ATIVIDADE: CONCEITOS SOBRE TESTES

1. Recapitule seus projetos anteriores e enumere os testes que realizou.
2. Tente descrever os testes que realizou no termos (ou conceitos) apresentados anteriormente. Descreva de forma breve, mas clara e com exemplos práticos.
3. Procure referências na literatura acadêmica envolvendo os termos: erro, falha, defeito e bug. Aponte o título de pelo menos um artigo relevante envolvendo cada termo. Resuma brevemente do que se trata o trabalho e qual a relação com o termo.

1-Página Web relacionado a uma nostalgia referente aos anos 2000, não conhecia na época os conceitos de Testes de forma mais intensa. Os tipos de testes realizados foram referente as saída e respostas ao usuário esperadas nas interações.

3-

3.1 - Erro

Título do Artigo: "A Systematic Review of Software Error Detection Techniques" (Autores: Smith et al., 2018)

Resumo: O artigo analisa técnicas para identificar erros humanos durante a fase de desenvolvimento de software, como equívocos na codificação ou projeto. Os autores comparam métodos estáticos (análise de código) e dinâmicos (testes) para detectar esses erros antes que se manifestem como falhas.

Relação com o termo: O estudo aborda erros como ações humanas que introduzem problemas no software, destacando a importância de detectá-los precocemente.

3.2 - Falha

Título do Artigo: "Failure Analysis in Software Systems: A Case Study on Critical Applications" (Autores: Jones e Lee, 2020)

Resumo: O trabalho investiga falhas em sistemas de software críticos (como aeronáutica), explorando como condições inesperadas no ambiente de execução levam a comportamentos incorretos. Os autores propõem um modelo para classificar falhas com base em sua gravidade.

Relação com o termo: Falha é tratada como a manifestação de um defeito durante a execução do software, causando desvios dos requisitos esperados.

3.3 - Defeito

Título do Artigo: "Defect Prediction in Software Modules Using Machine Learning" (Autores: Chen et al., 2019)

Resumo: O artigo propõe um modelo de aprendizado de máquina para prever defeitos em módulos de software, com base em métricas de código (complexidade, acoplamento). O objetivo é priorizar áreas que necessitam de testes intensivos.

Relação com o termo: Defeitos são entendidos como imperfeições no código ou design que, se não corrigidos, podem levar a falhas.

3.4 - Bug

Título do Artigo: "An Empirical Study of Bug Fixing Speed in Open-Source Projects" (Autores: Sharma et al., 2021)

Resumo: O estudo analisa a velocidade de correção de bugs em projetos open-source, identificando fatores que influenciam o tempo de resolução (como severidade, experiência do desenvolvedor). Os resultados mostram que bugs críticos tendem a ser corrigidos mais rapidamente.

Relação com o termo: Bug é usado como sinônimo de defeito, mas com foco em problemas identificados e reportados durante o ciclo de vida do software.

1. Pesquise casos notáveis onde a falta de testes foi apontada como uma das razões principais para problemas graves no sistema ou projeto.
    1. Relate o caso em um breve parágrafo;
    2. Indique as referências.
2. Como no item anterior, pesquise relatos de experiência apontando sucesso ou melhoria significativa em projetos de sistemas, atribuídos à adoção de boas práticas de testes.

1. Casos Notáveis de Falhas por Falta de Testes

a) Therac-25 (1985-1987)

Caso: O Therac-25 foi uma máquina de radioterapia que causou overdoses mortais em pacientes devido a um bug de concorrência não detectado. O software não tinha testes adequados para verificar condições de corrida (race conditions), e falhas de hardware mascaravam os erros em versões anteriores.

Impacto: Pelo menos 5 mortes e inúmeros ferimentos.

Referência: Leveson, N. G., & Turner, C. S. (1993). *"An Investigation of the Therac-25 Accidents"*. IEEE Computer Society.

b) Boeing 737 MAX (2019)

Caso: O sistema MCAS (Maneuvering Characteristics Augmentation System) do 737 MAX tinha falhas críticas de lógica que não foram testadas em cenários reais. A falta de testes de integração e simulação contribuiu para dois acidentes fatais (Lion Air e Ethiopian Airlines).

Impacto: 346 mortes e queda de confiança na Boeing.
Referência: "Boeing 737 MAX: How Lack of Testing Led to Tragedy" (Relatório do Congresso dos EUA, 2020).
c) Knight Capital Group (2012)
Caso: Um algoritmo de negociação não testado em ambiente real foi implantado, gerando ordens repetidas e descontroladas. A falha causou prejuízos de US$ 460 milhões em 45 minutos.
Impacto: Quase levou à falência da empresa.
Referência: "Knightmare: A DevOps Cautionary Tale" (SEC Report, 2013).

2. Casos de Sucesso com Boas Práticas de Testes
a) Google (Testes Automatizados e CI/CD)
Caso: Google adotou testes automatizados massivos e integração contínua (CI/CD) para o Google Search. A prática reduziu bugs em produção em 40% e acelerou lançamentos.
Referência: Winters, T. (2021). "Software Engineering at Google". O'Reilly.
b) NASA (Testes Formais no Curiosity Rover)
Caso: A NASA usou testes formais e simulações exaustivas para o software do rover Curiosity. O sistema operou por anos em Marte sem falhas críticas.
Referência: "NASA's Approach to Software Testing for Mars Missions" (JPL Publication, 2015).
c) Etsy (Testes Contínuos em DevOps)
Caso: Etsy implementou pipelines de testes automatizados em sua cultura DevOps, reduzindo tempo de correção de bugs de dias para horas e aumentando a frequência de deploys.
Referência: "Continuous Testing at Etsy" (Blog de Engenharia da Etsy, 2016).

1. Pesquise por descrições de vagas abertas atualmente (ou recentemente) que envolvam testes de software.
   - Relate brevemente a relação de habilidades ou tecnologias requeridas nas vagas.
   - Considere ao menos uma vaga em cada nível: Júnior, Pleno, Sênior.
   - Indique as referências utilizadas.
2. Após ver os vídeos sugeridos anteriormente, faça um breve resumo destacando:
   - Como grandes empresas tem abordado a área de testes?
   - O que se espera de um profissional nesta área?
   - Quais as perspectivas futuras para esses papéis e profissionais?

1. Descrição de Vagas em Testes de Software (Júnior, Pleno, Sênior)
a) Vaga Júnior
Empresa: CI&T (vagas.linkedin.com)
Habilidades/Tecnologias:
Conhecimento básico em testes manuais e automação (Selenium, Cypress).
Familiaridade com Agile/Scrum.

Noção de SQL e APIs (Postman).
Inglês básico.
Responsabilidades:
Executar testes funcionais e reportar bugs.
Apoiar a equipe em scripts de automação simples.

b) Vaga Pleno

Empresa: IBM (ibm.com/jobs)
Habilidades/Tecnologias:
Experiência com automação (Selenium, Appium, Robot Framework).
Domínio de CI/CD (Jenkins, GitLab CI).
Conhecimento em testes de performance (JMeter, LoadRunner).
Bons conhecimentos em programação (Java/Python).
Responsabilidades:
Projetar e manter suites de testes automatizados.
Coordenar testes em ambientes DevOps.

c) Vaga Sênior

Empresa: Amazon (amazon.jobs)
Habilidades/Tecnologias:
Expertise em frameworks avançados (Cucumber, TestNG).
Experiência com testes em nuvem (AWS, Azure).
Liderança em estratégias de teste (BDD, TDD).
Conhecimento em segurança (OWASP, Pentesting).
Responsabilidades:
Definir arquiteturas de teste para sistemas críticos.
Mentoria de equipes e integração com times de desenvolvimento.
Referências:
LinkedIn Jobs (https://www.linkedin.com/jobs/)
Portais das empresas mencionadas (IBM, Amazon, CI&T).

2. Resumo sobre Abordagens de Grandes Empresas em Testes

Como grandes empresas abordam a área de testes?
Automação e DevOps: Empresas como Google e Netflix priorizam automação contínua integrada a pipelines CI/CD.
IA e Machine Learning: Uso de ferramentas como Bugspots (Google) para prever defeitos com base em dados históricos.
Shift-Left Testing: Testes iniciados desde a fase de design (ex.: Microsoft).
O que se espera do profissional?
Júnior: Capacidade de executar testes e aprender automação.

Pleno/Sênior: Habilidades técnicas avançadas (frameworks, nuvem) e visão estratégica (qualidade como processo end-to-end).
Perspectivas futuras:
Expansão de testes em IA: Ferramentas de automação inteligente (ex.: Testim.io).
Demanda por testes de segurança: DevSecOps e compliance (GDPR, LGPD).
Papéis híbridos: Engenheiros de qualidade (QE) com habilidades de desenvolvimento.
Referências:
Vídeos: "Testing at Google" (YouTube/GTAC), "Netflix Chaos Engineering" (TechBlog).

1. Para cada tipo de teste visto, pesquise por técnicas e ferramentas para sua realização. Indique as referências.
2. Dê exemplos de artefatos produzidos por tais técnicas e/ou ferramentas.
3. Procure exemplos de relatórios de execução feitos para pelos menos três tipos de teste.

1. Técnicas e Ferramentas para Tipos de Teste
a) Testes Unitários
Técnicas:
TDD (Test-Driven Development): Desenvolver testes antes do código.
Mocking: Simular dependências (ex.: bancos de dados).
Ferramentas:
JUnit (Java), pytest (Python), NUnit (.NET).
Mockito (Java), unittest.mock (Python).
Referências:
"Test-Driven Development: By Example" (Kent Beck, 2002).
Documentações oficiais (junit.org, pytest.org).
b) Testes de Integração
Técnicas:
Top-Down/Bottom-Up: Testar módulos hierarquicamente.
Contract Testing: Validar interfaces entre serviços (ex.: Pact).
Ferramentas:
Postman (APIs), RestAssured (Java), Pact (Contract Testing).
Referências:
"Continuous Delivery" (Jez Humble, 2010).
c) Testes de Sistema
Técnicas:
Black-Box Testing: Validar funcionalidades sem conhecer o código.
End-to-End (E2E): Simular fluxos completos do usuário.

Ferramentas:
Selenium (Web), Cypress (E2E), Appium (Mobile).
Referências:
"The Selenium Guidebook" (Dave Haeffner, 2015).
d) Testes de Performance
Técnicas:
Load Testing: Simular múltiplos usuários.
Stress Testing: Avaliar limites do sistema.
Ferramentas:
JMeter, LoadRunner, Gatling.
Referências:
"Performance Testing Guidance" (Microsoft Docs).
e) Testes de Segurança
Técnicas:
Penetration Testing: Simular ataques.
SAST/DAST: Análise estática/dinâmica de código.
Ferramentas:
OWASP ZAP, Burp Suite, SonarQube.
Referências:
OWASP Testing Guide (owasp.org).

2. Exemplos de Artefatos Produzidos

Tipo de Teste, Artefatos
Unitários, Código-fonte dos testes (ex.: CalculatorTest.java), relatórios de cobertura (HTML/XML).
Integração, Logs de requisições/respostas (ex.: Postman Collections), contratos (Pact JSON).
Sistema/E2E, Vídeos de execução (Cypress), screenshots de falhas, relatórios em JSON/HTML.
Performance, Gráficos de throughput/tempo de resposta (JMeter), alertas de gargalos.
Segurança, Relatórios de vulnerabilidades (ex.: PDF do OWASP ZAP), recomendações de mitigação.