Testes de Software

Metodologias e Desenvolvimento de Software

Pedro Salgueiro
pds@di.uevora.pt
CLV-256

Testes de software

- Outline
  - Testes de desenvolvimento
  - Test-driven development
  - Testes de release
  - Testes de utilizadores

Testar programas

- Testes servem para mostrar que um programa faz o que deve fazer, e para descobrir problemas e erros antes de ser colocado em utilização.

- Como se testar o software
  - Executa-se o programa com dados artificiais
  - Verifica-se os resultados, procura-se erros, anomalias ou informação sobre os atributos não funcionais do sistema
- Pode revelar a presença de erros mas **não** a sua ausência
- Testes fazem parte do processo de verificação e validação mais genérico, que também inclui técnicas de validação estáticas

Testar programas

Objectivos

- Demonstrar ao programador e ao cliente que o software está de acordo com os requisitos
  - Software específico(feito à medida)
    - Pelo menos um teste para requisito no documento de requisitos.
  - Software genérico
    - Testes para todas as funcionalidades do sistema
- Encontrar situações em que o comportamento anómalos do software, que não estão de acordo com os requisitos
  - Testes de defeitos (defect testing) tem como objectivo remover comportamentos inesperados do sistema, “crashes” do sistema, interacções indesejadas, corrupção de dados, etc...

