Verificação e Validação
-
Teste de *Software*

Verificação e Validação

Objetivos

- Apresentar a verificação e validação de *software* e discutir a distinção entre elas
- Descrever o processo de inspeção de programa e seu papel em V&V
- Explicar a análise estática como uma técnica de verificação.

Tópicos abordados

- Planejamento de verificação e validação
- Inspeções de *software*
- Análise estática automatizada
- Verificação e métodos formais

Verificação x validação

- Verificação
  "Estamos construindo o produto correto?"
  - O *software* deve estar de acordo com sua especificação
- Validação
  "Estamos construindo o produto corretamente?"
  - O *software* deve fazer o que o usuário realmente deseja.

PEARSON Prentice Hall

O processo V&V

- É um processo de ciclo de vida completo
  - V&V deve ser aplicado a cada estágio do processo de *software*
- Tem 2 objetivos principais
  - Descobrir defeitos em um sistema
  - Avaliar se o sistema é útil e usável ou não em uma situação operacional.

PEARSON Prentice Hall

Verificação estática e dinâmica

- Inspeções de *software*
  - Relacionado à análise de representações estáticas de sistema para descobrir problemas (verificação estática)
  - Pode ser suplementado por um documento baseado em ferramenta e análise de código
- Teste de *software*
  - Relacionado ao exercício e à observação do comportamento do produto (verificação dinâmica)
  - O sistema é executado com dados de teste e seu comportamento operacional é observado.

PEARSON Prentice Hall

Teste de programa

- Pode revelar a presença de defeitos, NÃO a ausência
- É a principal técnica de validação para requisitos não funcionais, visto que o *software* é executado para ver como se comporta
- Deve ser usado em conjunto com a verificação estática para fornecer cobertura completa de V&V.

PEARSON Prentice Hall

Tipos de teste

- Teste de validação
  - Pretende mostrar que o *software* atende aos seus requisitos
  - Um teste bem sucedido é aquele que mostra que um requisito foi adequadamente implementado
- Teste de defeitos
  - Testes projetados para descobrir defeitos de sistema
  - Um teste de defeitos bem sucedido é aquele que revela a presença de defeitos em um sistema.

Teste e *debugging*

- Teste e *debugging* e de defeitos são processos distintos
- Verificação e validação está relacionada ao estabelecimento da existência de defeitos em um programa
- *Debugging* está relacionado à localização e repararação desses defeitos.

A estrutura de um plano de teste de *software*

- Processo de teste
- Rastreabilidade de requisitos
- Itens testados
- Cronograma de testes
- Procedimentos de registro de testes
- Requisitos de *hardware* e de *software*
- Restrições.

Inspeções de *software*

- Envolvem pessoas que examinam uma representação original com o objetivo de descobrir anomalias e defeitos
- Inspeções não requerem a execução de um sistema, por isso, podem ser usadas antes da implementação
- Podem ser aplicadas em qualquer representação do sistema (requisitos, projeto, dados de configuração, dados de teste, etc)
- Tem se mostrado uma técnica efetiva para descobrir erros de programa.

Inspeções de programa

- Abordagem formalizada para revisões de documentos
- Voltadas explicitamente para detecção de defeitos (não correção)
- Defeitos podem ser erros lógicos, anomalias no código que poderiam indicar uma condição errônea (por exemplo, uma variável não iniciada) ou não conformidade com padrões.

Checklists de inspeção

- Um *checklist* de erros comuns deve ser usado para direcionar a inspeção
- *Checklists* de erros são dependentes de linguagem de programação e refletem os erros característicos com maior probabilidade de surgimento na linguagem
- Em geral, a verificação de tipo 'fraco' é a maior parte do *checklist*
  - Exemplos: iniciação, denominação de constantes, terminação de loops, limites de vetores, etc.

Análise estática automatizada

- Analisadores estáticos são ferramentas de *software* para processamento de texto fonte
- Eles varrem o texto do programa e tentam descobrir condições potencialmente errôneas e chamam a atenção da equipe de V&V
- Eles são muito efetivos como um auxílio para as inspeções
  - são um suplemento, mas não um substituto para as inspeções.

PEARSON Prentice Hall

Estágios da análise estática

- Análise de fluxo de controle
  - Verifica loops com múltiplos pontos de saídas ou de entrada, encontra código inacessível, etc
- Análise de uso de dados
  - Detecta variáveis não iniciadas, variáveis escritas duas vezes sem uma tarefa de impedimento, variáveis que são declaradas mas nunca usadas, etc
- Análise de interface
  - Verifica a consistência das declarações de rotina e procedimentos e seus usos.

