Início Rápido em Teste e QA - Resumo e pontos (Seção 2)

Uma Breve História do Teste

A história dos testes de software reflete a **busca contínua por qualidade e a superação de falhas inerentes à natureza humana.** Desde Charles Babbage, que criou a primeira máquina automática de somar, até o crescimento da área nos anos 2000, a evolução dos testes acompanhou o avanço da tecnologia. A IBM teve um papel crucial na consolidação da computação, enquanto Glenn Ford Marx, com *A Arte de Testar Software*, se tornou referência na área. A regra dos "10x" destaca que quanto mais tarde um defeito é encontrado, maior o custo para corrigi-lo. No Brasil, as primeiras certificações impulsionaram a profissionalização do setor. Conhecer essa trajetória ajuda a evitar erros do passado e aprimorar as práticas de teste.

A Importância dos Testes - Danos e Bugs

Os testes de software são fundamentais para evitar **prejuízos financeiros**, danos à reputação e até riscos à vida. Bugs podem comprometer **negócios**, atrasando entregas e gerando perdas de confiança. No **setor público**, falhas expõem dados sigilosos, comprometem decisões estratégicas e facilitam ataques cibernéticos. No **setor militar**, erros podem causar falhas de comunicação e perdas estratégicas. Problemas em sistemas também afetam serviços essenciais, resultando em **falhas no transporte**, na saúde e na distribuição de direitos. Em casos mais graves, defeitos em sistemas críticos, como **controle aéreo** e **equipamentos médicos**, podem ser fatais. Além disso, **falhas ambientais** podem agravar desastres naturais, reforçando a necessidade de um controle rigoroso de qualidade no desenvolvimento de software.

Introdução ao Teste de Software

Princípios Fundamentais do QA

Testes Contínuos : Os testes devem ser iniciados desde as primeiras fases do projeto, garantindo evolução junto ao desenvolvimento, identificando e corrigindo falhas cedo, reduz e minimiza retrabalho.

Correção Preventiva : Erros devem ser corrigidos rapidamente para evitar que se tornem falhas maiores. Bem como um bug não tratado pode desencadear falhas em cascata, impactando outras funcionalidades.

Os 7 Fundamentos do Teste (ISTQB)

O primeiro fundamento do teste de software afirma que ele pode demonstrar a *presença de defeitos, mas nunca sua ausência*. Isso significa que, por mais que um software seja testado, sempre pode existir um erro não identificado.

***O teste não garante ausência de defeitos**:* Sempre pode haver um erro não identificado, independentemente da quantidade de testes realizados.

- *O processo de correção pode gerar novos defeitos:* Ao corrigir erros, outros problemas podem surgir, criando um ciclo contínuo de testes e ajustes.
- *Softwares críticos exigem testes exaustivos:* Em sistemas que envolvem riscos de vida, os testes continuam até que se alcance um nível aceitável de segurança.
- *Testes aumentam a qualidade, mas não garantem perfeição:* O objetivo é reduzir riscos e melhorar a experiência do usuário, mas nunca será possível garantir um software com "zero defeitos".

***Teste exaustivo não é possível**:* A quantidade de combinações de testes pode ser gigantesca, inviabilizando a verificação completa de um software.

- *Impossibilidade de tempo e dinheiro:* Testar todas as combinações possíveis seria inviável em termos de recursos e prazos.
- *Importância da priorização:* Testes devem focar no que é mais importante, mais frequente e com maior impacto.

Critérios de priorização:

- *Mais usado:* Testar primeiro os recursos e combinações mais utilizados pelos usuários.
- *Maior risco:* Priorizar testes onde os erros podem causar maior impacto ou prejuízo.
- *Foco na qualidade e eficiência:* Não é sobre testar tudo, mas sobre testar o que realmente importa para garantir um software funcional e confiável.

***Teste antecipado**:* Testar o mais cedo possível para evitar que problemas cresçam e fiquem mais caros de corrigir.

- *Defeitos se multiplicam:* Um "*bug*", um erro pode se espalhar pelo sistema e gerar mais falhas.
- Quanto antes um erro for identificado, menor será o custo de correção.
- Erros encontrados na fase de design custam menos do que na especificação, codificação, testes ou pós-produção.
- Erros descobertos pelo cliente são os mais caros.

