UNIVERSIDADE DO ESTADO DO PARÁ – UEPA
CENTRO DE CIÊNCIAS NATURAIS E TECNOLOGIA – CCNT
CURSO DE BACHARELADO EM ENGENHARIA DE SOFTWARE

Discente: Antonio Wanzeler Neto.

Docente: Fabrício Farias.

ATIVIDADE DE TESTE DE SOFTWARE

1. ESTUDO APROFUNDADO DE TESTES DE SOFTWARE: ABORDAGEM TEÓRICA

1.1 VISÃO GERAL DOS TIPOS DE TESTE

O teste de software constitui uma fase crítica no processo de desenvolvimento, destinada a avaliar a qualidade, o desempenho e a confiabilidade de sistemas computacionais. A literatura especializada classifica os testes em duas categorias principais: testes funcionais e não funcionais, cada qual com objetivos, abordagens e métricas distintas, porém complementares no processo de garantia de qualidade.

2. TESTES FUNCIONAIS: CONCEITOS FUNDAMENTAIS

2.1 DEFINIÇÃO E ABRANGÊNCIA

Os testes funcionais representam uma abordagem de verificação que examina se os componentes de software operam em conformidade com os requisitos funcionais especificados. Esta modalidade de teste concentra-se no comportamento externo do sistema, avaliando as funcionalidades oferecidas ao usuário final sem considerar a implementação interna.

Segundo Pressman (2016), o teste funcional tem como objetivo primordial “demonstrar que as funções do software são operacionais, que a entrada é propriamente aceita e que a saída é corretamente produzida”. Esta abordagem alinha-se estreitamente com a experiência do usuário, validando se o sistema executa as operações esperadas conforme definido nos requisitos.

2.2 UNIT TESTING (TESTE DE UNIDADE)

2.2.1 Conceituação Teórica

O teste de unidade constitui a base da pirâmide de testes, focalizando a menor unidade testável do software - tipicamente funções, métodos ou classes isoladas. Myers, Sandler e Badgett (2011) definem teste de unidade como “o processo de exercitar unidades individuais de código fonte para verificar se desempenham corretamente suas funções designadas”.

A fundamentação teórica dos testes unitários assenta no princípio do isolamento, onde cada unidade é testada independentemente de dependências externas, utilizando técnicas como mocks, stubs e dummies para simular comportamentos necessários.

2.2.2 Objetivos e Benefícios

A implementação sistemática de testes unitários proporciona múltiplos benefícios, incluindo:

- **Detecção Precoce de Defeitos**: Identificação de problemas na fase inicial de desenvolvimento
- **Facilitação de Refatoração**: Segurança para modificar código com confiança
- **Documentação Viva**: Servem como especificação executável do comportamento esperado
- **Design Melhorado**: Incentivam o desenvolvimento de código modular e coeso

2.2.3 Características de Testes Unitários Eficazes

Kaner, Falk e Nguyen (1999) estabelecem que testes unitários eficazes devem exibir:

- **Atomicidade**: Cada teste verifica uma única condição ou comportamento
- **Independência**: Testes não possuem dependências entre si
- **Rapidez**: Execução em milissegundos para permitir feedback contínuo
- **Determinismo**: Resultados consistentes independentemente do ambiente ou ordem de execução

2.3 INTEGRATION TESTING (TESTE DE INTEGRAÇÃO)

2.3.1 Fundamentação Conceitual

O teste de integração surge como etapa subsequente aos testes unitários, focando na interação entre componentes ou módulos integrados. Beizer (2003) caracteriza este teste como “a verificação das interfaces e interações entre componentes integrados para assegurar que cooperam conforme especificado”.

Esta modalidade aborda problemas emergentes que não se manifestam durante o teste de unidades isoladas, tais como incompatibilidades de interface, violação de pre/post-condições e problemas de timing.

2.3.2 Abordagens de Integração

A literatura especializada descreve diversas estratégias de integração:

- **Integração Big Bang**: Todos os componentes integrados simultaneamente
- **Integração Incremental**: Integração progressiva seguindo hierarquia de componentes
    - **Top-Down**: Integração a partir dos módulos de nível mais alto
    - **Bottom-Up**: Integração iniciando pelos módulos de nível mais baixo
    - **Sandwich**: Combinação das abordagens top-down e bottom-up

2.3.3 Desafios e Complexidades

O teste de integração apresenta desafios específicos, incluindo:

- **Gerenciamento de Dependências**: Coordenação entre times de diferentes componentes
- **Configuração de Ambiente**: Necessidade de ambientes que simulem integrações reais
- **Debug Complexo**: Dificuldade em isolar defeitos em sistemas distribuídos

3. TESTES NÃO FUNCIONAIS: PERSPECTIVA TEÓRICA

3.1 ENQUADRAMENTO CONCEITUAL

Os testes não funcionais distinguem-se por avaliar atributos de qualidade que transcendem o comportamento funcional do sistema. Conforme definido por ISO/IEC 25010, estes testes avaliam características como desempenho, confiabilidade, usabilidade, segurança e mantenabilidade.

