sonarqube

Natalia Marcinkowska, Maciej Maj, Damian Maślanka

Wstęp

SonarQube to platforma typu open source służąca do ciągłej analizy kodu statycznego pod względem jego jakości. Umożliwia automatyczne wykrywanie błędów, słabych punktów powodujących obniżenie bezpieczeństwa oraz tak zwanych "code smells".

Wspiera ponad 30 różnych języków programowania, co czyni go wyjątkowym narzędziem analitycznym.

Cele

- Poprawa jakości kodu SonarQube automatycznie analizuje kod, identyfikując problemy związane z jakością, takie jak duplikaty kodu, zła struktura czy nieefektywne rozwiązania.
- Zapewnienie bezpieczeństwa Narzędzie pomaga wykrywać luki bezpieczeństwa i problemy związane z podatnościami, np. SQL Injection, Cross-Site Scripting (XSS) czy ataki typu Buffer Overflow.



Cele ciąg dalszy

- Automatyzacja i optymalizacja pracy Narzędzie pozwala na automatyczne przeprowadzanie analiz w różnych fazach cyklu życia aplikacji, dzięki czemu błędy są wykrywane wcześniej i mogą być szybciej korygowane.
- Utrzymanie zgodności z najlepszymi praktykami – SonarQube wspiera deweloperów w przestrzeganiu standardów i zasad programowania, co jest kluczowe w zespołach developerskich, aby kod był spójny i łatwy w utrzymaniu.



Instalacja

- Pobierz bezpłatną wersję SonarQube ze strony: <u>https://www.sonarsour</u> <u>ce.com/products/sonar</u> <u>qube/downloads/</u>
- Rozpakuj pobrany folder



SonarQube wymaga posiadania Javy w wersji 11, albo 17

Instalacja cd.

SonarScanner CLI

SonarScanner Issue Tracker

6.2.1

FIPS support and improved SSL configuration

Download scanner for: Linux x64 Linux AArch64 Windows x64 macOS x6

Any (Requires a pre-installed JVM)

Release notes

- Pobierz SonarScanner ze strony: <u>https://docs.sonarsource.com/sonarq</u> <u>ube/latest/analyzing-source-</u> <u>code/scanners/sonarscanner/,</u> wybierając odpowiednie oprogramowanie.
- Rozpakuj
- Dodaj ścieżkę do katalogu bin SonarScannera do zmiennej środowiskowej PATH*.

*Wchodzimy w Ustawienia -> System -> Zaawansowane ustawienia systemu -> Zmienne środowiskowe -> W Zmiennych systemowych wybieramy PATH -> Edytuj

Instalacja cd.

- Wejdź w terminalu do katalogu bin SonarQube
- Wybierz folder z odpowiednim systemem operacyjnym i również do niego wejdź
- Aby uruchomić serwer SonarQube użyj komendy:
- StartSonar.bat lub sonar.sh



Instalacja cd.

- Wejdź na stronę: http://localhost:9000
- ✤ Zaloguj się:
- Login = admin
- Hasło = admin
- Ustaw swoje nowe hasło

Strona SonarQube



How do you want to create your project?

Do you want to benefit from all of SonarQube's features (like repository import and Pull Request decoration)? Create your project from your favorite DevOps platform.

First, you need to set up a DevOps platform configuration.

Import from Azure DevOps	Setup	1 Import from Bitbucket Cloud	Setup	Import from Bitbucket Server	Setup
Import from GitHub	Setup	₩ Import from GitLab	Setup		

Are you just testing or have an advanced use-case? Create a local project.

Create a local project



Bugs to błędy logiczne lub funkcjonalne w kodzie, które powodują, że program nie działa zgodnie z oczekiwaniami.



Vulnerabilities (podatności) to punkty w kodzie, które są narażone na ataki.



Code Smells (problemy stylistyczne lub projektowe) to fragmenty kodu, które choć nie powodują bezpośrednich błędów, wskazują na obszary wymagające poprawy pod kątem czytelności, wydajności lub utrzymania.



Każdy z tych problemów ma jeden z poziomów powagi (severity): blocker, critical, major, minor, info.

Typy błędów

Zakładka "Rules" służy do zarządzania regułami, które SonarQube
wykorzystuje do analizy kodu. Reguły definiują, jakie aspekty kodu są
sprawdzane pod kątem jakości, bezpieczeństwa, wydajności,
zgodności ze standardami oraz innych dobrych praktyk. Dzięki nim
SonarQube może znaleźć w kodzie bugs, code smells, issues itd.
Możemy filtrować je np. względem Javy.