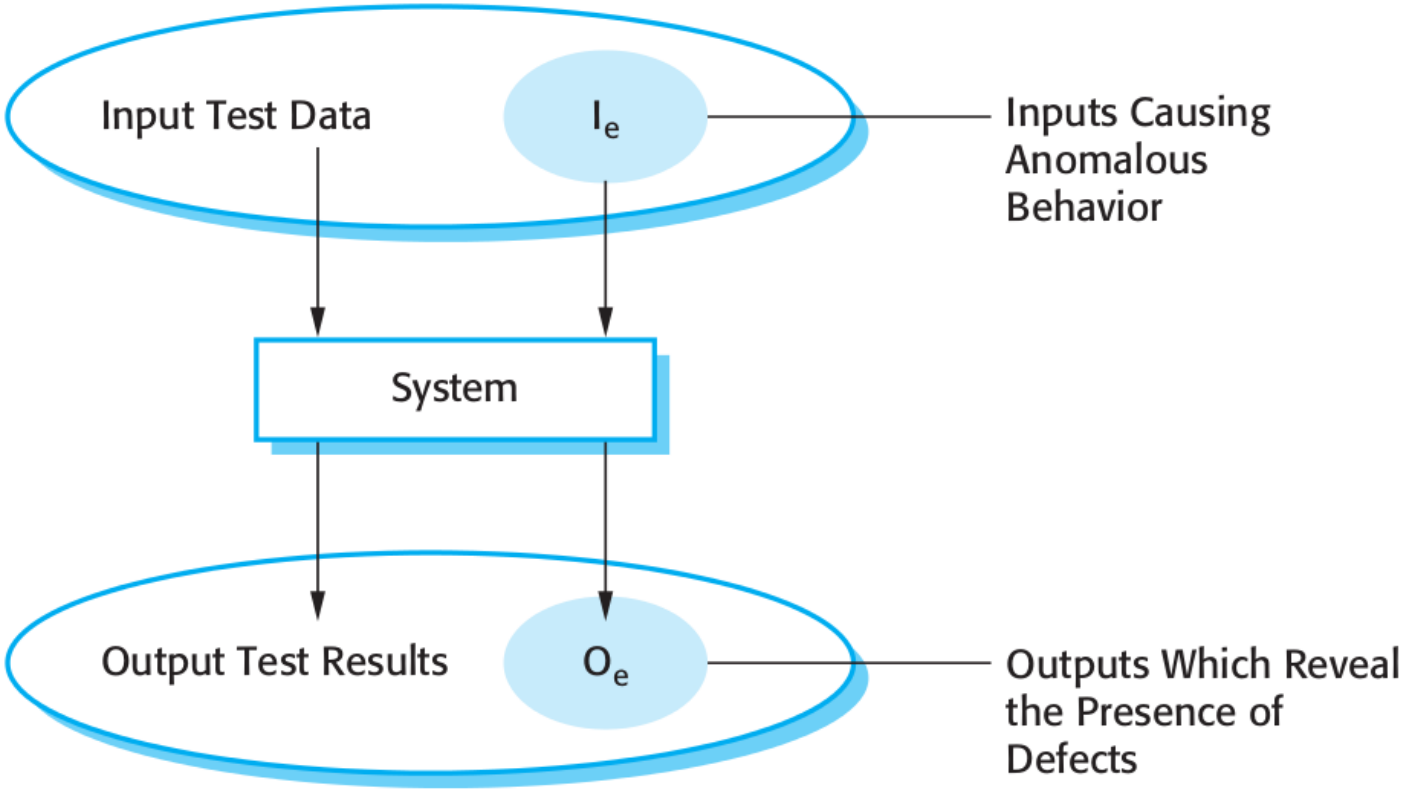
Testes de validação e de defeitos

- 1º objectivo
  - Dá origem aos testes de validação
    - Onde se espera que o sistema tenha o comportamento esperado, usando um conjunto de testes que reflectem a utilização normal do sistema
- 2º objectivo
  - Dá origem aos testes de defeito
    - Testes desenhados para revelar defeitos e erros.
    - Testes deliberadamente “obscuros” que não precisam de reflectir o comportamento normal do sistema

Processo de testes - objectivo

- Testes de validação
  - Demonstrar ao programador e ao cliente que o software está de acordo com os requisitos
  - Um bom teste mostra que o sistema funciona como deve funcionar
- Testes de defeito
  - Encontrar erros e problemas que dão origem a comportamentos anormais e que não estão de acordo com as especificações
  - Um bom teste faz com que o sistema funcione de forma incorrecta, expondo os seus erros e defeitos.

Teste de programas - Modelo de input-output



Verificação vs Validação

- Verificação
  - “Estamos a criar o produto de forma correcta?”
  - O software deve estar de acordo com as especificações

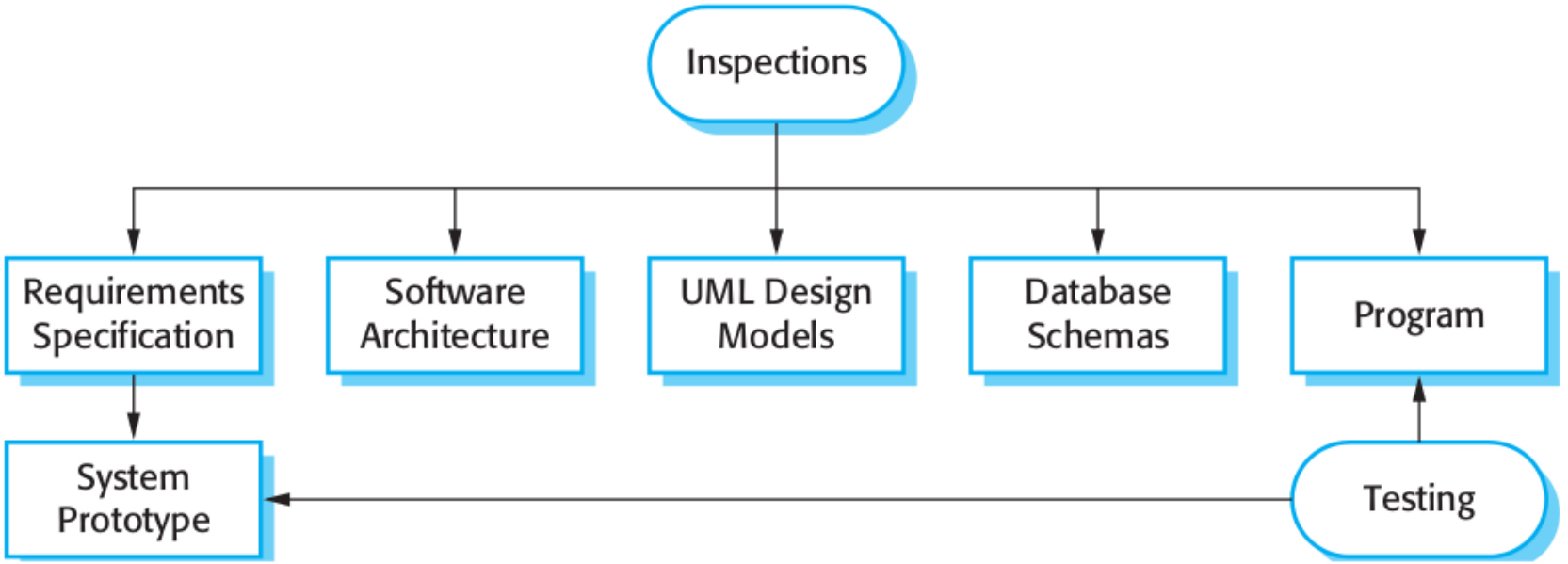
- Validação
  - “Estamos a construir o produto certo?”
  - O software deve fazer aquilo que o utilizador realmente necessita

