TESTES DE *SOFTWARE* E GERÊNCIA DE CONFIGURAÇÃO

Jeanine dos Santos Barreto

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Ferramentas de teste

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Identificar as ferramentas de software.
- Descrever as principais ferramentas de teste de software.
- Aplicar o teste de software com o uso do JUnit e testes com Java.

Introdução

As ferramentas de teste servem para automatizar as atividades de teste de software, garantindo maior produtividade e mais rapidez na sua execução. Existem muitas ferramentas que podem ser utilizadas em qualquer projeto de desenvolvimento de software. Há opções com licenças pagas ou gratuitas, mas nenhuma pode ser definida como a melhor. Cabe ao gestor do projeto definir a ferramenta mais adequada, o que depende do artefato produzido.

Nesse sentido, é importante que você conheça as ferramentas mais utilizadas pelos profissionais da área de qualidade de software. Neste capítulo, você vai estudar as ferramentas de software, as principais ferramentas de teste de software e ainda a aplicação do JUnit para os testes de software com Java.

Ferramentas de software

A realização dos testes, durante o desenvolvimento de um software, é a atividade que assegura a entrega de um produto de qualidade ao usuário final. Isso ocorre principalmente quando os testes são feitos ao longo de todo o ciclo de vida do projeto, pois eles garantem o alinhamento aos requisitos do software.

O teste de software é um aspecto do projeto que exige planejamento, tempo, conhecimento técnico, recursos, infraestrutura adequada e principalmente comprometimento — tanto dos gestores do projeto quanto dos analistas que trabalharão como testadores.

Como você pode imaginar, quanto mais tarde ocorre a correção de problemas identificados nos testes, maior é o custo decorrente dessas correções, tanto

financeiros quanto de imagem perante os usuários finais. Daí a importância da realização dos testes desde o início do projeto de software (PRESSMAN; MAXIM, 2016). Apesar disso, ainda existem muitas empresas que ignoram essa realidade e não dedicam tempo e recursos suficientes para a realização de testes de software. Isso gera incerteza, insegurança e falta de confiança dos clientes devido à presença de problemas na execução das aplicações.

É comum que os fabricantes de software deixem a realização dos testes para o final do projeto, após o término da codificação. Nesse momento, pode ser que muitos problemas identificados já não possam mais ser resolvidos, pois envolvem aspectos iniciais do projeto, como é o caso de um mal-entendido relacionado aos requisitos. Além disso, mesmo que os testes sejam realizados ao longo de todo o projeto, é praticamente impossível garantir que sejam identificados todos os problemas do software. Ou seja, quanto antes os testes se iniciarem, maior será a cobertura de problemas identificados.

Fique atento

Testar um software significa fazer averiguações e questionamentos em seu código-fonte e em sua interface com base em um planejamento de testes. A ideia é analisar as respostas obtidas e, assim, verificar se estão de acordo ou não com o produto idealizado inicialmente pelos usuários.

Existem basicamente três problemas que podem ser encontrados durante a realização dos testes em um software, como você pode ver a seguir (PRESSMAN; MAXIM, 2016).

- **Erro:** normalmente é causado por algo que o ser humano fez de errado. Pode ser um trecho de código de programação incorreto, um design inadequado em uma parte da interface, entre outros exemplos.
- **Defeito:** decorre de um erro, pois quando uma codificação ou um design são elaborados de maneira incorreta, isso gera um problema de documentação, codificação ou design de interface.
- **Falha:** decorre de um defeito, ou seja, quando um defeito ocorrer, pela tentativa de execução de uma parte do código ou da interface com problema, uma falha de sistema vai acontecer e o sistema não vai funcionar corretamente, podendo até mesmo sofrer uma interrupção na sua execução.

Com usuários cada vez mais exigentes e um mercado cada vez mais competitivo, entregar softwares de qualidade para o cliente final não é mais uma preocupação de poucos, mas de todos os fabricantes. Afinal, a qualidade agora é um requisito fundamental para que o software garanta a sua sobrevivência no mercado.

Hoje, as empresas fabricantes de software mais competitivas são aquelas que buscam melhorar continuamente seus processos para garantir um aumento crescente no nível de qualidade dos produtos oferecidos. Nesse sentido, é interessante adotar ferramentas que possam prestar suporte a esse processo de melhoria contínua. Atualmente, existem ferramentas que auxiliam durante todo o ciclo de vida do projeto de desenvolvimento de software, desde o seu início até a sua implantação junto ao usuário final.

