APOSTILA

Módulo 4:
Teste de Software

Curso de Treinamento Específico para Programadores

Sumário

1 Teste de Software

"*Testar é analisar um programa com a intenção de descobrir erros e defeitos.*" (Myers)

"*O teste de programas pode ser usado para mostrar a presença de defeitos, mas nunca para mostrar a sua ausência.*" (Dijkstra)

Testar é medir a qualidade e funcionalidade de um sistema.

1.1 Fundamentos do Teste de Software

Objetivos do Teste:

1. Reduzir a probabilidade de incidência de erro no cliente;
2. Minimizar riscos ao negócio do cliente;
3. Atender as necessidades do cliente (negócio, contratual, legal, etc.);
4. Resultando na maior satisfação do cliente.

Conceitos:

No que se refere às definições da disciplina de teste, vale ressaltar que não existe um conjunto universal de definições relativas ao teste de software. Alguns conceitos são importantes para o entendimento e possíveis adaptações dos processos, fazendo parte da base para a disciplina de teste. Nesse contexto, é imprescindível entender o conceito de erro, defeito e falha.

Embora esses termos e definições estejam diretamente ligados ao processo de desenvolvimento de software, vários profissionais da área de TI desconhecem o seu significado e suas respectivas diferenças.

a) Erro: trata-se de uma ação humana (ex.: não entendimento de como executar um cálculo)

b) Defeito: Causado por um erro de entendimento (ex.: código com fórmula de cálculo mal escrita)

c) Falha: Tentativa de execução de um defeito. (ex.: execução de um cálculo gerando resultados indevidos)

Falha é um evento; defeito é um estado do software, causado por um erro.

Os defeitos podem ser originados de vários fatores, tais como:

- Pressão dos interessados no software;

- Prazos de atendimento às demandas inadequados;
- Utilização de tecnologia inadequada;
- Falta de habilidade da equipe;
- Não entendimento das necessidades do cliente;
- Fator humano.

Os defeitos ocorrem no software porque os mesmos são escritos por pessoas, e por sua vez sabemos que as pessoas (profissionais) não conhecem e não dominam tudo. Entretanto, elas têm habilidades para tal responsabilidade, mas também não são perfeitas, o que nos leva a admitir que são suscetíveis de cometer erros. E por último, softwares são desenvolvidos sobre crescente pressão para entregá-los em prazos rigorosos, sem tempo para checar as atividades realizadas.

Por que testar?

Conforme afirmado em vários relatos de experiência e já percebido no mercado, os reais benefícios são a redução em 70% do índice de retrabalho na correção de falhas em produção, redução em 50% do tempo de homologação de uma nova versão. Além disso, aumenta em aproximadamente 90% o índice de falhas detectadas antes da produção onde o custo é bem mais baixo, e aumenta a abrangência dos testes. Portanto, de uma forma geral, os maiores benefícios são:

- Motivação por maior segurança aos clientes;
- Oferecer maior continuidade do serviço ao negócio do cliente;
- Melhoria da qualidade dos softwares;
- Busca pela confiabilidade do software junto aos clientes;
- Visando redução de gastos em correção de bugs.

Custo:

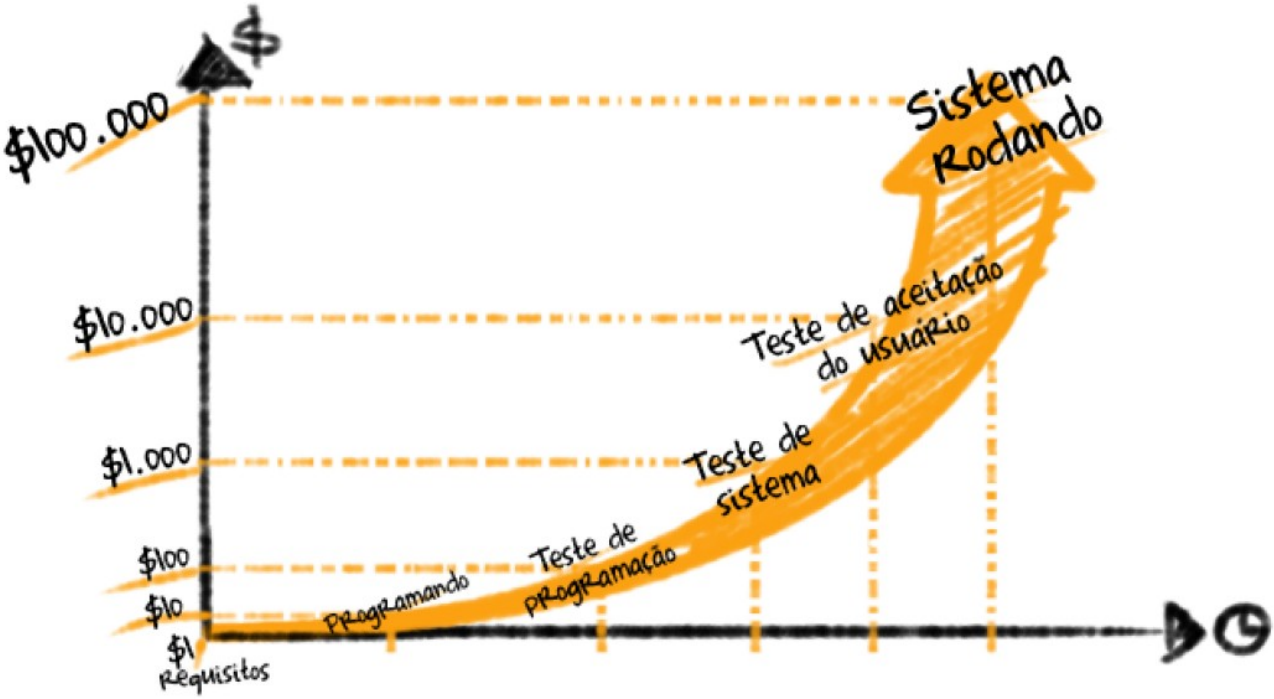
Para analisar os impactos das falhas encontradas no software é necessário avaliar o impacto de quando os erros foram encontrados. Portanto, o impacto de se encontrar e consertar os defeitos aumenta consideravelmente ao longo do tempo.

A regra 10 de Myers apresenta que o custo da correção de um defeito tende a aumentar quanto mais tarde ele for encontrado.

Os defeitos encontrados nas fases iniciais do projeto de desenvolvimento do software são mais baratos do que aqueles encontrados na produção.

Um estudo realizado observou que, quanto mais tempo levarmos para corrigir uma falha, mais custosa ela será. O gráfico apresentado tem como objetivo mostrar a proporção do aumento, onde, ainda na fase de levantamento de requisitos, o custo para

correção é considerado muito baixo, praticamente zero. Na fase de desenvolvimento do sistema, o custo aumenta em 10 vezes comparado com a fase anterior. Já na fase de teste unitário e teste de integração, o custo é de 100 vezes o custo inicial. Enquanto que na fase do teste de sistema, o custo será de mil (1000) vezes. O teste de aceitação custará 10 mil e quando o software está em produção, estima-se que o custo para correção de uma falha é de 100 mil vezes o custo, se a mesma fosse ajustada na fase de levantamento de requisitos (ver figura abaixo).



A Psicologia do Teste:

A forma de pensar utilizada enquanto se está testando e revisando é diferente da utilizada enquanto se está analisando e desenvolvendo. Com a sua forma de pensar, os desenvolvedores

estão aptos a testarem seus próprios códigos, mas a separação desta responsabilidade para um testador é tipicamente feita para ajudar a focalizar o esforço e prover benefícios adicionais, como uma visão independente, profissional e treinada de recursos de teste. Teste independente pode ser considerado em qualquer nível de teste.

Certo grau de independência (evitando a influência do autor) muitas vezes representa uma forma eficiente de encontrar defeitos e falhas. Independência não significa simplesmente uma substituição, tendo em vista que os desenvolvedores podem encontrar defeitos no código de maneira eficiente. Níveis de independência podem ser definidos como:

- Teste elaborado por quem escreveu o software que será testado (baixo nível de independência).
- Teste elaborado por outra(s) pessoa(s) (por exemplo, da equipe de desenvolvimento).
- Teste elaborado por pessoa(s) de um grupo organizacional diferente (ex: equipe independente de teste).
- Teste elaborado por pessoa(s) de diferentes organizações ou empresas (terceirizada ou certificada por um órgão externo).