***Agrupamento de defeitos**:* Bugs tendem a se concentrar em certas áreas do software, não sendo distribuídos de forma homogênea.

- *Partes mais problemáticas:* Áreas mais complexas, frequentemente alteradas ou críticas costumam ter mais defeitos.
- *Monitoramento de reclamações:* Áreas com maior índice de problemas relatados pelos clientes merecem mais atenção.
- *Foco nos riscos:* Testar mais intensamente partes do sistema onde falhas podem gerar grandes prejuízos.

***Definição de paradoxo**:* Relação em que uma ação esperada para gerar um efeito positivo pode, na verdade, causar um efeito negativo.

- *Paralelo com antibióticos:* Uso incorreto gera bactérias resistentes, tornando os medicamentos menos eficazes.
- É essencial revisar e atualizar os cenários de teste para detectar novas falhas.

Dependência do Contexto

- Cada software exige abordagens de teste específicas.
- O ambiente, arquitetura e perfil dos usuários influenciam diretamente na estratégia adotada.

Ilusão de Ausência de Erros

- A ausência de falhas nos testes não significa que o sistema esteja totalmente livre de erros.
- É necessário diversificar os testes para garantir uma cobertura mais eficaz.

Tipos de Testes Baseados na IEC/ISO 25010

Adequação Funcional

- Avalia se o software atende corretamente aos requisitos definidos.
- Critérios: Completude, Correção e Apropriação.

Usabilidade

- Mede a facilidade de uso e experiência do usuário.
- Critérios: Reconhecibilidade, Aprendizibilidade, Operabilidade, Proteção contra erro do usuário, Estética da interface e Acessibilidade.

Compatibilidade

- Verifica se o sistema funciona corretamente em diferentes ambientes e dispositivos.
- Critérios: Coexistência e Interoperabilidade.

Confiança

- Mede a estabilidade do sistema perante falhas inesperadas.
- Critérios: Maturidade, Disponibilidade, Tolerância a falhas e Recuperabilidade.

Eficiência no Desempenho

- Avalia a resposta do software sob diferentes condições de uso.
- Critérios: Comportamento em relação ao tempo, Utilização de recursos e Capacidade.

Manutenibilidade

- Mede a facilidade de manutenção e atualização do software.
- Critérios: Modularidade, Reusabilidade, Analisabilidade, Modificabilidade e Testabilidade.

Portabilidade

- Avalia a adaptabilidade do software a diferentes plataformas.
- Critérios: Adaptabilidade, Instalabilidade e Substitubilidade.

Segurança

- Garante a proteção contra ameaças e acessos indevidos.
- Critérios: Confidencialidade, Integridade, Não repúdio, Responsabilidade e Autenticidade.

Erro, Ocorrência, Defeito e Falha: Entendendo a Escalada de Problemas no Desenvolvimento de Software

- **Erro**

Um engano cometido por um desenvolvedor ou analista durante o desenvolvimento do software. Pode ocorrer por falta de conhecimento, desatenção ou erro de lógica.

Exemplo: *Um programador implementa uma função para calcular médias, mas esquece de dividir pelo número total de elementos.*

- **Ocorrência**

Um evento registrado no sistema, que pode ou não ser um problema real. Pode envolver alertas, logs e notificações sobre o comportamento do software.

Exemplo: *Um sistema de monitoramento registra um pico anormal de uso de memória. Isso pode indicar um problema, mas também pode ser apenas um comportamento esperado.*

- **Defeito (Bug)**

A manifestação de um erro no código, resultando em um comportamento incorreto do software. Nem todo erro gera um defeito imediatamente, mas quando um erro é codificado e afeta o funcionamento, ele se torna um defeito.

Exemplo: *O erro na função de cálculo de médias faz com que o sistema retorne um valor maior do que o esperado, impactando os relatórios financeiros.*

- **Falha**

Ocorre quando um defeito é acionado em condições reais de uso, afetando a experiência do usuário ou a operação do sistema.

