Fundamentos de Qualidade de Software
Carolina Santana Louzada

DEFINIÇÃO DE QUALIDADE

NBR/ISO 9000:2005: grau no qual um conjunto de características inerentes satisfaz a requisitos;
Peters(2002) : “A qualidade de software é avaliada em termos de atributos de alto nível chamado fatores, que são medidos em relação a atributos de baixo nível, chamados critérios.”
Sanders(1994): “ Um produto de software apresenta qualidade dependendo do grau de satisfação das necessidades dos clientes sob todos os aspectos do produto.”
Pressman: “Qualidade de software é a conformidade a requisitos funcionais e de desempenho que foram explicitamente declarados, a padrões de desenvolvimento claramente documentados, e a características implícitas que são esperadas de todo software desenvolvido por profissionais.”
ISO/IEC 25010:2011 : “capacidade do produto de software de satisfazer necessidades declaradas e implícitas sob condições especificadas”
IEEE Standard(2014): “ o grau em que um produto de software atende aos requisitos estabelecidos; no entanto a qualidade depende do grau em que esses requisitos representam com precisão as necessidades, desejos e expectativas das partes interessadas ”

Requisitos de software são a base para medir a qualidade; padrões específicos definem o conjunto de critérios de desenvolvimento; Existem requisitos implícitos que afetam diretamente a qualidade.

Visão transcendental - reconhecimento por experiência, histórico…
Visão do usuário - de acordo com a necessidade do cliente
Visão de manufatura - conformidade aos requerimentos
Visão de produto - produto com boas propriedades internas implicando em boa qualidade externa
Visão baseada em valor - custo-benefício na visão do cliente

NORMAS E PADRÕES DE QUALIDADE

Padronização é importante para uma boa comunicação - todo mundo “falando a mesma língua”

PROCESSOS DE GERENCIAMENTO DE QUALIDADE

Atividades de gerenciamento:

- Planejamento de qualidade (determinar padrões e processos de qualidade - indentificar quais normas serão usadas / definir metas de qualidade)
- Garantia de qualidade (garantir que o que está sendo produzido realmente foi pedido pelo cliente - realizar atividades que definem e avaliam os processos de software provendo evidências que estabeleça confiança no produto)

- Controle de qualidade (examinar os artefatos do processo para ver se padrões estão sendo seguido / avaliar os produtos intermediários e finais / observar as entradas e saídas dos requerimentos)
- Melhoria de processos (melhorar a eficiência, efetividade e outras características que tenham como meta a melhoria da qualidade de software)

GERENCIAMENTO DE DEFEITOS

CONTROLE DE QUALIDADE

Teste é uma forma de controle de qualidade
Análise estática - Avaliação de documentação do software e código-fonte (métodos formais)
Análise dinâmica - Técnicas com o código em execução
Verificação - garantir que o produto esteja sendo construído corretamente
Validação - estar em conformidade com o que o usuário pediu (produto correto)

CARACTERIZANDO DEFEITOS

Rastrear os defeitos
Entender os tipos de defeitos encontrados
Facilitar a correção do processo ou produto
Reportar o status do produto
Alinhar revisões - time de desenvolvimento

O que é defeito?
Anomalia encontrada no produto
Erro - ação humana que produz resultado incorreto
Defeito - imperfeição ou deficiência relacionada aos requerimentos e especificações do produto
Falha no sistema - evento em que o sistema não executa uma função sob limites específicos

Motivos de erros:
Pressão, falha humana, inexperiência, falta de comunicação, complexidade de código/ modelagem/ arquitetura, complexidade de tecnologia, condições ambientais inesperadas…

CICLO DE VIDA DO BUG

New: Defeito é identificado e cadastrado pela primeira vez
Assigned: defeito é atribuído para desenvolvedor avaliar
Open: desenvolvedor inicia análise e correção
Fixed: Desenvolvedor finaliza correção
Pending Retest: Estado de espera para o time de teste
Retest: Estado de execução do reteste
Verified: Defeito corrigido
Reopen: Defeito não-corrigido.
Closed: Corrigido + testado + aprovado
Duplicate: efeito já encontrado anteriormente