Pessoas e projetos são direcionados por objetivos. Pessoas tendem a alinhar seus planos com os objetivos da gerência e

outros envolvidos ("*stakeholders*") para, por exemplo, encontrar defeitos ou confirmar que o software funciona. Desta forma, é importante ter objetivos claros do teste. Identificar falhas durante o teste pode ser considerado uma crítica contra o produto e o autor (responsável pelo produto). Teste é, nestes casos, visto como uma atividade destrutiva, apesar de ser construtiva para o gerenciamento do risco do produto. Procurar por falhas em um sistema requer curiosidade, pessimismo profissional, um olhar crítico, atenção ao detalhe, comunicação eficiente com os profissionais do desenvolvimento e experiência para encontrar erros. Se os erros, defeitos ou falhas são comunicados de uma forma construtiva, podem-se evitar constrangimentos entre as equipes de teste, analistas e desenvolvedores, tanto na revisão quanto no teste.

O testador e o líder da equipe de teste precisam ter boa relação com as pessoas para comunicar informações sólidas sobre os defeitos, progresso e riscos de uma forma construtiva. A informação do defeito pode ajudar o autor do software ou documento a ampliar seus conhecimentos. Defeitos encontrados e resolvidos durante o teste trará ganho de tempo e dinheiro, além de reduzir os riscos.

Problemas de comunicação podem ocorrer, especialmente se os testadores forem vistos somente como mensageiros de más

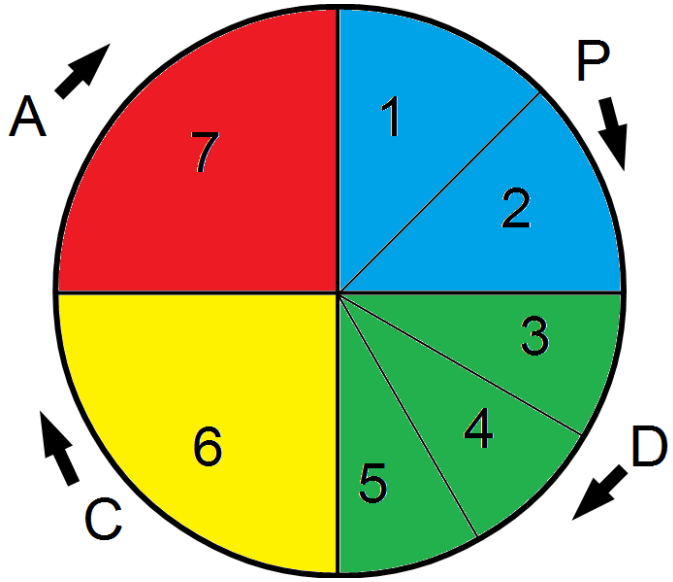
notícias ao informar os defeitos. De qualquer forma, existem formas de melhorar a comunicação e o relacionamento entre os testadores e os demais:

- Começar com o espírito de colaboração, em vez de disputa (conflitos), onde todos têm o mesmo objetivo para alcançar a melhor qualidade do sistema.
- Comunicar os erros encontrados nos produtos de uma forma neutra, dar foco no fato sem criticar a pessoa que o criou, por exemplo, escrevendo objetivamente o relatório de incidentes.
- Tentar compreender como a pessoa se sente ao receber a notícia e interpretar sua reação.
- Confirmar que a outra pessoa compreendeu o que você relatou e vice-versa.

Processo de Teste - PDCA (*Plan, Do, Check, Act* – Planejar, Fazer, Checar, Agir)

1. Definir Meta
2. Definir Método
3. Educar e Treinar
4. Executar
5. Coletar Dados
6. Checar

7. Ação: corretiva, preventiva, melhoria



Fundamentos do Processo de Teste de Software

Um processo de teste básico deve contemplar:

- Planejamento e controle;
- Análise e modelagem;
- Implementação e execução;
- Avaliação de critérios de saída e relatórios;
- Atividade de encerramento dos testes.
- Melhoria/Corretiva/Preventiva.