As ferramentas de software permitem maior cobertura de identificação de problemas e diminuem o esforço dos testadores e de toda a equipe, tanto no apontamento dos problemas quanto na sua correção. Essas ferramentas podem ser divididas nas categorias listadas a seguir. Elas são utilizadas ao longo do ciclo de vida de desenvolvimento do software, como você pode ver na Figura 1 (CAETANO, 2007).

- Ferramentas para gestão de defeitos
- Ferramentas para gestão de testes
- Ferramentas para gestão de projetos
- Ferramentas para automação de testes funcionais e de aceitação
- Ferramentas para automação de testes de performance
- Ferramentas para controle de versões
- Ferramentas de apoio

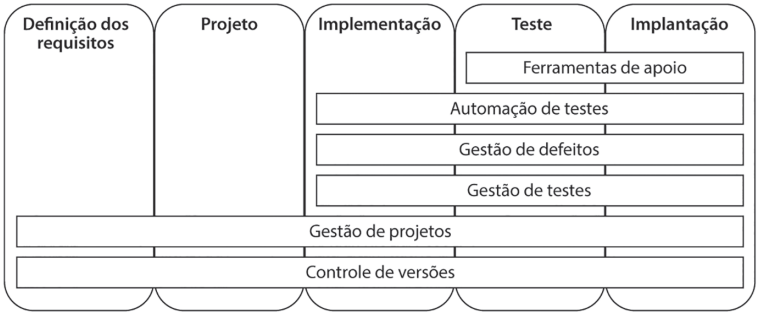

Figura 1. Utilização das ferramentas de software ao longo do ciclo de vida do produto.
Fonte: Adaptada de Caetano (c2019, documento on-line).

A utilização de ferramentas na realização das atividades de testes de software proporciona alguns benefícios muito importantes (CAETANO, 2007). Veja:

- os testes podem ser repetidos em sua integralidade, de maneira exaustiva, sem que isso torne a atividade incômoda e maçante para os testadores;
- os casos de teste podem ser armazenados e executados, e seus resultados podem ser documentados de maneira facilitada, ficando disponíveis para que os demais integrantes da equipe de projeto os utilizem em seu trabalho;
- os testes podem ser reproduzidos de maneira fiel, pois existe documentação para cada atividade realizada no âmbito da ferramenta;
- são gerados arquivos de log apontando tudo o que foi feito no decorrer das atividades de teste, com indicação dos problemas encontrados.

Assim como a maneira de desenvolver softwares se modifica ao longo do tempo, as ferramentas e tecnologias também mudam. O objetivo das mudanças é aumentar a produtividade e a qualidade e garantir a satisfação do cliente por meio da entrega de um produto confiável.

Principais ferramentas de teste de software

Existem incontáveis ferramentas de teste de software disponíveis no mercado. Várias delas são pagas e outras tantas são *open source*. Não existe uma ferramenta que possa ser indicada como a melhor de todas, mas é possível destacar as principais delas, por serem as mais utilizadas. Cabe aos analistas responsáveis pelo projeto decidir que ferramenta se adapta melhor a cada caso.

Saiba mais

Open source (código aberto) é um modelo de desenvolvimento que libera o licenciamento para o design ou a codificação de um produto de software, permitindo a sua distribuição universal e ainda a consulta, a análise e a modificação do produto. Para isso, não é necessário pagar uma licença, o que torna esse tipo de produto uma produção intelectual colaborativa.

Não existe uma ferramenta ideal para ser aplicada em todos os casos. Contudo, uma ferramenta automatizada de teste de software precisa envolver necessariamente os seguintes aspectos para ser considerada um instrumento de auxílio na realização das atividades de teste (CAETANO, 2007):

- cada uma das menores partes do software que precisa ser testada deve ser executada de maneira isolada e independente de todas as outras unidades testadas;
- todos os erros, defeitos e falhas identificados devem ser documentados;
- a definição de cada unidade de teste, ou menor parte do software que precisa ser testada, deve ser definida de maneira simples e não complexa.

TestLink

O TestLink é uma ferramenta automatizada de testes de software *open source* escrita na linguagem de programação PHP. O principal objetivo dessa ferramenta é dar suporte às atividades realizadas para a gestão de testes. Como padrão, o TestLink é instalado na língua inglesa, mas é possível alterar essa configuração para o idioma português.