Amber (2014) salienta que “enquanto testes funcionais verificam se o sistema faz o que deve fazer, testes não funcionais avaliam quão bem o sistema executa essas funcionalidades”.

3.2 LOAD TESTING (TESTE DE CARGA)

3.2.1 Base Teórica

O teste de carga avalia o comportamento do sistema sob condições normais e de pico de utilização, medindo métricas de desempenho como tempo de resposta, throughput e utilização de recursos. Jain (1991) define teste de carga como “a prática de modelar o uso

esperado do software simulando múltiplos usuários acessando o sistema concurrentemente".

3.2.2 Objetivos Principais

Os testes de carga visam:

- **Identificar Gargalos**: Detecção de limitações de performance antes da produção
- **Validar Capacidade**: Verificação se o sistema suporta volumes esperados de usuários
- **Otimizar Alocação de Recursos**: Determinação da infraestrutura necessária
- **Estabelecer Linhas de Base**: Criação de métricas de performance para comparação futura

3.2.3 Metodologias e Abordagens

A implementação de testes de carga segue metodologias estruturadas:

- **Modelagem de Carga**: Definição de perfis de usuário e padrões de uso realistas
- **Instrumentação**: Coleta de métricas de performance durante a execução
- **Análise de Resultados**: Interpretação de dados para identificar problemas de performance

3.3 STRESS TESTING (TESTE DE ESTRESSE)

3.3.1 Fundamentação Teórica

O teste de estresse submete o sistema a condições extremas além dos limites operacionais normais, visando avaliar sua robustez e capacidade de recuperação. Segundo Weyuker e Vokolos (2000), "testes de estresse examinam o comportamento do software sob condições anormais para verificar como lida com falhas e se recupera de situações de overload".

3.3.2 Objetivos Específicos

Esta modalidade de teste busca:

- **Avaliar Resiliência**: Verificar como o sistema comporta-se sob condições extremas

- **Identificar Pontos de Falha**: Descobrir componentes que falham catastropicamente
- **Testar Mecanismos de Recuperação**: Validar procedimentos de fallback e recovery
- **Preparar para Cenários Extremos**: Antecipar comportamentos em situações de pico inesperado

3.3.3 Técnicas e Abordagens

Os testes de estresse empregam diversas técnicas:

- **Spike Testing**: Aplicação súbita de carga extrema
- **Soak Testing**: Aplicação de carga sustentada por períodos prolongados
- **Failover Testing**: Simulação de falhas de componentes críticos

4. CONSIDERAÇÕES FINAIS TEÓRICAS

A literatura especializada evidencia que testes funcionais e não funcionais representam dimensões complementares e igualmente essenciais para a garantia de qualidade de software. Enquanto os testes funcionais validam a correção lógica do sistema em relação aos requisitos especificados, os testes não funcionais avaliam atributos de qualidade que determinam a experiência prática de uso e a robustez operacional.

A implementação balanceada de ambas as abordagens, seguindo princípios estabelecidos na literatura de engenharia de software, constitui prática recomendada para o desenvolvimento de sistemas confiáveis, eficientes e adequados aos propósitos para os quais foram concebidos.

REFERÊNCIAS BIBLIOGRÁFICAS

PRESSMAN, R. S. Engenharia de Software: Uma Abordagem Profissional. 8. ed. Porto Alegre: AMGH, 2016.

MYERS, G. J.; SANDLER, C.; BADGETT, T. The Art of Software Testing. 3. ed. Hoboken: John Wiley & Sons, 2011.

KANER, C.; FALK, J.; NGUYEN, H. Q. Testing Computer Software. 2. ed. New York: Wiley, 1999.

BEIZER, B. Software Testing Techniques. 2. ed. Boston: Thomson Learning, 2003.

ISO/IEC 25010: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models, 2011.

AMBER, S. W. Fundamental of Software Testing. New Delhi: Prentice Hall India, 2014.

JAIN, R. The Art of Computer Systems Performance Analysis. New York: Wiley, 1991.

WEYUKER, E. J.; VOKOLOS, F. I. Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study. IEEE Transactions on Software Engineering, v. 26, n. 12, p. 1145-1154, 2000.