Atividades de Verificação e Validação:

A Validação e a Verificação são conceitos de grande importância para a disciplina de teste, onde Validação é a avaliação da veracidade do produto baseado nas necessidades e requisitos definidos pelo cliente. Devemos, no decorrer do

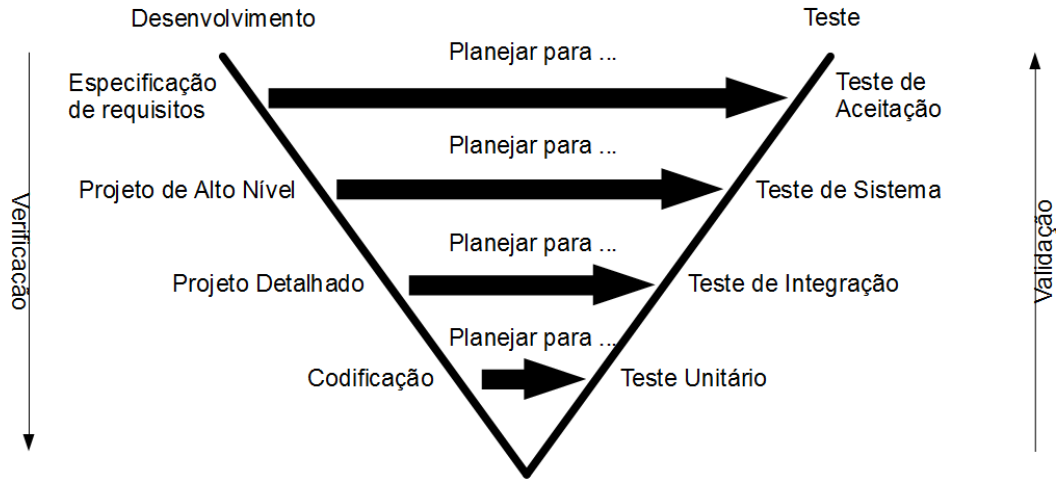
processo, questionar se estamos desenvolvendo o produto correto e se o sistema está de acordo com a especificação definida.

A Verificação, por sua vez, é a avaliação se o objeto foi desenvolvido da maneira prevista, onde devemos nos questionar se estamos desenvolvendo o produto corretamente. Para isso existem diversas técnicas como, por exemplo, a inspeção e revisão

A Validação e Verificação asseguram que o software cumpra com suas especificações e atenda às necessidades dos usuários. O processo de teste é dividido em duas grandes áreas:

A estrutura do modelo V é uma aproximação do processo de testes que pode ser integrada com todo a processo de desenvolvimento. O modelo V representa o desenvolvimento versus o teste. O modelo V focaliza-se em testar durante todo o ciclo de desenvolvimento para conseguir uma detecção adiantada dos erros.

Processo em “V”



1.2 Testes Unitários

O teste unitário tem por objetivo testar a menor parte testável do sistema (unidade), que pode ser um módulo, objeto ou classe.

Idealmente, o teste unitário é independente de outros testes, validando assim cada parte ou funcionalidade individualmente.

Nos casos em que existe dependência de métodos ou classes que não estão sob teste, são usados *mocks* (objetos criados para simular o comportamento de objetos reais nos testes) para simular estes objetos ou métodos que são usados pela unidade, desacoplando a unidade sob teste dos componentes dos quais ela é dependente.

O teste unitário também é caracterizado pelo uso de controladores (*drivers*) e simuladores (*stubs*).

Todos os tipos de teste podem ser executados de forma unitária, ou seja, tantos os testes funcionais, não-funcionais, estruturais, desempenho, cobertura, etc, podem ser executados de forma unitária.

Não há registro de defeitos identificados, visto que são corrigidos imediatamente.

TDD (*Test Driven Development* - Desenvolvimento Orientado

a Testes) é uma técnica de desenvolvimento de software em que, ao contrário do que em geral estamos acostumados, primeiro são escritos os testes e só depois é produzido um código que possa validá-lo. A base para o TDD é um ciclo curto de desenvolvimento.

Em um primeiro momento, não precisamos nos preocupar com todos os casos possíveis, isso se dá forma gradual a cada teste implementado, evitando que se perca muito tempo quebrando a cabeça para resolver um problema complexo quando ele pode ser resolvido em partes e refatorado.

Além disso, ao implementar o teste primeiro, pensando apenas na funcionalidade que desejamos implementar, evitamos de nos basear no próprio código para montar os testes, esquecendo algumas vezes de alguns casos que não foram previstos inicialmente.

