Static Analyzers

Static analysis tools are essential in software development, focusing on improving code quality, identifying bugs, and ensuring adherence to coding standards without executing the code. They serve multiple purposes:

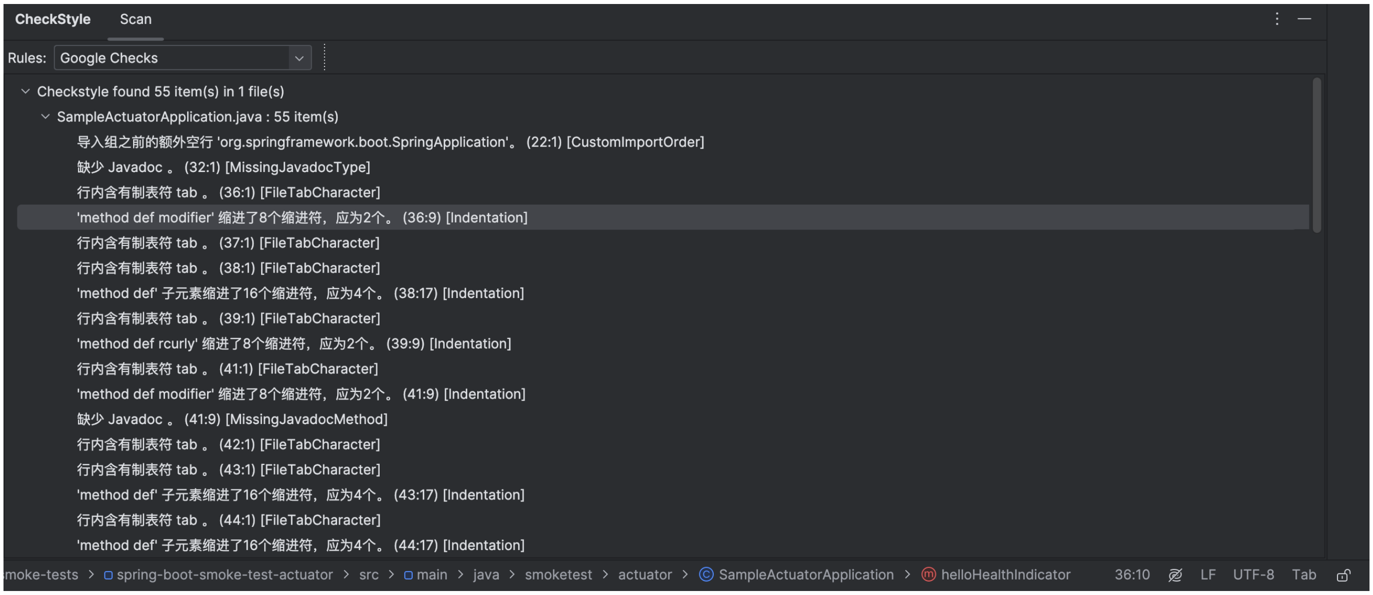
1. Code Quality: Enhance code readability, maintainability, and efficiency.
2. Bug Detection: Identify and fix errors early in the development process.
3. Security: Detect potential security vulnerabilities.
4. Performance: Highlight inefficient code patterns for optimization. These tools are used throughout the development lifecycle, particularly in:
5. Development: Integrated into IDEs, they provide real-time feedback to developers.
6. Code Review: Automate the detection of potential issues before peer review.
7. Continuous Integration: Included in CI pipelines to enforce code quality standards.
8. Compliance: Ensure code adheres to industry-specific standards.

> Tools like Checkstyle, PMD, and SonarQube are popular choices, each offering unique features to aid developers in creating robust, efficient, and secure Java applications.

Spring boot: CheckStyle and PMD

Both CheckStyle and PMD are easy to install in IDEA plugin. You can also integrate it inside the dependency through Gradle, I try to add the Spotbugs into Gradle at first, but it does not function and cause horrible issue. Then I give up and turn to CheckStyle and PMD.

