Semana 11 - Aula 1

Tópico Principal da Aula: Testes funcionais e não funcionais

Subtítulo/Tema Específico: Testes não funcionais: desempenho, segurança e usabilidade

Código da aula: [SIS]ANO1C3B2S11A1

Objetivos da Aula:

- Compreender os fundamentos para melhorias de usabilidade em testes de aplicação de software.
- Conhecer a importância da otimização de desempenho em aplicações web e móveis.
- Entender os conceitos de análise de carga e stress, e otimização de recursos.

Recursos Adicionais (Sugestão, pode ser adaptado):

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

Exposição do Conteúdo:

Referência do Slide: Slide 04 - Otimização de desempenho em aplicações web e móveis

- **Definição:** A otimização de desempenho em aplicações web e móveis é um tema vital no desenvolvimento de software, pois o desempenho das aplicações influencia diretamente a experiência do usuário e a eficiência operacional. Uma aplicação lenta ou instável pode levar à perda de usuários e receita.
- **Aprofundamento/Complemento (se necessário):** O desempenho de uma aplicação não se limita apenas à velocidade, mas também à sua capacidade de resposta, estabilidade e utilização eficiente dos recursos do sistema. Em um mercado cada vez mais competitivo, onde a atenção do usuário é um recurso escasso, aplicações com desempenho superior tendem a se destacar. Isso envolve uma série de técnicas e práticas que vão desde a arquitetura do sistema até a codificação e a infraestrutura.
- **Exemplo Prático:** Imagine um e-commerce durante a Black Friday. Se a plataforma não estiver otimizada para lidar com um alto volume de acessos e transações, ela pode ficar lenta, apresentar erros ou até mesmo sair do ar. Isso resultaria em uma péssima experiência para o usuário, perda de vendas e danos à reputação da empresa. Um exemplo de otimização seria a

implementação de cache para requisições frequentes, reduzindo o tempo de carregamento de páginas de produtos.

Referência do Slide: Slide 05 - Análise de carga e stress

- **Definição:** Análise de carga e stress são técnicas de teste não funcionais que ajudam a entender como a aplicação se comporta sob diferentes níveis de demanda. A análise de carga simula o comportamento da aplicação sob um volume de usuários e requisições esperado, enquanto os testes de stress levam a aplicação ao seu limite para identificar o ponto de ruptura.
- **Aprofundamento/Complemento (se necessário):** A análise de carga visa garantir que o sistema possa suportar o número de usuários e transações previstos em condições normais e de pico. Já o teste de stress busca identificar a capacidade máxima do sistema, a resiliência a condições extremas e como ele se recupera de falhas. Essas análises são cruciais para planejar a infraestrutura necessária e identificar gargalos antes que a aplicação entre em produção.
- **Exemplo Prático:** Para um aplicativo de streaming de vídeo, um teste de carga simularia mil usuários simultâneos assistindo a diferentes vídeos, para verificar se o servidor consegue entregar o conteúdo sem travamentos. Um teste de stress, por outro lado, poderia simular dez mil usuários tentando acessar o mesmo vídeo ao mesmo tempo para ver como o sistema se comporta sob essa pressão extrema e onde ele falha.

Referência do Slide: Slide 06 - Otimização de recursos

- **Definição:** A otimização de recursos envolve estratégias para melhorar o uso de memória, CPU e outros recursos do sistema, como rede e disco. O objetivo é garantir que a aplicação utilize o mínimo de recursos possível para executar suas funções, aumentando a eficiência e reduzindo custos.
- **Aprofundamento/Complemento (se necessário):** A otimização de recursos é um pilar da performance. Código ineficiente, algoritmos inadequados ou o uso excessivo de recursos podem levar a lentidão, alto consumo de energia e custos de infraestrutura desnecessários. Técnicas como a refatoração de código, a escolha de estruturas de dados mais eficientes, a compressão de dados e a otimização de consultas a bancos de dados são exemplos de otimização de recursos.
- **Exemplo Prático:** Em um aplicativo mobile, a otimização de imagens, o descarte adequado de objetos na memória para evitar vazamentos (memory leaks) e a redução do número de requisições à API para carregar dados são exemplos de otimização de recursos que levam a um aplicativo mais rápido e com menor consumo de bateria.

Referência do Slide: Slide 07 - Monitoramento e relatório de desempenho

- **Definição:** Monitoramento e relatório de desempenho são o uso de ferramentas e processos para acompanhar continuamente o desempenho de uma aplicação em tempo real, coletando dados e gerando relatórios sobre métricas importantes. Isso permite identificar problemas de forma proativa e tomar decisões baseadas em dados para otimizar a performance.
- **Aprofundamento/Complemento (se necessário):** Ferramentas de Application Performance Monitoring (APM) como New Relic, Dynatrace ou Google Analytics (mencionado nos slides) são essenciais para isso. Elas fornecem dashboards com métricas de uso de CPU, memória, tempo de resposta, taxa de erros, e permitem a visualização de gargalos e a detecção de anomalias. O monitoramento contínuo é fundamental para garantir que as otimizações implementadas realmente tragam os resultados esperados e para identificar novos problemas que possam surgir.
- **Exemplo Prático:** Usar o Google Analytics para monitorar o tempo de carregamento das páginas de um site, a taxa de rejeição e o número de usuários ativos. Se o tempo de carregamento aumentar significativamente em um determinado horário, o time de desenvolvimento pode investigar a causa e aplicar correções.

Semana 11 - Aula 2

Tópico Principal da Aula: Testes funcionais e não funcionais

Subtítulo/Tema Específico: Testes não funcionais: desempenho, segurança e usabilidade

Código da aula: [SIS]ANO1C3B2S11A2

Objetivos da Aula:

- Compreender os fundamentos para melhorias de usabilidade de testes em aplicação de software.
- Identificar as principais técnicas utilizadas para prevenção de invasões e como podem ser aplicadas.
- Compreender técnicas para mitigar invasões a partir de casos reais.

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Recurso audiovisual para exibição de vídeos e imagens.

Exposição do Conteúdo:

Referência do Slide: Slide 04 - Segurança de aplicações: prevenção de ataques e proteção de dados

- **Definição:** A segurança de aplicações é crítica para proteger dados sensíveis e manter a confiança do usuário. A vulnerabilidade pode levar a perdas financeiras significativas e causar danos à reputação da empresa. Este tema aborda as práticas e técnicas para garantir que o software seja robusto contra ataques maliciosos e que os dados dos usuários sejam protegidos.
- **Aprofundamento/Complemento (se necessário):** A segurança não é um "extra" no desenvolvimento de software, mas uma preocupação fundamental desde o início do ciclo de vida. Ataques cibernéticos podem resultar em roubo de dados, fraude, interrupção de serviços e multas regulatórias. A implementação de uma cultura de segurança ("Security by Design") e a adoção de padrões como o OWASP Top 10 são essenciais para construir aplicações mais seguras.
- **Exemplo Prático:** Um aplicativo bancário deve ser construído com segurança como prioridade. Isso significa criptografar as comunicações, proteger as informações do usuário armazenadas, e implementar mecanismos robustos de autenticação. Se houver uma falha de segurança que exponha dados de clientes, o banco pode sofrer milhões em multas e perder a confiança de seus usuários.

Referência do Slide: Slide 05 - Testes de invasão e vulnerabilidade

- **Definição:** Testes de invasão (pentests) e de vulnerabilidade são técnicas que visam identificar e mitigar falhas de segurança em uma aplicação. Os testes de vulnerabilidade buscam por falhas conhecidas, enquanto os testes de invasão simulam ataques reais para explorar essas vulnerabilidades.
- **Aprofundamento/Complemento (se necessário):** Os testes de vulnerabilidade podem ser automatizados com ferramentas que escaneiam o código ou a aplicação em busca de padrões de falha. Já os testes de invasão são realizados por especialistas em segurança (ethical hackers) que tentam "quebrar" a segurança do sistema de forma controlada. Ambos são complementares e essenciais para uma estratégia de segurança robusta. Eles ajudam a identificar brechas antes que atacantes mal-intencionados as explorem.
- **Exemplo Prático:** Uma equipe de segurança pode usar uma ferramenta de varredura de vulnerabilidades para encontrar potenciais falhas de injeção de SQL ou cross-site scripting (XSS) em um site. Posteriormente, um pentester pode tentar explorar essas vulnerabilidades, por exemplo, inserindo código malicioso em um campo de formulário para roubar cookies de sessão de outros usuários.

Referência do Slide: Slide 06 - Autenticação e autorização

- **Definição:** Autenticação e autorização são pilares da segurança de aplicações, garantindo que apenas usuários autorizados tenham acesso aos recursos apropriados. Autenticação verifica a identidade do usuário (quem você é), enquanto autorização determina o que o usuário pode fazer (o que você pode acessar).
- **Aprofundamento/Complemento (se necessário):** Sistemas de autenticação modernos incluem múltiplos fatores (MFA), senhas fortes, biometria e single sign-on (SSO). A autorização é frequentemente implementada por meio de controle de acesso baseado em papéis (RBAC - Role-Based Access Control) ou controle de acesso baseado em atributos (ABAC - Attribute-Based Access Control), onde as permissões são concedidas com base nas funções ou características do usuário.
- **Exemplo Prático:** Ao fazer login em um sistema, a autenticação verifica seu nome de usuário e senha. Após a autenticação, a autorização entra em jogo: um usuário comum pode visualizar seu perfil, mas não pode acessar a seção de administração do sistema. Um administrador, por sua vez, teria permissão para gerenciar outros usuários e configurações.

Referência do Slide: Slide 07 - Compliance e padrões de segurança

- **Definição:** Compliance e padrões de segurança referem-se à adesão a normas de segurança e à privacidade de dados impostas por regulamentações e boas práticas da indústria. Isso é fundamental para evitar multas, proteger a reputação e garantir a conformidade legal.
- **Aprofundamento/Complemento (se necessário):** Exemplos de padrões e regulamentações incluem a Lei Geral de Proteção de Dados (LGPD) no Brasil, o General Data Protection Regulation (GDPR) na Europa, e padrões específicos da indústria como PCI DSS para transações com cartão de crédito. A conformidade exige a implementação de políticas, processos e controles de segurança, além de auditorias regulares.
- **Exemplo Prático:** Uma empresa que processa dados de saúde de pacientes deve estar em conformidade com regulamentações específicas do setor (como HIPAA nos EUA, por exemplo) e LGPD no Brasil. Isso significa que o software deve ser projetado para criptografar dados sensíveis, restringir o acesso a informações confidenciais e manter registros de auditoria de quem acessou o quê. A não conformidade pode resultar em penalidades severas.

Semana 11 - Aula 3

Tópico Principal da Aula: Testes funcionais e não funcionais

Subtítulo/Tema Específico: Testes não funcionais: desempenho, segurança e usabilidade

Código da aula: [SIS]ANO1C3B2S11A3

Objetivos da Aula:

- Compreender os fundamentos para melhorias de usabilidade em testes de aplicação de software.
- Compreender a importância do desenvolvimento voltado para a acessibilidade, uma vez que o software precisa ser pensado de maneira completa e inclusiva.
- Na prática, a partir de um cenário, entender quais ações de software podem ser utilizadas para sua compreensão e desenvolvimento.

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Recurso audiovisual para exibição de vídeos e imagens.

Exposição do Conteúdo:

Referência do Slide: Slide 04 - Melhoria da experiência do usuário por meio de testes de usabilidade

- **Definição:** A melhoria da experiência do usuário (UX - User Experience) por meio de testes de usabilidade impacta diretamente a satisfação e a eficiência com que os usuários interagem com a aplicação. Uma boa usabilidade pode aumentar a retenção e a satisfação do cliente, tornando o software intuitivo e fácil de usar.
- **Aprofundamento/Complemento (se necessário):** Testes de usabilidade não focam apenas na funcionalidade ("o que o sistema faz"), mas em "como o sistema faz". Eles buscam entender se os usuários conseguem atingir seus objetivos de forma eficiente, eficaz e satisfatória. Uma interface confusa ou difícil de navegar pode levar ao abandono do produto, mesmo que ele tecnicamente funcione.
- **Exemplo Prático:** Em um aplicativo de delivery de comida, se o processo de seleção de pratos, adição ao carrinho e checkout for complicado e demorado, os usuários podem desistir da compra. Testes de usabilidade revelariam esses pontos de atrito, permitindo que a equipe de desenvolvimento simplifique a jornada do usuário.

Referência do Slide: Slide 05 - Avaliação heurística

- **Definição:** Avaliação heurística é uma análise baseada em princípios estabelecidos de design de UX (User Experience). Especialistas avaliam a interface de um sistema comparando-a com um conjunto de "heurísticas" (regras de ouro) para identificar problemas de usabilidade.
- **Aprofundamento/Complemento (se necessário):** As heurísticas de Nielsen (Visibility of System Status, Match Between System and Real World, User Control and Freedom, Consistency and Standards, Error Prevention, Recognition Rather Than Recall, Flexibility[1] and Efficiency of Use, Aesthetic and Minimalist Design, Help Users Recognize, Diagnose, and Recover from Errors, Help and Documentation) são as mais conhecidas. Essa avaliação é rápida e pode ser feita no início do projeto, ajudando a identificar problemas de design antes de gastar muito tempo e recursos no desenvolvimento.
- **Exemplo Prático:** Um avaliador heurístico poderia analisar um formulário de cadastro online e identificar que ele não fornece feedback claro sobre o status de preenchimento (ex: "Você está no passo 2 de 5"). Isso viola a heurística de "Visibilidade do Status do Sistema", indicando um problema de usabilidade.

A avaliação heurística(***deriva do grego "heuriskein", que significa "encontrar" ou "descobrir"***), no contexto do desenvolvimento de software e, mais especificamente, da usabilidade, é um método de inspeção de usabilidade onde especialistas avaliam uma interface de usuário para identificar problemas de usabilidade. Essa avaliação é baseada em um conjunto de "heurísticas" (princípios ou regras gerais) estabelecidas de design de UX (User Experience - Experiência do Usuário).

Em resumo:

- **O que é:** Um método de avaliação da usabilidade de um sistema (como um software, site ou aplicativo) realizado por especialistas.
- **Como funciona:** Os avaliadores examinam a interface e julgam se ela segue princípios de usabilidade reconhecidos.
- **Objetivo:** Identificar problemas de usabilidade que podem dificultar a interação dos usuários com o sistema, antes mesmo de realizar testes com usuários reais.

Aprofundamento/Complemento:

As heurísticas mais conhecidas e amplamente utilizadas foram propostas por Jakob Nielsen em 1990 (e revisadas em 1994). As dez heurísticas de Nielsen são:

1. **Visibilidade do status do sistema:** O sistema deve sempre manter os usuários informados sobre o que está acontecendo, por meio de feedback apropriado dentro de um tempo razoável.[1]
2. **Correspondência entre o sistema e o mundo real:** O sistema deve falar a linguagem do usuário, com palavras, frases e conceitos familiares, em vez de termos técnicos. Seguir convenções do mundo real, tornando as informações em uma ordem natural e lógica.
3. **Controle e liberdade do usuário:** Os usuários frequentemente realizam ações por engano. Eles precisam de uma "saída de emergência" claramente marcada para sair da ação indesejada sem ter que passar por um diálogo extenso. Apoiar as funções de desfazer e refazer.
4. **Consistência e padrões:** Os usuários não deveriam ter que questionar se diferentes palavras, situações ou ações significam a mesma coisa. Siga as convenções da plataforma.
5. **Prevenção de erros:** É melhor um design cuidadoso que previna a ocorrência de problemas em primeiro lugar do que boas mensagens de erro.
6. **Reconhecimento em vez de memorização:** Minimize a carga de memória do usuário tornando objetos, ações e opções visíveis. O usuário não deve ter que se lembrar de informações de uma parte[2] para outra do diálogo.
7. **Flexibilidade e eficiência de uso:** Permitir que usuários novatos e experientes usem o sistema de maneira eficaz. Fornecer aceleradores para usuários experientes (atalhos, personalização).
8. **Estética e design minimalista:** Diálogos não devem conter informações irrelevantes ou raramente necessárias. Toda informação extra em um diálogo compete com as unidades de informação relevantes e diminui sua visibilidade relativa.
9. **Ajude os usuários a reconhecer, diagnosticar e recuperar-se de erros:** Mensagens de erro devem ser expressas em linguagem clara e simples, indicar precisamente o problema e sugerir uma solução construtiva.
10. **Ajuda e documentação:** Embora seja melhor que o sistema não precise de nenhuma documentação adicional, pode ser necessário fornecer ajuda e documentação. Qualquer ajuda deve ser fácil de pesquisar, focada na tarefa do usuário, listar passos concretos a serem realizados e não ser muito grande.

Exemplo Prático:

Imagine que você está avaliando um novo aplicativo de banco. Durante uma avaliação heurística, um especialista em usabilidade pode, ao tentar realizar uma transferência bancária:

- **Aplicar a heurística "Visibilidade do status do sistema":** Observar se o aplicativo exibe uma mensagem de carregamento durante o processamento da transação ou se o usuário fica sem saber se a ação foi registrada.
- **Aplicar a heurística "Controle e liberdade do usuário":** Verificar se há um botão "Cancelar" claro em todas as etapas da transferência, permitindo que o usuário desista da operação a qualquer momento.
- **Aplicar a heurística "Prevenção de erros":** Checar se o aplicativo solicita a confirmação do valor da transferência antes de finalizar, ou se ele impede a inserção de caracteres inválidos nos campos de conta e agência.

Ao final da avaliação, o especialista compila uma lista de problemas de usabilidade encontrados, classificando-os por gravidade, o que serve como base para melhorias no design da interface.

Referência do Slide: Slide 06 - Testes com usuários reais

- **Definição:** Testes com usuários reais envolvem a coleta de feedback direto dos usuários enquanto eles interagem com a aplicação. É uma forma eficaz de identificar problemas de usabilidade que talvez não sejam evidentes para os desenvolvedores ou designers.
- **Aprofundamento/Complemento (se necessário):** Esses testes podem ser moderados (com um facilitador observando e fazendo perguntas) ou não moderados (onde os usuários interagem livremente). Eles podem ser realizados em laboratório, em campo ou remotamente. O feedback qualitativo obtido é extremamente valioso para entender as dificuldades e preferências dos usuários.
- **Exemplo Prático:** Convidar 5-10 usuários representativos do público-alvo para realizar tarefas específicas em um novo site (ex: "encontre um produto e adicione ao carrinho"). Enquanto eles realizam as tarefas, observe suas ações, expressões e peça que pensem em voz alta. Isso pode revelar que um botão importante não está visível ou que o fluxo de navegação é confuso para eles.

Test-Driven Development (TDD), ou Desenvolvimento Orientado a Testes, é uma abordagem de desenvolvimento de software onde os testes são escritos antes do código real. É um ciclo iterativo que geralmente segue os passos "Red-Green-Refactor" (Vermelho-Verde-Refatorar):

- **Red (Vermelho)**: Você escreve um teste unitário para uma nova funcionalidade que ainda não existe. O teste deve falhar.
- **Green (Verde)**: Você escreve apenas o código mínimo necessário para fazer o teste passar.
- **Refactor (Refatorar)**: Uma vez que o teste passa, você otimiza e refatora o código recém-escrito, garantindo que ele seja limpo, eficiente e mantenha a

conformidade com os critérios de simplicidade, sem quebrar os testes existentes.

Esse ciclo se repete para cada nova funcionalidade até que o produto esteja completo.

Vantagens do TDD:

- **Cobertura Abrangente de Testes**: Garante que todo o código novo seja coberto por pelo menos um teste, levando a um software mais robusto.
- **Confiança no Código**: Desenvolvedores ganham maior confiança na confiabilidade e funcionalidade do código.
- **Código Bem Documentado**: O processo naturalmente resulta em código bem documentado, pois cada teste esclarece o propósito do código que ele testa.
- **Clareza de Requisitos**: Incentiva uma compreensão clara dos requisitos antes de iniciar a codificação.
- **Redução de Bugs**: Ajuda a identificar e corrigir erros em estágios iniciais do desenvolvimento, minimizando a frequência e a gravidade dos bugs.
- **Design Aprimorado**: Força o desenvolvedor a pensar na interface e no design do código, resultando em um design mais modular e testável.

TDD é uma metodologia ágil que transforma a abordagem de testes, tornando-a um processo contínuo e iterativo, em vez de ocorrer apenas no final do ciclo de desenvolvimento.

Para mais informações, você pode consultar as seguintes fontes:

- Test-driven development - Wikipedia (em inglês)
- Test-driven development (TDD) explained - CircleCI (em inglês)
- What is Test Driven Development (TDD)? - Agile Alliance (em inglês)
- Test-driven development – Wikipédia, a enciclopédia livre (em português)

Referência do Slide: Slide 07 - Acessibilidade e inclusão

- **Definição:** Acessibilidade e inclusão em software significam garantir que a aplicação possa ser usada por todas as pessoas, independentemente de suas habilidades físicas, sensoriais ou cognitivas. Isso envolve projetar e desenvolver o software para ser utilizável por pessoas com deficiência.
- **Aprofundamento/Complemento (se necessário):** A acessibilidade não é apenas uma questão de conformidade legal ou ética, mas também de mercado. Uma aplicação acessível atinge um público maior e demonstra responsabilidade social. Isso inclui o uso de contraste de cores adequado, suporte a leitores de tela, navegação por teclado, legendas em vídeos, entre outros. As Web Content Accessibility Guidelines (WCAG) são um padrão internacional importante.

- **Exemplo Prático:** Um site de notícias que não possui texto alternativo (alt text) para as imagens torna-se inacessível para usuários com deficiência visual que utilizam leitores de tela. Da mesma forma, um formulário que exige apenas o uso do mouse para ser preenchido impede que usuários com deficiência motora, que usam apenas o teclado, consigam interagir com ele. Garantir que esses elementos sejam acessíveis torna o site inclusivo.