Por meio do TestLink, você pode elaborar planos de testes e gerar relatórios com diversas informações para acompanhar a realização das atividades de testes. Essa ferramenta também possibilita registrar e organizar todos os requisitos levantados junto ao usuário no início do projeto e, posteriormente, associar cada caso de teste criado ao requisito correspondente.

Link

Para fazer o download e saber mais sobre a ferramenta de testes TestLink, acesse o link a seguir.

https://goo.gl/1WBFFE

As principais funcionalidades oferecidas pela ferramenta TestLink são (CAETANO, 2007):

- controle de acesso e níveis de permissão, dependendo do perfil do usuário (por exemplo, testador ou gerente de projeto);

- organização dos casos de teste em pacotes hierárquicos que podem ser organizados por prioridade;
- classificação dos casos de teste por palavras-chave, a fim de facilitar a pesquisa e organizar ainda mais a disposição dos casos de teste;
- criação ilimitada de projetos e de casos de teste dentro de cada projeto, ou seja, não há limite de quantidade para nenhum deles;
- atribuição dos planos e casos de teste a testadores específicos;
- geração de relatórios e gráficos com informações sobre os testes que podem ser exportadas para diversas extensões, como CSV, PDF, Microsoft Excel e Microsoft Word;
- possibilidade de integração com outras ferramentas automatizadas de gestão de testes e de defeitos, como o Mantis.

Selenium

A ferramenta de testes de software chamada Selenium surgiu nos anos 2000, evoluiu ao longo dos anos e atualmente é uma das mais conhecidas ferramentas de automação de teste *open source* para aplicativos web (CAETANO, 2007).

Link

Para fazer o download e saber mais sobre a ferramenta de testes Selenium, acesse o link a seguir.

https://goo.gl/52Qs4M

O Selenium é multiplataforma, ou seja, funciona em diversos sistemas operacionais, como Windows, Mac e Linux, e ainda em diversos navegadores web, como Chrome, Internet Explorer e Firefox. Além disso, os seus *scripts* de teste podem ser escritos em diversas linguagens de programação, como Python, Ruby, Perl e Java.

Apesar de o Selenium poder ser utilizado em diversos ambientes de programação diferentes, além de suportar casos de teste complexos que atendem a altos níveis de complexidade de sistemas, é necessário ter muito conhecimento técnico para trabalhar com ele. É preciso ter habilidades avançadas

de programação. Além disso, é requerido bastante esforço para a criação de estruturas automatizadas e bibliotecas que atendam às necessidades de teste, por mais simples que sejam.

IBM Rational Functional Tester

A ferramenta de teste chamada IBM Rational Functional Tester é paga e serve para a realização de testes funcionais e testes de regressão. Os *scripts* dos casos de teste são escritos em Visual Basic, .NET e Java (CAETANO, 2007).

Link

Para fazer o download e saber mais sobre a ferramenta de testes IBM Rational Functional Tester, acesse o link a seguir.

https://goo.gl/oEAWD1

O IBM Rational possui um recurso inédito entre as ferramentas de teste: o teste do *storyboard*. Nele, as ações dos usuários e testadores são registradas e podem ser visualizadas posteriormente no formato de *storyboard*, feito por meio de capturas da tela da ferramenta.

TestComplete

O TestComplete é uma ferramenta de testes paga considerada muito poderosa para a realização de testes em aplicações web, dispositivos móveis e desktops. Essa ferramenta suporta a elaboração de *scripts* de casos de teste em JavaScript, VBScript, C++ e Python (CAETANO, 2007).

Link

Para fazer o download e saber mais sobre a ferramenta de testes TestComplete, acesse o link a seguir.

https://goo.gl/UBu3QL

A ferramenta TestComplete possibilita que os testadores realizem testes orientados por palavras-chave e dados. Além disso, oferece um recurso que permite a gravação dos testes realizados e a sua reprodução, de maneira simples e fácil de usar.

Teste de software com JUnit e testes com JAVA

Na área da engenharia de software, uma das atividades mais importantes é o teste de software. Ele precisa ser feito ao longo de todo o ciclo de vida do projeto de desenvolvimento do software. Isso se torna mais evidente quando a equipe de projeto trabalha com *extreme programming*, que tem uma visão voltada principalmente para os testes (CAETANO, 2007).

