Engenharia de Software II

Aula 6

http://www.ic.uff.br/~bianca/engsoft2/

Ementa

- Processos de desenvolvimento de software
- **Estratégias e técnicas de teste de software** (Caps. 13 e 14 do Pressman)
- Métricas para software
- Gestão de projetos de software: conceitos, métricas, estimativas, cronogramação, gestão de risco, gestão de qualidade e gestão de modificações
- Reengenharia e engenharia reversa

Estratégias de teste de software

- Software é testado para se descobrir erros de projeto e construção.
- Uma estratégia de teste de software fornece um roteiro que deve responder a questões do tipo:
  - Quais os passos a serem conduzidos como parte do teste?
  - Quando os passos são planejados e executados?
  - O programa deve ser testado como um todo ou em partes?
  - Testes que já foram conduzidos devem ser re-executados quando o software é modificado?
  - Quando o cliente deve ser envolvido?

Características genéricas das estratégias de teste

- Para realizar um teste efetivo, uma equipe de software deve conduzir revisões técnicas formais.
  - Com isso, muitos erros são eliminados antes do início do teste.
- O teste começa no nível de componente e prossegue "para fora", em direção à integração de todo o sistema.
- Diferentes técnicas são adequadas em diferentes momentos.
- O teste é conduzido pelo desenvolvedor do software ou (para projetos grandes) um grupo de teste independente.
- O progresso deve ser mensurável.

Verificação e Validação

- O teste é somente um elemento de um conceito mais amplo da engenharia de software, conhecido como Verificação e Validação (V&V).
- Verificação = Estamos construindo o produto corretamente?
- Validação = Estamos construindo o produto certo?
- Outras atividades são necessárias para V&V como as revisões técnicas e formais, auditoria de qualidade e configuração, monitoramento de desempenho, etc.
  - O teste deve ser o último recurso para avaliar a qualidade.
- Existe um debate se o teste só fornece verificação ou se também pode fornecer validação.

Organização do Teste de Software

- Conflito de interesses
  - O desenvolvedor tem interesse em demonstrar que o programa está livre de erros e que funciona de acordo com os requisitos.
  - Assim, o desenvolvedor projeta e executa testes que demonstram que o programa funciona, em vez de descobrir erros.
  - Um grupo de teste independente não tem esse conflito.
- Concepções equivocadas
  - Os desenvolvedores não devem fazer nenhum teste.
  - O grupo de teste independente (ITG) deve se envolver com o projeto somente quando os passos de teste estão para começar.
- Organização recomendada
  - O desenvolvedor testa as unidades individuais e faz testes de integração até a arquitetura do software estar completa.
  - Depois, o ITG trabalha junto com o desenvolvedor para garantir que testes rigorosos serão conduzidos.

Uma Estratégia Global para Arquiteturas Convencionais

1. Teste de unidade
   - Focaliza cada componente individualmente, garantido que funciona.
   - Faz uso intensivo de técnicas que exercitam caminhos específicos na estrutura de controle.
2. Teste de integração
   - Focaliza o pacote de software completo e trata da verificação do programa como um todo.
   - Faz uso de técnicas de projeto de casos de teste que enfocam as entradas e saídas, além de exercitar caminhos específicos.
3. Teste de validação
   - Critérios de avaliação estabelecidos durante a análise de requisitos são avaliados.
4. Teste de sistema
   - Testa a combinação do software com outros elementos do sistema (como hardware, pessoal e bancos de dados).
   - Verifica se a função/desempenho global do sistema é alcançada.

Uma Estratégia Global para Arquiteturas Orientada a Objeto

- É idêntica em filosofia à estratégia para arquiteturas convencionais, mas difere na abordagem.
  - O menor elemento testado é uma classe ou pacote de classes que colaboram.
    - Abrange atributos e operações e implica comunicação e colaboração.
  - À medida que as classes são integradas, uma série de testes é feita para descobrir erros devido a comunicação e colaboração entre classes.