Due to the limitation of the plugin, We use `SampleActuatorApplication.java` in actuator smoke as a single control group for testing. Here is the result:

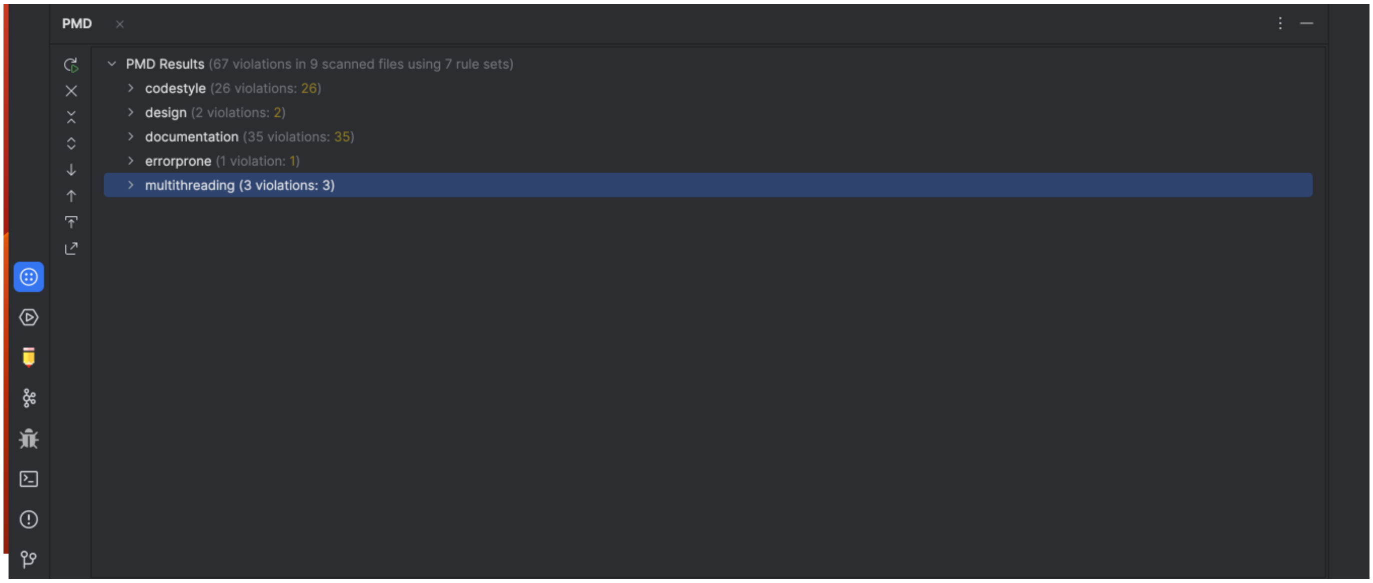


```
Checkstyle found 55 item(s) in 1 file(s)
  SampleActuatorApplication.java : 55 item(s)
    导入组之前的额外空行 'org.springframework.boot.SpringApplication'。 (22:1) [CustomImportOrder]
    缺少 Javadoc 。 (32:1) [MissingJavadocType]
    行内含有制表符 tab 。 (36:1) [FileTabCharacter]
    'method def modifier' 缩进了8个缩进符，应为2个。 (36:9) [Indentation]
    行内含有制表符 tab 。 (37:1) [FileTabCharacter]
    行内含有制表符 tab 。 (38:1) [FileTabCharacter]
    'method def' 子元素缩进了16个缩进符，应为4个。 (38:17) [Indentation]
    行内含有制表符 tab 。 (39:1) [FileTabCharacter]
    'method def rcurly' 缩进了8个缩进符，应为2个。 (39:9) [Indentation]
    行内含有制表符 tab 。 (41:1) [FileTabCharacter]
    'method def modifier' 缩进了8个缩进符，应为2个。 (41:9) [Indentation]
    缺少 Javadoc 。 (41:9) [MissingJavadocMethod]
    行内含有制表符 tab 。 (42:1) [FileTabCharacter]
    行内含有制表符 tab 。 (43:1) [FileTabCharacter]
    'method def' 子元素缩进了16个缩进符，应为4个。 (43:17) [Indentation]
    行内含有制表符 tab 。 (44:1) [FileTabCharacter]
    'method def' 子元素缩进了16个缩进符，应为4个。 (44:17) [Indentation]
```

