Qualidade de software

Professor Emiliano S. Monteiro

[Versão 10 Ago/2019]

24.2. Métricas de confiabilidade

1. Probabilidade de Falha na Demanda (PFD)

- PFD = 0,001
- Para uma em cada 1000 solicitações, o serviço falha por unidade de tempo.
- 2. Taxa de Ocorrência de Falha (RFO)
- RFO = 0,02
- Duas falhas para cada 100 unidades operacionais de tempo de operação.

24.1. Confiabilidade de Software

- Definido como: a probabilidade de falha operação livre de um programa de computador em um ambiente especificado para um período de tempo especificado.
- Pode ser medido diretamente e estimado usando dados históricos e de desenvolvimento (ao contrário de muitos outros fatores de qualidade de software).
- Problemas de confiabilidade de software geralmente podem ser rastreados até erros de design ou implementação.

- Usando sistemas de bug trackers (sistema O.S.) !

24.2. Métricas de confiabilidade

- Tempo médio de falha (MTTF)
- Tempo médio entre falhas observadas (ou MTBF)
- Disponibilidade = MTBF / (MTBF + MTTR)
  - MTBF = tempo médio entre falhas MTTR = tempo médio de reparo
- Confiabilidade = MTBF / (1 + MTBF)

24.2. Métricas de confiabilidade

- MTBF = Mean Time Between Failure (tempo médio entre falhas)
- MTTF = Mean Time To Failure (Tempo médio até a falha)
- MTTR = Mean Time To Repair (Tempo médio para reparar)
- MTBF = MTTF + MTTR

- Disponibilidade: (MTTF / (MTTF+MTTR))/100%

24.3. Métricas: Unidades de tempo

- Outras unidades:
  - Tempo de execução bruto de um sistema sem interrupção.
  - Hora (no calendário)... se o sistema tiver padrões de uso regulares.
  - Número de transações do sistemas transacional por tipo de operação.

25. Testes

25.1. Revisão em pares

É difícil para um desenvolvedor encontrar defeitos no produto de seu trabalho.

Geralmente é solicitado que outro faça uma revisão.

A revisão por outro desenvolvedor é chamada de revisão por pares.

Cada artefato do projeto desenvolvido é revisado por outro integrante da equipe.

25.1.1. Por que realizar revisões por pares?

1. Para melhorar a qualidade.
2. Captura 80% de todos os erros se feito corretamente.
3. Captura erros de codificação.
4. Captura erros de projeto.
5. Implica em treinamento dos pares envolvidos.

25.2. Revisão formal

- Objetivos:
  - Descobrir erros de uma função ou lógica de implementação
  - Verificar requisitos*
  - Verificar se atende a determinados padrões
  - Corrigir erros de saída do sistema
  - Promover o conhecimento sobre o sistema

25.3. Reunião de revisão

- Deve conter um número impar de pessoas
  - Recomenda-se de 3 a 5 pessoas
- Não deve exigir mais de 1 a 2 horas de preparação de cada participantes
- Deve ter uma pessoa que realiza anotações e distribui entre os demais o que foi acordado.
- As reuniões devem ter prazo fixo de duração e momento para terminar, algo em torno de 2 horas ou menos.
- Os resultados devem ser comunicados às partes interessadas e registrados na documentação do projeto.

25.4. Teste de componentes

Teste de componentes:

1. Testes de componentes individuais do programa.
2. Geralmente a responsabilidade é do desenvolvedor do componente.
3. Os testes são derivados da experiência do desenvolvedor.

Dependências de pacotes !
apt-get install mc (verificação de dependência)

25.4. Teste de integração

Teste de integração:

1. Testes de grupos de componentes integrados para criar um sistema ou subsistema.
2. A responsabilidade de uma equipe de testes independente.
3. Os testes são baseados em uma especificação do sistema (caixa-preta)

ACBr automação comercial

Avaliação em conjunto de artefatos.

25.5. Black-box

Teste de caixa preta:

- Uma abordagem de teste onde o programa é considerado como uma "caixa-preta“.
- Os casos de teste do programa são baseados na especificação do sistema.
- O planejamento de testes pode começar cedo no processo de software.
- Testa funcionalidades sem importar-se com a forma de codificação.

25.6. White-box

Teste de caixa branca:

1. Antigamente era conhecida como teste estrutural.
2. Derivação de casos de teste de acordo com a estrutura do programa.
3. O conhecimento do programa é usado para identificar casos de teste adicionais.
4. O objetivo é executar todas as instruções do programa (nem todas as combinações de caminho).
5. As medidas de cobertura de teste garantem que todas as declarações tenham sido executadas pelo menos uma vez.
6. Interessado no código do sistema.

26.7. Sala limpa – Clear room

- O nome faz alusão a uma sala limpa para fabricação sob atmosfera controlada.
- É orientado a equipes.
- Tem base em controle estatístico.
- Princípios:
  - Fazer certo da primeira vez – evitar ciclos e testes de erros
  - Desenvolvimento incremental, somente após cada fase ser aprovada
  - Testes formais de programas
  - Cada teste é uma amostra de uma população e ajuda a estimar a qualidade de um produto.

26.8. Validação

- Validação é o processo de afirmar que a especificação de uma etapa ou parte do sistema é apropriada e consistente com os requisitos dos clientes
  - Validação tenta avaliar se a construção do artefato segue o que foi predefinido.

26.8. Verificação

- Verificação é o processo de determinar se a saída de cada etapa esta de acordo com os requisitos especificados na etapa anterior.
  - Verificar não é demonstrar que a saída etapa do desenvolvimento é correta, mas verificar se o software esta de acordo com as especificações determinadas anteriormente.
  - Verificar consiste em, identificar defeitos e possíveis problemas de um pacote pronto.

26.10. Plano de testes padrão IEEE 829-1998

- Propósito
- Identificador
- Introdução
- Itens
- Funcionalidades
- Abordagem
- Critérios de aceite
- Suspensão
- Produtos
- Tarefas de testes
- Ambiente
- Responsabilidades
- Cronograma

26.9. Classificação ortogonal de defeitos da IBM

- Foi desenvolvida na década de 90 e permite classificar os defeitos de acordo com diversas dimensões para capturar o máximo de informações.

26.9. Classificação ortogonal de defeitos da IBM

- Dimensões:
  - Atividade – o momento em que detectado o defeito
  - Gatilho – Causa da aparição do defeito
  - Impacto – Efeitos sobre o usuário/cliente
  - Alvo – Qual a parte que deve ser corrigida
  - Tipo de defeito – Correção necessária
  - Qualificador – classifica por: omissão, incorreção, excesso
  - Idade – se o defeito foi encontrado em um componente novo, reescrito ou já corrigido
  - Fonte: identifica se o componente nasceu com defeito dentro da empresa ou se veio de um módulo de fora da empresa.

27. Métricas

27.1. Métrica de software

- Uma métrica de software é uma propriedade de um produto de software, processo ou documentação relacionada que leva um valor numérico que pode ser medido em Linhas de código (LOC) em um programa, número de dias-pessoa necessários para desenvolver um componente, etc.

- É importante medir (quantificar) a qualidade de um produto de software.

- Dificuldade principal: distância entre o que queremos saber (geralmente um atributo de qualidade externo) e o que podemos medir (normalmente um atributo interno).

27.2. Tipos de métricas do produto

Métricas dinâmicas:

1. São coletadas por medições feitas de um programa em execução.
2. Estão intimamente relacionados com atributos de qualidade de software, como eficiência e confiabilidade.
3. É relativamente fácil medir o tempo de resposta de um sistema (atributo de desempenho) ou o número de falhas (atributo de fiabilidade).

27.2. Tipos de métricas do produto

Métricas estáticas:

1. São coletados por medições feitas das representações do sistema (arquivos de origem, documentação, etc.).
2. Tem uma relação indireta (e difícil de estabelecer) com atributos de qualidade, tais como complexidade, compreensão e manutenção.

27.3. Exemplo de métricas

1. Número de itens de dados diferentes
2. Número total de campos no banco de dados
3. Número total de campos de entrada nos formulários
4. Número médio de campos por formulário
5. Quantidade de usuários trabalhando simultâneamente