Quality Profiles

Zakładka "Quality Profiles" pozwala na grupowanie reguł w zestawy, które są dostosowane do konkretnych technologii, typów projektów, lub wymagań. Profile jakości są stosowane do projektów, aby określić, jakie reguły mają być stosowane podczas analizy kodu.

Copy Profile "Sonar way" - Java

This new quality profile won't inherit from a built-in profile. It will not benefit from automatic updates when new rules are introduced.

If you want to benefit from automatic updates, consider extending a built-in quality profile instead.

Create a new quality profile as a replica of "Sonar way". The two profiles will then evolve independently.

All fields marked with * are required

New name *

A

Sonar way-custom



Sonar way-custom			Updated: 20 seconds ago	Used: Never See C	hangelog
nheritance Change Parent			Rule breakdown		
This quality profile does not inherit from a built-i To benefit from automatic updates, set the corre	n profile. It will not benefit from automatic updates when	new rules are introduced.	Clean Code Categories	Active	Inactive
	opononig cont in promo do trio parent or your quanty pro		Consistency	79	53
onar way-custom 512 active rule	s 155 inactive rules	0 overridden rules	Intentionality	347	80
Projects Change Projects			Adaptability	36	19
lo projects are explicitly associated to the profile.			Responsibility	14	2
Permissions			Software Qualities	Active	Inactive
Isers with the global "Administer Quality Profiles" permi	ssion and those listed below can manage this quality pro	file.	Security	29	3
			Reliability	161	11
Grant permissions to more users			Maintainability	293	140
			Activate More		

Quality Gates

Zakładka "Quality Gates" umożliwia definiowanie kryteriów, które projekt musi spełnić, aby zostać uznanym za gotowy do wdrożenia. Są to zestawy warunków, które muszą być spełnione przez kod po analizie, aby można było uznać go za akceptowalny.

Conditions ?

Conditions on New Code

Metric	Operator	Value	
Issues	is greater than	0	
Security Hotspots Reviewed	is less than	100%	
Coverage	is less than	80.0%	1

Generacja raportu (Maven)

1. Dodaj projekt

		Create Project. ~
My Favorites All		
	Q Search for projects Perspective Overall Status V Sort by	🖌 🖌 From GitLab
Filters		Local project
Quality Gate	☆ 'SonarCube-maven' PUBLIC	Import from other DevOps Platforms
	Last analysis: 7 minutes ago - 200 Lines of Code - XML, Java	
✓ Passed 2		
× Failed 0	E 1 A 0 A 29 A — 0 0.0% Security Reliability Maintainability Hotspots Reviewed Coverage Duplications	
Security		
A ≥ 0 info issues 0	☆ 'sonarcube_4' PUBLIC	✓ Passed
B ≥ 1 minor issue 0	Last analysis: 1 day ago • 107 Lines of Code • XML, Java	
C ≥ 1 major issue 0		
D ≥ 1 critical issue 0	E 1 A 0 A 4 A - O 0.0% • 0.0%	
E ≥ 1 blocker issue 2	Security Reliability Maintainability Hotspots Reviewed Coverage Duplications	

Create a local project

Project display name *

SonarCube-M	Maven	C
Project key *	SonarCube-maven	
SonarCube-N	Maven	C

Main branch name *

main

The name of your project's default branch Learn More 🖸



Generacja raportu (Maven)

2. Wygeneruj Token

Generacja raportu (Maven)

3. Wybierz opcje Maven, wejdź w konsoli do folderu ze swoim projektem i uruchom poniższą komende bez \

1 Provide a token	Analyze "SonarCube-Maven1": sqp_3c4a0703ec4ecb23c90a929ecf877d5c708585ef
2 Run analysis on your project What option best describes your project?	
Execute the Scapper for Mayen	
Running a SonarQube analysis with Maven is straighforward. You just need to run the following command in your project's folder.	
<pre>mvn clean verify sonar:sonar \ -Dsonar.projectKey=SonarCube-Maven1 \ -Dsonar.projectName='SonarCube-Maven1' \ -Dsonar.host.url=http://localhost:9000 \ </pre>	Сору
-Dsonar.token=sqp_3c4a0703ec4ecb23c90a929ecf877d5c708585ef	

