Finalizar a atividade 10 e 11

Semana 15 - Aula 1

Tópico Principal da Aula: Testes funcionais e não funcionais

Subtítulo/Tema Específico: Estratégias e técnicas de testes funcionais e não funcionais

Código da aula: [SIS]ANO1C3B2S15A1

Objetivos da Aula:

- Compreender a importância de testes funcionais no contexto de desenvolvimento de software para empresas.

Recursos Adicionais:

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 04 - Testes funcionais

- **Definição:** **Testes funcionais são um tipo de teste de software que verifica se o sistema está operando de acordo com os requisitos especificados. Em outras palavras, eles asseguram que o software faz "o que se espera que faça", ou seja, que todas as funções e recursos operam conforme o projetado.** São cruciais para garantir a conformidade do software com as expectativas do usuário e as especificações de negócio.
- **Aprofundamento/Complemento (se necessário):** Os testes funcionais focam na validação das funcionalidades do sistema, independentemente de sua implementação interna. Eles são frequentemente baseados nas especificações de requisitos e nos casos de uso do sistema. A execução desses testes geralmente não requer conhecimento da estrutura interna do código, sendo por isso classificados como "testes de caixa preta".
- **Exemplo Prático:** Em um sistema de e-commerce, um teste funcional para a funcionalidade de "adição de item ao carrinho" verificaria se, ao clicar no botão "Adicionar ao Carrinho", o item é de fato adicionado, a quantidade é atualizada corretamente e o preço total do carrinho reflete a adição do item.

Referência do Slide: Slide 05 - Verificação de requisitos

- **Definição: A verificação de requisitos, no contexto de testes funcionais, é o processo de confirmar se o software em desenvolvimento realmente atende a todas as especificações e necessidades do usuário que foram previamente definidas. É a etapa onde se garante que o sistema construído corresponde ao sistema desejado.**
- **Aprofundamento/Complemento (se necessário):** Este processo é fundamental para evitar o "desalinhamento" entre o que foi pedido e o que foi entregue. A verificação de requisitos é contínua e deve ocorrer em todas as fases do ciclo de vida do desenvolvimento de software, mas é particularmente enfatizada durante os testes funcionais, onde cada requisito é mapeado para um ou mais casos de teste.
- **Exemplo Prático:** Para um requisito que diz "O usuário deve ser capaz de fazer login com seu nome de usuário e senha registrados", a verificação envolveria a criação de casos de teste para login bem-sucedido com credenciais válidas, falha de login com credenciais inválidas e tratamento de senhas em branco.

Referência do Slide: Slide 06 - Técnicas de teste de caixa preta

- **Definição: Testes de caixa preta são técnicas de teste funcional que avaliam a funcionalidade de um software sem conhecer sua estrutura interna, código ou implementação. O foco é dado nas entradas e saídas do sistema, tratando o software como uma "caixa preta" da qual apenas o comportamento externo é observado.**
- **Aprofundamento/Complemento (se necessário):** As principais técnicas de teste de caixa preta incluem:
    - **Teste de Equivalência (ou Particionamento de Equivalência):** Divide os dados de entrada em partições de dados válidos e inválidos. Se uma condição é verdadeira para um valor em uma partição, ela é assumida como verdadeira para todos os valores nessa partição.
    - **Teste de Valor-Limite (ou Análise de Valor Limite):** Concentra-se nos valores nos limites das partições de equivalência, pois são historicamente os pontos mais propensos a erros. Por exemplo, se uma entrada aceita números de 1 a 100, os valores a serem testados seriam 0, 1, 2, 99, 100, 101.
    - **Tabela de Decisão:** Usada para testar sistemas que têm regras de negócio complexas e várias condições e ações. Mapeia combinações de condições a ações específicas.
    - **Grafos de Causa e Efeito:** Representam as relações entre causas (condições de entrada) e efeitos (saídas ou resultados).
- **Exemplo Prático:** Para um campo de idade que aceita valores de 18 a 65:
    - **Particionamento de Equivalência:** Partições seriam: idade < 18 (inválido), 18 <= idade <= 65 (válido), idade > 65 (inválido).
    - **Análise de Valor Limite:** Testaríamos 17, 18, 19, 64, 65, 66.

Referência do Slide: Slide 07 - Automação de testes funcionais

- **Definição:** **A automação de testes funcionais envolve o uso de software e ferramentas para executar casos de teste funcional de forma automática, sem a necessidade de intervenção manual em cada execução.** Isso acelera o processo de teste, melhora a repetibilidade e reduz a ocorrência de erros humanos.
- **Aprofundamento/Complemento (se necessário):** A automação é particularmente útil em ambientes de integração e entrega contínua (CI/CD), onde os testes precisam ser executados frequentemente. Ferramentas populares para automação de testes funcionais incluem Selenium (para web), Cypress, Playwright, Appium (para mobile), entre outras. O objetivo não é substituir completamente os testes manuais, mas otimizar os ciclos de teste e permitir que os testadores se concentrem em casos mais complexos ou exploratórios.
- **Exemplo Prático:** Utilizar Selenium para automatizar o fluxo de login em um site:
- Python

```
from selenium import webdriver
from selenium.webdriver.common.by import By

# Inicializa o driver do navegador (ex: Chrome)
driver = webdriver.Chrome()
driver.get("https://www.seusite.com/login")

# Encontra os elementos de input e preenche-os
username_input = driver.find_element(By.ID, "username")
password_input = driver.find_element(By.ID, "password")
login_button = driver.find_element(By.ID, "loginButton")

username_input.send_keys("usuario_teste")
password_input.send_keys("senha_teste")
login_button.click()

# Verifica se o login foi bem-sucedido (ex: verifica o título da página)
assert "Bem-vindo" in driver.title

driver.quit()
```

-
-

Referência do Slide: Slide 08 - Integração de testes funcionais em pipelines de CI/CD

- **Definição: A integração de testes funcionais em pipelines de Integração Contínua (CI) e Entrega Contínua (CD) significa que os testes automatizados são executados automaticamente a cada nova alteração de código ou a cada construção do software**. Isso garante que quaisquer regressões ou novos defeitos sejam identificados rapidamente, mantendo a qualidade do código consistentemente alta.
- **Aprofundamento/Complemento (se necessário):** Em um pipeline de CI/CD, quando um desenvolvedor faz um commit de código, o servidor de CI (como Jenkins, GitLab CI, GitHub Actions) automaticamente compila o código, executa os testes unitários e, em seguida, os testes funcionais automatizados. Se todos os testes passarem, o código pode ser integrado ao ramo principal (branch master/main) e, posteriormente, implantado em ambientes de teste ou produção através do CD. Isso cria um feedback rápido para os desenvolvedores e assegura que apenas código testado e funcional seja entregue.
- **Exemplo Prático:** Configurar um pipeline em Jenkins que, após cada commit no repositório Git, execute os seguintes passos:
    1. **Build:** Compilar o projeto.
    2. **Testes Unitários:** Executar todos os testes unitários.
    3. **Testes Funcionais:** Executar os testes funcionais automatizados (por exemplo, scripts Selenium).
    4. **Relatório:** Gerar um relatório de testes e notificar a equipe em caso de falha.
    5. **Deploy (se testes passarem):** Realizar o deploy da aplicação em um ambiente de homologação.

Semana 15 - Aula 2

Tópico Principal da Aula: Testes funcionais e não funcionais

Subtítulo/Tema Específico: Estratégias e técnicas de testes funcionais e não funcionais

Código da aula: [SIS]ANO1C3B2S15A2

Objetivos da Aula:

- Conhecer os aspectos de desenvolvimento para testes não funcionais voltados para programas empresariais.

Recursos Adicionais:

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 04 - Testes não funcionais

- **Definição:** Testes não funcionais são um tipo de teste de software que avalia os "atributos de qualidade" de um sistema, em vez de suas funcionalidades específicas. Eles garantem que o sistema atenda a critérios como desempenho, segurança, usabilidade, escalabilidade, compatibilidade, confiabilidade, entre outros. São cruciais para assegurar que o software não apenas faça o que foi projetado, mas que o faça de forma eficiente, segura e utilizável.
- **Aprofundamento/Complemento (se necessário):** Enquanto os testes funcionais respondem à pergunta "O software faz o que deveria fazer?", os testes não funcionais respondem à pergunta "O software funciona bem?". Esses testes são fundamentais para a experiência do usuário e para a robustez do sistema, pois um software funcional, mas lento ou inseguro, não atenderá às expectativas do negócio.
- **Exemplo Prático:** Para um aplicativo bancário, um teste não funcional de desempenho poderia verificar se a transação de pagamento é concluída em menos de 2 segundos, mesmo com 1000 usuários simultâneos. Um teste de segurança poderia tentar injetar código SQL malicioso para acessar dados confidenciais.

Referência do Slide: Slide 05 - Testes de desempenho

- **Definição:** Testes de desempenho são um subconjunto dos testes não funcionais focados em avaliar a velocidade, a escalabilidade, a estabilidade e a capacidade de resposta de um sistema sob diferentes cargas. O objetivo é identificar gargalos e garantir que o software possa lidar com o volume de usuários e transações esperados.
- **Aprofundamento/Complemento (se necessário):** Os principais tipos de testes de desempenho incluem:
    - **Teste de Carga:** Simula o número esperado de usuários ou transações para verificar o comportamento do sistema sob carga normal.
    - **Teste de Stress:** Empurra o sistema além de seus limites normais para verificar como ele se comporta em condições extremas e onde ele falha.
    - **Teste de Pico:** Simula aumentos repentinos na carga de usuários para ver como o sistema lida com picos de tráfego.
    - **Teste de Volume:** Testa o desempenho do sistema com uma grande quantidade de dados no banco de dados.
    - **Teste de Estabilidade (ou Teste de Fadiga):** Avalia o desempenho do sistema sob carga contínua por um longo período para verificar a estabilidade.

    - Ferramentas comuns para testes de desempenho incluem JMeter, LoadRunner e Gatling.
- **Exemplo Prático:** Utilizando o JMeter, simular 500 usuários acessando simultaneamente uma página de produto em um e-commerce para medir o tempo de resposta do servidor e o uso da CPU. O resultado esperado seria que o tempo de resposta médio não exceda 1 segundo.

Referência do Slide: Slide 06 - Testes de segurança

- **Definição:** Testes de segurança são um tipo de teste não funcional que visa identificar vulnerabilidades, ameaças e riscos no software que possam levar à perda de dados, acesso não autorizado ou outras falhas de segurança. O objetivo é proteger os dados e a funcionalidade do sistema contra ataques maliciosos.
- **Aprofundamento/Complemento (se necessário):** Os testes de segurança podem incluir:
    - **Testes de Vulnerabilidade:** Usam ferramentas automatizadas para escanear o software em busca de vulnerabilidades conhecidas.
    - **Testes de Penetração (Pentesting):** Simulam ataques reais por parte de hackers para explorar vulnerabilidades e avaliar a resiliência do sistema. Podem ser de "caixa branca" (com conhecimento total do sistema) ou "caixa preta" (sem conhecimento prévio).
    - **Análise de Código Estática (SAST):** Analisa o código-fonte em busca de falhas de segurança sem executá-lo.
    - **Análise de Código Dinâmica (DAST):** Testa o software em tempo de execução para identificar vulnerabilidades.
    - **Testes de Autenticação e Autorização:** Verificam se os mecanismos de login e permissões estão funcionando corretamente.
- **Exemplo Prático:** Tentar realizar um ataque de injeção SQL em um formulário de login para ver se é possível bypassar a autenticação. Ou, em um teste de penetração, tentar explorar uma vulnerabilidade de cross-site scripting (XSS) em um campo de comentário.

Referência do Slide: Slide 07 - Usabilidade e compatibilidade

- **Definição:**
    - **Usabilidade:** Refere-se à facilidade com que os usuários podem aprender a usar o software, operá-lo e alcançar seus objetivos de forma eficiente e satisfatória. Testes de usabilidade avaliam a interface do usuário (UI) e a experiência do usuário (UX).
    - **Compatibilidade:** Refere-se à capacidade do software de operar corretamente em diferentes ambientes, como diferentes sistemas operacionais, navegadores, dispositivos (desktop, mobile) e versões de software.
- **Aprofundamento/Complemento (se necessário):**

    - **Testes de Usabilidade:** Podem ser realizados por meio de:
        - **Testes com Usuários Reais:** Observar usuários reais interagindo com o software.
        - **Avaliação Heurística:** Especialistas em usabilidade avaliam a interface com base em princípios de design.
        - **Questionários e Pesquisas:** Coleta de feedback direto dos usuários.
    - **Testes de Compatibilidade:** Envolvem a execução do software em diversas configurações para garantir que ele se adapte e funcione bem em todas elas. Isso é crucial para atingir um público amplo e garantir uma experiência consistente.
- **Exemplo Prático:**
    - **Usabilidade:** Observar um grupo de usuários tentando completar uma tarefa específica em um aplicativo (ex: fazer uma reserva de voo) e registrar dificuldades, tempo gasto e nível de satisfação.
    - **Compatibilidade:** Testar um site em diferentes navegadores (Chrome, Firefox, Edge, Safari) e em diferentes dispositivos (smartphone Android, iPhone, tablet, desktop) para garantir que o layout e as funcionalidades se adaptam corretamente.

Semana 15 - Aula 3

Tópico Principal da Aula: Testes funcionais e não funcionais

Subtítulo/Tema Específico: Estratégias e técnicas de testes funcionais e não funcionais

Código da aula: [SIS]ANO1C3B2S15A3

Objetivos da Aula:

- Conhecer os fundamentos sobre as estratégias de testes e sua gestão no ambiente corporativo.

Recursos Adicionais:

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 04 - Estratégias de teste e gestão

- **Definição:** A estratégia de teste é um plano de alto nível que define a abordagem geral para o processo de teste de software. Ela descreve os tipos

de testes a serem executados, as ferramentas a serem utilizadas, os ambientes de teste, as responsabilidades da equipe e os critérios de entrada e saída para cada fase de teste. A gestão de testes, por sua vez, é o processo de planejar, organizar, monitorar e controlar as atividades de teste ao longo do ciclo de vida do desenvolvimento de software.

- **Aprofundamento/Complemento (se necessário):** Uma estratégia de teste bem definida é essencial para o sucesso de qualquer projeto de software, pois garante que os recursos de teste sejam utilizados de forma eficiente, que os riscos sejam mitigados e que a qualidade do software seja assegurada. A gestão de testes envolve a coordenação de equipes, o rastreamento do progresso, a análise de métricas e a comunicação com as partes interessadas.
- **Exemplo Prático:** Para um projeto de desenvolvimento de um novo sistema ERP, a estratégia de teste pode definir que serão realizados testes unitários pelos desenvolvedores, testes de integração por uma equipe de QA, testes funcionais automatizados e manuais, testes de desempenho e testes de aceitação do usuário (UAT) com os clientes finais. A gestão de testes acompanharia o progresso de cada fase, os defeitos encontrados e a conformidade com os cronogramas.

Referência do Slide: Slide 05 - Planejamento de testes

- **Definição:** O planejamento de testes é a atividade de criar um plano de teste abrangente que detalha o escopo do teste, os objetivos, os recursos necessários, o cronograma, as tarefas, os riscos e os critérios de sucesso. É a base para todas as atividades de teste subsequentes.
- **Aprofundamento/Complemento (se necessário):** Um plano de teste eficaz deve incluir:
    - **Escopo do Teste:** O que será e o que não será testado.
    - **Objetivos do Teste:** O que se espera alcançar com os testes.
    - **Estratégia de Teste:** A abordagem geral (ex: testes de caixa preta, caixa branca, automação, manual).
    - **Ambiente de Teste:** As configurações de hardware e software necessárias.
    - **Recursos:** Equipe, ferramentas, tempo.
    - **Cronograma:** Datas de início e fim para as atividades de teste.
    - **Critérios de Entrada e Saída:** Condições para iniciar e encerrar uma fase de teste.
    - **Gerenciamento de Riscos:** Identificação e mitigação de riscos relacionados ao teste.
    - **Deliverables:** Documentos e relatórios a serem produzidos.
- **Exemplo Prático:** No desenvolvimento de um aplicativo móvel de delivery de comida, o planejamento de testes definiria que os testes funcionais focariam nas funcionalidades de busca de restaurantes, adição de itens ao carrinho e

checkout. O cronograma alocaria 2 semanas para testes funcionais e 1 semana para testes de usabilidade, com a expectativa de atingir 95% de cobertura dos requisitos funcionais antes do lançamento.

Referência do Slide: Slide 06 - Gestão de defeitos

- **Definição:** A gestão de defeitos (ou gerenciamento de bugs) é o processo de identificar, registrar, rastrear, analisar e resolver defeitos (bugs) encontrados durante o ciclo de vida do desenvolvimento de software. O objetivo é garantir que os defeitos sejam corrigidos de forma eficiente e que o software final tenha a melhor qualidade possível.
- **Aprofundamento/Complemento (se necessário):** As etapas típicas da gestão de defeitos incluem:
    1. **Descoberta e Relato:** Um testador encontra um defeito e o registra em uma ferramenta de gestão de defeitos (ex: Jira, Bugzilla, Azure DevOps).
    2. **Triagem (Triage):** A equipe avalia o defeito, prioriza-o e o atribui a um desenvolvedor.
    3. **Análise e Correção:** O desenvolvedor analisa o defeito e implementa uma correção.
    4. **Verificação:** O testador verifica se o defeito foi corrigido.
    5. **Fechamento:** Se a correção for validada, o defeito é fechado.
    6. **Geração de Relatórios:** Análise das tendências de defeitos e métricas de qualidade.
- **Exemplo Prático:** Um testador encontra um bug onde, ao adicionar 10 unidades de um produto no carrinho, o preço total está incorreto. Ele registra o defeito com detalhes (passos para reproduzir, comportamento esperado vs. atual, screenshots) em uma ferramenta. O time de desenvolvimento prioriza, o desenvolvedor corrige o erro na lógica de cálculo, e o testador retesta para confirmar que o bug foi resolvido antes de fechar o item.

Referência do Slide: Slide 07 - Melhores práticas em gestão de testes

- **Definição:** Melhores práticas em gestão de testes são um conjunto de diretrizes e abordagens comprovadas que visam otimizar o processo de teste de software, resultando em maior eficiência, eficácia e qualidade do produto final.
- **Aprofundamento/Complemento (se necessário):** Algumas das melhores práticas incluem:
    - **Envolvimento Antecipado do QA (Shift-Left Testing):** Iniciar as atividades de teste o mais cedo possível no ciclo de desenvolvimento, desde a fase de requisitos.
    - **Automação Estratégica:** Automatizar testes repetitivos e de regressão, focando nos casos que trazem maior retorno sobre o investimento.

    - **Testes Contínuos:** Integrar testes em todas as fases do pipeline de CI/CD para feedback rápido.
    - **Gerenciamento de Dados de Teste (TDM):** Criar e gerenciar dados de teste realistas e reutilizáveis.
    - **Métricas de Teste:** Coletar e analisar métricas (ex: cobertura de código, taxa de defeitos, tempo médio de reparo) para monitorar o progresso e identificar áreas de melhoria.
    - **Colaboração entre Equipes:** Promover a comunicação e a colaboração entre desenvolvedores, testadores e stakeholders.
    - **Gerenciamento de Ambientes de Teste:** Garantir que os ambientes de teste sejam estáveis, consistentes e representem o ambiente de produção.
- **Exemplo Prático:** Uma equipe de desenvolvimento adota a prática de "Shift-Left Testing", onde os QAs participam das reuniões de levantamento de requisitos para identificar possíveis cenários de teste e critérios de aceitação desde o início. Eles também implementam a automação de testes de regressão para cada build noturno e usam um dashboard para monitorar a cobertura de código e a taxa de defeitos abertos versus fechados.