Engenharia de Software
Testes de Software

Marcello Thiry
marcello.thiry@gmail.com

LQPS
http://www.univali.br/lqps

Para pensar

- "*Program testing can be used to show the presence of bugs, but never to show their absence!*" (Edsger Dijkstra, 1972)

- "*Beware of bugs in the above code; I have only proved it correct, not tried it.*" (Donald Knuth, 1977)
- "*A relatively small number of causes will typically produce a large majority of the problems or defects (80/20 Rule).*" (Pareto principle)

(Joseph Juran, 1904-2008)

(Vilfredo Pareto, 1848-1923)

Por que testar?

- Encontrar e documentar defeitos
- Fornecer uma base para a percepção da qualidade do software
- Fornecer confirmação de suposições feitas no projeto (design) e especificações de requisitos por meio de demonstrações concretas
- Avaliar se o produto de software funciona como projetado (interação entre objetos, integração entre todos os componentes do software)
- Avaliar se o produto de software funciona como esperado (os requisitos foram implementados apropriadamente?)
- Identificar e garantir que os problemas encontrados sejam endereçados aos responsáveis pela implementação antes de liberar a distribuição do software

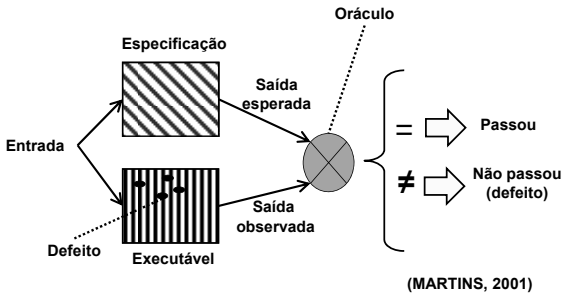
Definição de teste de software

- "*Executar o produto de software com o intuito de detectar a presença de defeitos*" (MYERS, 1979)
- "Processo de execução de um sistema ou componente sob condições específicas para detectar diferenças entre os resultados obtidos e os esperados" (IEEE 610, 1990)
- "Compreende qualquer atividade para a avaliação de um atributo ou capacidade de um programa ou sistema voltada para determinar se os resultados requisitados foram alcançados" (HETZEL, 1993)

Definição de teste de software

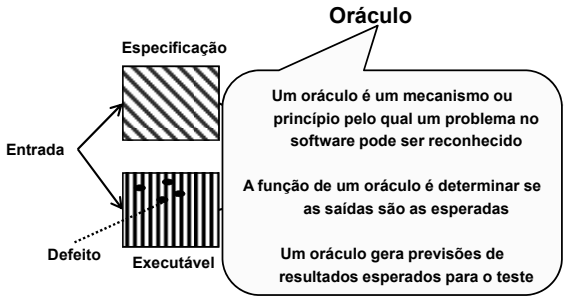
- Testes consistem na verificação **dinâmica** do funcionamento de um programa em um conjunto **finito** de casos de teste, cuidadosamente **selecionado** dentro de um domínio infinito de entradas, contra seu funcionamento **esperado** (SWEBOK, 2004)
  - **Dinâmico** – Execução
  - **Finito** – Existem muitos casos de teste
  - **Selecionado** – Técnicas diferem na seleção
  - **Esperado** – Funcionamento esperado deve ser verificado

Princípio do teste de software



(MARTINS, 2001)

Princípio do teste de software



Oráculo

- Como você sabe se os resultados do sistema de software são corretos?
- Oráculo: prediz o resultado esperado
  - Manual: você definiu o resultado esperado
  - Automático: outro sistema definiu o resultado esperado

Oráculo

- Um oráculo é qualquer programa, processo ou conjunto de dados que fornece o resultado esperado de um teste para o projetista de teste
- Mecanismo que estabelece os critérios que decidem se um determinado teste passou ou não
- Oráculo de teste é um mecanismo para verificar o comportamento da execução de teste
  - Automação é cara e está sujeita a erros durante a verificação
  - Projeto do oráculo é crítico para o planejamento de teste

