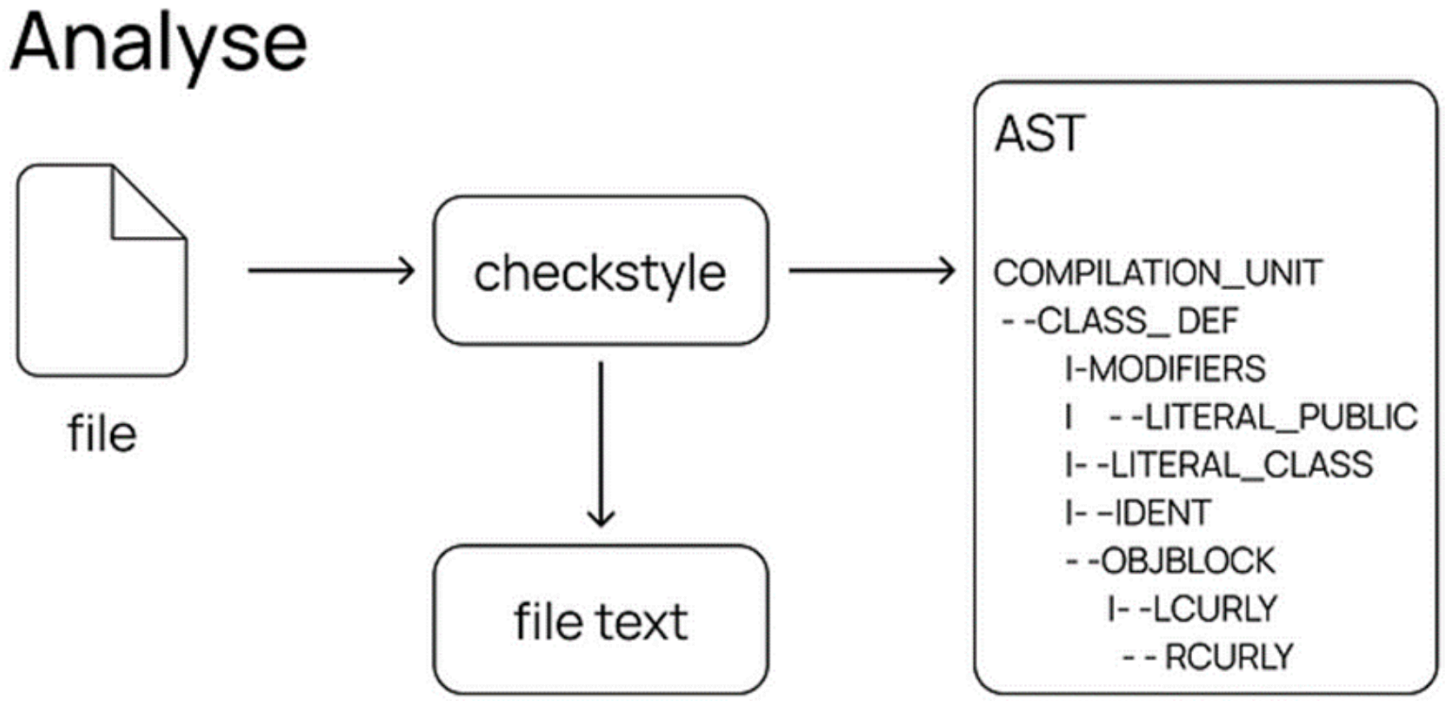
1

2 **mvn checkstyle:check**

3

File.java:10: Missing a Javadoc comment. [JavadocType]
File.java:14: Line is longer than 80 characters (found 85). [LineLength]

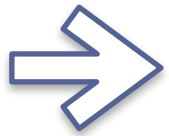
How does CheckStyle work?