Please visit the official documentation of the Scanner for Maven [2] for more details

package com.example;

import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication;

```
@SpringBootApplication
public class App {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
```

```
int additionResult = addNumbers( a: 5, b: 10);
System.out.println("Addition result: " + additionResult);
```

```
int subtractionResult = subtractNumbers( a: 15, b: 10);
System.out.println("Subtraction result: " + subtractionResult);
```

```
private static int addNumbers(int a, int b) { 1usage
    int result = a + b + 0;
    return a + b + 0;
```

```
•
```

private static int subtractNumbers(int a, int b) { 1usage int result = a + b; result = a + b; return result;

Przykładowy kod

× Quality Gate P Failed			Last analysis 5
A The last analysis has warnings. <u>See details</u>			
New Code 2 failed Overall Code			
New Code: Since October 30, 2024 Started 3 days ago			
2 conditions failed	New issues FAILED		Accepted issues
4 Issues	4 Required = 0	ب د	O Valid issues that were not fixed
O 0.0% Coverage is less than 80.0%			
	Coverage		Duplications
Fix issues before they fail your Quality Gate with <u>SonarLint</u> ⊖ in your IDE. Power up with	0.0%	0	0.0%

	Replace this use of System.out by a logger.	Adaptabilit
	Maintainability 📀	bad-practice cert
> Software Quality	○ Open ~ Not assigned ~	L15 = 10min effort = 1 day ago = 🛞 Code Smell = 🗞 Ma
	Remove this useless assignment to local variable "result".	Intentionalit
> Severity ?	Maintainability	cwe cert
> Type	○ Open ∽ Not assigned ∽	L19 • 1min effort • 6 minutes ago • 🕃 Code Smell • 💿 Ma
	Remove this unused "result" local variable.	Intentionalit
> Scope	Maintainability 📀	unused
	○ Open ~ Not assigned ~	L19 = ⊖ = 5min effort = 6 minutes ago = & Code Smell = ♥ Mir
> Status		
> Security Category		
, , ,		



Badanie pokrycia kodu - coverage

- Pokrycie kodu pomaga upewnić się, że kod jest dobrze przetestowany, co przyspiesza wykrywanie nowych błędów.
- Analiza pokrycia wskazuje nieprzetestowane obszary, które mogą być podatne na błędy przy zmianach.
- JaCoCo (Java Code Coverage) to narzędzie do analizy pokrycia kodu przez testy w projektach opartych na Javie.
- JaCoCo pozwala programistom skupić się na kluczowych ścieżkach, co usprawnia budowanie testów i zarządzanie czasem ich wykonania.
- JaCoCo wspiera utrzymanie standardów i efektywne zarządzanie testowaniem, szczególnie w dużych projektach.
- JaCoCo jest dostępne jako wtyczka do Maven i Gradle, co ułatwia jego integrację z popularnymi narzędziami do budowania projektów Java.



Jak korzystać z JaCoCo

- Dodaj wtyczkę do projektu: W pliku pom.xml dodaj wtyczkę JaCoCo, aby była uruchamiana w fazie testowej.
- Uruchom testy z JaCoCo: Po dodaniu wtyczki wystarczy wykonać komendę mvn test, aby wygenerować raport pokrycia kodu.
- Analiza wyników: Po zakończeniu testów JaCoCo wygeneruje raporty pokrycia kodu w formacie HTML. Wygenerowane pliki można znaleźć w katalogu target/site/jacoco.

<build> <plugins> <plugin> <groupId>org.jacoco</groupId> <artifactId>jacoco-maven-plugin</artifactId> <version>0.8.7</version> <executions> <execution> <goals> <goal>prepare-agent</goal> <goal>report</goal> </goals> </execution> </executions> </plugin> </plugins> </build>

Co robi JaCoCo?

 JaCoCo generuje kompleksowe raporty, które dokładnie pokazują, w jakim stopniu kod aplikacji został objęty testami. Raporty te dostarczają szczegółowej analizy pokrycia na różnych poziomach:

• Instrukcje (Linie Kodów)

Analiza pokrycia poszczególnych linii kodu, która wskazuje, ile instrukcji zostało przetestowanych. Stanowi najniższy poziom szczegółowości, identyfikując linie pominięte przez testy.

• Ścieżki Decyzyjne (Gałęzie)

Pokrycie logiki warunkowej, które śledzi wszystkie możliwe ścieżki wykonania, np. w instrukcjach warunkowych if-else. Dzięki temu raporty pomagają dostrzec potencjalnie pominięte scenariusze.