moke-tests > spring-boot-smoke-test-actuator > src > main > java > smoketest > actuator > SampleActuatorApplication > helloHealthIndicator
36:10 LF UTF-8 Tab

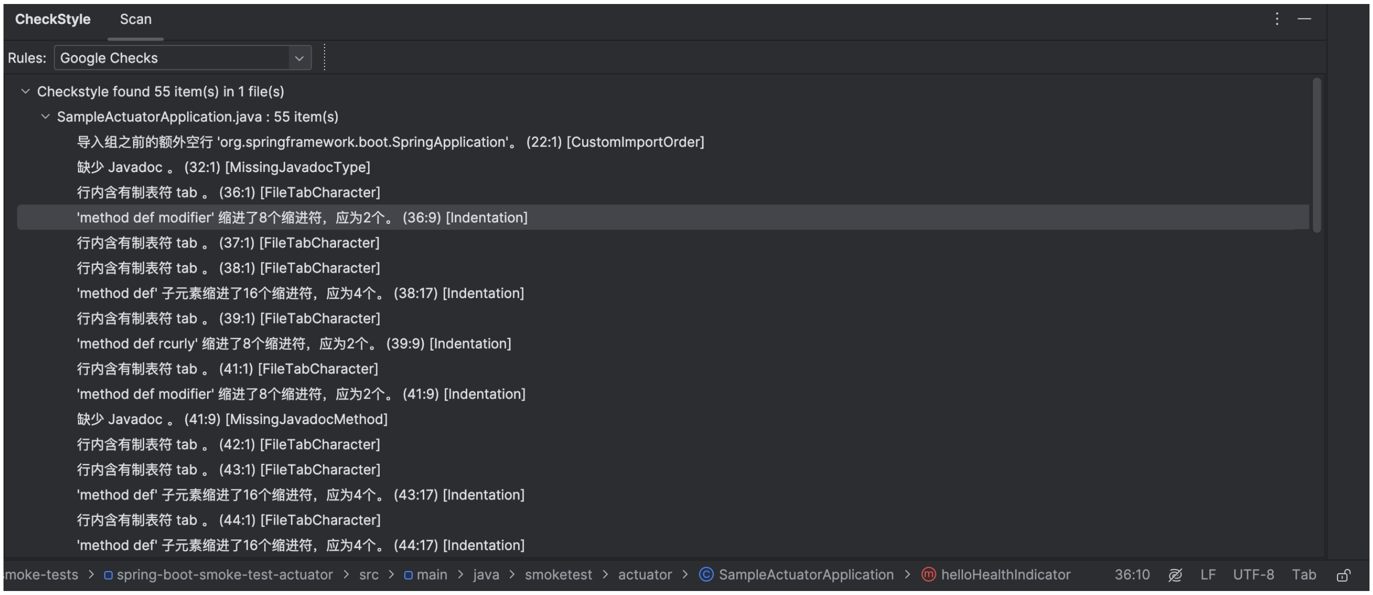
I don't know why it automatically generates the chinese output, it did convent for me but also make more work for me. You can see in the rules section on the left right corner we use `google checks` , we have two default configurations, one is call `Sun checks` , I choose google checks since sun checks generate a more horrible result than Google, but a Google result is not very good either. I'll talk about it later.

Let's take a look at how PMD performance.