Abstract Syntax Tree (AST)

```
public class Main {

  public static void main(String[] args)
    System.out.println("Hello world!");
  }
}
```



```
CLASS_DEF -> CLASS_DEF [1:0]
|--MODIFIERS -> MODIFIERS [1:0]
|   `--LITERAL_PUBLIC -> public [1:0]                    1
|--LITERAL_CLASS -> class [1:7]                          2
|--IDENT -> Main [1:13]                                  3
`--OBJBLOCK -> OBJBLOCK [1:17]
    |--LCURLY -> { [1:17]
    |--METHOD_DEF -> METHOD_DEF [2:4]
    |   |--MODIFIERS -> MODIFIERS [2:4]
    |   |   |--LITERAL_PUBLIC -> public [2:4]
    |   |   `--LITERAL_STATIC -> static [2:11]
    |   |--TYPE -> TYPE [2:18]
    |   |   `--LITERAL_VOID -> void [2:18]
    |   |--IDENT -> main [2:23]
    |   |--LPAREN -> ( [2:27]
    |   |--PARAMETERS -> PARAMETERS [2:34]
    |   |   `--PARAMETER_DEF -> PARAMETER_DEF [2:34]
    |   |       |--MODIFIERS -> MODIFIERS [2:34]
    |   |       |--TYPE -> TYPE [2:34]
    |   |       |   `--ARRAY_DECLARATOR -> [ [2:34]
    |   |       |       |--IDENT -> String [2:28]
    |   |       |       `--RBRACK -> ] [2:35]
    |   |       `--IDENT -> args [2:37]
    |   |--RPAREN -> ) [2:41]
    |   `--SLIST -> { [2:43]
```

CheckStyle Custom Behavior

RegExp

XPath

Pure Java

RegExp

widely used

>> analyse scripts, configuration files

Examples: check lines, whitespaces and etc

Example (RegExp)

Example #1: You need to check the config files where **each maven command should contain two flags**: **-e** and **-no-transfer-progress**

```
<module name="RegexpSingleline">
  <property name="format" value="^[^#]*mvn (?!(-e –no-transfer-progress))"/>
  <property name="fileExtensions" value="sh, yml"/>
</module>
```

Example (RegExp)

Example #2: You need to check that **each test class should contain two test methods with @Test**

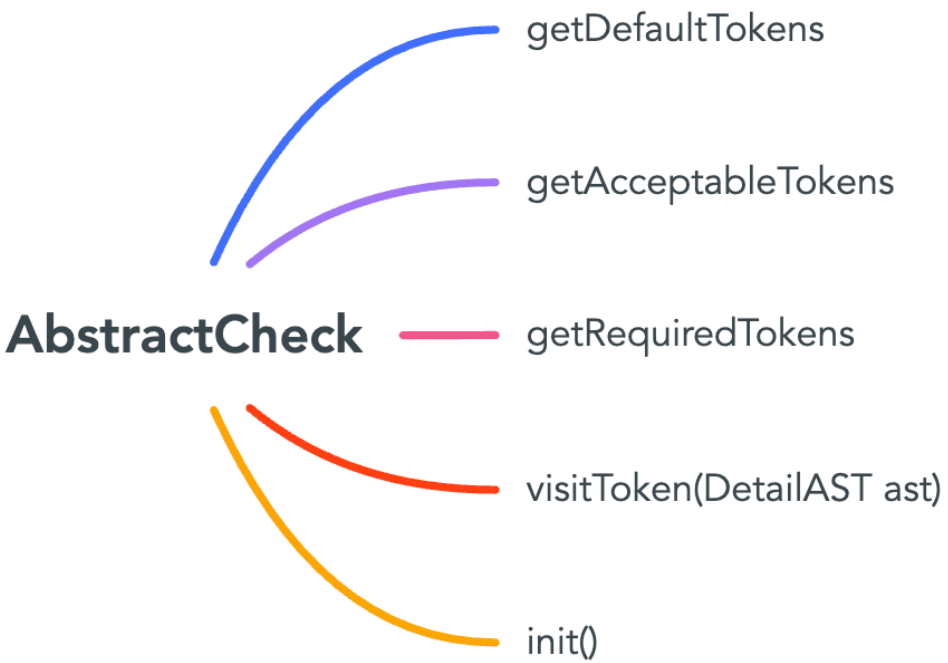
```
<module name="RegexpSingleline">
  <property name="format" value="@Test"/>
  <property name="minimum" value="2"/>
  <property name="fileExtensions" value="java"/>
</module>
```

Example (XPath)

Example #1: You are **not allowed to use @Issue** in your project.

```
<module name="MatchXpath">
  <property name="query" value="//ANNOTATION[./IDENT[@text='Issue']]"/>
</module>
```