Inspecções e testes

- Inspecções de software
  - Análise e verificação da representação estática do sistema por forma a encontrar problemas
    - Verificação estática

- Testes de software
  - Observação do comportamento do sistema
    - Verificação e dinâmica
    - Sistema é executado com dados de testes, verificando-se o seu comportamento

Inspecções e testes



Inspecção de software

- Análise da representação do código por humanos, com o objectivo de descobrir anomalias e defeitos
- Inspecções não necessitam de executar o sistema
  - Podem ser usadas antes da sua implementação
- Pode ser feita a qualquer representação do sistema
  - Requisitos, desenho, dados de configuração, dados de testes, etc…
- Técnica eficaz para encontrar erros

Inspecção de software

Vantagens

- Durante os testes, os erros podem esconder outros erros.
  - Como as inspecções são um processo estático, não existe a preocupação com interacções entre erros.
- Versões incompletas do sistema podem ser inspeccionadas sem custos adicionais
  - Se o programa estiver incompleto, é necessário desenvolver testes específicos para testar os componentes disponíveis.
- Para além de se procurarem defeitos
  - Inspecções podem ser úteis para encontrar atributos de qualidade de um programa: satisfação de standards, portabilidade e manutenção

Inspecções e testes

- Inspecções e testes são actividades complementares
- Ambas devem ser usadas no processo de validação e verificação
- Inspecções podem verificar se o sistema está de acordo com as especificações
  - Não podem ser usadas para verificar se o sistema está de acordo com as reais necessidades do cliente
- Inspecções não podem ser usadas para verificar características não funcionais
  - Desempenho, usabilidade, etc...

Testes - etapas

- Testes durante a fase de desenvolvimento
  - *Development testing*
  - Sistema é testado durante a etapa de desenvolvimento
  - Procurar erros e defeitos
- Testes *de Release*
  - Testes a uma versão completa do sistema (release)
  - Equipa de testes diferente
  - Antes de ser entregue aos utilizadores
- Testes de utilizador
  - Testes feitos ao sistema por utilizadores reais, no seu âmbiente de trabalho

Testes durante a fase de desenvolvimento

- Testes de desenvolvimento incluem todas as actividades de testes realizadas durante o desenvolvimento do sistema
  - Testes unitários
    - “Unidades” ou objectos individuais do sistema são testados
    - Testar funcionalidades de objectos ou métodos
  - Testes de componentes
    - Várias unidades são integradas por forma a criar componentes
    - Testar o interface entre os vários componentes
  - Testes ao sistema
    - Todos os componentes são integrados
    - Teste do sistema como um todo
    - Testar interacções entre componentes

