

# **JUnit Ferramenta de testes em Java**

*Pedro Beck Bratfich Penteado  
pedrobbp22@gmail.com  
pedro.penteado@fatec.sp.gov.br*

## **Resumo**

Este estudo visa apresentar a ferramenta JUnit, frequentemente empregada para a execução de testes unitários em programas Java. A seleção da ferramenta foi motivada pela sua importância no cenário do desenvolvimento baseado em testes (TDD), uma prática que vem ganhando cada vez mais destaque no mercado de software. A abordagem utilizada consistiu na análise teórica da ferramenta, seguida pela implementação prática em projetos menores desenvolvidos ao longo do semestre. Por meio da prática, pude observar como a utilização do JUnit auxilia na detecção antecipada de falhas, na manutenção do código e na diminuição do retrabalho. Os achados reforçam a relevância de implementar testes automatizados desde os estágios iniciais do desenvolvimento, consolidando práticas recomendadas de engenharia de software.

Palavras-chaves: JUnit. Java. TDD.

## **Introdução**

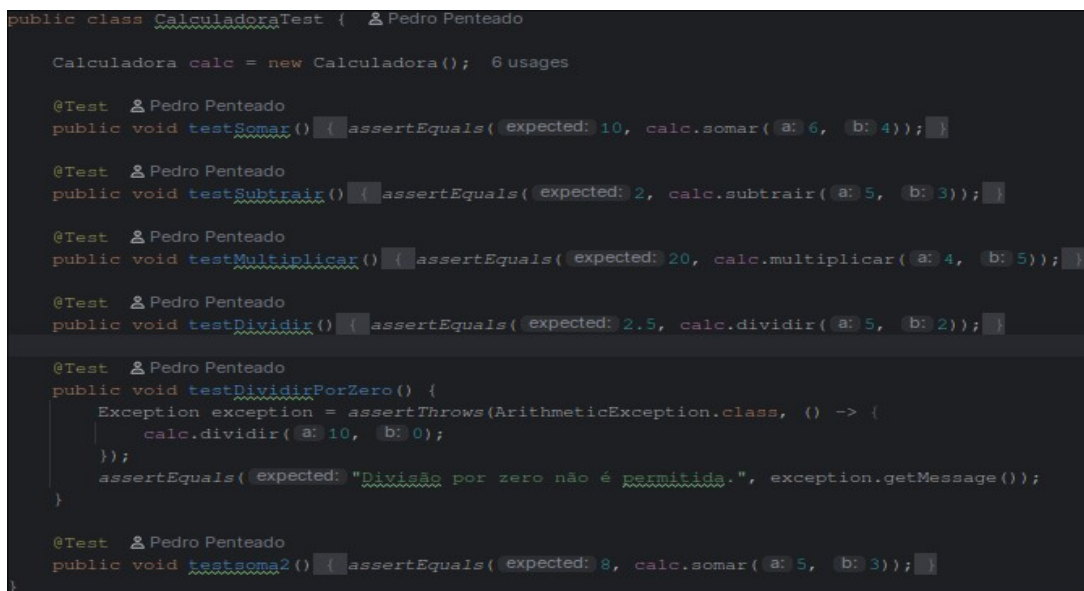
A criação de software de alto padrão requer a implementação de práticas que assegurem a confiabilidade, manutenção e expansibilidade do código. Neste cenário, os testes automatizados têm um papel crucial, particularmente quando aplicados desde os estágios iniciais do desenvolvimento. Este estudo se concentra na análise do JUnit, uma das ferramentas mais utilizadas para a execução de testes unitários em aplicações criadas em Java. A finalidade principal desta pesquisa é entender a operação e a utilidade do JUnit no cenário do desenvolvimento guiado por testes (TDD - Test Driven Development). Dentre as metas específicas, destacam-se: reconhecer as principais características do software, executar testes em projetos básicos e avaliar as vantagens do uso de testes automatizados no cotidiano do programador. A seleção do tema é justificada pela demanda crescente por especialistas em boas práticas de testes, bem como a importância do JUnit no ecossistema Java. Além de ferramentas de visualização de testes como Java Code Coverage e SonarQube, onde o Code Coverage se destaca em visualização única e precisa de código enquanto o SonarQube se sobressai na visualização de diversos scripts ao mesmo tempo.

## Desenvolvimento

O JUnit é um framework de testes unitários para aplicações Java, amplamente utilizado em projetos acadêmicos e profissionais. Ele permite validar o comportamento de métodos e classes por meio de testes automatizados, contribuindo para a detecção precoce de falhas, o aumento da confiabilidade do sistema e a manutenção contínua do código. A prática de desenvolvimento orientado a testes (TDD), que incentiva a escrita dos testes antes mesmo da implementação das funcionalidades, é fortemente suportada por essa ferramenta.

Durante o desenvolvimento deste trabalho, foram exploradas as principais funcionalidades do JUnit, como as anotações `@Test`, `@BeforeEach`, `@AfterEach`, entre outras, e os métodos de asserção como `assertEquals`, `assertTrue` e `assertThrows`.

A figura 1 ilustra a o uso das anotações no IntelliJ IDEA para os testes:



```
public class CalculadoraTest { @ Pedro Penteadó

    Calculadora calc = new Calculadora(); 6 usages

    @Test @ Pedro Penteadó
    public void testSomar() { assertEquals(expected: 10, calc.somar(a: 6, b: 4)); }

    @Test @ Pedro Penteadó
    public void testSubtrair() { assertEquals(expected: 2, calc.subtrair(a: 5, b: 3)); }

    @Test @ Pedro Penteadó
    public void testMultiplicar() { assertEquals(expected: 20, calc.multiplicar(a: 4, b: 5)); }

    @Test @ Pedro Penteadó
    public void testDividir() { assertEquals(expected: 2.5, calc.dividir(a: 5, b: 2)); }

    @Test @ Pedro Penteadó
    public void testDividirPorZero() {
        Exception exception = assertThrows(ArithmeticException.class, () -> {
            calc.dividir(a: 10, b: 0);
        });
        assertEquals(expected: "Divisão por zero não é permitida.", exception.getMessage());
    }

    @Test @ Pedro Penteadó
    public void testSoma2() { assertEquals(expected: 8, calc.somar(a: 5, b: 3)); }
```

Fonte: próprio autor

Conforme observado na figura 1, foram desenvolvidos projetos práticos que aplicam testes em unidades específicas do código.

Além da construção dos testes, foi investigado o conceito de **cobertura de código** (*code coverage*), que avalia o percentual do código fonte que está sendo exercitado pelos testes. Ferramentas como **JaCoCo** foram utilizadas para medir essa cobertura, gerando relatórios que indicam trechos não testados e incentivando a melhoria da qualidade dos testes escritos.

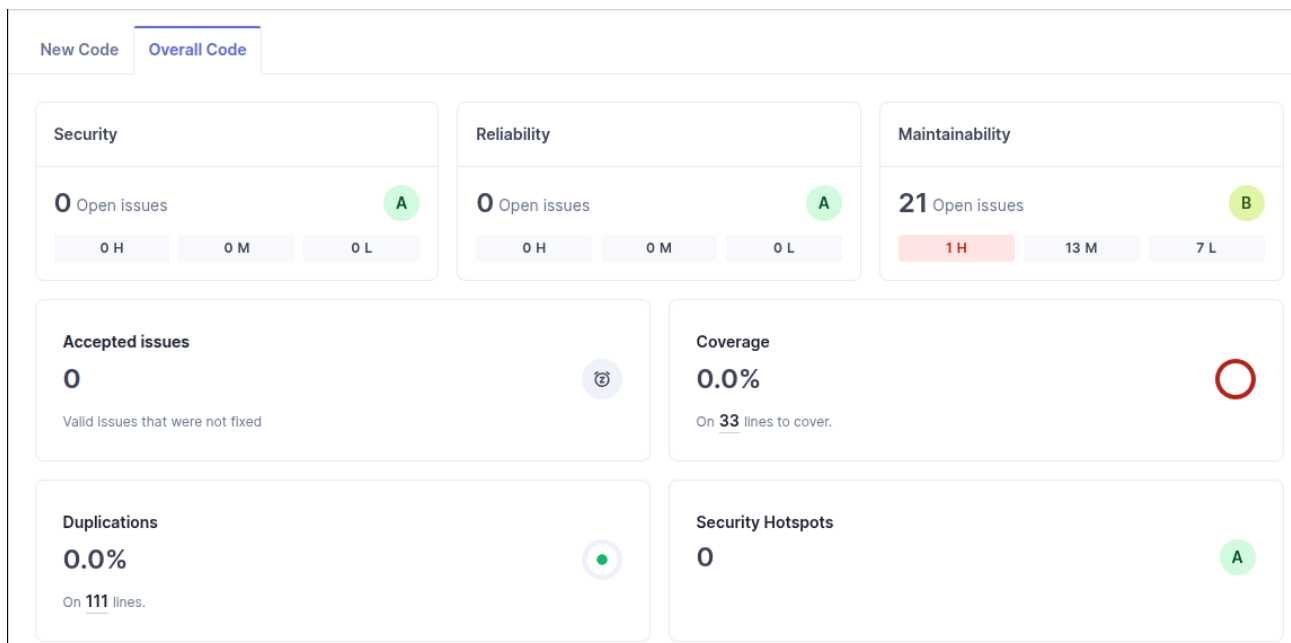
A figura 2 ilustra a o uso do **JaCoCo** no IntelliJ IDEA nativo mostrando testes:

Coverage Main ×				
<div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div>				
Element ^	Class, %	Method, %	Line, %	Branch, %
<div> <div></div> <div>org.example</div> </div>	100% (2/2)	40% (2/5)	57% (15/26)	12% (1/8)
<div> <div></div> <div>Calculadora</div> </div>	100% (1/1)	25% (1/4)	16% (1/6)	0% (0/2)
<div> <div></div> <div>Main</div> </div>	100% (1/1)	100% (1/1)	70% (14/20)	16% (1/6)

Fonte: próprio autor

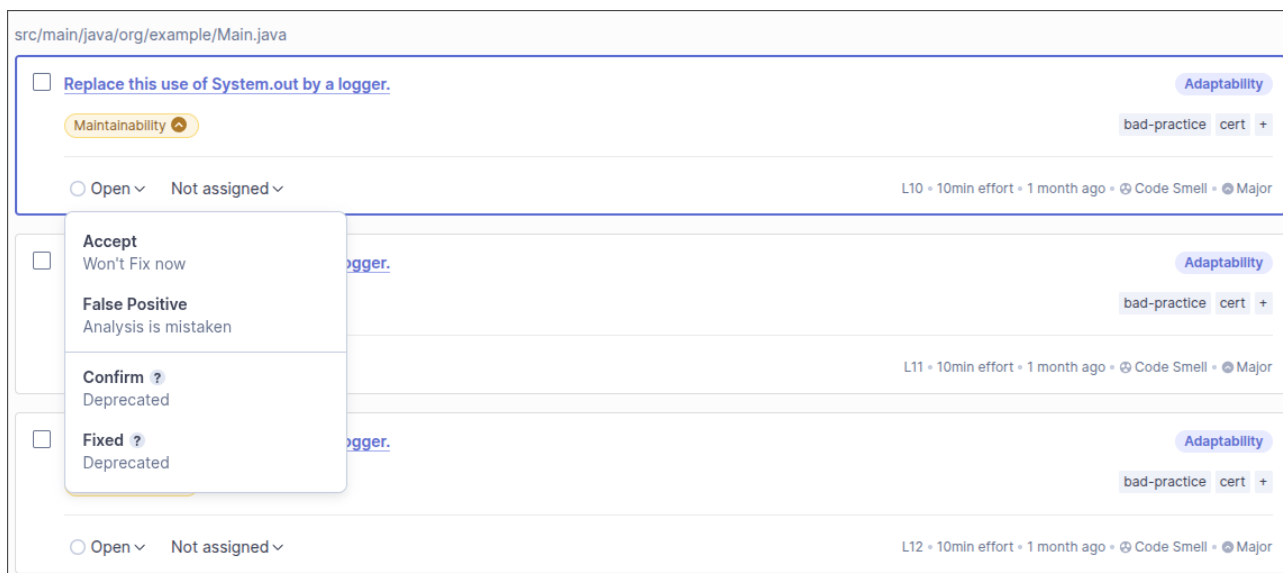
Para complementar a análise de qualidade, foi utilizado o **SonarQube**, uma plataforma que realiza a inspeção contínua de qualidade do código. O SonarQube fornece métricas detalhadas sobre duplicação de código, complexidade ciclomática, presença de *code smells* (alerta de que algo no seu código pode estar mal projetado) e vulnerabilidades de segurança. A integração com projetos Java e com o JUnit permite uma visão unificada da saúde do software, sendo extremamente útil para equipes que desejam manter altos padrões de qualidade.

A figura 3 ilustra a o uso do **SonarQube** mostrando como ele mostra as metricas do projeto:



Fonte: próprio autor

Conforme observado na imagem, o projeto em questão tem uma segurança e confiabilidade alta mas tem mantimento um pouco a baixo. Já na parte de code smells mostrado na figura 4:



*Fonte: próprio autor*

Aqui ele mostra que o uso do “*System.out*” está desatualizado e que deve ser trocado por uma solução permanente.

Conforme observado, a combinação do JUnit com ferramentas de cobertura de código e análise estática fortalece o processo de desenvolvimento de software, promovendo práticas mais profissionais, seguras e sustentáveis. A experiência obtida no uso dessas tecnologias destacou a importância de não apenas escrever testes, mas também de analisar sua efetividade e manter o código limpo e confiável ao longo do tempo.

## Considerações Finais

A execução deste trabalho possibilitou uma compreensão robusta acerca da relevância dos testes automatizados no desenvolvimento de software, bem como do papel fundamental do JUnit nesse contexto. A ferramenta provou ser acessível, eficaz e profundamente integrada ao ecossistema Java, possibilitando a elaboração de testes robustos e reutilizáveis. A implementação prática consolidou a ideia de desenvolvimento orientado a testes (TDD), demonstrando que a elaboração de testes desde as fases iniciais tem um impacto considerável na qualidade, manutenibilidade e confiabilidade do código.

A utilização do JaCoCo para análise de cobertura de código possibilitou uma avaliação objetiva da eficácia dos testes realizados, evidenciando áreas do sistema que ainda não haviam sido adequadamente validadas. Por outro lado, a integração com o SonarQube aprimorou essa análise ao

fornecer métricas detalhadas a respeito da qualidade do código-fonte, identificando code smells, vulnerabilidades e duplicações que poderiam afetar a saúde do projeto.

Em conclusão, a prática regular de testes automatizados, aliada a ferramentas de inspeção e métricas de qualidade, constitui um diferencial importante para o desenvolvedor. Além de minimizar retrabalho e riscos, essas práticas promovem uma cultura de excelência na criação de software.

## **Bibliografia**

**FATEC ARARAS.** Faculdade de Tecnologia de Araras. Disponível em:  
<https://fatecararas.cps.sp.gov.br/>. Acesso em: 11 nov. 2024.

**SARAIVA JUNIOR, Orlando.** ebook\_qualidade\_teste\_software. GitHub. Disponível em:  
[https://github.com/orlandosaraivajr/ebook\\_qualidade\\_teste\\_software](https://github.com/orlandosaraivajr/ebook_qualidade_teste_software). Acesso em: 06 abril. 2025.

**SONARSOURCE.** *SonarQube Documentation*. Disponível em:  
<https://docs.sonarsource.com/sonarqube/>. Acesso em: 18 jun. 2025.

**ECLIPSE FOUNDATION.** *JaCoCo Java Code Coverage Library*. Disponível em:  
<https://www.jacoco.org/jacoco/>. Acesso em: 20 jun. 2025.

**BECK BRATFICH PENTEADO, Pedro.** Ebook\_Pedro\_Beck. GitHub. Disponível em:  
[https://github.com/Dreppo/Ebook\\_Pedro\\_Beck](https://github.com/Dreppo/Ebook_Pedro_Beck). Acesso em: 24 jun. 2025.