Exemplo: *O erro de cálculo nos relatórios financeiros gera valores errados, levando a decisões erradas e prejuízo financeiro para a empresa.*

Relato de um Problema Real com Erro, Ocorrência, Defeito e Falha

Cenário

Uma empresa financeira desenvolveu um sistema para calcular juros de investimentos de seus clientes. Durante o desenvolvimento, um dos programadores cometeu um **erro** ao implementar a fórmula de juros compostos.

O desenvolvedor esqueceu de converter a taxa de juros para a base correta antes de aplicá-la ao cálculo. Em vez de dividir a taxa anual por 12 para obter a taxa mensal, ele utilizou a taxa anual diretamente na equação.

Erro no código:

```
juros = capital * (1 + taxa_anual) ** meses  # Deveria ser taxa_anual / 12
```

Ocorrência

O sistema de monitoramento registrou um aumento inesperado nos valores de juros processados. No entanto, nenhum alarme crítico foi disparado, pois os valores ainda estavam dentro de um intervalo aceitável.

Log da ocorrência:

```
[INFO] Valores de juros processados acima da média esperada. Verificar comportamento.
```

Defeito

Após alguns clientes relatarem que seus rendimentos estavam muito altos, a equipe de QA testou a funcionalidade e encontrou a inconsistência no cálculo. O **defeito** foi identificado como a aplicação incorreta da taxa de juros, resultando em valores superestimados.

Comportamento esperado vs. Real:

Capital	Taxa Anual	Meses	Valor Correto	Valor Errado (Bug)
R$ 10.000	12%	12	R$ 11.268,25	R$ 31.210,00

Falha

O erro só foi identificado depois que clientes começaram a sacar valores muito maiores do que deveriam. Isso gerou um grande prejuízo para a empresa, levando à suspensão temporária da plataforma e a uma auditoria nos cálculos financeiros.

Impacto:
A empresa precisou reverter transações, comunicar os clientes sobre o erro e corrigir os relatórios financeiros. Além disso, sofreu um dano à reputação e possíveis processos por clientes afetados.

Resumo

- **Erro**: Engano na fórmula do cálculo de juros.
- **Ocorrência**: Registro de valores inesperados no log do sistema.
- **Defeito**: Código incorreto gerando cálculos errados.
- **Falha**: Impacto real nos saques dos clientes e prejuízo financeiro.

Esse tipo de situação demonstra como um simples erro pode escalar até uma falha grave se não for detectado a tempo.

Tipos de Testes

Teste Funcional

- Verifica se o sistema atende aos requisitos esperados e funciona corretamente.

Teste Não-Funcional

- Avalia aspectos como desempenho, usabilidade e segurança.

Teste Manual

- Executado por testadores sem o uso de scripts, sendo útil para testes exploratórios e de UI/UX.

Teste Automatizado

- Utiliza scripts e ferramentas para validar funcionalidades repetitivas, sendo essencial para testes regressivos, de carga e desempenho.

Metodologias de Teste

Testes Tradicionais

- Seguem processos estruturados e documentados, geralmente aplicados em metodologias como Waterfall.

Testes Ágeis

- Incorporam testes contínuos e automação para acompanhar o ritmo do desenvolvimento ágil, como no Scrum e DevOps.

Uma estratégia de testes bem definida não apenas melhora a qualidade do software, mas também reduz custos, aumenta a confiabilidade e proporciona uma melhor experiência para o usuário.

Comparação Resumida

Testes Tradicionais	Testes Ágeis
Ocorrem após o desenvolvimento	Ocorrem continuamente
Separação entre devs e testadores	Testes feitos em colaboração
Planos de testes rígidos	Flexibilidade para mudanças
Pouca automação	Uso intenso de testes automatizados

Conclusão: Os testes tradicionais são mais formais e estruturados, enquanto os testes ágeis são rápidos, contínuos e colaborativos. No cenário moderno, os testes ágeis são mais comuns, pois reduzem falhas e aceleram a entrega de software.