UNIVERSIDADE ESTADUAL DA PARAÍBA
CENTRO DE CIÊNCIAS E TECNOLOGIA
CIÊNCIA DA COMPUTAÇÃO

LUCAS DE LUCENA SIQUEIRA

Testes de Software

CAMPINA GRANDE
2022

SUMÁRIO

1. Resumo

Serão abordados os principais tópicos referentes aos testes de software, assim como os pré requisitos necessários para um bom entendimento do mesmo, dos vário tipos e metodologias de testes existentes e adotados no mundo corporativo, serão abordadas as conceituações, assim como às exemplificações do funcionamento dos Testes de Caixa-Preta, Testes de Caixa-Branca, Testes de Integração, Testes de Segurança, incluindo a prática de *Bug Bounty*, Testes Automatizados de Unidade, Testes Automatizados de Interface Gráfica, Testes Automatizados de Serviços WEB. Será abordado também, um tópico que irá tratar da atribuição da responsabilidade dos testes no contexto de metodologias ágeis. Ao final, será possível ter uma visão básica de como funciona a aplicações, assim como a importância dos testes no desenvolvimento de software.

2. Introdução

Os testes de software são referentes a um conjunto de atividades que tem por objetivo validar e garantir a qualidade de uma aplicação que esteja sendo desenvolvida ou mantida. Esse tipo de prática está presente em diversas etapas do desenvolvimento de um software, a depender da metodologia de desenvolvimento, pode chegar a estar em todas, ou pelo menos na maioria delas.

Sendo um conjunto de procedimentos que tem por objetivo encontrar *bugs*, e diversos tipos de erros, podendo ser eles funcionais ou não funcionais, como uma funcionalidade chave da aplicação ou o suporte a vários acessos simultâneos por exemplo.

A equipe que estiver responsável pelos testes tem por objetivo garantir que a maior quantidade de cenários possíveis tenham sidos postos em análise, proporcionando por consequência, um produto final bem elaborado e com a menor quantidade de riscos possível.

Segundo a *ISTQB* (*International Software Qualifications Board*), que é um selo de qualidade internacional fundado em 2002, sendo ele direcionado aos testes de software, as validações que os testes propõem são de suma importância pelas seguintes justificativas:

- Garantir a boa integração de todos os componentes;
- Garantir a adesão de todos os requisitos para sua funcionalidade;
- Redução com custos de manutenção corretiva.

Dentre os diversos tipos e as diversas dinâmicas de teste, podem se destacar os testes de unidade, que validam um trecho de código, os testes de integração, que valida se os incrementos de código estão sendo realizados da melhor maneira, de forma com que entregue um bom funcionamento em conjunto, os testes de caixa preta e caixa branca, que põem em jogo o uso da aplicação pelos *testers* com ou sem acesso ao código desenvolvido, testes funcionais e não funcionais, testes de segurança, que devem garantir a segurança de toda a aplicação e seus respectivos dados, e por fim, os testes de interface, que visam garantir uma boa navegabilidade ao usuário final, porém esses não são os

únicos tipos de teste utilizados, já que com o passar dos dias novas tecnologias acabam disponibilizando um novo cenário para as equipes de desenvolvimento, como é o caso da advenção da automação de testes por exemplo.

3. SDLC (Software Development Life Cycle)

Em primeira instância é necessário descrever o *SDLC* (Software Development Life Cycle), que se refere ao ciclo de vida do desenvolvimento de um software, havendo por sua vez, diversas fases que serão mediadas e aprimoradas pelo próprio *SDLC*. O desenvolvimento de um software pautado pelo *SDLC* é dividido em 6 etapas, sendo elas as etapas de Planejamento, Análise de Requisitos, Design e Arquitetura do produto, Desenvolvimento, Testagem e por fim, a Entrega conjunta com a Manutenção do produto.