• Metody

Analiza pokazująca, czy wszystkie metody w kodzie zostały uruchomione podczas testów, co pozwala ocenić kompletność testów na poziomie poszczególnych funkcji.

• Klasy

Ocena pokrycia klas w aplikacji. Ten poziom wskazuje, czy każda klasa została w pełni objęta testami, co pozwala kontrolować pokrycie na poziomie ogólnych jednostek kodu.



Co to są raporty JaCoCo?



Czym są raporty JaCoCo?

Raporty generowane przez JaCoCo pokazują, jaki procent kodu aplikacji został pokryty testami, analizując szczegółowo klasy, metody i linie kodu.



Jak działają?

JaCoCo monitoruje wykonanie testów jednostkowych i integracyjnych, generując raporty w różnych formatach (HTML, XML, CSV). Wyniki pozwalają szybko zidentyfikować nieprzetestowane części kodu. Można go zintegrować z narzędziami CI/CD, np. Jenkins, dla automatycznego raportowania.



Zalety

Raporty JaCoCo umożliwiają szczegółowe monitorowanie pokrycia kodu testami, pomagając zespołowi identyfikować nieprzetestowane fragmenty kodu i zwiększając ogólną stabilność oraz jakość aplikacji.

Element	Missed Instructions +	Cov. 🗢	Missed Branches 🔅	Cov. 🗢	Missed	Cxty≑	Missed	Lines	Missed	Methods	Missed	Classes
<u> </u>		97%		91%	143	1,537	125	3,631	19	746	2	147
employed and a constraint of the constraint of t		58%		64%	24	53	97	193	19	38	6	12
<u> / @erg.jacoco.agent.rt</u>	=	75%		83%	32	130	75	344	21	80	7	22
iacoco-maven-pluging		90%		82%	35	193	49	465	8	116	1	23
<u> </u>	=	97%		100%	4	109	10	275	4	74	0	20
<u> </u>		99%		99%	4	572	2	1,345	1	371	0	64
/ <u>≇org.jacoco.ant</u>	=	98%		99%	4	162	8	428	3	110	0	19
<u> / erg.jacoco.agent</u>		86%		75%	2	10	3	27	0	6	0	1
Total	1,438 of 28,925	95%	183 of 2,386	92%	248	2,766	369	6,708	75	1,541	16	308

- **Tabela pokrycia kodu**: Zawiera wskaźniki procentowe dla instrukcji, gałęzi, metod, klas.
- **Raport szczegółowy dla klas**: Pozwala sprawdzić dokładne linie kodu pokryte i niepokryte przez testy. Linie pokryte mogą być zaznaczone na zielono, a niepokryte na czerwono.
- Podsumowanie pokrycia projektu: Procentowe zestawienie pokrycia dla całego projektu, które można używać do mierzenia postępów w testach regresyjnych.

Raporty

Badanie pokrycia kodu testami

- Stworzenie projektu Maven
- Dodanie JUnita5 do pom.xml

<dep< th=""><th>endencies></th></dep<>	endencies>
	<dependency></dependency>
	<proupid>org.junit.jupiter</proupid>
	<artifactid>junit-jupiter-api</artifactid>
	<version>5.11.3</version>
	<scope>test</scope>
<td>pendencies></td>	pendencies>

Badanie pokrycia kodu testami

 Dodanie dowolnego testu, np.. w pliku MainTest.java i stworzenie test testGetText

package pl.edu.agh.kis.pz1;

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
public class MainTest {
    @Test
    public void testGetText() {
        // Test the getText method
        String expected = "Some text 2";
        assertEquals(expected, Main.getText(),
```

message: "The getText method should return 'Some text 2'");



<plugins> <plugin>

<groupId>org.jacoco</groupId>

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

<version>0.8.12</version>

<executions>

<execution>

<goals>

<goal>prepare-agent</goal>

</goals>

</execution>

<execution>

<id>report</id>

<phase>test</phase>

<goals>

<goal>report</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

Badanie pokrycia kodu testami

 Stworzenie raportów code coverage poprzez jacoco

- Dodanie jacoco w pom.xml
- Po odpaleniu "mvn test" pojawi się raport w pliku:

target/site/jacoco/index.html

Badanie pokrycia kodu testami

• Dodanie SonarQube do projektu w pliku pom.xml:



Stworzenie pliku na token np.. Token.txt. Przykładowa zawartość pliku: token.txt (token wygenerowany ze strony http://localhost:9000): sqp_cb6eecdb88c9e94fe85437962972bc2c5ee7966

Badanie pokrycia kodu testami

 Stworzenie pliku script.sh, który będzie budować projekt i odpali statyczną analizę i pokrycie kodu:

```
TOKEN="$(cat ./token.txt)"
mvn clean verify sonar:sonar \
    -Dsonar.projectKey=sonarqube-demo-1 \
    -Dsonar.projectName='SonarQube Demo 1' \
    -Dsonar.host.url=http://localhost:9000 \
    -Dsonar.token="$TOKEN"
```

Po odpaleniu skryptu za pomocą komendy ./script.sh w panelu SonarQube powinno pojawić się Issues oraz informacja o pokryciu kodu.

Jeśli po stworzeniu pliku script.sh brakuje uprawnień to używamy komendy chmod +x script.sh



Badanie pokrycia kodu testami

Issues Security Hotspo	ts Measures	Code Activity	Project Settings V P		
Filtere		src/main/java/pl/edu/agh/Kis/pz1/Main.java		.1	
Filters		Replace this use of System.out by a logger.	Adaptability	4	<pre>public static void main(String[] args) {</pre>
Issues in new code		(Maintainability 🕤	bad-practice cert +	5	<pre>System.out.println("Hello World! " + getText());</pre>
 Clean Code Attribute 		○ Open ~ Not assigned ~	L5 - 10min effort - 29 minutes ago - 😔 Code Smell - 🚇 Major		
Consistency	0	Remove this method and declare a constant for this value.	Intentionality		Replace this use of System.out by a logger.
Intentionality	5	(Maintainability 🛇)	confusing +		
Adaptability	1			6	}
Responsibility	0	O open < . Not assigned <	L9 * Smin errort + 29 minutes ago + 9 Code Smith + 9 Minor	7	
 Software Quality 		src/test/java/pl/edu/agh/kis/pz1/MainTest.java		8	<pre>public static String getText() {</pre>
Security	0	Remove this unused import 'java.io.ByteArrayOutputStream'.	Intentionality	9	return "Some text 2";
Reliability	0	(Maintainability 🔘	unused +		······································
Maintainability	6	Open ✓ Not assigned ✓	L6 - 😑 - 1min effort - 29 minutes ago - 🟵 Code Smell - 🛛 Minor		Domovo this method and declars a constant for this va
					Remove this method and declare a constant for this va

Badanie pokrycia kodu testami

Measures

Maintainability ?	>
Security Review ?	>
Coverage	~
Overview	
Overall Code	
Coverage	25.0%
Lines to Cover	4
Uncovered Lines	3
Line Coverage	25.0%

-

SonarQube Demo 1 > src > main/.../pl/edu/agh/kis/pz1 > Main.java

Lines to Cover 4

1		<pre>package pl.edu.agh.kis.pz1;</pre>
2		
3		<pre>public class Main {</pre>
4		<pre>public static void main(String[] args) {</pre>
5	•	<pre>System.out.println("Hello World! " + getText());</pre>
5		}
7		
В		<pre>public static String getText() {</pre>
9	•	<pre>return "Some text 2";</pre>
10		}
11		}
12		

Wykres pokrycia kodu

- Oś Y Coverage (Pokrycie): Wskaźnik pokrycia kodu testami jednostkowymi, wyrażony w procentach.
- **Oś X Czas (np. 5min, 10min, 15min)**: Przedstawia szacowany czas potrzebny do rozwiązania problemów z jakością kodu aby spełniał wymagania jakościowe.
- Każde kółko reprezentuje projekt lub część kodu, przy czym:
- Kolor kółka Ocena (Rating): Kolor zielony oznacza, że kod ma dobrą ocenę pod względem jakości i bezpieczeństwa
- Rozmiar kółka Lines of Code (Linie kodu): Wielkość kółka obrazuje ilość linii kodu.



Przydatne linki

Instalacja:

https://www.sonarsource.com/products/sonarqube/ downloads/

https://docs.sonarsource.com/sonarqube/latest/anal yzing-source-code/scanners/sonarscanner/

Dokumentacja:

https://docs.sonarsource.com/sonarqube/9.9/userguide/issues/

Bibliografia

https://www.altkomsoftware.com/pl/blog/sonarqub e-pierwsze-kroki/ https://docs.sonarsource.com/sonarqube/9.9/userguide/issues/

https://www.eclemma.org/jacoco/

Dziękujemy za uwagę!