Saiba mais

A eXtreme Programming (XP), que em português significa "programação extrema", é uma metodologia ágil de desenvolvimento de software. Ela é aplicada principalmente quando a equipe desenvolvedora do sistema não conta com requisitos muito precisos, ou ainda quando os requisitos sofrem mudanças constantes, como no caso de softwares financeiros.

Na metodologia XP, a codificação e os testes são realizados e acompanhados de maneira constante e contínua, para que os ajustes sejam feitos ao longo do projeto de desenvolvimento. Na metodologia ágil, ou no desenvolvimento ágil de software, o produto de software é fabricado aos poucos, em pedaços e períodos predefinidos, na tentativa de diminuir o máximo possível os riscos com o desenvolvimento.

Quando o produto de software precisa ser desenvolvido na linguagem de programação Java, os testes de software podem ser implementados por meio do JUnit. Atualmente, ele é um dos *frameworks* mais conhecidos para a realização de testes em Java.

O JUnit é um *framework* que pertence a uma família de ferramentas de testes de software chamada xUnit. Essas ferramentas são aplicadas em produtos de software desenvolvidos em C++, SmallTalk, Java e outras linguagens de programação. Apesar de o JUnit ser bem simples de se utilizar, ele traz um ambiente completo para a realização dos testes unitários e dos testes de regressão em códigos escritos na linguagem Java, facilitando o trabalho do desenvolvedor.

Os testes unitários, que são aqueles realizados por meio do JUnit, asseguram que cada método projetado trabalhe da maneira como se espera, averiguando as menores partes do código desenvolvido, que, no caso da metodologia ágil, são os métodos.

Exemplo

Para entender melhor o que significam essas menores partes do código, basta você imaginar um sistema de uma escola em que existe uma classe chamada Aluno, com os métodos calcularMedia() e calcularMensalidade(). Para averiguar se existe algum erro que envolva esses métodos, é necessário testar cada um deles a fim de entender de onde o problema está vindo. Para a realização desses testes unitários, uma boa saída são os testes automatizados, que aumentam a produtividade do desenvolvedor pois diminuem o tempo gasto com testes.

Essa ferramenta é dividida em duas partes: uma com métodos e anotações que servem para a elaboração dos casos de teste; e outra que serve para realizar a execução dos casos de teste criados pela primeira parte (CAETANO, 2007).

A execução dos testes, por meio do JUnit, é feita de forma totalmente automatizada e sem que seja necessária a intervenção humana. Além disso, a apresentação do resultado dos testes realizados pelos testadores é feita por meio de algo semelhante a um semáforo: a cor verde significa que o teste foi executado da maneira como estava escrito no caso de teste; a cor azul significa que o teste apresentou algum erro de validação; e a cor vermelha significa que houve algum erro de exceção na escrita do código em Java.

Para trabalhar com o JUnit, você deve tomar algumas precauções durante o planejamento dos testes (BITTENCOURT; TESSMANN; SCHIRMER, 2019):

- definir um pacote em que fiquem armazenadas todos os casos de teste;
- levar em consideração que cada caso de teste é responsável pelos testes de uma classe do sistema;
- criar vários procedimentos de teste, de modo que cada um deles realize pequenos testes.

Tomando essas medidas, a cobertura e a identificação dos testes serão muito mais abrangentes. Caso sejam encontrados problemas ou erros, eles poderão ser identificados de maneira muito mais rápida e simples ao longo dos casos de teste.

Fique atento

Para começar a trabalhar com o JUnit, você precisa fazer o download do *framework*, que está disponível de maneira gratuita no link a seguir. É comum também que os ambientes de desenvolvimento integrado (*Integrated Development Environments* — IDEs) mais utilizados pelos desenvolvedores já tragam o JUnit embutido em seus pacotes (BITTENCOURT; TESSMANN; SCHIRMER, 2019).

https://junit.org

Na versão atual, o JUnit se divide em sete classes e interfaces, que são Assert, TestResult, Test, TestListener, TestCase, TestSuite e BaseTestRunner. As principais são a TestCase, a TestSuite e a BaseTestRunner.