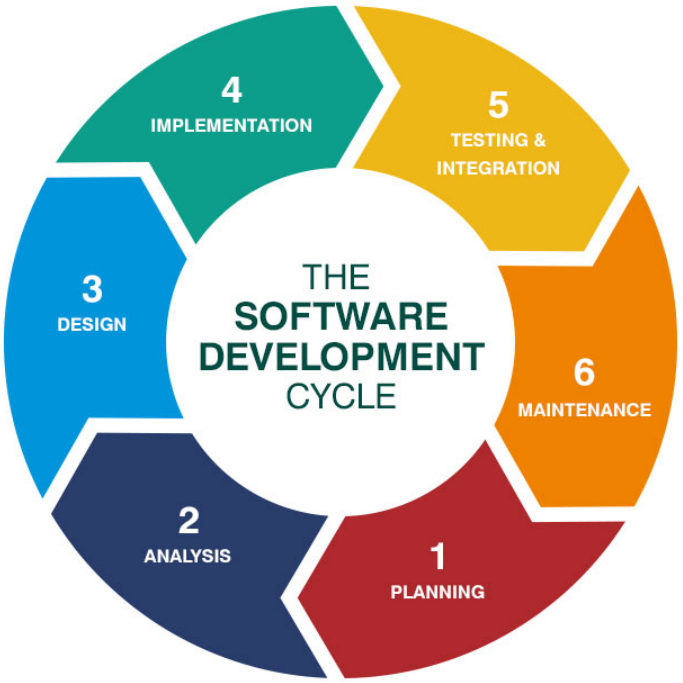

Figura 1 - The Software Development Cycle

Logo, se torna indispensável a etapa de testes, sendo válido ressaltar sua importância no processo de desenvolvimento, já que eles contribuem para o desenvolvimento de um Produto Correto e um Produto que está sendo desenvolvido Corretamente. Apesar dos conceitos de “Desenvolver o Produto Correto” e “Desenvolver um Produto Corretamente” serem semelhantes, são análogos aos conceitos de “Verificação” e “Validação”, sendo que o primeiro termo se refere a cumprir o propósito do software em questão, estando dentro do escopo e cedendo as funcionalidades definidas, já o segundo se refere a produzir um software da maneira correta, seguindo corretamente as convenções estabelecidas anteriormente e todas as boas práticas.

4. Tipos de Testes

Existem várias metodologias que podem ser abordadas em um ciclo de desenvolvimento de software, cada uma delas tem uma peculiaridade que visa focar em algum quesito mais específico e pontual. Quando um software chega em alguma fase de teste, quesitos funcionais e não funcionais devem ser bem testados e analisados, a fim de trazer um ótimo produto.

Com a ajuda de novas tecnologias e frameworks as rotinas de teste tem se tornado cada vez mais abrangentes, principalmente com a advenção dos testes automatizados, acrescentando novas abordagens de testes, como os “Testes de Unidade Automatizados”, “Testes de *Web Services*”, “Testes de *GUI*”, que serão abordados posteriormente.

A fim de exemplificação para todos os demais tópicos, será tomado como referência um software que tem como propósito o gerenciamento de uma assistência técnica a computadores. As funcionalidades que serão levadas em consideração dizem respeito à criação, edição, pesquisa e exclusão de ordens de serviço (e.g. troca de tela de notebook acer nitro 5), dos clientes da assistência e funcionários da mesma.

4.1. Testes Manuais

Nesse tipo de testes, as baterias de teste são realizadas manualmente, ou seja, verificando se algumas entradas de dados geram as saídas a partir de uma iniciativa manual (e.g. via terminal).

4.1.1. Testes Funcionais

São os testes referentes às funcionalidades do software, ou seja, que são direcionados à validação do código que foi desenvolvido e consequentemente às suas respectivas funções e propósitos. De uma forma geral, os testes funcionais auxiliam na identificação de funções incorretas, possíveis erros de desempenho, interface, acessos, inicializações ou términos por exemplo, entregando um produto livre de *bugs* e com uma alta qualidade.

Com a disseminação de ferramentas como o *Selenium* e *JUnit*, por exemplo, tornou-se possível realizar diversos tipos de testes, que cumpririam o propósito de testar recursos de uma página web, de execução de funcionalidades chaves de um sistema ou até de aceitação do usuário.

4.1.1.1. Testes de Caixa-Preta

Nos testes de caixa-preta, a equipe responsável por realizar os testes não têm acesso algum ao código que foi desenvolvido, sendo um teste com completa imparcialidade quanto ao método de desenvolvimento, já que os *testers* não terão preocupação alguma quanto ao código propriamente dito, mas sim com o seu funcionamento.