Critérios para Completamento do Teste

- Quando devemos terminar o teste?
- Não há uma resposta definitiva, pois nunca podemos garantir com 100% de certeza que não há erros.
- Algumas respostas “cínicas”:
  - O teste nunca acaba, a tarefa passa do engenheiro de software para o cliente.
  - O teste deve acabar quando o tempo ou o dinheiro acabarem.
- Respostas baseadas em critérios estatísticos:
  - Coletando métricas e usando modelagem estatística e teoria da confiabilidade de software, podem ser desenvolvidos modelos de falha de software.
  - Os modelos estatísticos permitem dizer, por exemplo: “Com 95% de confiança, a probabilidade de mil horas de operação de CPU livre de falhas é de 0,995”.

Diretrizes Estratégicas I

- Especifique os requisitos do produto de um modo quantificável muito antes de testar.
  - Isso inclui características como portabilidade e usabilidade.
- Enuncie explicitamente os objetivos do teste.
  - A cobertura, o tempo médio entre falhas, a densidade restante de defeitos, o número de horas de trabalho por teste, etc.
- Entenda os usuários de software e desenvolva um perfil para cada categoria.
  - Focaliza o teste no uso real do produto.
- Desenvolva um plano de teste que enfatize um ciclo rápido (2% do esforço do projeto).
  - Gera uma realimentação rápida para o projeto.

Diretrizes Estratégicas II

- Construir software robusto.
  - O software deve ser capaz de auto-diagnosticar certas classes de erro.
  - O projeto deve acomodar automação de teste e teste de regressão.
- Usar revisões técnicas formais como filtro.
  - Reduzem a quantidade de testes necessária para produzir software de alta qualidade.
- Conduzir revisões técnicas formais para avaliar a estratégia de teste.
  - Podem descobrir inconsistências, omissões e erros na abordagem de teste.
- Desenvolver uma abordagem de aperfeiçoamento contínuo para o teste.

Teste de Unidade

- Focaliza na menor unidade de projeto do software: o componente ou módulo.
- Pode ser conduzido em paralelo para os diversos componentes.
- Caminhos de controle importantes são testados para descobrir erros dentro dos limites do módulo.
- A complexidade dos testes é limitada pelo escopo restrito.

Considerações do Teste de Unidade

- Teste de Interface
  - Garante que a informação flui adequadamente pra dentro e para fora da unidade.
- Teste das Estruturas de Dados
  - Garante que os dados armazenados temporariamente mantenham sua integridade durante todos os passos da execução.
- Teste das Condições-Limite
  - Garante que o módulo opere adequadamente nos limiares estabelecidos para limitar ou restringir o processamento.
- Teste de Caminhos Independentes
- Teste de Caminhos de Manipulação de Erros.
  - Condições de erros são antecipadas e caminhos são estabelecidos para redirecionar ou claramente terminar o processamento.

Erros comumente encontrados nos testes de unidade I

- Erros devidos a cálculos errados
  - Precedência aritmética errada
  - Operações em modo misto
  - Inicialização incorreta
  - Falta de precisão
  - Representação incorreta de uma expressão simbólica
- Erros devidos a comparações erradas
  - Comparação de tipos de dados diferentes
  - Operadores ou precedência lógica incorretos
  - Expectativa de igualdade quando erro de precisão torna a igualdade improvável
  - Comparação incorreta de variáveis
  - Terminação de ciclo inadequada ou inexistente
  - Variáveis de ciclo inadequadamente modificadas

Erros comumente encontrados nos testes de unidade II

- Erros na manipulação de erros
  - A descrição do erro é ininteligível
  - O erro mencionado não corresponde ao erro encontrado
  - A condição de erro provoca a intervenção do sistema antes da manipulação do erro
  - A descrição do erro não fornece informação suficiente para manipular o erro