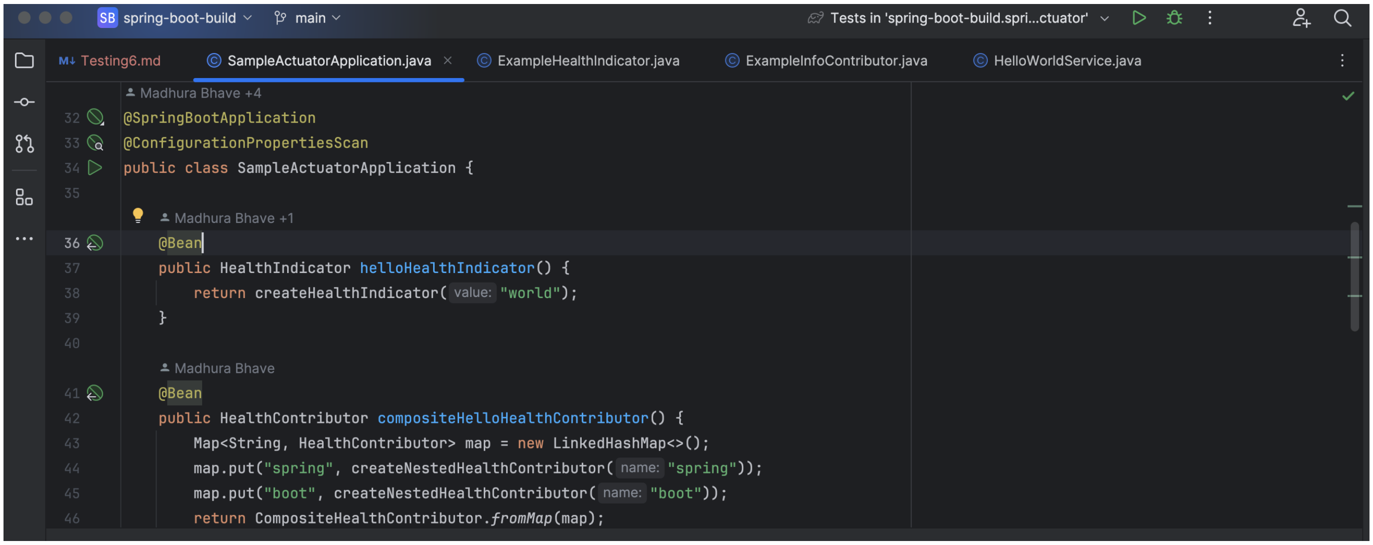
```
PMD Results (67 violations in 9 scanned files using 7 rule sets)
  codestyle (26 violations: 26)
  design (2 violations: 2)
  documentation (35 violations: 35)
  errorprone (1 violation: 1)
  multithreading (3 violations: 3)
```

PMD plugin supports multi file scan, and I ask it to check some test system. And only 10 files generate 67 violations. Its shocking result, but let's see what really happen.


CheckStyle
Scan
Rules: Google Checks
moke-tests > spring-boot-smoke-test-actuator > src > main > java > smoketest > actuator > SampleActuatorApplication > helloHealthIndicator
36:10
LF
UTF-8
Tab

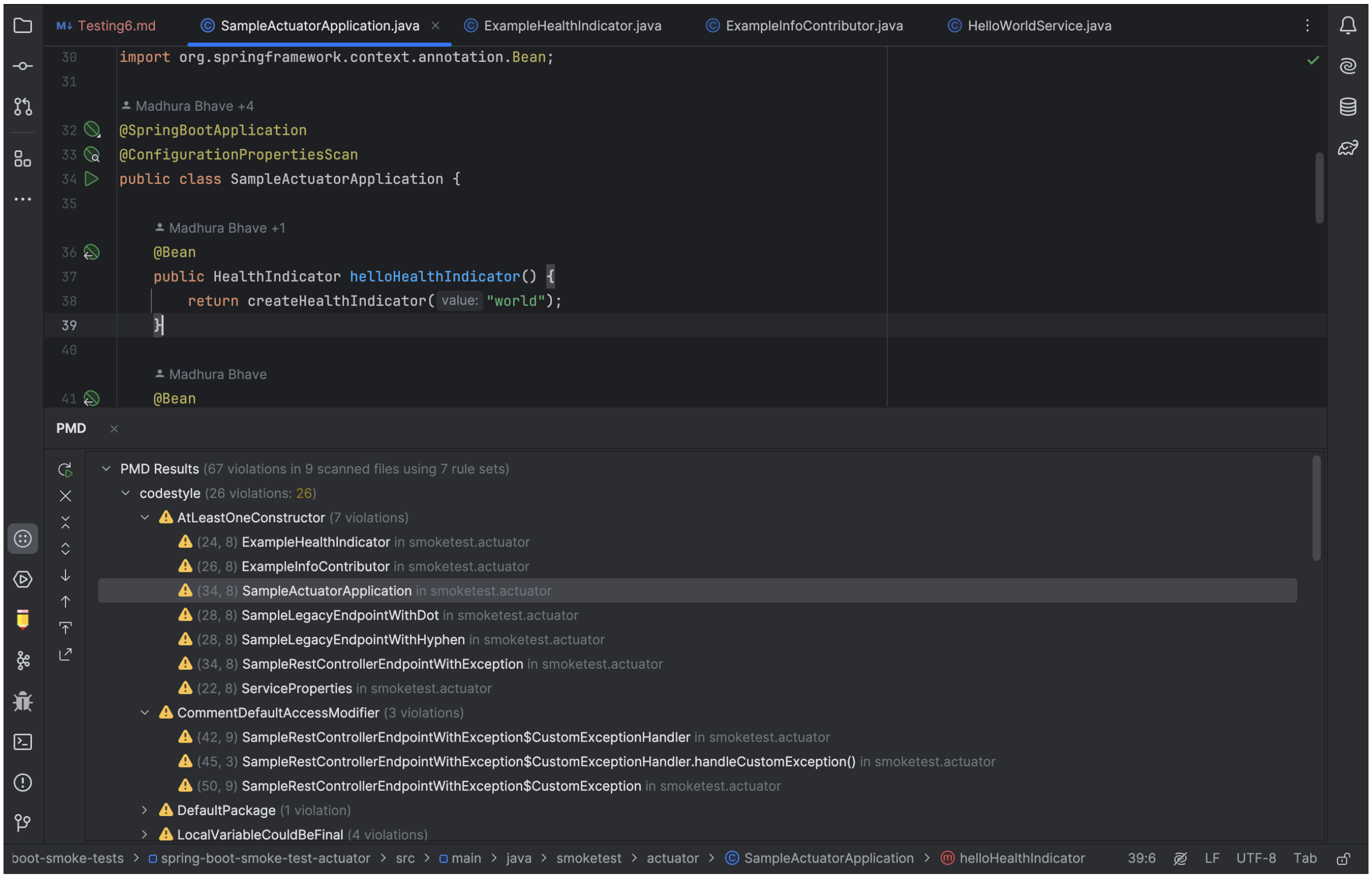
```
Checkstyle found 55 item(s) in 1 file(s)
  SampleActuatorApplication.java : 55 item(s)
    导入组之前的额外空行 'org.springframework.boot.SpringApplication'。 (22:1) [CustomImportOrder]
    缺少 Javadoc 。 (32:1) [MissingJavadocType]
    行内含有制表符 tab 。 (36:1) [FileTabCharacter]
    'method def modifier' 缩进了8个缩进符，应为2个。 (36:9) [Indentation]
    行内含有制表符 tab 。 (37:1) [FileTabCharacter]
    行内含有制表符 tab 。 (38:1) [FileTabCharacter]
    'method def' 子元素缩进了16个缩进符，应为4个。 (38:17) [Indentation]
    行内含有制表符 tab 。 (39:1) [FileTabCharacter]
    'method def rcurly' 缩进了8个缩进符，应为2个。 (39:9) [Indentation]
    行内含有制表符 tab 。 (41:1) [FileTabCharacter]
    'method def modifier' 缩进了8个缩进符，应为2个。 (41:9) [Indentation]
    缺少 Javadoc 。 (41:9) [MissingJavadocMethod]
    行内含有制表符 tab 。 (42:1) [FileTabCharacter]
    行内含有制表符 tab 。 (43:1) [FileTabCharacter]
    'method def' 子元素缩进了16个缩进符，应为4个。 (43:17) [Indentation]
    行内含有制表符 tab 。 (44:1) [FileTabCharacter]
    'method def' 子元素缩进了16个缩进符，应为4个。 (44:17) [Indentation]
```