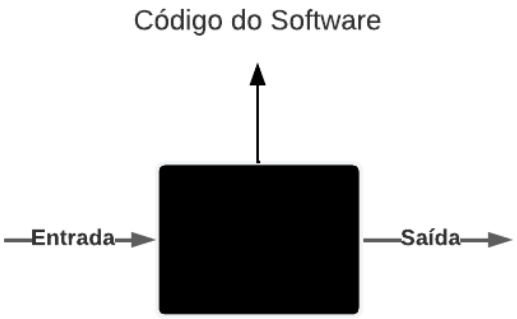

Figura 2 - Abstração do Teste de Caixa-Preta

4.1.1.2. Testes de Caixa-Branca

Nos testes de caixa-branca, os *testers* terão outra preocupação além de validar e verificar o que foi desenvolvido, que seria avaliar também, o comportamento interno dos componentes de software que estão sendo testados. Diferentemente dos testes de caixa-preta, nesse tipo de testagem, os desenvolvedores terão acesso ao código desenvolvido, prestando atenção em aspectos mais lógicos e de mais baixo nível, como no fluxo de dados, no ciclo dos dados e na lógica de programação utilizada. Um framework amplamente utilizado nesse contexto é o JUnit, disponível para a linguagem Java com suporte à criação de testes automatizados.

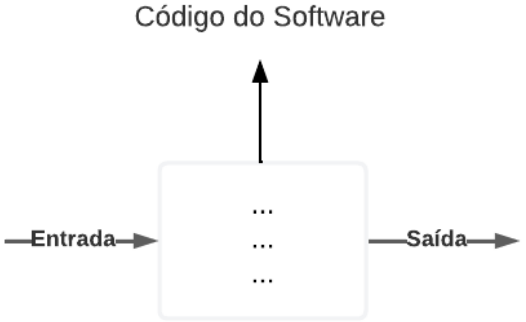

Figura 3 - Abstração do Teste de Caixa-Branca

4.1.1.3. Testes de Integração

Os teste de integração são referentes ao acréscimo de uma nova unidade no projeto como um todo, ou seja, uma pequena parte que será incrementada ao projeto é testada de forma isolada (teste de unidade) e só após é integrada no sistema que possa estar sendo desenvolvido. Tendo por objetivo encontrar falhas na integração de novas unidades, essa prática irá garantir que toda a aplicação continuará funcionando corretamente, com todas as

funcionalidades que já estavam presentes sem nenhuma alteração referente ao seu funcionamento e a nova funcionalidade inserida funcionando e interagindo corretamente com o restante da aplicação.

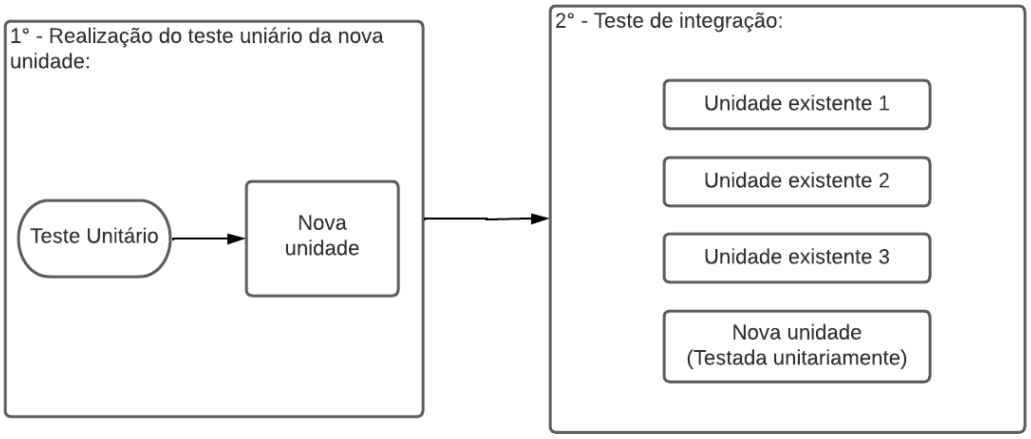

Figura 4 - Abstração de Teste de Integração

4.1.2. Testes Não Funcionais

São mais referentes às restrições do software, ou seja, que tem como propósito verificar aspectos como escalabilidade, desempenho e performance.

4.1.2.1. Testes de Segurança

No contexto de segurança da informação temos que um software pode conter diversas informações sigilosas que devem ser bem resguardadas, e já que sistemas e aplicações com esse tipo de informações são grandes alvos para acesso ilegal ou não autorizado, o que pode gerar um grande prejuízo. Para contornar esse tipo de situação, no processo de desenvolvimento de software, a equipe de desenvolvimento deve ter uma preocupação especial para garantir uma boa segurança no software desenvolvido.

Adjuntamente aos desenvolvedores, a equipe de testes deve garantir que esses mecanismos de segurança estão funcionando corretamente e que são suficientes para o projeto e os seus respectivos requisitos.

No processo de procura por vulnerabilidades realizadas pela equipe de *testers*, devem ser postos em prática a maior quantidade de cenários possíveis, observando os servidores, o ambiente dos administradores, usuários finais, os canais de comunicação.

Dentre as alternativas atuais que tem por objetivo aplicar uma camada de segurança estão os firewalls e as criptografias, por exemplo, que devem ser bem estruturados e testados.

Vale ressaltar uma diferença entre as violações de segurança e de privacidade, tal ela, que as violações de segurança se referem ao caso de alguém realizar alguma ação que está acima das permissões concedidas para ela, já as violações de privacidade dizem respeito à divulgação não autorizada de algum dado sensível, como dados bancários cadastrados por exemplo.

4.1.2.1.1. *Bug Bounty*

O *Bug Bounty* é um tipo de programa em que empresas se cadastram com o propósito de buscar vulnerabilidades e falhas com o apoio de pessoas externas ao projeto ou à empresa, sendo em sua grande maioria, os hackers éticos.

As pessoas que irão testar as aplicações cadastrada pelas empresas terão permissão legal para poder tentar invadir elas, buscando vulnerabilidades como informações sigilosas e falhas de segurança por exemplo, mas de uma forma construtiva, dispondo para a empresa o ponto exato que necessita de mudança ou melhoria. Os hackers éticos que participam dos programas de *Bug Bounty* recebem em troca uma remuneração disponibilizada pelas empresas participantes em caso do encontro de vulnerabilidades, sendo ela proporcional ao nível de gravidade da falha encontrada, o que pode tornar o processo de desenvolvimento no quesito de segurança mais barato para as empresas.

4.2. Testes Automatizados

Já nesse tipo de teste, são utilizados *scripts* e *frameworks* para realizar o controle de qualidade e os testes de forma automatizada, a exemplo do *JUnit*, *Selenium* e *Mockito*.

4.2.1. Testes de Unidade (Unit Tests)

Os testes automatizados de unidade são referentes aos tipos de testes que têm uma interação direta com o código-fonte da aplicação em questão, ou seja, com as rotinas e funções que venham a ter sido desenvolvidas pela equipe de desenvolvimento. Nesse tipo de teste, problemas referentes à sintaxe, má estruturação de variáveis e incoerências com a interface das funções e métodos podem ser facilmente identificados (e.g. um método que deve retornar a soma de dois números estar retornando a multiplicação deles).

Nos teste de unidade automatizados, existirão três blocos de código caracterizados pelo "*triplo A"*, ou seja, o *arrang*e (preparo), *act* (ação) e *assert* (afirmação).

No momento do *arrange*, é "preparado o terreno", ou seja, são realizados todos os procedimentos que a unidade que vai ser testada necessite, como definição de variáveis por exemplo.

Já no *act*, é realizada a ação que a unidade que está sendo testada propõe fazer, a exemplo de somar uma lista de números.

Por fim, no *assert*, é realizada a verificação entre o resultado esperado e o resultado advindo da etapa do *act*.

Vamos considerar um simples teste de unidade em python que tem por objetivo testar uma função *sum_positive_numbers*, que deve somar todos os números positivos de uma lista:

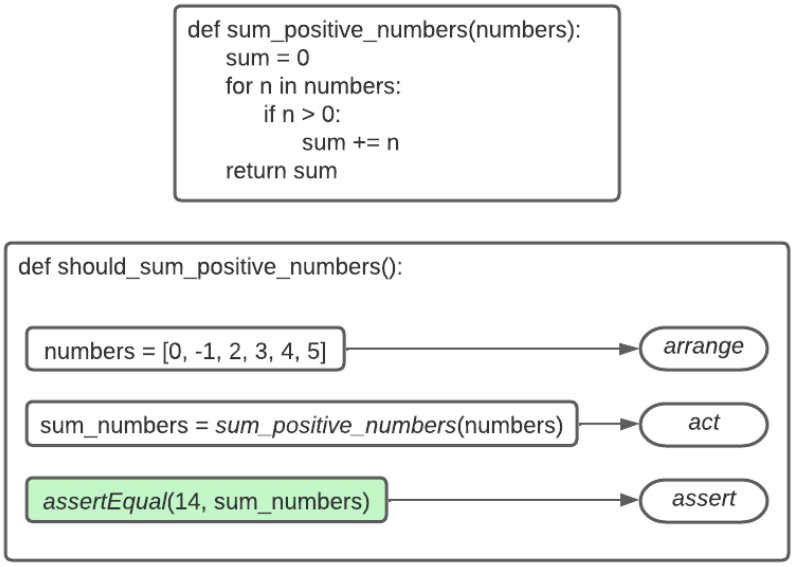

Figura 5 - Exemplo de Teste de Unidade em Python

Como é possível observar acima, existem todas as etapas descritas acima de uma forma prática, contendo a função “sum_positive_numbers”, descrita anteriormente, a definição de uma variavel “numbers” com uma lista de números (*arrange*), a ação referente ao teste da função “sum_positive_numbers” (*act*), e por fim, a conferência entre o resultado esperado e o resultado do *act* (*assert*).

4.2.2. Testes de Serviços Web (Web Services Tests)

Os serviços web, sendo eles mais conhecidos pelo termo *APIs* (Application Programming Interface), são recursos que utilizam de protocolos que permitem interações e integrações entre diversas aplicações e entre sistemas web. As *APIs* servem de “intermédio” entre uma aplicação e um banco de dados na grande maioria das situações, contendo por exemplo, recursos que implementam requisitos relativos à segurança de seu uso, que permitiria certas funções apenas para certos usuários.

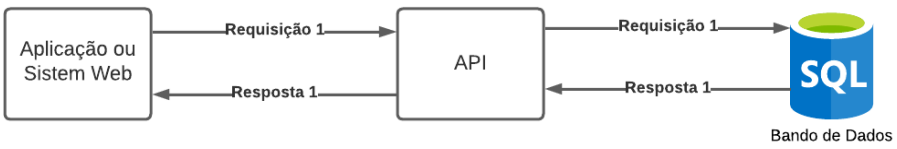

Figura 6 - Abstração do Teste de Serviços Web

Com a advinda de frameworks e tecnologias como o *Mockito* e o *JUnit*, sendo ambos frameworks utilizados para automação de testes no java, as automações dos testes também passaram a ser aplicadas aos serviços web, reduzindo trabalho, aumentando a produtividade e confiabilidade do software que esteja sendo desenvolvido ou mantido.

A estrutura é bem semelhante à dos testes de unidade, tendo em seu corpo algumas divisões referentes ao preparo do teste e à conferência entre o resultado esperado e o resultado obtido.

Considerando dessa vez um exemplo em Java, estará sendo testada a funcionalidade GET, que é uma funcionalidade referente ao ato de obter alguns dados como resposta (e.g. funcionário cadastrado com o CPF: 123.123.123-12) de uma *API* hospedada em "https://domain.com/api/v1":

```
public testResponseCode() {
    Response resp = new Response();  // arrange

    resp = RestAssured.get("https://domain.com/api/v1");  // act
    int responseCode = resp.getStatusCode();

    Assert.assertEquals(code, 200);  // assert
}
```

Figura 7 - Exemplo de Teste de Serviço Web em Java

A partir do exemplo acima é possível observar a semelhança entre os testes de serviços web e os testes de unidade, já que ambas tem como base o conceito do "tiple A" (*arrange*, *act* e *assert*). No método "testResponseCode" listado acima, tem-se o arrange, que está sendo responsável por definir a variável "resp", que tem por objetivo guardar o conteúdo da resposta da requisição que estará sendo feita posteriormente no "*act*", por fim, é realizada a conferência entre o código de status obtido na resposta recebida anteriormente e o código de status que é esperado.

4.2.3. Testes de Interface Gráfica (GUI Tests)

Quando é falado em testes de interface gráfica, diferentemente dos outros dois tipos de testes apresentados anteriormente, o foco passa a ser o desenvolvimento front-end, que é responsável por toda a responsividade e toda a exibição gráfica de uma aplicação web para o seu usuário final.

Com a advinda de ferramentas como o *Selenium*, que é um framework que suporta diversas linguagens de programação, como o Java, Python e NodeJs por exemplo, as automações

dos testes voltados para a interface gráfica de uma aplicação passaram a ser cada vez mais requisitadas e propostas, já que irão garantir mais confiabilidade e garantia ao produto.

```
def should_press_login_button_and_login:
    # Arrange
    driver = webdriver.Chrome()
    usernameData = 'lucaslucena'
    passwordData = '12345'

    # Act
    driver.find_element(By.NAME, 'usernameField').send_keys(usernameData)
    driver.find_element(By.NAME, 'passwordField').send_keys(passwordData)

    # Assert
    driver.find_element(By.ID, 'trta1-mfs-later')
```

Figura 8 - Exemplo de Teste de Interface Gráfica com Selenium e Python

Acima temos um exemplo prático utilizando o framework Selenium conjuntamente com a linguagem Python, nesse exemplo há o teste de um formulário de login, o qual está testando a inserção dos campos “username”, “password”, também está testando o botão de login e por fim conferindo se o elemento que confirma que o login foi realizado com sucesso está presente.

5. Responsabilidade dos Testes em uma Metodologia de Desenvolvimento

Serão abordadas três metodologias de desenvolvimento que variam do modelo em cascata ao modelo ágil, cada uma tendo uma diferente atribuição referente à responsabilidade dos testes no contexto de desenvolvimento de softwares.

5.1. Metodologia Cascata

A metodologia de desenvolvimento cascata tem por característica uma rígida atenção às documentações com uma perspectiva de progressão constante no que havia sido planejado, dando pouca ou quase nenhuma abertura para algum tipo de replanejamento ou ajuste anterior. Sabendo disso, tem-se que os **Analistas de Testes** serão os responsáveis pela rotina de testes do projeto, se apropriando dos testes funcionais, procurando automatizar os testes.
Nesse contexto, os desenvolvedores da equipe não eram responsáveis pelas rotinas de teste, mas sim pela correção de bugs encontrados pelos testadores. Vale ressaltar que essa divisão poderia provocar algum tipo de “disputa”, gerando assim uma perda de produtividade e até alguns desentendimentos dentro da equipe.

5.2. Metodologia Ágil em Transição

A metodologia ágil tem toda uma mentalidade que prega o trabalho em conjunto com a equipe, mas geralmente, a empresa que estiver realizando a adoção da metodologia ágil irá cometer alguns erros relativos à organização do fluxo de informações e de entregas, caracterizando uma “Metodologia Ágil em Transição”. Tendo isso em mente, o time como um todo terá os testes sob sua responsabilidade.
Nessa transição há a adoção de mais um integrante na equipe, os ***Quality Assurances*** (QA), que por sua vez irão mentorear o time quanto às práticas dos testes de software, podendo também, codificar de forma a auxiliar a equipe de desenvolvimento.

5.3. Metodologia Ágil

Como já foi expressado, a metodologia ágil tem como característica uma segregação de funções, porém um trabalho conjunto entre todas elas, proporcionando um alto rendimento de trabalho com entregas constantes e interativas entre cada parte da equipe de desenvolvimento. Dessa forma, os testes ficam sob responsabilidade dos **Engenheiros de Software** da equipe envolvida no projeto, o que acaba sendo uma quebra de paradigma de uma métrica que era adotada pelo mercado, condizente com a tese de que os engenheiros de software não teriam todos os conhecimentos necessários para a condução dos testes de software.
Os engenheiros de software, além de codificar, devem também trabalhar nos testes e suas respectivas automações, não inteirando a necessidade de um *QA*, já que os próprios engenheiros entendem a qualidade de software como uma cultura, não como uma necessidade, adotando também a adoção de frameworks para automação dos testes.