Escrevendo os testes e definindo o comportamento dos métodos antes, como no exemplo anterior em que é esperada que seja lançada uma exceção já pré-definida, permite que outros desenvolvedores saibam como o código vai se portar antes mesmo dele ser escrito, além de ser possível definir os requisitos do projeto ou da funcionalidade para o desenvolvedor.

Pode-se pensar como um contraponto do TDD, ou aos testes unitários em si, o tempo e o trabalho necessário para desenvolver

os testes, porém, esse tempo "perdido" é recompensado mais para frente, quando é necessário refatorar um código ou procurar aquele bug que resolveu aparecer dois anos depois naquele código que ninguém mexe faz séculos.

Em alguns casos, os testes unitários praticamente substituem a função de um *debugger*, já que, modificando o código e verificando quais dos testes deles falham, é possível saber onde exatamente está o problema.

Um outro ponto muito importante, que diz respeito também aos testes unitários, é com relação à posterior manutenção do código. Não é necessário se preocupar com o fato de que possíveis refatorações ou alterações resultem em bugs novos e inesperados, já que todas as funcionalidades do sistema estão sendo testadas.

Dessa forma, acaba o medo de refatorar aquele trecho pré-histórico que ninguém mais sabe de onde surgiu, ou mesmo de codificar uma funcionalidade nova que pode alterar o funcionamento de outras antigas, já que se surgir algum imprevisto, algum teste falhará, e muitas vezes só de olhar qual foi o teste que falhou já é possível identificar o erro. Além do fato de que ninguém pode pôr a culpa se algum bug que surgir em algum código seu, já que, quando você desenvolveu, todos os testes passaram.

Outros Níveis de Teste:

- **Integração**
  - Testa a integração entre os diversos componentes de software criados;
- **Sistema**
  - Validação do comportamento do sistema;
- **Aceite**
  - Validação realizada pelos *stakeholders* (clientes, usuários, etc.);
  - Teste de Aceite Contrato (cumprimento de acordo: legislação, etc.);

1.3 Testes Funcionais

Testes Funcionais são testes que avaliam o funcionamento da aplicação (está construindo o produto certo?) considerando o comportamento externo do software. Dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado a um resultado esperado.

O teste funcional é aplicável a todas as fases do teste (unitário, integração, sistema e aceitação). Testa as funcionalidades, requerimentos, regras de negócio presentes na

documentação.

Outros tipos de testes incluem:

- **Não funcional**
  - Busca a validação de aspectos não funcionais (como está funcionando?) do software como:
    - Desempenho, Carga, Stress, Usabilidade, etc.
- **Estrutural**
  - Busca a validação de aspectos estruturais do software como:
    - Cobertura de código, sentença (comandos, funções), decisão, etc.
- **Mudanças**
  - Teste de Confirmação
    - valida a remoção de defeitos;
  - Teste de Regressão
    - executado quando ocorre alteração de software ou ambiente;
    - valida se não foram gerados novos defeitos.

Classificação dos Testes:

Técnica Caixa-preta:

Deriva condições e casos de teste de documentações formais. Funcionais e não-funcionais do software. Não considera aspectos internos (código).

- Transição de estados:
  - Utiliza-se do diagrama de transição de estado;
  - Os estados são identificáveis e finitos;
  - Identificar e exercitar todas transições válidas e inválidas.
- Caso de Uso:
  - Interações entre usuário e sistema gerando resultado relevante
  - Preocupação com as pré-condições e pós-condições;
  - Cobrir os fluxos existentes (principal, alternativos e exceção);
  - Tratar os pontos de extensão.

Técnica Caixa-branca:

Considera aspectos internos do software (código). Foco nas estruturas dos códigos (cobertura). Deriva condições e casos de testes de forma sistêmica.

- Teste e cobertura de Sentença (Comando):
  - Análise de cobertura de comandos executáveis existentes

no código;

- Utilizados para aumentar a cobertura dos testes.

- Teste e cobertura de Decisão:
  - Teste de controle de fluxo;
  - Utilizados para aumentar a cobertura dos testes.

Observação: **100% de cobertura de decisão, garante 100% de cobertura de sentença, mas NÃO vice-versa.**