Pure Java



Example (Pure Java)

```
private DetailAST findLastChildWhichHasType(DetailAST ast, int type) {
  DetailAST child = ast.getLastChild();
  while (child != null && child.getType() != type) {
    child = child.getPreviousSibling();
  }
  return child;
  //empty line
  //empty line
}
```

Avoid leaving empty lines at the end of methods

Example (Pure Java)

```
public void visitToken(DetailAST ast) {
  DetailAST openingBrace = ast.findFirstToken(TokenTypes.SLIST);
  DetailAST closingBrace =
      (openingBrace != null) ? findLastChildWhichHasType(openingBrace, TokenTypes.RCURLY)
    : null;

  if (openingBrace != null && closingBrace != null) {
    int open = openingBrace.getLineNo();
    int end = closingBrace.getLineNo();
    checkEmptyLinesAfter(open, end);
    checkEmptyLinesBefore(end, open);
  }
}
```

What should you also know?

Ignore violations:

1

```
<module name="SuppressionFilter">
    <property name="file" value="checkstyle-suppressions.xml"/>
</module>
```

CheckStyle GUI:

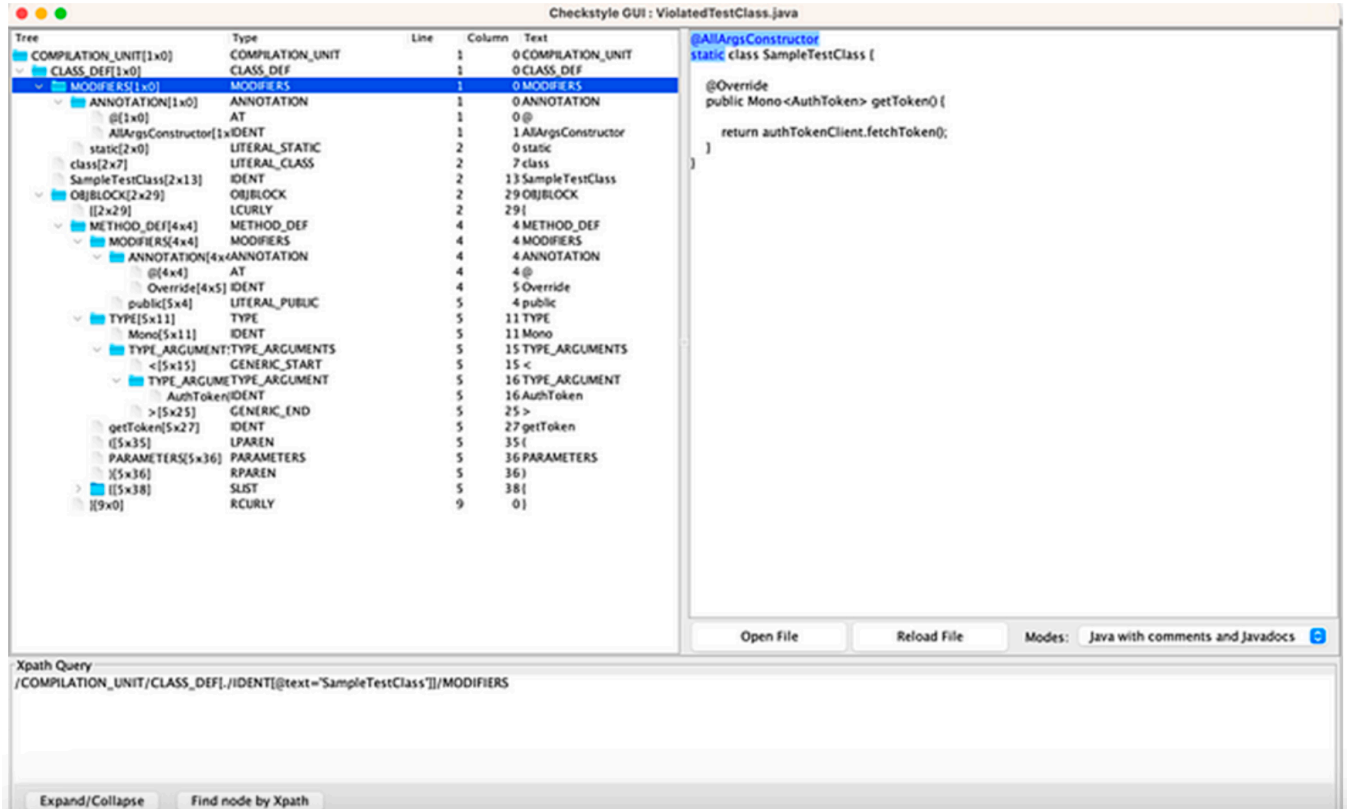
2

```
java -cp checkstyle-10.12.2-all.jar
com.puppycrawl.tools.checkstyle.gui.Main
```

Additional custom rules:

3

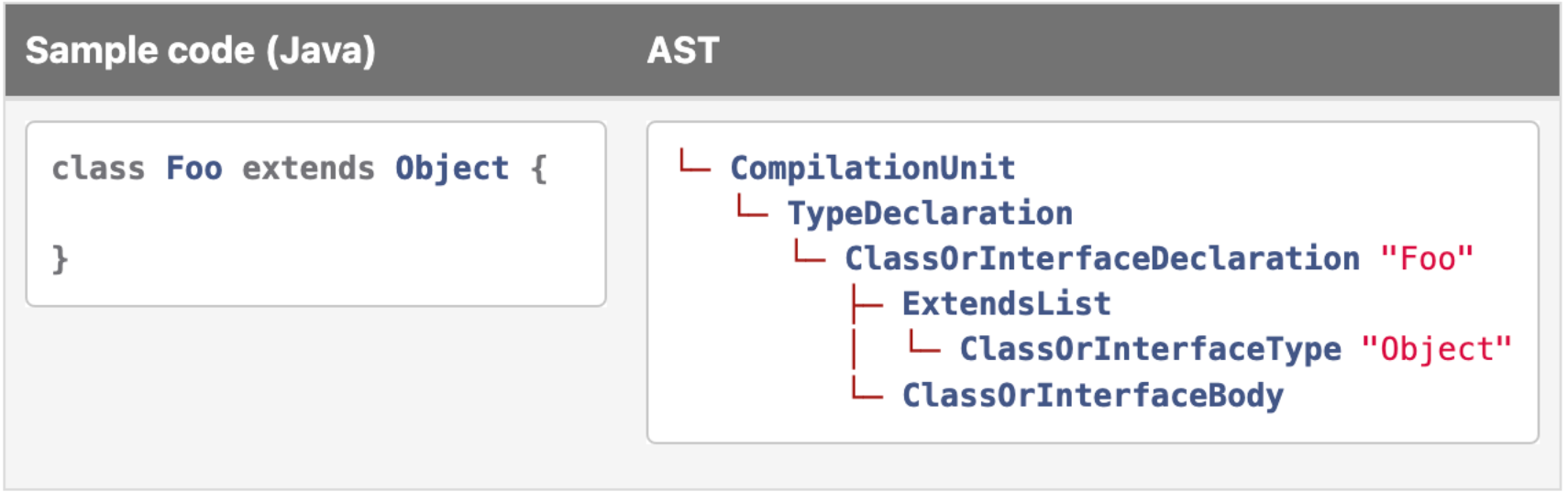
github.com/sevntu-checkstyle/sevntu.checkstyle
github.com/checkstyle-addons/checkstyle-addons



PMD Customization

Before running rules, PMD parses the source file into a data structure called an **abstract syntax tree (AST)**.
This tree represents the syntactic structure of the code, and encodes syntactic relations between source code elements.

