



COBERTURA (JAVA)

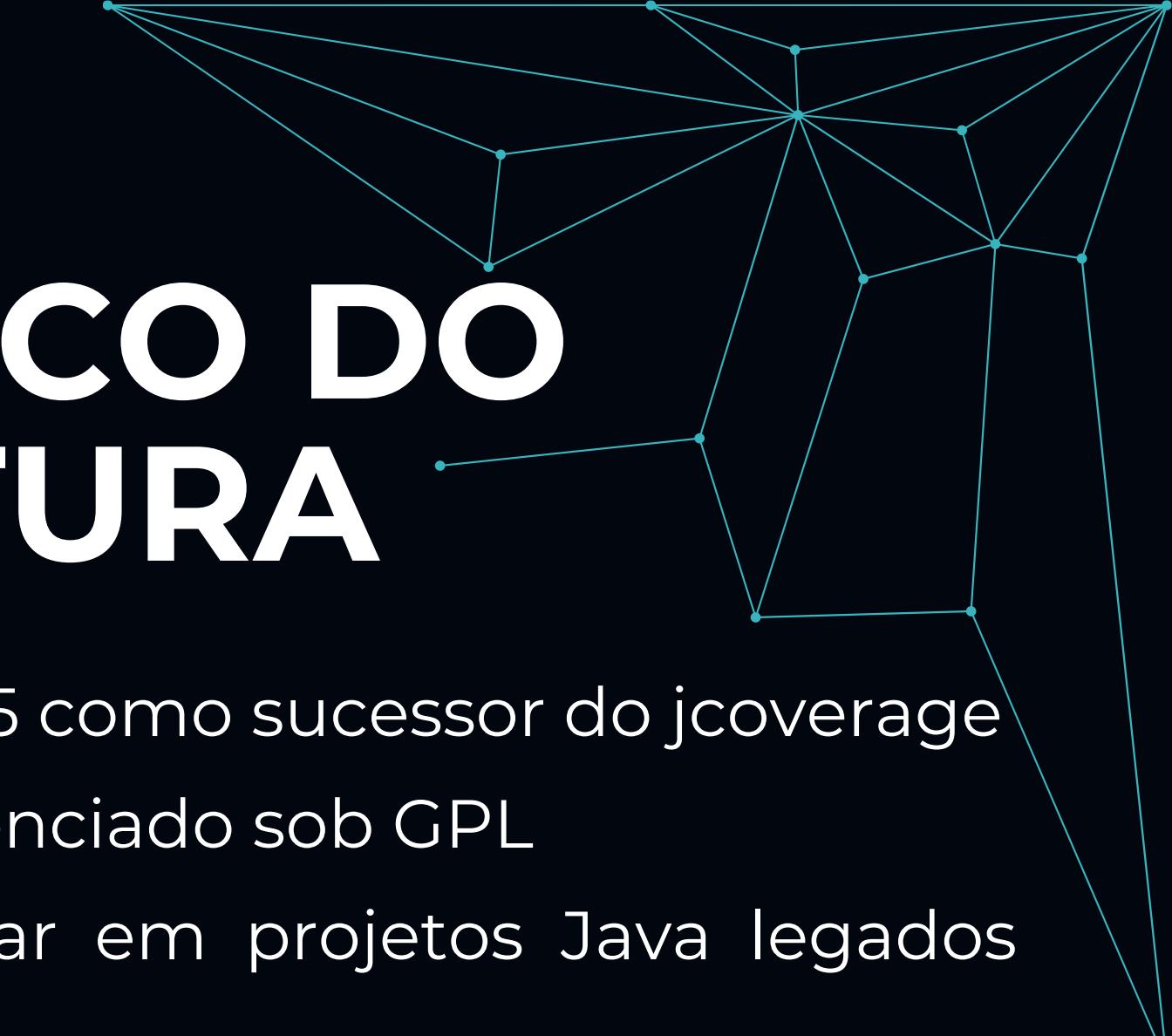
MEDINDO A COBERTURA DE TESTES

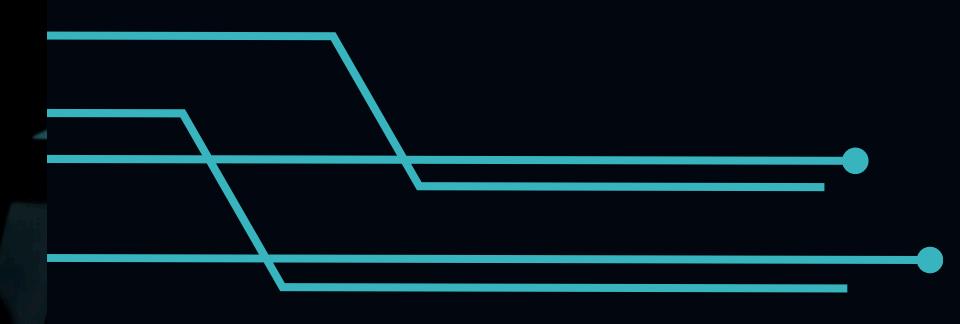


Segurança e Testes de Software
Lucas Vieira e Gabriel Teles



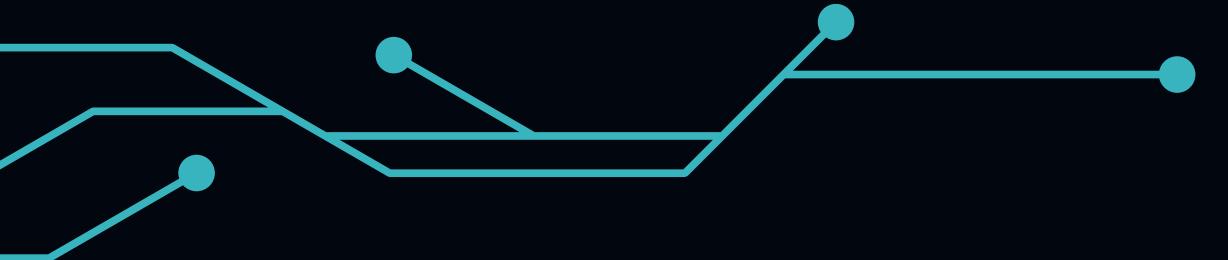
HISTÓRICO DO COBERTURA



- Lançado em 2005 como sucessor do jcoverage
 - Open-source, licenciado sob GPL
 - Tornou-se popular em projetos Java legados (Java 5+)
 - Hoje tem manutenção mínima, mas segue disponível no GitHub
 - Inspirou outros frameworks, como JaCoCo
- 