PEARSON Prentice Hall

Estágios da análise estática

- Análise de fluxo de informação
  - Identifica as dependências das variáveis de entrada e de saída
  - Não detecta anomalias em si, mas destaca as informações para inspeção ou revisão de códigos
- Análise de caminho
  - Identifica caminhos através do programa e estabelece as declarações executadas naquele caminho
  - Novamente, é potencialmente útil no processo de revisão
- Ambos os estágios geram uma grande quantidade de informações
  - por isso devem ser usados com cuidado.

Verificação e métodos formais

- Métodos formais podem ser usados quando uma especificação matemática do sistema é produzida
- Eles são a técnica fundamental de verificação estática
- Envolvem análise matemática detalhada da especificação, e podem desenvolver argumentos formais que um programa está em conformidade com a sua especificação matemática.

Pontos-chave

- Verificação e validação não são a mesma coisa
  - A verificação mostra a conformidade com a especificação
  - já a validação mostra que o programa atende às necessidades do cliente
- Planos de teste devem ser definidos para guiar o processo de teste
- Técnicas de verificação estática envolvem o exame e a análise do programa para detecção de erros.

PEARSON Prentice Hall

Pontos-chave

- Inspeções de programa são eficientes para encontrar erros
- Em inspeções, o código de programa é sistematicamente verificado por uma equipe pequena para localizar defeitos de *software*
- Ferramentas de análise estática podem descobrir anomalias de programa que podem ser uma indicação de defeitos no código.

PEARSON Prentice Hall

Teste de *Software*

Objetivos

- Discutir as distinções entre teste de validação e teste de defeito
- Descrever os princípios de teste de sistemas e teste de componentes
- Descrever estratégias para gerar casos de teste de sistema
- Compreender as características essenciais de ferramentas usadas para automação de testes

Tópicos abordados

- Teste de sistema
- Teste de componente
- Projeto de casos de teste
- Automação de testes

O processo de teste

- Teste de componentes
  - Teste de componentes individuais de programa
  - Geralmente é de responsabilidade do desenvolvedor do componente (exceto algumas para sistemas críticos)
- Teste de sistema
  - Teste de grupos de componentes integrados para criar um sistema ou um subsistema
  - A resposabilidade é de uma equipe independente de teste
  - Os testes são baseados em uma especificação de sistema.

Teste de defeitos

- A meta do teste de defeitos é descobrir defeitos em programas
- Um teste de defeitos bem sucedido é aquele que faz um programa se comportar de uma maneira anômala
- Os testes mostram a presença e não a ausência de defeitos.

Metas do processo de teste

- Teste de validação
  - Utilizado para demonstrar ao desenvolvedor e ao cliente do sistema que o *software* atende aos seus requisitos
  - Um teste bem sucedido mostra que o sistema opera conforme pretendido
- Teste de defeitos
  - Utilizado para descobrir faltas ou defeitos no *software* nos locais em que o comportamento não está correto ou não está em conformidade com a sua especificação
  - Um teste bem sucedido é aquele que faz o sistema executar incorretamente e, assim, expor um defeito no sistema.

Políticas de teste

- Somente testes exaustivos podem mostrar que um programa está livre de defeitos
  - Contudo, testes exaustivos são impossíveis
- As políticas de teste definem a abordagem a ser usada na seleção de testes de sistema
  - Todas as funções acessadas por meio de menus devem ser testadas
  - As combinações de funções acessadas por meio dos mesmos menus devem ser testadas
  - Onde as entradas de usuário são fornecidas, todas as funções devem ser testadas com entradas corretas e incorretas.

Teste de integração

- Envolve a construção de um sistema a partir de seus compontes e o teste do sistema resultante dos problemas ocorridos nas interações entre componentes
- Integração *top-down*
  - Desenvolver o esqueleto do sistema e preenchê-lo com componentes
- Integração *bottom-up*
  - Integrar componentes de infra-estrutura e, em seguida, adicionar componentes funcionais
- Para simplificar a localização de erros, os sistemas devem ser integrados incrementalmente.

Teste de *releases*

- É o processo de teste de um release de sistema que será distribuído aos clientes
- A meta primária é aumentar a confiança do fornecedor de que o sistema atende aos seus requisitos
- Teste de *releases* é, geralmente, um teste caixa-preta ou funcional
  - É baseado somente na especificação de sistema
  - Os testadores não têm conhecimento da implementação do sistema.