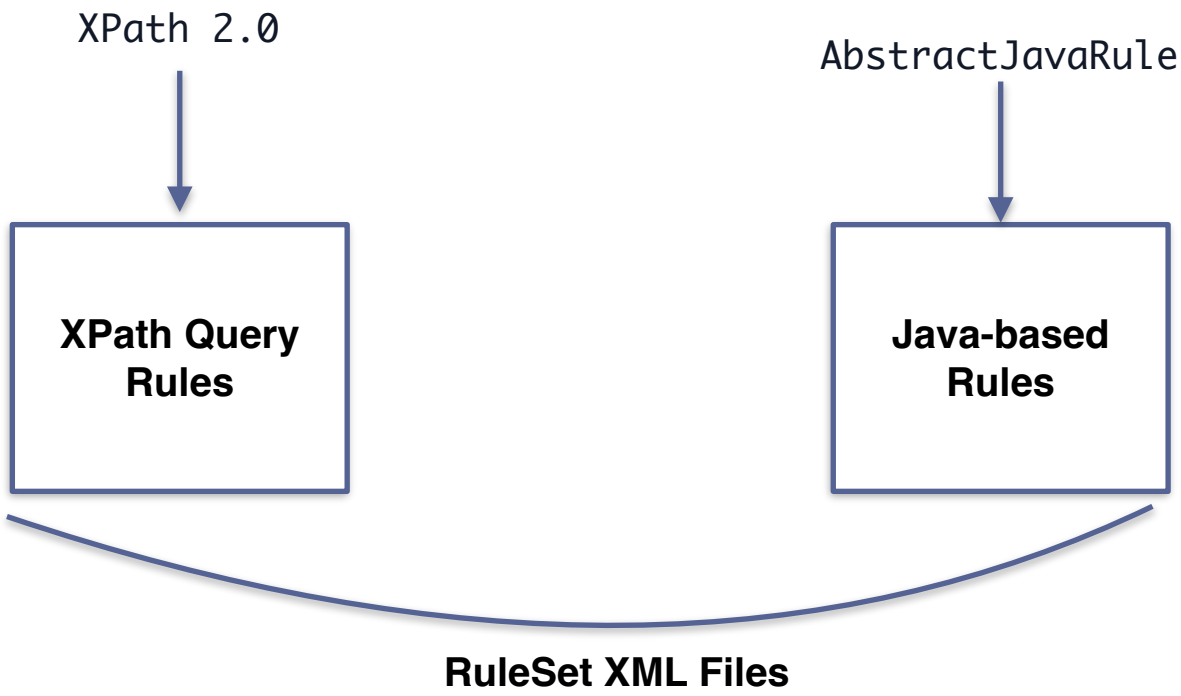
For instance, in Java, method declarations belong to a class: in the AST, the nodes representing method declarations will be descendants of a node representing the declaration of their enclosing class.



PMD Customization

PMD Designer

PMD Custom Behavior



docs.pmd-code.org/latest/pmd_userdocs_extending_testing.html

Xpath Custom Rules

1
```
Mono
    .just("dear")
    .switchIfEmpty(Mono.just(getRandomString())); // non-compliant
```

2
```
Mono
    .just("dear")
    .switchIfEmpty(Mono.error(getException())); //non-compliant
```

3
```
Mono
    .just("dear")
    .switchIfEmpty(Mono.defer(() -> Mono.just(getRandomString()))); // compliant
```

4
```
Mono
    .just("dear")
    .switchIfEmpty(Mono.error(() -> getException())); //compliant
```

Xpath Custom Rules

```
<ruleset name="Custom Rules for Emirates Code Base"
         xmlns="http://pmd.sourceforge.net/ruleset/2.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://pmd.sourceforge.net/ruleset/2.0.0 https://pmd.sourceforge.io/ruleset_2_0_0.xsd">

    <description>
        Rules which enforce generally accepted best practices.
    </description>

    <rule name="SwitchIfEmptyIncorrectUsage"
          language="java"
          message="Avoid direct call of Mono.switchIfEmpty with non-constant parameter and use Mono.defer instead"
          class="net.sourceforge.pmd.lang.rule.XPathRule">
        <description>
            By design Mono.switchIfEmpty is being evaluated eagerly and it's always trigger alternative no matter what during stream
            construction. To address this you can defer evaluation of a second mono by using Mono.defer.
        </description>
        <priority>1</priority>
        <properties>
            <property name="xpath">
                <value>
                    <![CDATA[
//PrimarySuffix[@Image = "switchIfEmpty"]
[
(following-sibling::PrimarySuffix/
    Arguments/
    ArgumentList/
    Expression/
    PrimaryExpression/
    PrimaryPrefix/
    Name/@Image != "Mono.defer"
and
following-sibling::PrimarySuffix/
    Arguments/
    ArgumentList/
    Expression/
    PrimaryExpression/
```

Pay attention

Pure Java Custom Rules

Test must have TestId annotation

```
@ParameterizedTest
@ValueSource(strings = {"apple", "banana"})
@TestCaseId(2)
void testValueSource(String fruit) {
    assertNotEquals("pineapple", fruit);
}
```