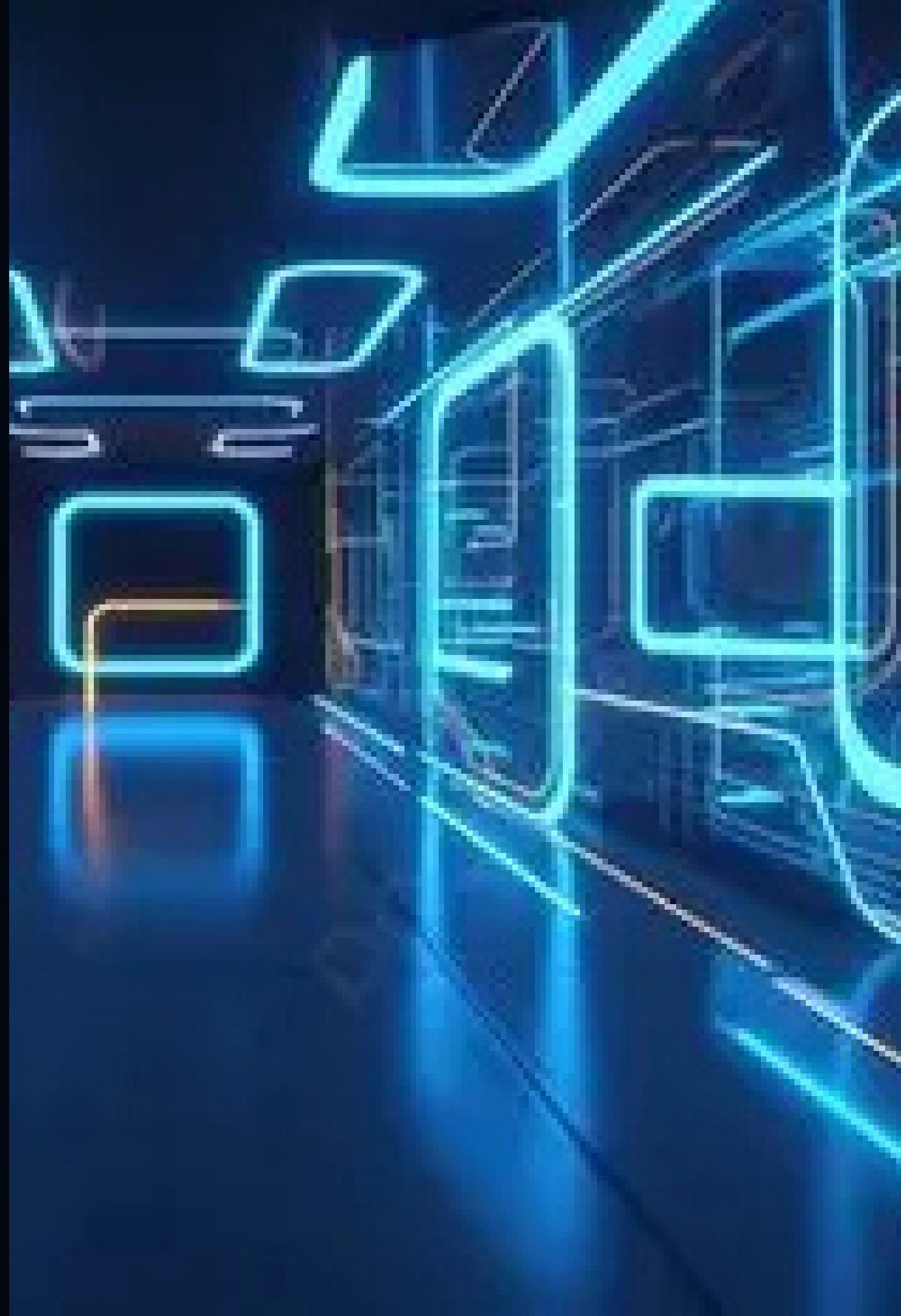
O QUE É? CARACTERÍSTICAS

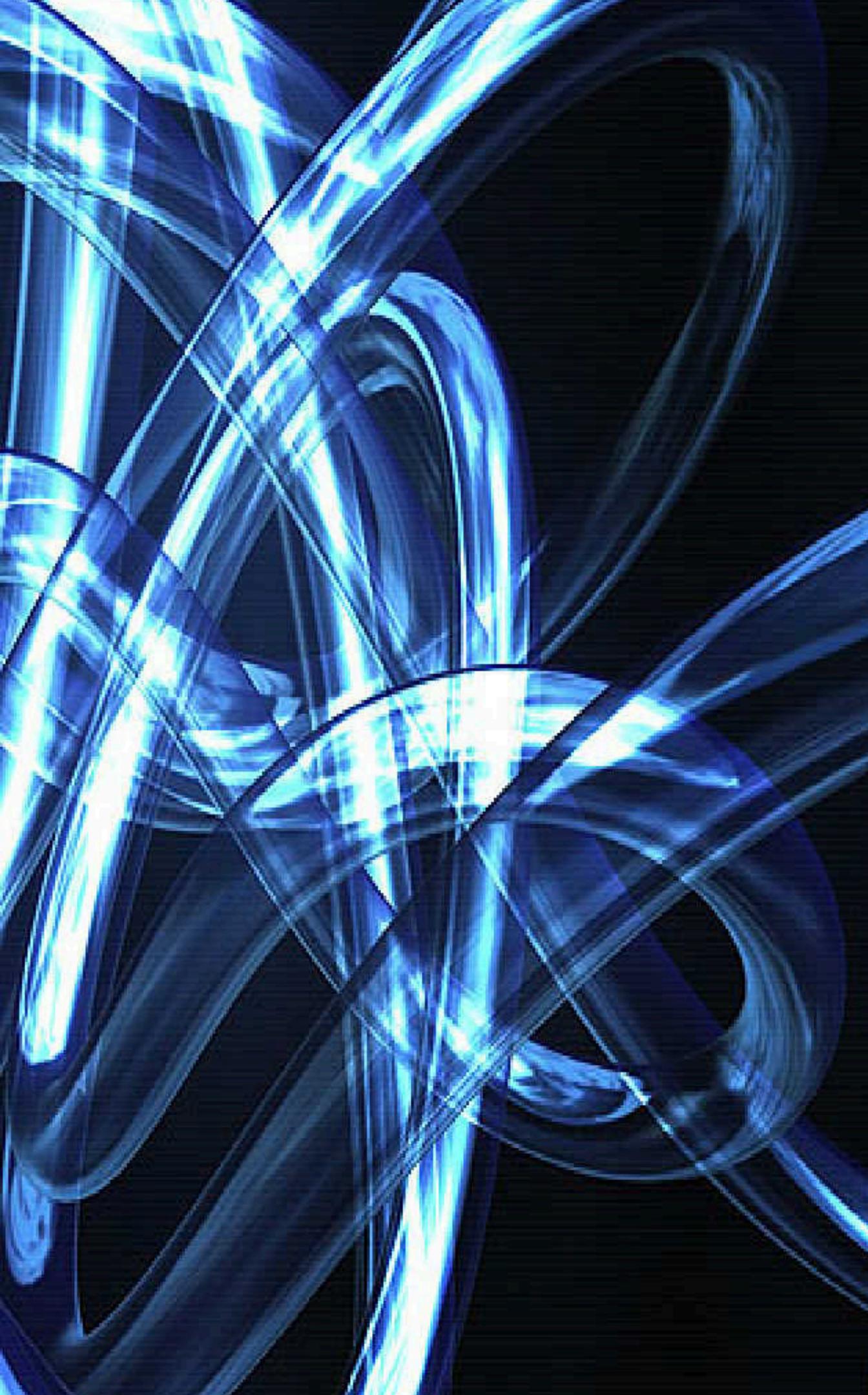
- Não é um framework de testes
- Ferramenta para medir cobertura
- Mede *line coverage* e *branch coverage*
- Open-source
- Gera relatórios em HTML, XML e CSV
- Não executa testes
- Apenas instrumenta o bytecode



PIRÂMIDE DE AUTOMAÇÃO DE TESTES

- Cobertura não substitui testes, apenas mede cobertura
- Focado em *unit tests* (testes de unidade) e *integration tests* (testes de integração)
- Ajuda a avaliar a qualidade e alcance dos testes
- Útil para monitorar regressões em pipelines de CI





TIPOS DE TESTES

CAIXA-BRANCA/PRETA/CINZA

- Funciona com os testes de unidade e com os testes de integração
- Apoia especialmente white-box testing
- Pode ser usado em black-box – Desde que execute o código
- Útil também para gray-box testing



COMO FUNCIONA?



Instrumentação offline do bytecode

Ocorre durante o processo de build



Modifica classes compiladas

Insere instruções adicionais no bytecode



Contadores coletam dados

Incrementam valores quando um bloco é alcançado no código



Gera relatórios de cobertura

HTML, XML e CSV



Diferente do JaCoCo

Não faz uso de agente em tempo de execução

FUNCIONALIDADES



**Mede cobertura
de linhas e
branches**



**Gera relatórios
HTML interativos e
XML Cobertura**



**Permite filtrar e
excluir pacotes
ou classes**



**Integra com Maven,
Gradle e Ant**



**Suporte a
projetos
legados
(Java 1.5+)**

MÉTRICAS DE COBERTURA

Line Coverage

Branch Coverage

Method Coverage

Class/Package
Coverage





LIMITAÇÕES TÉCNICAS



- Menor suporte a versões modernas do Java
- Problemas conhecidos com projetos multi-módulo
- Instrumentação offline adiciona overhead em builds
- Comunidade e atualizações reduzidas nos últimos anos



BOAS PRÁTICAS

Não perseguir 100% de cobertura a todo custo

Priorizar cobertura de código crítico e de negócio

Cobertura baixa em código gerado ou de bibliotecas é aceitável

Acompanhar cobertura em pull requests via CI

Integrar com análise de qualidade (ex.: SonarQube)