PEARSON Prentice Hall

Diretrizes de teste

- Diretrizes são recomendações para a equipe de teste para auxiliá-los a escolher os testes que revelarão defeitos no sistema
  - Escolher entradas que forcem o sistema a gerar todas as mensagens de erro
  - Projetar entradas que causem *overflow* dos buffers
  - Repetir a mesma entrada ou série de entradas várias vezes
  - Forçar a geração de saídas inválidas
  - Forçar resultados de cálculo a serem muito grandes ou muito pequenos.

PEARSON Prentice Hall

Teste de desempenho

- Parte do teste de releases pode envolver teste de propriedades emergentes de um sistema, tais como desempenho e confiabilidade
- Testes de desempenho envolve, geralmente, o planejamento de uma série de testes onde a carga é constantemente aumentada até que o desempenho do sistema se torne inaceitável.

Teste de estresse

- São exercícios do sistema além de sua carga máxima de projeto
  - O estresse de um sistema causa, frequentemente, o surgimento de defeitos
- O estresse de sistema testa o comportamento de falha, pois os sistemas não devem falhar catastroficamente
  - O teste de estresse verifica uma perda inaceitável de serviço ou de dados
- O teste de estresse é particularmente relevante para sistemas distribuídos que podem exibir degradação severa quando uma rede se torna sobrecarregada.

Teste de componentes

- Teste de componente ou unitário é o processo de teste de componentes individuais isolados
- É um processo de teste de defeitos
- Os componentes podem ser
  - Funções individuais ou métodos de um objeto
  - Classes de objeto com vários atributos e métodos
  - Componentes compostos com interfaces definidas usadas para acessar sua funcionalidade.

Teste de classe de objeto

- A abrangência do teste completo de uma classe envolve
  - Teste de todas as operações associadas com um objeto
  - Atribuir e interrogar todos os atributos de objeto
  - Exercício do objeto em todos os estados possíveis
- A herança torna mais difícil o projeto de testes de classe de objeto quando as informações a serem testadas não são localizadas.

Teste de interfaces

- Os objetivos são detectar defeitos devido a erros de interface ou suposições inválidas sobre interfaces
- É particularmente importante para o desenvolvimento orientado a objetos quando os objetos são definidos pelas suas interfaces.

Projeto de casos de teste

- Envolve o projeto de casos de teste (entradas e saídas) usados para testar o sistema
- A meta do projeto de casos de teste é criar um conjunto de testes que sejam eficazes em validação e teste de defeitos
- Abordagens de projeto
  - Teste baseado em requisitos
  - Teste de partições
  - Teste estrutural.

Teste baseado em requisitos

- Um princípio geral de engenharia de requisitos é que os requisitos devem ser testáveis
- O teste baseado em requisitos é uma técnica de teste de validação onde você considera cada requisito e deriva um conjunto de testes para esse requisito.

Teste estrutural

- Conhecido como “teste caixa-branca”
- É a derivação de casos de teste de acordo com a estrutura do programa
  - O conhecimento do programa é usado para identificar casos de teste adicionais
- O objetivo é exercitar todas as declarações do programa (não todas as combinações de caminhos).

Teste de caminho

- O objetivo do teste de caminho é assegurar que o conjunto de casos de teste é tal que cada caminho pelo programa é executado pelo menos uma vez
- O ponto de partida do teste de caminho é um fluxograma de programa que mostra os nós que representam as decisões do programa e arcos que representam o fluxo de controle
- Declarações com condições são, portanto, nós no fluxograma.

PEARSON Prentice Hall

Automação de teste

- Teste é uma fase dispendiosa do processo
  - Os *workbenches* de teste fornecem uma variedade de ferramentas para reduzir o tempo necessário e os custos totais de teste
- Sistemas tais como o JUnit apóiam a execução automática de testes
- A maioria dos *workbenches* de teste são sistemas abertos porque as necessidades de teste são específicas da organização
- Eles são, algumas vezes, difíceis de integrar com *workbenches* de projeto e análise fechados.

PEARSON Prentice Hall

Pontos-chave

- Os testes podem mostrar a presença de defeitos em um sistema
  - Eles não podem provar que não haja defeitos remanescentes
- Desenvolvedores de componentes são responsáveis pelo teste de componentes
  - já o teste de sistema é de responsabilidade de uma equipe separada
- O teste de integração é o teste de incrementos do sistema
  - o teste de *release* envolve teste de um sistema a ser liberado para um cliente.

PEARSON Prentice Hall

Pontos-chave

- Teste de interfaces é projetado para descobrir defeitos nas interfaces dos componentes compostos
- A análise estrutural baseia-se na análise de um programa e na derivação de testes a partir desta análise
- A automação de testes reduz os custos pelo apoio ao processo de teste com uma variedade de ferramentas de *software*.

PEARSON Prentice Hall