Solution

```
public class TestMustHaveTestIdAnnotationRule extends AbstractJavaRule {

    @Override
    public Object visit(final ASTMethodDeclaration node, final Object data) {
        ASTAnnotation testAnnotation = node.getAnnotation(Test.class.getCanonicalName());
        ASTAnnotation parameterizedTestAnnotation =
node.getAnnotation(ParameterizedTest.class.getCanonicalName());

        if (testAnnotation != null || parameterizedTestAnnotation != null) {
            ASTAnnotation testCaseIdAnnotation =
node.getAnnotation(TestCaseId.class.getCanonicalName());
            if (testCaseIdAnnotation == null) {
                addViolation(data, node);
            }
        }
        return super.visit(node, data);
    }
}
```

1

2

3

4

How to create a Custom SonarQube Rule?

SonarQube UI

Pure JAVA

https://dilumpathi.medium.com/adding-sonarqube-coding-rules-using-java-for-the-test-automation-suite-c4083405278a

https://github.com/SonarSource/sonar-java/blob/master/docs/CUSTOM_RULES_101.md

Pure Java

1

```
<dependency>
    <groupId>org.sonarsource.sonarqube</groupId>
    <artifactId>sonar-plugin-api</artifactId>
    <version>${sonarqube.version}</version>
    <scope>provided</scope>
</dependency>
```

2

`MyJavaRulesPlugin implements Plugin`	*Entry point of your plugin containing your custom rules*
`MyJavaRulesDefinition implements RulesDefinition`	*Declare rule metadata in server repository of rules. That allows to list the rules in the page "Rules".*
`MyJavaFileCheckRegistrar implements CheckRegistrar`	*Provide the checks (implementations of rules) classes that are going be executed during source code analysis.*
`@Rule(key = "MyFirstCustomRule") public class MyFistCustomCheck extends IssuableSubscriptionVisitor`	*Overide nodesToVisit and visitNode*

3

Put your jar to Sonar instance

`extensions/plugins`

EK Payments

1. We have a bom file

2. We have separate dependency specific to PMD, CheckStyle rules

3. We use bom in each pom services

   If needed we override parameters for each plugin

4. We wouldn’t allow to build PR with violations

EK Payments Customization

PMD	CheckStyle	SonarQube
Basic Rules Set	Basic Rules Set	Basic Rules for each language
+ Custom Rules	+ Custom Rules (JavaDoc for new code in PRs)	

Conclusion

1. It was an brief overview of PMD, CheckStyle, SonarQube

2. Customization PMD (Xpath, PureJava - AbstractJavaRule)

3. We have looked at how we could customize CheckStyle (Xpath, RegExp, AbstractCheck)

4. We have looked at how we could customize SonarQube (UI, plugin system)

5. We have looked at how we adopted these tools to URP projects in EK

6*. Live Demonstration (time permitting)

QA

1. https://www.sonarsource.com/blog/what-makes-checkstyle-pmd-findbugs-and-macker-complementary/
2. https://www.linkedin.com/pulse/customize-checkstyle-rules-your-development-process-artem-sokovets-kxuyf/
3. https://medium.com/@rahul.gite11/checkstyle-all-you-need-to-know-5b6e7a37072a
4. https://medium.com/@norsag/custom-pmd-rules-488cbd332ad9
5. https://github.com/NorsaG/evla-custom-pmd/blob/master/src/main/resources/rules.xml
6. https://datamify.medium.com/java-pmd-how-to-check-java-source-code-with-pmd-2df2783e9f30
7. https://github.com/pmd/pmd-examples/tree/main/maven/simple-project
8. https://docs.pmd-code.org/pmd-doc-6.55.0/pmd_userdocs_extending_designer_reference.html
9. https://dilumpathi.medium.com/adding-sonarqube-coding-rules-using-java-for-the-test-automation-suite-c4083405278a
10. https://github.com/SonarSource/sonar-java/blob/master/docs/CUSTOM_RULES_101.md
11. https://medium.com/@artsokovets/customize-checkstyle-rules-for-your-development-process-bbf7b60c05a5

Live Demo*

github.com/NorsaG/evla-custom-pmd

github.com/hadiinz/sonar-custom-rule-training

github.com/artsok/custom-checkstyle-rules

- MissingJavaDocMethodUrpCheck
- IntermediateEmptyLinesCheck
- EmptyLineAtTheEndChecker
- AnnotationsWithoutEmptyLinesChecker