INTEGRAÇÕES



JUnit

Testes executados sobre bytecode instrumentado



Maven/Gradle/Ant

- Gradle/Ant – Suporte via plugins
- Maven – Plugins pelo pom.xml



CI/CD

Relatórios XML usados em GitLab, GitHub Actions e Jenkins

FERRAMENTAS SEMELHANTES

JACOCO
Java Code Coverage

JACOCO

Padrão moderno
Agente on-the-fly

OpenClover

**CLOVER /
OPENCLOVER**

Comercial
Híbrido

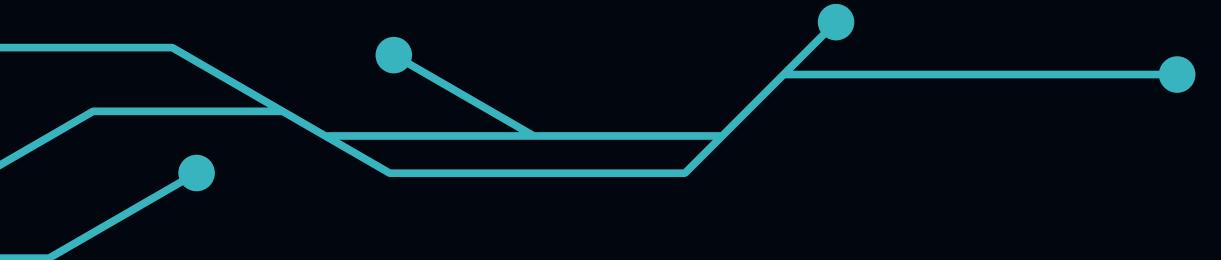
 **MANUAL software**

EMMA

Predecessor do Cobertura
Hoje descontinuado

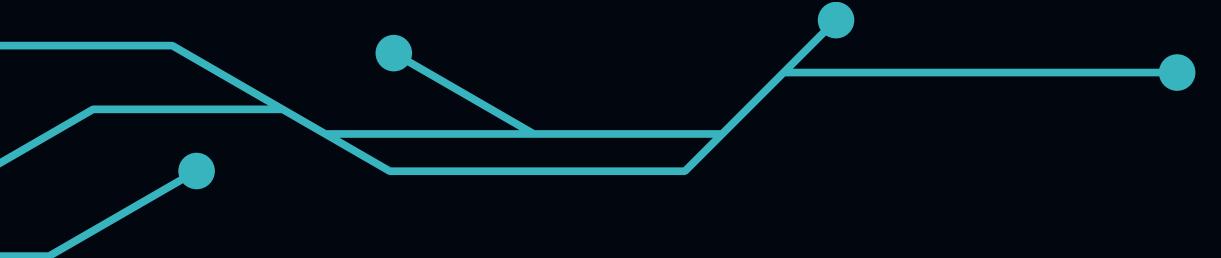
VANTAGENS

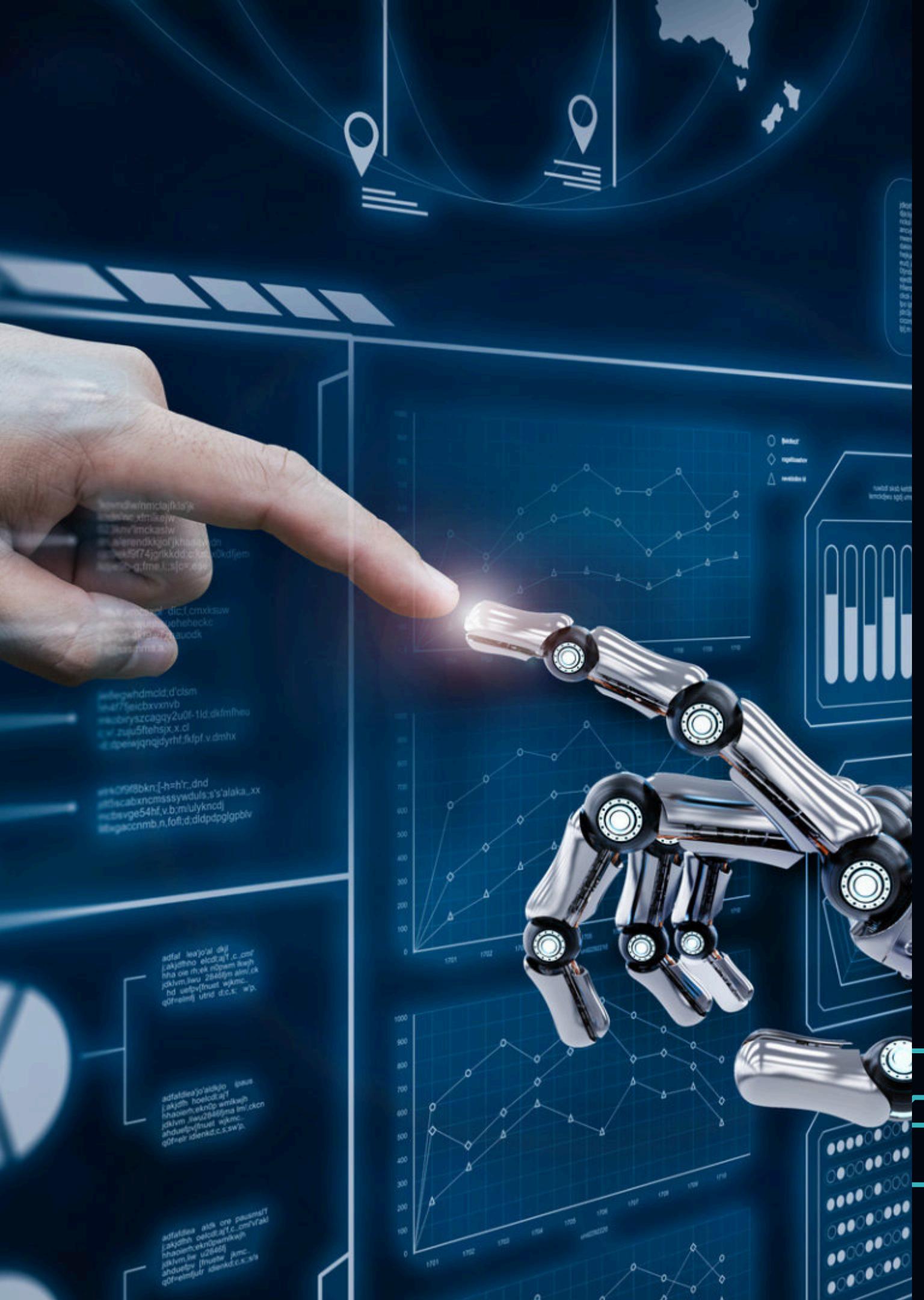