6. Referências Bibliográficas

[1] WHAT is Software Testing?. [*S. l.*], 17 jul. 2019. Disponível em: https://www.youtube.com/watch?v=cDQ34z0oqnQ. Acesso em: 14 out. 2022.

[2] SOFTWARE Development Life Cycle (SDLC). [*S. l.*], 8 abr. 2019. Disponível em: https://bigwater.consulting/2019/04/08/software-development-life-cycle-sdlc/. Acesso em: 14 out. 2022.

[3] TESTES, DO CASCATA AO ÁGIL: A evolução da responsabilidade em testes. [*S. l.*], 17 mar. 2020. Disponível em: https://www.youtube.com/watch?v=P4M6mMnGjdk. Acesso em: 13 out. 2022.

[4] TESTES automatizados: saiba o que são, como e porquê usar!. [*S. l.*], 9 maio. 2022. Disponível em: https://blog.betrybe.com/tecnologia/teste-automatizados/. Acesso em: 14 out. 2022.

[5] O PADRÃO Triple A (Arrange, Act, Assert). [*S. l.*], 28 fev. 2020. Disponível em: https://medium.com/@pablodarde/o-padrão-triple-a-arrange-act-assert-741e2a94cf88#:~:tex

t=O%20modelo%20Triple%20A%20é,Assert%20(Verificar%20as%20asserções). Acesso em: 14 out. 2022.

[6] TUTORIAL de teste de automação de API REST. [*S. l.*], 26 maio. 2021. Disponível em: https://www.loadview-testing.com/pt-br/blog/tutorial-de-teste-de-automacao-de-api-rest/. Acesso em: 14 out. 2022.

[7] TÉCNICAS de caixa branca e preta para teste de software. [*S. l.*], 2020. Disponível em: https://blog.geekhunter.com.br/tecnicas-de-caixa-preta-e-branca-para-teste-de-software/. Acesso em: 14 out. 2022.

[8] CONHEÇA a importância de identificar bugs preventivamente no desenvolvimento de software. [*S. l.*], 10 mar. 2021. Disponível em: https://testingcompany.com.br/blog/conheca-a-importancia-de-identificar-bugs-preventivamente-no-desenvolvimento-de-software#:~:text=Segundo%20o%20ISTQB%20(International%20Software,se%20manterem%20competitivas%20no%20mercado. Acesso em: 14 out. 2022.

[9] IMPORTÂNCIA dos testes de software na qualidade do sistema. [*S. l.*], 1 out. 2020. Disponível em: https://www.treinaweb.com.br/blog/importancia-dos-testes-de-software-na-qualidade-do-sistema#:~:text=Os%20testes%20de%20software%20são,ser%20vistos%20como%20algo%20opcional.&text=Testes%20de%20software%20é%20um,para%20verificar%20seu%20correto%20funcionamento. Acesso em: 14 out. 2022.

[10] TESTES de SOFTWARE — Teste CAIXA-BRANCA e CAIXA-PRETA. [*S. l.*], 15 nov. 2017. Disponível em: https://www.youtube.com/watch?v=QyXN_zAhqJA. Acesso em: 21 out. 2022.

[11] #2 NÍVEIS DE TESTES | O QUE SÃO TESTES DE INTEGRAÇÃO?. [*S. l.*], 27 out. 2020. Disponível em: https://www.youtube.com/watch?v=ewAwf02QVnQ. Acesso em: 21 out. 2022.

[12] ENGENHARIA de Software - Teste de segurança. [*S. l.*], 11 abr. 2022. Disponível em: https://www.youtube.com/watch?v=-HBYN1kebR0. Acesso em: 21 out. 2022.

[13] O QUE É BUG BOUNTY - VALE A PENA? - HackerSec. [*S. l.*], 13 jul. 2021. Disponível em: https://www.youtube.com/watch?v=ee5rZY55xK0. Acesso em: 21 out. 2022.