Let's go back to the Check style: On the `highlight text` it said: 'method def modifier' indent 8 and should be 2. The above line says： there is tab in line. Both problem happen in line 36, lets see what line 36 do.


SB spring-boot-build
main
Tests in 'spring-boot-build.spri...ctuator'
Testing6.md
SampleActuatorApplication.java
ExampleHealthIndicator.java
ExampleInfoContributor.java
HelloWorldService.java

```
     Madhura Bhave +4
32   @SpringBootApplication
33   @ConfigurationPropertiesScan
34   public class SampleActuatorApplication {
35
         Madhura Bhave +1
36       @Bean
37       public HealthIndicator helloHealthIndicator() {
38           return createHealthIndicator( value: "world");
39       }
40
         Madhura Bhave
41       @Bean
42       public HealthContributor compositeHelloHealthContributor() {
43           Map<String, HealthContributor> map = new LinkedHashMap<>();
44           map.put("spring", createNestedHealthContributor( name: "spring"));
45           map.put("boot", createNestedHealthContributor( name: "boot"));
46           return CompositeHealthContributor.fromMap(map);
```

This is line 36 in `SampleActuatorApplication` class, we can see this is a @Bean annotation, which is a spring boot characteristic. It's clear that the default Checkstyle configuration cannot identify the annotation and treat this as an unexpected indentation. Let's see how PMP did.


M↓ Testing6.md
SampleActuatorApplication.java ×
ExampleHealthIndicator.java
ExampleInfoContributor.java
HelloWorldService.java

```
30  import org.springframework.context.annotation.Bean;
31
    Madhura Bhave +4
32  @SpringBootApplication
33  @ConfigurationPropertiesScan
34  public class SampleActuatorApplication {
35
        Madhura Bhave +1
36      @Bean
37      public HealthIndicator helloHealthIndicator() {
38          return createHealthIndicator( value: "world");
39      }
40
        Madhura Bhave
41      @Bean
```

```
PMD ×
PMD Results (67 violations in 9 scanned files using 7 rule sets)
  codestyle (26 violations: 26)
    AtLeastOneConstructor (7 violations)
      (24, 8) ExampleHealthIndicator in smoketest.actuator
      (26, 8) ExampleInfoContributor in smoketest.actuator
      (34, 8) SampleActuatorApplication in smoketest.actuator
      (28, 8) SampleLegacyEndpointWithDot in smoketest.actuator
      (28, 8) SampleLegacyEndpointWithHyphen in smoketest.actuator
      (34, 8) SampleRestControllerEndpointWithException in smoketest.actuator
      (22, 8) ServiceProperties in smoketest.actuator
    CommentDefaultAccessModifier (3 violations)
      (42, 9) SampleRestControllerEndpointWithException$CustomExceptionHandler in smoketest.actuator
      (45, 3) SampleRestControllerEndpointWithException$CustomExceptionHandler.handleCustomException() in smoketest.actuator
      (50, 9) SampleRestControllerEndpointWithException$CustomException in smoketest.actuator
    DefaultPackage (1 violation)
    LocalVariableCouldBeFinal (4 violations)
```