Rejected: Defeito não é novo.
Deferred: Será corrigido em versões futuras.
Not a bug: Quando a anomalia não é de fato um erro depois de analisado

O time todo deve estar de acordo com o fluxo de rastreamento de defeitos
Defeitos podem ser rastreados a qualquer momento do ciclo de vida do processo de software
Usar ferramentas de rastreio e report de bugs (

Reporte deve conter:
Um identificador único
Título resumindo o problema
Data/autor
Identificação do item sob teste e do ambiente
Fase do ciclo de vida no qual o defeito foi observado
Descrição completa do defeito para reprodução
Evidências de auxílio na resolução:

- logs
- dumps de banco de dados
- screenshots
- gravações

Resultado esperado
Severidade
Urgência/Prioridade
Estado do defeito
Conclusões/Sugestões
Impactos
Histórico
Referência do teste

INTRODUÇÃO A TESTES

O que é teste?
Processo de avaliar e reduzir risco de falhas

Objetivos de testar:

- Evitar defeitos
- Verificar cumprimento de requisitos
- Validar se o produto funciona como o cliente espera
- Criar confiança no nível de qualidade do produto
- Reduzir riscos
- Atuar junto ao cliente em tomadas de decisão

Depuração é o processo de investigação e correção do erro no processo de desenvolvimento do código (normalmente acaba sendo feito apenas pelo desenvolvedor)

Princípios de teste:

1. Teste mostra presença de defeitos e não a ausência

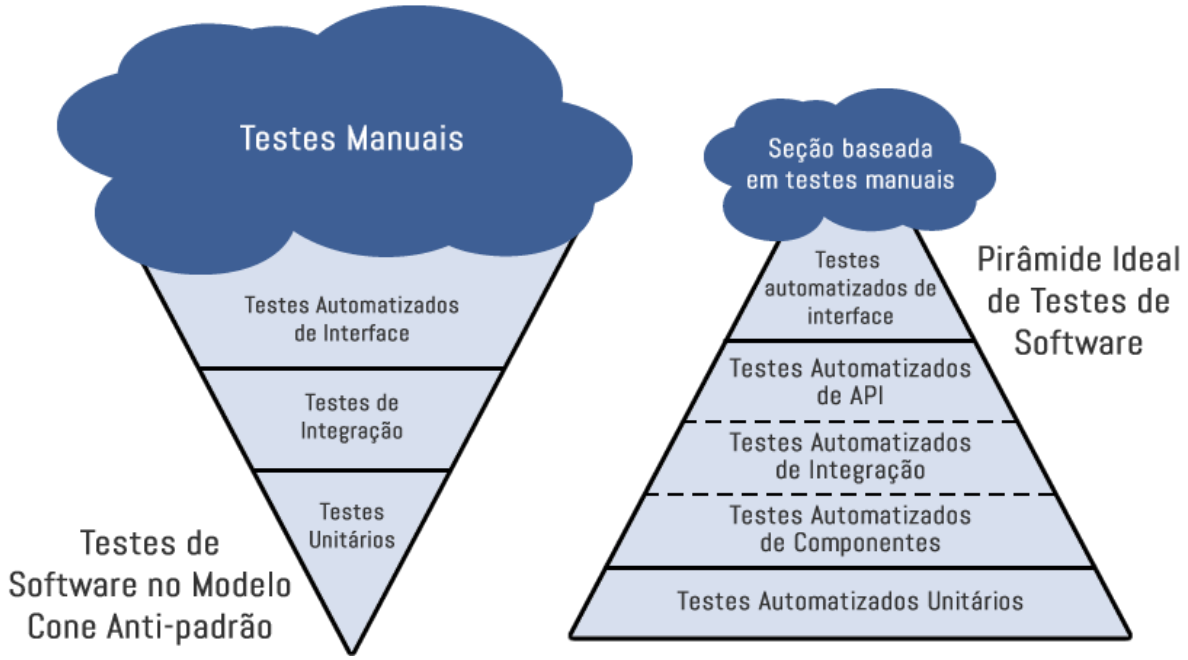
2. Testes exaustivos são impossíveis
3. Testes iniciais economizam tempo e dinheiro
4. Defeitos se agrupam
5. O mesmo teste não encontra novos defeitos -> atenção com testes de regressão
6. O teste depende do contexto
7. Ausência de erros é ilusão

Atividades de teste:

- Planejamento (Definir propósito do teste e abordagem com restrição de contexto; especificar tarefas e estimativas de prazos; definir estratégias)
- Monitoramento e controle do teste (comparação do progresso real com o plano - relatório de progresso)
- Análise (olhar para a base de testes e definir o que testar de acordo com critérios pré-estabelecidos - especificações de requisitos, documentos de arquitetura, fluxograma, código-fonte…) (definir e priorizar condições de teste)
- Modelagem (como testar? elaborar casos de teste - priorizar casos de teste e conjuntos de casos - verificar infraestrutura e projetar ambiente de teste)
- Implementação (desenvolver e priorizar procedimentos de teste - automatizar - criar suítes de testes - preparar os dados de teste)
- Execução (Executar conforme planejado - comparar resultados com o esperado - analisar anomalias para estabelecer prováveis causas - reportar e registrar anomalias - retestar após correção)
- Conclusão (coletar dados de testes já concluídos - revisar o que foi feito - criar relatório - finalizar e arquivar dados e registros dos testes)

NÍVEIS DE TESTES
São grupos de atividades de testes - tem relação com o nível de desenvolvimento

- Teste de componentes ou Unidade (código)
- Teste de integração (design de alto nível - verificar interfaces)
- Teste de sistema (requisitos funcionais e não funcionais - verificar percurso de usabilidade)
- Teste de aceite (validação do software)



TIPOS DE TESTES

Grupo de atividades de testes que verifica características específicas do sistema - baseado em objetivos específicos

1. Avaliar características funcionais (Caixa preta - o que o sistema deve executar - relacionado principalmente a testes de interface - desenvolvido a partir de especificações de requisitos, histórias de usuários)
2. Avaliar características não funcionais (eficiência de performance, segurança, usabilidade - todos os níveis de testes)
3. Avaliar estrutura ou arquitetura de componente/sistema (Caixa branca - teste de unidade e de integração - avalia fluxo de dados)
4. Avaliar efeitos de alterações em outras partes do código (testes de mudanças - Teste de confirmação e de regressão)

TÉCNICAS DE TESTE

Caixa preta: fundamentada em documentos de requisitos, histórias de usuário, o que é esperado do sistema - Testes funcionais ou não funcionais - foco nas entradas e saídas do teste, não foca na estrutura interna)

1. Particionamento de equivalência (dividir os dados em partes ou classes de equivalência)
2. Análise de valor limite (parecido com particionamento, mas avalia valores máximos e mínimos)
3. Tabela de decisão (testar requisitos que combinados possuem diferentes resultados)
4. Transição de estado (avaliar como o sistema reage a diferentes tipos de eventos)
5. Caso de uso (entende o que o usuário deseja - projeta casos básicos, alternativos e de erros)

Caixa branca: Visivel a estrutura interna - usada em nível de componentes

1. Cobertura de instruções (testa instruções executáveis do código - expressa em porcentagem “nº de instruções executadas”/”total de instruções”)

2. Cobertura de decisões (testar as condicionais - Cobertura = número de resultados de decisão executados/ total de resultados de decisão no objeto

Por experiência: técnicas por feeling, identifica situações não encontradas nos métodos sistemáticos

1. Suposição de erro
2. Teste exploratório
3. Baseado em checklist