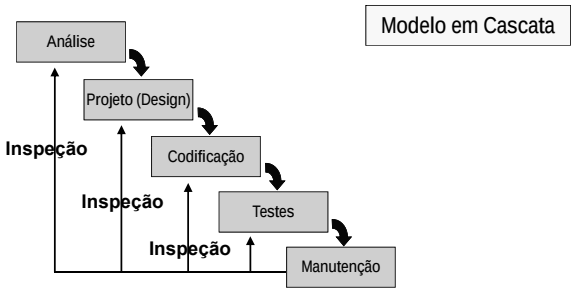
5 fontes para Oráculos (BEIZER, 1990)

- **Oráculo básico (Kiddie Oracle)**: apenas executar o programa e ver as saídas; se elas parecem estar corretas, então elas devem estar corretas
- **Suítes de Testes de Regressão**: executar o programa e comparar a saída com os resultados dos mesmos testes executados em uma versão anterior do programa
- **Dados validados**: executar o programa e comparar os resultados contra um padrão como uma tabela, fórmula ou outra definição aceita como saída válida
- **Suítes de Testes Adquiridos**: executar o programa contra uma suíte padronizada de teste que tenha sido previamente criada e validada; programas como compiladores, navegadores web, e processadores SQL são freqüentemente testados contra tais suítes
- **Programa existente**: executar o programa e comparar a saída com outra versão do programa (por exemplo, no caso de uma migração)

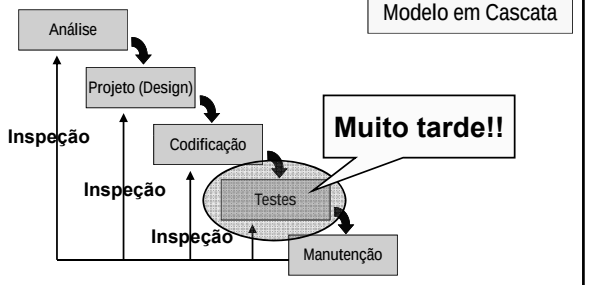
O que é considerado um defeito?

- O software não faz algo que a especificação diz que deve fazer
- O software faz algo que a especificação diz que não deve fazer
- O software faz algo que a especificação não menciona
- O software é difícil de entender, difícil de usar, lento ou será considerado pelo usuário final como simplesmente não correto
- O software não está de acordo com padrões definidos

Quando testar?



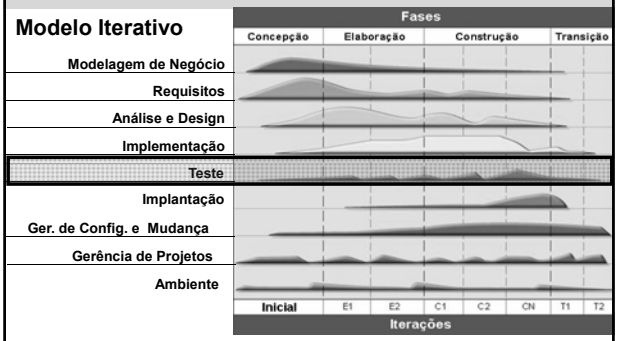
Quando testar?



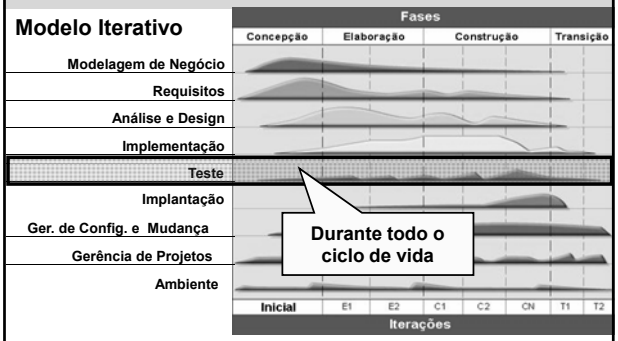
Quando testar?

- Testes devem ser uma parte integrada do desenvolvimento
- As atividades de teste devem ser iniciadas cedo
- Procedimentos de teste podem ser descritos desde a fase de especificação

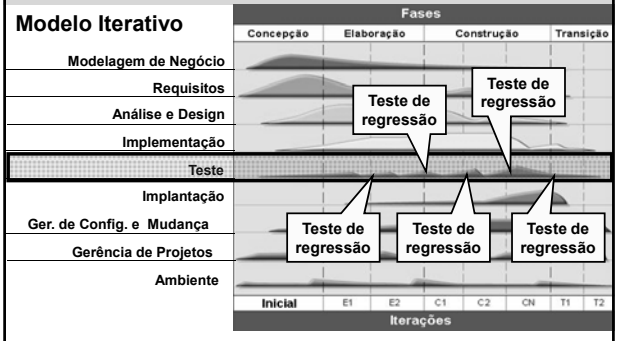
Quando testar?



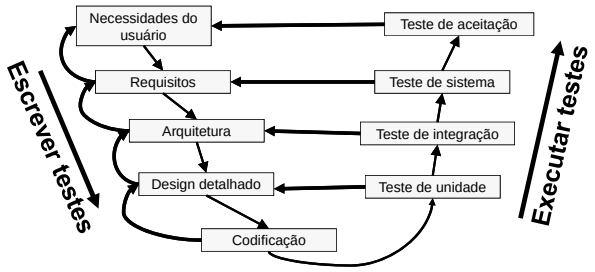
Quando testar?



Quando testar?



Quando testar (modelo V)?



Observações gerais

- Planejamento dos testes
- Casos de uso e cenários
- Validação e verificação incremental
- Integrações parciais

Caso de teste

- Especificam uma forma de testar o sistema:
  - O que testar?
  - Condições de teste
  - Resultado esperado
  - Procedimento
- Dados de teste: dados de entrada para os testes

Exemplo (cenário)

- Em uma loja virtual, um cliente pode selecionar uma das seguintes opções para entrega:
  a) Expressa
  b) 2 dias
  c) UPS
- O custo é baseado no preço do produto e no método de entrega
- Se nenhum método de entrega é selecionado, o cliente recebe uma mensagem de erro "*Por favor, selecione uma opção de entrega*"
- O processo do pedido não pode ser continuado até que a opção de entrega seja selecionada e confirmada

Exemplo (cenário)

Tipo de entrega	Valor total da compra			
	Menor que $ 20,00	**Entre $ 20,00 e $ 29,99**	**Entre $ 30,00 e $39,00**	**Maior que $ 40,00**
Expressa	$12,00	$ 16,00	$ 22,00	$ 27,00
2 dias	$ 8,00	$ 6,00	$ 4,00	$ 0,00
UPS	$ 6,00	$ 6,00	$ 6,00	-

Exemplo (caso de teste 1)

ID#	Condição de teste	Resultado esperado	Procedimento	Passou/ Falhou?	Defeito encontrado
1	Cliente pede produtos totalizando menos que $20,00 e seleciona entrega expressa	O custo da entrega a ser aplicado é de $12,00	1. O cliente acessa a página segura de pedido 2. O cliente adiciona produtos no carrinho, totalizando menos que $20,00 3. O cliente seleciona a opção de entrega expressa 4. O cliente confirma os itens do carrinho, o total do pedido e a opção de entrega		

Exemplo (caso de teste 2)

ID#	Condição de teste	Resultado esperado	Procedimento	Passou/ Falhou?	Defeito encontrado
2	Cliente pede produtos totalizando menos que $30,00 e mais que $20,00, e seleciona entrega expressa				

Exemplo (dados de teste)

- Pedido (dados de teste 1):
  - 1 cópia do livro “Engenharia de Software”
- Pedido (dados de teste 2):
  - 1 DVD do show “Iron Maiden: Rock in Rio”
  - 1 DVD do filme “2 Filhos de Francisco”

Casos de teste

- **Casos de teste positivos:**
  - O programa **faz o que deve fazer**?
  - Exercitar uma funcionalidade do sistema
- **Casos de teste negativos:**
  - O sistema **não faz o que não deve fazer**?
  - Exercitar a robustez / tratamento de erros

Procedimentos de teste

- Especificam como executar casos de teste
- Instruções para o testador (testes manuais)
- Instruções para interação com a ferramenta de automação de teste para criar, integrar e executar componentes de teste
- Freqüentemente, representam descrições de fluxo de eventos, incluindo como fazer a entrada de dados, como verificar resultados

Qualidade de casos de teste

- Nenhum defeito foi detectado:
  - PARABÉNS!! → O programa está ok!

  OU
  - CUIDADO!! → O caso de teste pode não ser adequado?

Qualidade de casos de teste

- Nenhum defeito foi detectado:
  - PARABÉNS!! → O programa está ok!

  OU
  - CUIDADO!! → O caso de teste pode não ser adequado?
- **Nenhum defeito detectado = não existem defeitos?**

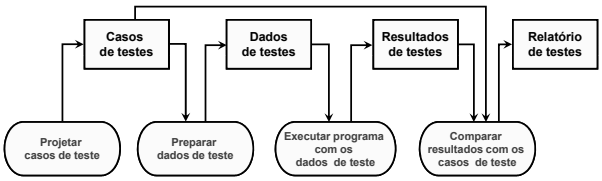
Qualidade de casos de teste

- Nenhum defeito foi detectado:
  - PARABÉNS!! → O programa está ok!

  OU
  - CUIDADO!! → O caso de teste pode não ser adequado?
- **Nenhum defeito detectado = não existem defeitos?**
  - Impossível afirmar
  - Os casos de teste têm a cobertura necessária?
  - Qual a cobertura necessária?
  - E os dados de teste? Eles são adequados?
  - Como os testes foram executados?

Processo de teste

- Um teste bem sucedido para a detecção de defeitos é aquele que faz com que o sistema opere incorretamente e, como conseqüência, apresente um defeito existente



Processo de teste para a detecção de defeitos (Sommerville, 2003)

O que deve ser testado?

- Identificar os componentes e funcionalidades (*features*) do software que devem ser testados
- Priorizar
  - Positivos/negativos
  - Os mais graves
  - Os mais prováveis
  - Falhas anteriores
  - …

O que deve ser testado?

- Considerar aspectos funcionais: o que o sistema deve fazer
- Considerar aspectos não-funcionais: desempenho, usabilidade, robustez, restrições, etc
- Considerar tempo (prazos), recursos, ambiente, riscos
- Nem tudo pode ser testado, e o que é testado não pode ser completamente testado

Quem deve testar?

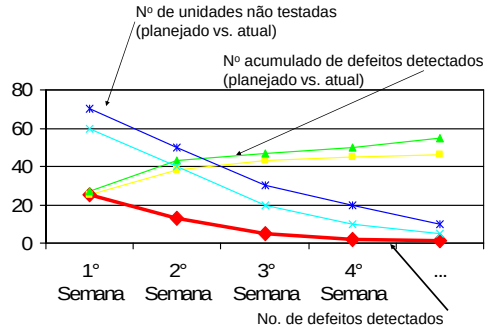
Desenvolvedor

- **Conhece o sistema**
- **Mas testa "gentilmente"**
- **Direcionado a entrega**

Testador independente

- **Não conhece o sistema**
- **Foco é detectar defeitos**
- **Direcionado a qualidade**

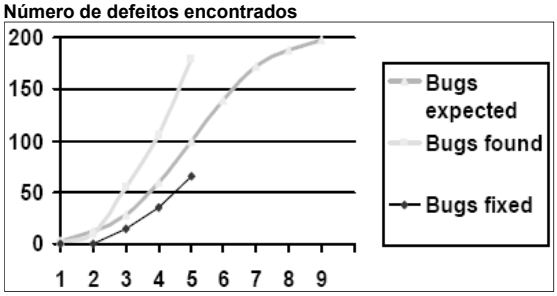
Monitoramento do processo de teste



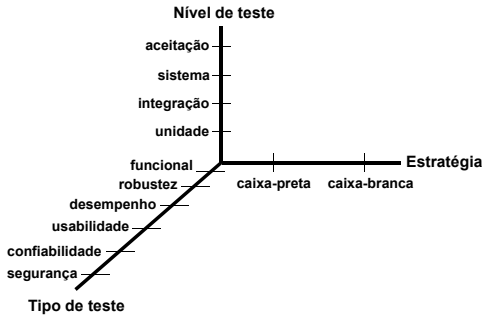
Analisando o processo de teste

- Exemplos de medidas:
  - Número de casos de testes definidos (por área funcional)
  - Número de casos de testes executados vs. número total de casos de testes
  - Taxa de casos de teste que passaram, que falharam, que não foram executados
  - Número de defeitos detectados por dia/projeto/etc., ou esforço de teste por defeito detectado
  - Defeitos classificados por "criticidade", por *feature*, etc...

Analisando o processo de teste

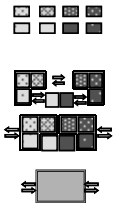


Dimensões

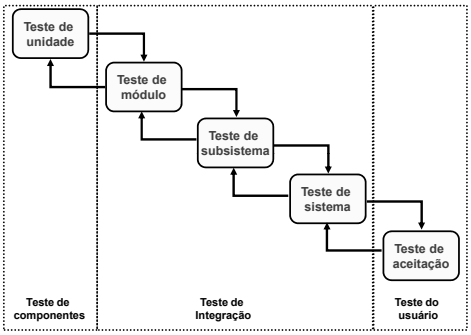


Níveis de Teste

- Nível de teste é um grupo de atividades de teste focado num certo nível do alvo de teste:
  - **Teste de Unidade**: comparar uma unidade de código (classes, funções, ...) com a especificação do *design* detalhado
  - **Teste de Integração**: grupos de classes/ subsistemas/ componentes são testados contra a especificação do *design* alto-nível para assegurar consistência de interfaces e aderência ao design
  - **Teste de Sistema**: comparar um sistema integrado com os requisitos do sistema de software
  - **Teste de Aceitação:** validar a aceitação dos sistema de software pelo usuário



Níveis de Teste



Tipos de teste

- Tipo de teste é um grupo de atividades visando a avaliação de um sistema com respeito a um conjunto associado de **características de qualidade**
- Exemplos:
  - Teste funcional
  - Teste de desempenho
  - Teste de carga (stress)
  - Teste de usabilidade

Testes com ou sem roteiros

- **Testes com roteiros**: casos de teste são pré-definidos e documentados detalhadamente
  - Um *scripts* pode ser manual ou automatizado
- **Testes sem roteiros**: são usualmente testes manuais sem a adoção de casos de teste detalhados
  - Pode ser feito de forma disciplinada, planejada e bem documentada
  - ou de forma ad-hoc ...

Quanto testar?

- Como saber **quanto** devemos **testar** e **quando** podemos **parar** de testar?
  - Testar tudo?
  - **Cobertura de teste**
  - **Critério de completude / grau de adequação dos testes**

Critério de completude

- Baseado, por exemplo, no grau da cobertura de testes, taxas de defeitos, "criticidade" dos defeitos
- Somente prazo não é um bom critério!
- Medidas:
  - Medidas de cobertura
    - Quando o percentual "suficiente" da estrutura do sistema foi exercitado?
  - Garantia empírica
    - Quando a taxa de defeitos/testes se torna um valor mínimo constante
  - Plantação de erros (*mutant testing*)
    - Percentual de defeitos detectados que foram plantados é proporcional ao número de defeitos reais detectados

Teste com mutantes

- *Mutating testing* é uma variação de *fault seeding* (plantação de defeitos)
- Introduzir pequenas modificações no código original (substituir operações, constantes, etc.) para criar um conjunto de unidades similares ("mutantes")
- Executar os casos de teste originais com cada mutante
- Se um mutante passa por todos os casos de teste (se o mutante não é "morto"), então:
  - ou o mutante é equivalente ao código original ⇨ é ignorado
  - ou não é equivalente ⇨ devem ser desenvolvidos mais casos de teste (a qualidade dos casos de teste não é adequada)
- A taxa de mutantes "sobreviventes" dá uma estimativa da taxa de defeitos ainda não detectados que possivelmente existem no código

Exemplo: triângulo

- Como entrada, o programa recebe três valores inteiros
- Os três valores são interpretados como os comprimentos dos lados de um triângulo (informados em cm)
- Como resultado, o sistema apresenta uma mensagem informando se o triângulo é escaleno, isósceles ou eqüilátero
  - Escaleno: todos os lados têm um comprimento diferente
  - Isósceles: dois lados têm o mesmo comprimento
  - Eqüilátero: todos os lados têm o mesmo comprimento
- Em um triângulo válido, nenhum lado deve ser de comprimento zero ou negativo e cada lado precisa ser menor que a soma de todos os lados dividido por 2

Exemplo: triângulo

- Como entrada, o programa recebe três valores inteiros
- Os três valor… …os dos lados de um triân…
- Como resu… …nformando se o triângu…
  - Escaleno:…
  - Isósceles:…
  - Eqüilátero…
- Em um triângulo válido, nenhum lado deve ser de comprimento zero ou negativo e cada lado precisa ser menor que a soma de todos os lados dividido por 2

<u>Exercício</u>:
defina um conjunto de dados de teste (especifique valores de entrada) para testar adequadamente este programa!

Princípios de teste (MYERS, 1979)

- Caso de teste = entrada (dados de teste) + saída esperada
- Dados de teste devem ser definidos para dados válidos, inválidos e inoportunos
- Evite testar seus próprios programas, a menos que seja com auxílio de uma ferramenta
- Determine se o software faz o que é esperado, mas também, se ele não faz algo indesejável
- Nunca planeje testes assumindo que os defeitos não serão encontrados

Princípios de teste (MYERS, 1979)

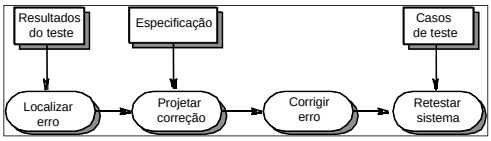
- Nunca jogue fora os casos de teste, a não ser que esteja jogando fora o software também
- A probabilidade de detectar defeitos em uma parte do software é proporcional ao nº de defeitos já detectados
- Um bom caso de teste é aquele que tem uma alta probabilidade de detectar novos defeitos
- Verifique cuidadosamente os resultados de cada caso de teste
- Testes devem ser planejados desde o início do desenvolvimento

Dificuldades e limitações

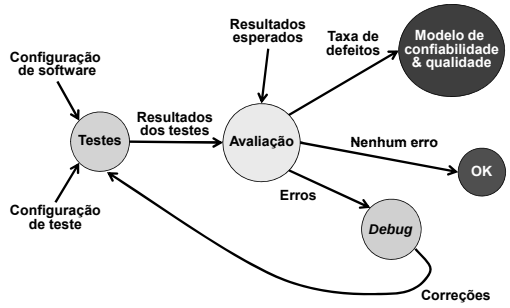
- Detecção de defeitos se dá através da ocorrência de falhas
- Testes sistemáticos necessitam especificação:
  - Defeitos nos requisitos podem não ser detectados
  - Especificação incompleta ou ambígua pode levar a resultados incorretos ou inadequados
- É impossível testar exaustivamente um software
- Certas atividades de teste não podem ser automatizadas (problemas intratáveis e *indecidíveis*)
- Qualidade dos testes depende dos dados de teste selecionados

Testes x Depuração

- **Testes de defeitos ≠ depuração (*debugging*)**
  - Testes (V&V): relacionado com a **detecção** de defeitos
  - Depuração: relacionado com a **localização e correção** de erros que ocorrem em tempo de execução
- Depuração compreende a formulação de uma hipótese sobre o comportamento do sistema e então os testes destas hipóteses para localizar o erro



Visão geral



Referências

- ABRAN, A. e MOORE, J. (eds). (2004). *Guide to the Software Engineering Body of Knowledge - SWEBOK*, IEEE.
- BEIZER, B. (1990). *Software Testing Techniques* (2nd ed.). Van Nostrand Reinhold.
- HETZEL, B. (1993). *The Complete Guide to Software Testing*. Wiley & Sons.
- IEEE 610.12:1990. *Standard Glossary of Software Engineering Terminology*, IEEE.
- MARTINS, E. (2001). *Notas de aula*. UNICAMP.
- MYERS, J. (1979). *The Art of Software Testing*. Wiley & Sons.
- SOMMERVILLE, I. (2003). *Engenharia de Software* (6ª ed.), Pearson.

Contato

Marcello Thiry
thiry@univali.br
marcello.thiry@gmail.com

LQPS
http://www.univali.br/lqps