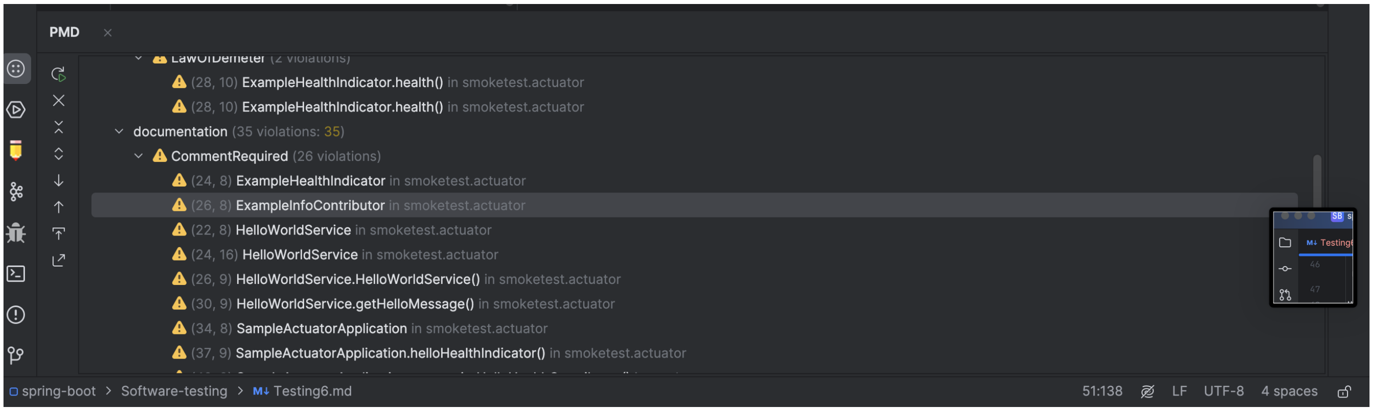
boot-smoke-tests > spring-boot-smoke-test-actuator > src > main > java > smoketest > actuator > SampleActuatorApplication > helloHealthIndicator
39:6
LF
UTF-8
Tab

Similarly, PMP doesn't perform very well in default, in line 34, it said it should declare at least one constructor, but actually, spring boot will automatically offer a default constructor when there is no explicit constructor exist.

Hence, both static analyzers may not function well without specific configuration. And nowadays, we did have a very powerful static analyzer call IDEA.

Check Overlap

Some functions both static analyzers provided are similar, for example, the code style. In PMD, its located in the `documentation section` and `comment size` :



```
PMD ×
    LawOfDemeter (2 violations)
      (28, 10) ExampleHealthIndicator.health() in smoketest.actuator
      (28, 10) ExampleHealthIndicator.health() in smoketest.actuator
  documentation (35 violations: 35)
    CommentRequired (26 violations)
      (24, 8) ExampleHealthIndicator in smoketest.actuator
      (26, 8) ExampleInfoContributor in smoketest.actuator
      (22, 8) HelloWorldService in smoketest.actuator
      (24, 16) HelloWorldService in smoketest.actuator
      (26, 9) HelloWorldService.HelloWorldService() in smoketest.actuator
      (30, 9) HelloWorldService.getHelloMessage() in smoketest.actuator
      (34, 8) SampleActuatorApplication in smoketest.actuator
      (37, 9) SampleActuatorApplication.helloHealthIndicator() in smoketest.actuator
```

The document section is ask you to add some comments on the class and constructor, and the comment size will inform you when the comment is too large. I do not see the overlap appear. In summary, the checkStyle is focusing on the formatting only, but the PMD is mainly used to find potential problems in the code, such as complex code that may cause errors, unused parameters or variables, overly complex expressions, etc. It focuses more on the logical and structural quality of the code. However, without targeted configuration, both plugins are far from the level that can be deployed in a production environment.