- Simples de integrar em projetos Java antigos
- Relatórios legíveis (HTML, XML padrão Cobertura)
- Boa compatibilidade com ferramentas CI
- Sem agente em runtime



DESVANTAGENS

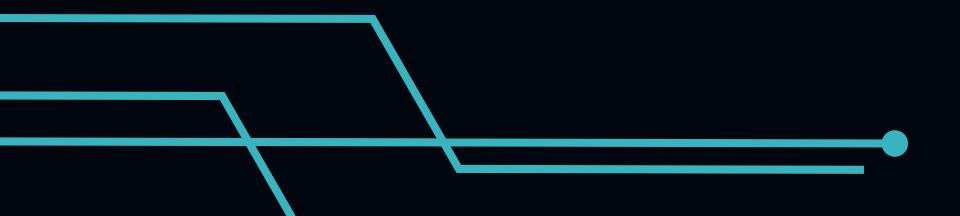
- Menos ativo em termos de manutenção
- Instrumentação offline é menos flexível
- Problemas conhecidos em builds multi-módulo
- Concorrência forte do JaCoCo





FUTURO DO COBERTURA



- Manutenção mínima e poucas funcionalidades
 - Comunidade migra gradualmente para JaCoCo
 - Ferramenta permanece útil em projetos que exigem formato Cobertura XML
 - Pode se tornar ferramenta de nicho para legados
- 

CONCLUSÃO

Quando Adotar

- Projetos legados
- Projetos Maven ou ANT
- Ambientes restritos

Quando Não Adotar

- Solução moderna em Java
- Suporte da Comunidade
- Relatórios Avançados





DEMONSTRAÇÃO

```
<build>
    <plugins>
        <!-- Plugin do Cobertura -->
        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>cobertura-maven-plugin</artifactId>
            <version>2.7</version>
            <executions>
                <execution>
                    <goals>
                        <goal>cobertura</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

```
package com.codigo;

public class Calculadora {
    public int soma(int a, int b) {
        return a + b;
    }

    public int subtrai(int a, int b) {
        return a - b;
    }

    public int multiplica(int a, int b) {
        return a * b;
    }

    public int divide(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Não pode dividir por zero!");
        }
        return a / b;
    }
}
```

```
package com.codigo;

import org.junit.Test;
import static org.junit.Assert.*;

public class CalculadoraTest {
    private Calculadora calculadora = new Calculadora();

    @Test
    public void testSoma() {
        assertEquals(expected:5, calculadora.soma(a:2, b:3));
    }

    @Test
    public void testSubtrai() {
        assertEquals(expected:1, calculadora.subtrai(a:3, b:2));
    }

    @Test
    public void testMultiplica() {
        assertEquals(expected:6, calculadora.multiplica(a:2, b:3));
    }

    @Test
    public void testDivide() {
        assertEquals(expected:2, calculadora.divide(a:6, b:3));
    }

    // @Test(expected = ArithmeticException.class)
    // public void testDividePorZero() {
    //     calculadora.divide(1, 0);
    // }
}
```

Coverage Report - com.codigo.Calculadora

Classes in this File	Line Coverage	Branch Coverage	Complexity
Calculadora	100% 	100% 	1.5

```
1 package com.codigo;
2
3 public class Calculadora {
4
5     public int soma(int a, int b) {
6         return a + b;
7     }
8
9     public int subtrai(int a, int b) {
10    return a - b;
11}
12
13    public int multiplica(int a, int b) {
14    return a * b;
15}
16
17    public int divide(int a, int b) {
18        if (b == 0) {
19            throw new ArithmeticException("Não pode dividir por zero!");
20        }
21        return a / b;
22    }
23}
```

Report generated by [Cobertura](#) 2.1.1 on 10/25 5:12 PM.

```
package com.codigo;

public class Calculadora {
    public int soma(int a, int b) {
        return a + b;
    }

    public int subtrai(int a, int b) {
        return a - b;
    }

    public int multiplica(int a, int b) {
        return a * b;
    }

    public int divide(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Não pode dividir por zero!");
        }
        return a / b;
    }
}
```

```
package com.codigo;

import org.junit.Test;
import static org.junit.Assert.*;

public class CalculadoraTest {
    private Calculadora calculadora = new Calculadora();

    @Test
    public void testSoma() {
        assertEquals(expected:5, calculadora.soma(a:2, b:3));
    }

    @Test
    public void testSubtrai() {
        assertEquals(expected:1, calculadora.subtrai(a:3, b:2));
    }

    @Test
    public void testMultiplica() {
        assertEquals(expected:6, calculadora.multiplica(a:2, b:3));
    }

    @Test
    public void testDivide() {
        assertEquals(expected:2, calculadora.divide(a:6, b:3));
    }

    // @Test(expected = ArithmeticException.class)
    // public void testDividePorZero() {
    //     calculadora.divide(1, 0);
    // }
}
```

Coverage Report - com.codigo.Calculadora

Classes in this File	Line Coverage	Branch Coverage	Complexity
Calculadora	85%  6/7	50%  1/2	1.5

```
1 package com.codigo;
2
3 4 public class Calculadora {
4
5     public int soma(int a, int b) {
6         1     return a + b;
7     }
8
9     public int subtrai(int a, int b) {
10    1     return a - b;
11 }
12
13    public int multiplica(int a, int b) {
14    1     return a * b;
15 }
16
17    public int divide(int a, int b) {
18    1     if (b == 0) {
19    0         throw new ArithmeticException("Não pode dividir por zero!");
20     }
21    1     return a / b;
22 }
23 }
```