Testes unitários

- Processo de testar componentes individuais de forma isolada
- Processo de testes de defeitos
- “Unidades” podem ser
  - Funções ou métodos individuais de um objecto
  - Objectos com vários atributos e métodos
  - Componentes com interfaces bem definidos para aceder às suas funcionalidades

Testar objectos

- Cobertura completa dos testes a uma classe envolve
  - Testar todas as operações associadas com um objecto
  - Especificar e ler todos os atributos de um objecto
    - *Setters* e *getters*
  - Testar o objecto em todos os possíveis estados
- Herança pode tornar difícil o desenho de testes a objectos
  - Informação não está localizada

Testes autónomos

- Sempre que possível, os testes unitários devem ser automatizados
  - Executar e verificar os testes sem intervenção manual
- *Frameworks* de testes
  - Escrever e executar os testes
  - e.g.: JUnit
  - Fornecem classes de testes genéricas
    - Estendidas para criar testes específicos
  - Executar todos os testes implementados
    - Gerar relatórios

Testes autónomos

Componentes

- Setup
  - Inicialização do sistema e do teste
    - Inputs
    - Outputs esperados
- Chamada/Execução
  - Método, função ou objecto é testado
- Verificação/Assertion
  - Comparação do output da chamada com a output esperado
    - Teste passou com sucesso: True
    - Teste não passou: False

Testes unitários

Eficácia

- Testes unitários devem mostrar
  - Componentes testados fazem o esperado
  - Se houver defeitos ou problemas, devem ser revelados pelos testes

- Origem a dois tipos de testes
  - Testes que reflectem o uso normal do sistema e mostrar que o componente funciona de acordo com o esperado
  - Testes que usam inputs pouco comuns ou anormais
    - Verificar que o sistema é capaz de os processar e não fazem o sistema “crashar”