Para testar o código, o JUnit oferece o método *assert*, que é muito fácil de entender. Um método de asserção, ou *assert*, recebe o valor que é o esperado como resultado para o teste, ou seja, o valor correto, e um outro que é o resultado obtido pelo teste. A comparação entre eles é realizada e o teste falha se os valores forem diferentes, mas é realizado com sucesso se eles forem iguais.

Conhecendo esse conceito de *assert*, é possível realizar o teste em todo o código desenvolvido. É preciso apenas conhecer os requisitos do sistema (a ponto de saber quais valores devem ser retornados em cada teste) e os resultados obtidos com os testes (para tecer as comparações).

Uma boa maneira de escrever um teste unitário eficiente é colocar asserções para os valores realmente críticos ou relevantes para o sistema. Ou seja, talvez não seja necessário testar valores intermediários das classes, apenas resultados finais. Porém, outras vezes talvez os resultados intermediários interfiram diretamente no resultado final e seja interessante inserir asserções para eles também.

Ao adicionar casos de teste no JUnit, você deve ter em mente que um valor pode corresponder a qualquer objeto Java. Por isso, é fundamental implementar

o *equals* e o *hashcode* dos objetos de resposta que precisarão ser testados com o JUnit (BITTENCOURT; TESSMANN; SCHIRMER, 2019).

Como exemplo de implementação de método de teste no JUnit, considere o seguinte:

```
@Test
public void metodoTeste() {

}
```

Como o método criado é um *void*, ele não vai receber nenhum argumento, ou seja, será uma classe de teste mesmo. A partir disso, é preciso inserir asserções para que se possa compreender o funcionamento do método:

```
Assert. assertEquals("2 dividido por 2 precisa resultar em 1", 1, 2/2);
```

Nesse código, o primeiro argumento é a mensagem que vai aparecer caso o teste falhe. Para visualizar a mensagem de erro, basta alterar o último argumento para que traga um resultado incorreto. Essa é a maneira mais fácil e básica de entender como os testes são realizados por meio do JUnit.

Caso o teste precise envolver elementos mais complexos, o *script* do caso de teste pode se parecer com o que é mostrado na Figura 2, a seguir.

```
1   private String stringLocal;
2
3   @Before
4   public void inicializaLocal() {
5       System.out.println("Inicializou.");
6       this.stringLocal = "inicializada";
7   }
8
9   @Test
10  public void testInicializadaLocal1() {
11      Assert.assertEquals("A String deveria estar inicializada.","inicializada",
12              this.stringLocal);
13  }
14
15  @Test
16  public void testInicializadaLocal2() {
17      Assert.assertEquals("A String deveria estar inicializada.","inicializada",
18              this.stringLocal);
19  }
20
21  @After
22  public void setaNull() {
23      System.out.println("Setou nulo.");
24      this.stringLocal = null;
25  }
```

Figura 2. *Script* mais complexo para testes com JUnit.

Fonte: Bittencourt, Tessmann e Schirmer (2019, p. 10).

É importante você observar que a *string* é inicializada e recebe valor *null* por três vezes, porque existem três métodos de teste. Os métodos @Before são executados antes dos métodos de teste, e o @After é executado depois deles. As anotações @Before e @After, da maneira como foram utilizadas, permitem criar cenários de teste sempre zerados e corretamente inicializados a cada ciclo de teste (BITTENCOURT; TESSMANN; SCHIRMER, 2019).

Referências

BITTENCOURT, G.; TESSMANN, P. V.; SCHIRMER, T. *Teste de software:* uma introdução ao uso do JUnit em testes unitários. [2019]. Disponível em: https://www.academia.edu/17303382/Teste_de_Software_Uma_introdu%C3%A7%C3%A3o_ao_uso_do_Junit_em_testes_unit%C3%A1rios?auto=download. Acesso em: 09 mar. 2019.

CAETANO, C. *Automação e gerenciamento de testes:* aumentando a produtividade com as principais soluções open source e gratuitas. São Paulo: Novatec, 2007.

CAETANO, C. *Automação e gerenciamento de testes:* aumentando a produtividade com as principais soluções open source e gratuitas (2a edição). c2019. Disponível em: http://www.linhadecodigo.com.br/artigo/1566/automacao-e-gerenciamento-de-testes-aumentando-a-produtividade-com-as-principais-solucoes-open-source-e-gratuitas-2a-edicao.aspx. Acesso em: 09 mar. 2019.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software:* uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2011.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS