# Qualidade e Automação de Testes

## Formas e Qualidade de Testes

A função de um bom teste é evitar que um software tenha muitos bugs (erros). Para tanto, a realização dos testes de softwares deve seguir quatro etapas, as quais funcionam como uma espiral, partindo do menor teste para o maior.

- **Etapa 01 – Teste Unitário:** Feito na menor unidade de um código. Neste momento, os desenvolvedores testam os códigos de forma unitária à medida que escrevem as instruções. É claro que é inviável testar o código a cada linha de programa desenvolvida, mas deve-se fazer isso, pelo menos, a cada sequência de comando criada e nas fronteiras destas sequências.
- **Etapa 02 – Teste de Integração:** Verifica unidades diferentes. Nesse caso, pode ser definida uma rotina de testagem a cada incorporação de uma nova unidade ou a cada número fixo de unidades incorporadas, dependendo do tamanho do código. Este teste garante que as unidades funcionem de forma coordenada e em conjunto.
- **Etapa 03 – Teste de Sistema:** Testa variados elementos, bem como o código. Além de testar o código como um todo e sua funcionalidade, também testa aspectos como segurança, performance e robustez. Normalmente, ele é seguido de um teste de aceitação do programa pelo cliente.
- **Etapa 04 – Teste de Manutenção:** Avalia o software a cada atualização. É feito para verificar se o programa está funcionando corretamente, se ainda é seguro contra o ataque de hackers e se as novas funcionalidades (atualizações) foram incorporadas de maneira correta.

## Quiz – Automação de Testes

Exercite os seus conhecimentos sobre testes de softwares, escolhendo a alternativa que preenche corretamente a lacuna de cada uma das sentencias apresentadas a seguir:

**1.** Os testes de _____ são realizados sem a execução da aplicação ou do software.
- **a) Verificação.**
- **b)** Validação.
- **c)** Performance.

**2.** Nos testes _____, o desenvolvedor utiliza diretamente o código da solução.
- **a)** Caixa cinza.
- **b)** Caixa preta.
- **c) Caixa branca.**

**3.** Os testes _____ aplicam-se a uma versão de software liberada para o usuário final.
- **a)** Alpha.
- **b)** Beta.
- **c) Gama.**

**4.** O _____ é aplicado sempre que ocorre uma mudança em partes do código.
- **a) Teste de regressão.**
- **b)** TDD.
- **c)** Teste mocado.

5. A técnica chamada _____ objetiva evitar casos de testes mal definidos ou mal construídos.
   **a)** Code smells.
   **b)** **Testes smells.**
   **c)** Cobertura de testes.

Por meio do menu a seguir, reforce e amplie os seus conhecimentos sobre os principais conceitos estudados:

1. **Metodologias Ágeis:** Basicamente, as metodologias ágeis dividem um grande projeto em projetos menores, em curtos períodos, com mais iterações e, consequentemente, riscos e custos reduzidos, devido à possibilidade de identificar e corrigir problemas mais rapidamente, em virtude do maior número de iterações do que uma gestão de projeto tradicional. Os resultados práticos observados em empresas que implementaram as metodologias ágeis são: Menores custos, melhoria da qualidade, cumprimento dos prazos de entrega e confiabilidade.
2. **Ferramentas CASE:** As ferramentas CASE (Computer Aided Software Engineering) são "um conjunto de técnicas e ferramentas informatizadas que auxiliam o engenheiro de software no desenvolvimento de aplicações" (Ramos, 2011). De acordo com Peloso (2014), tais ferramentas podem ser divididas em três categorias:

   - **Lower CASE (Back-End):** As ferramentas da categoria Lower CASE trabalham em ambientes mais simples, auxiliando na criação dos códigos dos softwares, nos seus testes, na depuração e na manutenção deles.
   - **Upper CASE (Front-End):** As ferramentas da categoria Upper CASE trabalham em ambientes mais complexos. As tarefas de análise, de projetos e de geração de código são mais automatizadas do que na categoria Lower CASE.
   - **Integrated CASE (Lower + Upper):** As ferramentas da categoria Integrated CASE trabalham integradas em ambientes que relacionem entradas e saídas. Isso permite o controle dos dados de forma consistente.

3. **Definição da Linguagem:** Existem diversas linguagens de programação com diferentes abordagens para o desenvolvimento de um software. Nesse contexto, o recomendado é que a definição dessa linguagem observe algumas regras:

   - Utilizar a linguagem de programação determinada pelo cliente.
   - Se o cliente não especificar a linguagem, utilizar a que a equipe tenha mais "fluência".
   - Escolher o nome das variáveis de acordo com a sua função.
   - Elaborar documentação detalhada e clara.

   Além disso, ao se escolher uma linguagem de programação, devem-se considerar algumas métricas:

   - Quantidade de linhas do código.
   - Tempo de execução do algoritmo.
   - Eficiência do software nas diversas linguagens.

4. **Escopo e Requisitos:** Hoje, com o aumento visível das demandas e da complexidade de um programa, bem como da concorrência, temos a necessidade da definição de escopos, os quais levam à produção de uma lista de requisitos que englobam as necessidades dos

clientes. Esses requisitos, por sua vez, trazem os atributos relacionados às funcionalidades (Sbrocco, Macedo, 2012). Dentre os requisitos, destacam-se:

- **Prioridade:** Ordena as funcionalidades de acordo com sua relevância para o projeto. Ela é dividida em:
  - **Prioridade Crítica:** Cuja funcionalidade deve ser inserida na próxima atualização ou versão do software.
  - **Prioridade Importante:** Cuja funcionalidade possui grau de importância, mas pode ter sua implementação adiada no caso de problemas técnicos.
  - **Prioridade Útil:** Cuja não implementação em uma próxima versão pode ser adiada e não trará prejuízos ao funcionamento do software.
- **Complexidade:** Pode ser dividida em baixa, média e alta, e está relacionada com a dificuldade de implementação de uma ou mais funcionalidades solicitadas pelo software.
- **Risco:** É um requisito que auxilia na definição das prioridades dentro do escopo. A partir da análise de riscos, são identificadas as funcionalidades críticas e que precisam de mais atenção. Geralmente, as funcionalidades mais críticas são colocadas primeiro dentro do cronograma do projeto.
- **Esforço-Tempo:** É um requisito de fundamental importância para a definição do orçamento do software. É medido a partir do número de integrantes do projeto versus tempo, por isso, é chamado esforço-tempo. As variáveis que devem ser analisadas são: O tamanho do projeto, o tempo total para execução do projeto, o número de pessoas no projeto e as ferramentas utilizadas.

5. **Hackers:** A definição dada aos hackers – pessoas com alto conhecimento de informática e, mais especificamente, de redes e programação (Ulbrich, Della Vale, 2004) – contrasta com muitos hackers que escolheram o caminho criminoso, os chamados crackers. No entanto, o conhecimento profundo de segurança da informação permite aos hackers do bem serem peças fundamentais para evitar, descobrir falhas de segurança e implementar maiores camadas de segurança em redes, sistemas, estruturas e softwares. De acordo com Rocha (2016), podemos elencar os seguintes tipos de ataques de hackers:
   - **BotNet:** Vírus que infectam computadores sem os usuários perceberem. As vítimas podem ser desde usuários finais a grandes empresas, bancos e órgãos governamentais.
   - **DdoS/Dos (Distributed Deny of Service Attack):** Ataque de negação de serviço, através de muitos acessos simultâneos que sobrecarregam servidores. Esse tipo de ataque cria um mercado de botnets.
   - **BruteForce:** Trata-se de um script de internet para gerar senhas e acessar sistemas.
   - **Phishing:** São os ataques mais comuns, nos quais, através de e-mails e mensagens falsas, são solicitados dados sensíveis, atualizações cadastrais, entre outras fraudes, o que permite ao criminoso acessar esses dados e contas bancárias.
   - **Engenharia Social:** Envolve, além do phishing, a utilização de ferramentas de neurolinguística, convencimento e outros acessos “físicos” a dados sensíveis de empresas, como listas e informações impressas, além de criminosos se passando por funcionários.

## Métodos Ágeis: BDD e TDD

A seguir, saiba diferenciar duas práticas de desenvolvimento ágil:

- **Métodos Ágeis:** Visam ao aumento da performance nas organizações. Portanto, trata-se de um método para padronizar ações nas organizações, independentemente do tipo de mercado ou do seu potencial produtivo. Quanto mais rápido e com menor custo, melhor. Além disso, esses métodos buscam a excelência, ou seja, a qualidade dentro dos processos.
- **BDD (Behavior Driven Development, ou Desenvolvimento Guiado por Comportamento):** Para Soares (2011), essa prática pode ser definida como a junção de diversas práticas ágeis na formulação de softwares, com foco na linguagem e nas conexões usadas no período do processo de desenvolvimento, permitindo uma melhor comunicação entre o desenvolvedor e a equipe, uma vez que se utiliza uma linguagem de visão unificada e compartilhada entre a equipe envolvida no desenvolvimento do projeto.

    - **Benefícios:** Podemos citar as seguintes vantagens do BDD:

        - **Melhor Comunicação:** Entre os times de desenvolvimento.
        - **Divisão de Conhecimento:** Compartilhando informações e criando sinergia dentro de uma equipe multifuncional.
        - **Documentação Dinâmica:** Uma vez que os frameworks de BDD possibilitam a geração de documentação de maneira mais prática, evitando esforços secundários.
        - **Visão Holística:** Sugerindo que tanto os analisas como os testadores escrevam uma base, um cenário, antecipadamente ao desenvolvimento dos códigos em si, tornando possível que eles tenham um panorama geral das características solicitadas para o projeto (North, 2006).

    - **Processo:** Por meio do BDD, é possível extrair as características do software solicitadas pelo cliente durante o levantamento dos requisitos do projeto. Essa prática se baseia no uso de cenários utilizados para registrar os comportamentos que são esperados em determinado software, fazendo com que seja possível automatizar esses cenários para dar validade a algum comportamento. É importante observar que existem três critérios para a aceitação dos cenários, são eles: Given (dado), definição dos requisitos, When (quando), ocorrência de um fenômeno, e Then (então), verificação do resultado.

- **TDD (Test Driven Developmente, ou Desenvolvimento Guiado por Testes):** Essa metodologia, utilizada por equipes de projetos ágeis, baseia-se nos testes. A escrita do teste é realizada antes mesmo da própria escrita do código. Isso faz com os que desenvolvedores se assegurem de que uma parte considerável do seu sistema possua um teste para validar sua operação (Aniche, 2014).

    - **Benefícios:** Beck (2003) explica que o TDD deixa os projetos de software mais limpos, à medida que ele direciona o projeto indicando os problemas na hora certa, e isso reduz a quantidade de falhas no projeto. O autor ainda afirma que esse método encoraja a simplificação dos códigos e que códigos simples inspiram maior confiança.
    - **Processo:** A técnica do TDD se concentra em um ciclo curto de ações, composto por três fases:

        - **Red (Vermelho):** Escrever um teste simples que falhe e que não complique o início.
        - **Green (Verde):** Fazer o teste funcionar de forma rápida, entretanto escrevendo o suficiente para que isso ocorra.
        - **Refactor (Refatorar):** Excluir os códigos duplicados, criados apenas para fazer o teste funcionar.