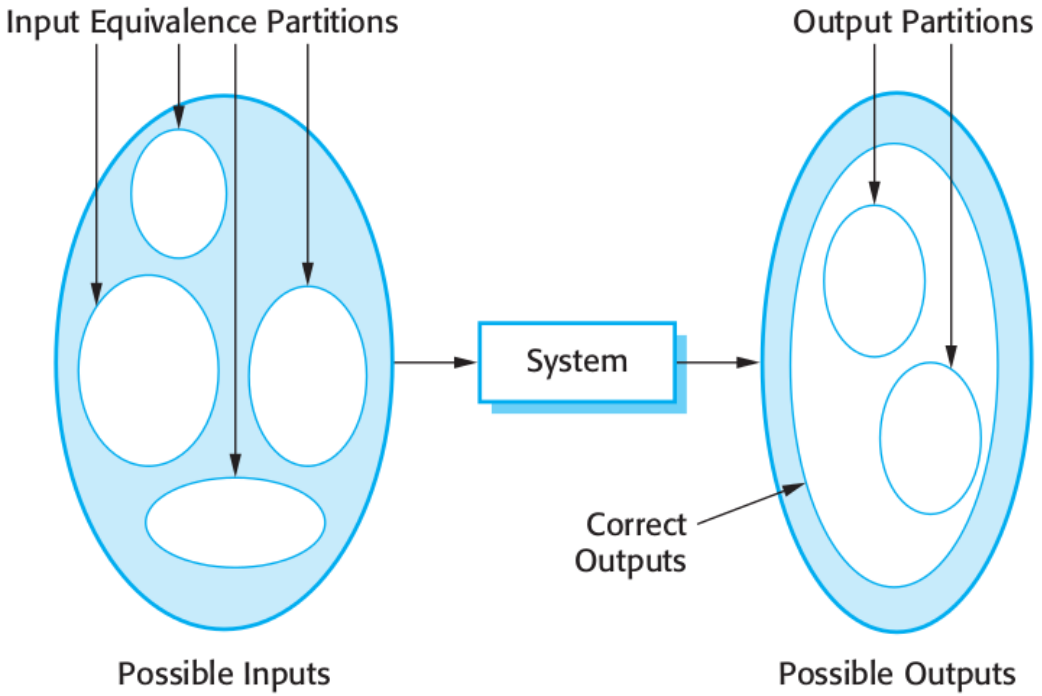
Estratégias de testes

- Particionar os testes
  - Identificar grupos de inputs que têm características comuns e devem ser processados da mesma forma
    - Escolher testes das categorias identificadas
- Baseado em regras/guias
  - Testes escolhidos de acordo com algumas regras
  - Regras/guias baseadas na experiência dos programadores
    - Nos erros comuns de programação

Particionar os testes

- Dados de input e de output enquadram-se em diferentes classes
  - Todos os membros de uma classe estão relacionados
- Cada classe
  - Partição equivalente
  - Sistema comporta-se de forma igual para cada membro da classe
- Casos de teste devem ser escolhidos a partir de cada partição

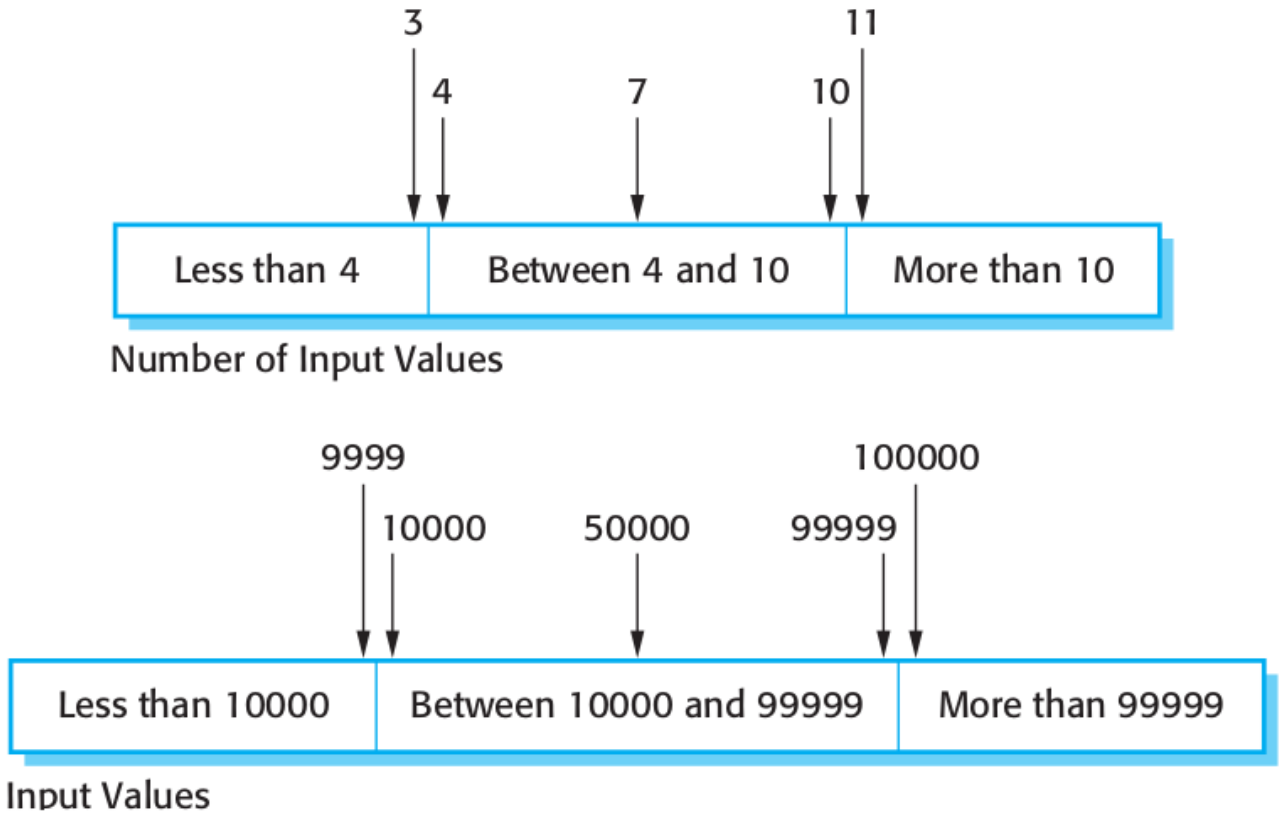
Partições equivalentes



Partições equivalentes – exemplo

- Um programa
  - aceita entre 4 e 10 inputs
  - cada input
    - 5 dígitos
    - > 10,000

Partições equivalentes - exemplo



Testar sequências

Guia

- Sequências
  - listas, arrays, etc…

- Testar usando sequências apenas com um valor
- Testar o primeiro, ultimo e elemento do meio
- Testar sequências com tamanho zero

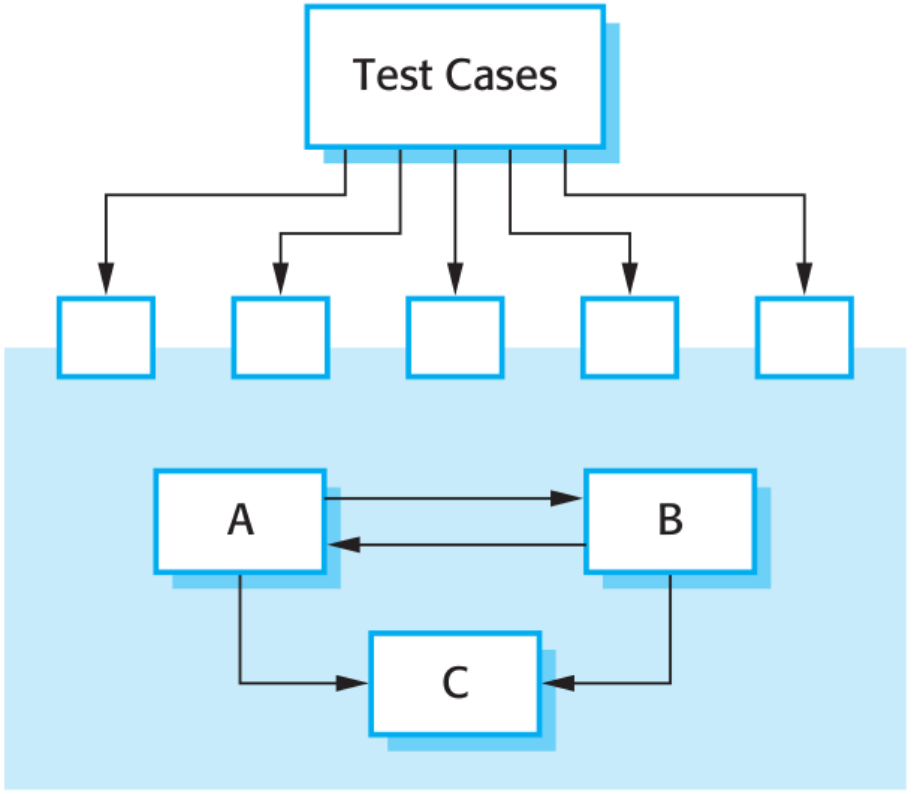
Guias de testes genérico

- Escolher inputs que forcem o sistema a gerar todas as mensagens de erros possíveis
- Escolher inputs que possam causar “buffer overflows”
- Repetir os mesmos inputs várias vezes seguidas
- Forçar a geração de outputs inválidos
- Forçar processos que produzam resultados muito pequenos ou muito grandes

Teste de componentes

- Componentes de software
  - Componentes compostos
  - Resultam da interacção entre objectos
- Acede-se à funcionalidade do objecto através do seu interface
- Teste de componentes compostos deve focar-se em mostrar que o interface tem um comportamento de acordo com a sua especificação
  - Deve-se assumir que os testes unitários estão terminados (e completos)

Testes de componentes



Testar interfaces

- Objectivo
  - Detectar falhas
    - Erros de interface
    - Suposições erradas

Erros de interfaces

- Má utilização
  - Chamada de componente usando o interface de forma errada, e.g.: ordem errada dos parâmetros
- Interface mal percebido
  - Utilização de componente de forma errada, devido a suposições erradas
- Problemas de sincronização
  - Acesso a dados desactualizados

Testar interfaces

Guia

- Desenhar testes de forma a que parâmetros tomem valores extremos do seu domínio
- Testar parâmetros do tipo apontadores com valores nulos
- Desenhar testes que façam os componentes falhar
- Colocar o sistema sob *stress*
- Variar a forma com os componentes são chamados/activados

Teste de sistema

- Teste de sistema durante a fase desenvolvimento
  - Integrar componentes
  - Criar uma versão do sistema
  - Testar o sistema integrado
- Foco
  - Testar interacções entre os componentes
- Objectivo
  - Testar se os componentes são compatíveis
  - Se os dados correctos são transferidos através dos interfaces
  - Testar o comportamento global do sistema

Testes de sistema e de componentes

- Testar a integração de componentes do sistema
  - Integração de componentes reutilizáveis
  - Integração de componentes desenvolvidos por outras equipas

Testes baseados em Use Cases

- Use Cases
  - Identificam interacções do sistema
  - Podem ser usados como base dos testes do sistema
- Cada Use Case
  - Envolve vários componentes
  - Testar um Use Case força a interacção entre os vários componentes

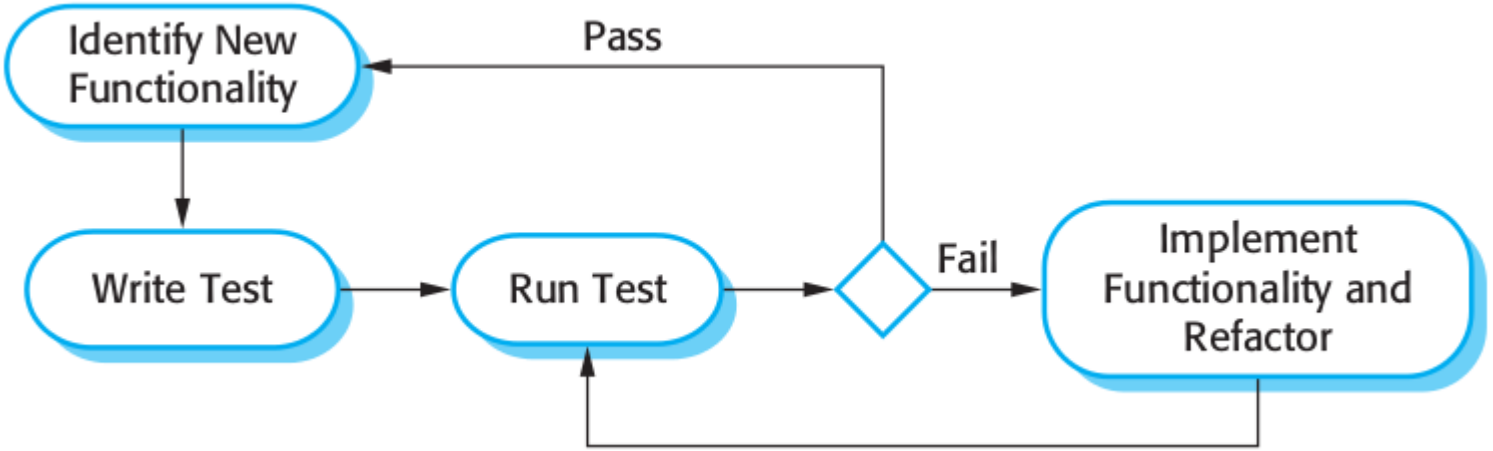
“Políticas”/Regras de testes

- Testes exaustivos
  - É impossível
  - Necessário criar regras que estabeleçam uma cobertura mínima
- Exemplos
  - Todas as funções acessíveis via menus devem ser testadas
  - Combinações de funções que são acedidas através do mesmo menu devem ser testadas
  - Todas funções que trabalho com dados introduzidos pelos utilizadores devem ser testadas

Test-driven development

- Método de programação
  - Intercalar testes com desenvolvimento de código
- Testes escritos antes de implementar código
  - Factor critico no desenvolvimento: Testes que “passem” com sucesso
- Código desenvolvido de forma incremental
  - Sempre com testes para esse incremento
  - Não se avança para outro incremento sem que todos os testes passem com sucesso
- Base dos métodos ágeis
  - Pode ser usado em metodologias baseadas em planos

Test-driven development



Test-driven development

Benefícios

- Cobertura do código com testes
  - Todos os segmentos de código estão associados pelo menos com um teste
- Testes de regressão
  - Criados de forma incremental à medida que o sistema é desenvolvido
- Debug simplificado
  - Quando um teste falha, torna-se óbvio qual o problema
- Documentação do sistema
  - Testes servem de documentação ao sistema

Teste de regressão

- Testar o sistema por forma a verificar se alterações não “partiram” código previamente funcional

- Processo manual
  - Difícil e dispendioso

- Processo automático
  - Simples
  - Todos os testes são executados sempre que existe uma alteração no sistema

- Todos os testes devem passar antes do código ser “commited”

Testes de Release

- Testar uma release específica do sistema
  - Com o objectivo de se colocar em uso fora do ambiente de desenvolvimento
- Objectivo
  - Mostrar que o sistema está pronto para ser usado
  - Mostrar que o sistema implementa o que especificado
- Testes numa “black-blox”
  - Ignora-se a implementação do sistema
  - Testes criados a partir das especificações

Testes de release e de sistema

- Testes de release
  - Tipo de testes de sistema

- Diferenças importantes
  - Testes de release não devem ser feitos pela equipa de desenvolvimento
  - Testes de sistema
    - Encontrar problemas
  - Testes de release
    - Verificar se o sistema cumpre os requisitos

Testes de desempenho

- Fazem também parte dos testes de release
  - Desempenho e fiabilidade
- Devem
  - reflectir o perfil de utilização do sistema
  - variar a carga do sistema de forma até que o sistema se torne inutilizável
- Testes de stress
  - Tipo de teste de desempenho
  - Sistema é sobrecarregado por forma a avaliar o seu comportamento quando falha

Testes de utilizador

- Utilizadores ou clientes indicam como testar o sistema
  - Ajudam na fase de testes

- Fase de testes essencial
  - Mesmo quando são feitos os testes de sistema e de release
  - Porquê?
    - Influencias dos utilizadores podem ter grande impacto na fiabilidade, desempenho e usabilidade do sistema. Estas influências não podem ser replicadas nos testes de sistema ou de release.

Testes de utilizador

Tipos

- Testes alfa
  - Utilizadores trabalham com a equipa de desenvolvimento para testar o software no ambiente de desenvolvimento
- Testes beta
  - Utilizadores testam uma release do sistema para podem experimentar o sistema e encontrar problemas, juntamente com a equipa de desenvolvimento
- Testes de aceitação
  - Decidir se o sistema está pronto para começar a ser usado

Métodos ágeis e testes de aceitação

- Métodos ágeis
  - Utilizador/Cliente faz parte da equipa de desenvolvimento
    - Responsáveis por decisões relacionadas com a aceitação do sistema
- Testes são definidos pelo utilizador/cliente e integrados com outros testes
  - Corridos de forma automática
- Não existe etapa (separada) de testes de aceitação
- Problema
  - Garantir se o cliente/utilizador representa os interesses de todos os interessados do sistema

Bibliografia

- Software Engineering, Ian Sommerville, 9th Edition, Addison-Wesley, 2010. Capítulo 8.