Laboratório de Engenharia de Software

Princípios de Teste 1

Arndt von Staa
Departamento de Informática
PUC-Rio
Março 2015

Laboratório de Engenharia de Software

Especificação

- Objetivo da aula
  - discutir os conceitos e princípios básicos que governam a realização de testes.
- Justificativa
  - estabelecer uma terminologia comum
  - estabelecer o embasamento conceitual
- Texto
  - Pezzè, M.; Young, M.; Teste e Análise de Software; Porto Alegre, RS: Bookman; 2008, capítulos 2 e 3
  - Staa, A.v.; Programação Modular; Rio de Janeiro: Campus; 2000; capítulo 15

Limitação intrínseca

Laboratório de Engenharia de Software

- *Teste somente sinaliza a presença de defeitos, nunca a sua ausência.*
  - E.W. Dijkstra

Limitação intrínseca

Laboratório de Engenharia de Software

- A criação de suítes de testes é restrita pela necessidade (econômica) de selecionar poucos casos de teste de um gigantesco* conjunto de possíveis casos
  - Independentemente de quão cuidadoso você seja, você deixará de incluir casos relevantes
  - Independentemente da perfeição do seu trabalho, você nunca encontrará o último defeito e, se encontrar, jamais o saberá
    - Kaner, C.; Falk, J.; Nguyen, H.Q.

Pergunta básica: um teste é um exemplo de uso. Pode-se inferir a corretude total a partir de poucos exemplos?

* Gigantesco: na maioria dos casos quer dizer: *virtualmente infinito*

Limitação intrínseca

- Indecidibilidade: não pode existir um programa *T* capaz de determinar: se para todas as entradas *D* fornecidas a um programa qualquer *P*, *P* eventualmente pára ou não. (Turing 1936)
- Corolário 1: é impossível criar um programa *V* capaz de verificar se dois programas *E* e *P* são equivalentes.
  - equivalente: para todas as entradas válidas geram exatamente os mesmos resultados.
- Corolário 2: é impossível criar um programa *V* capaz de verificar se um programa *P* é uma implementação exata de uma especificação *E*.

Terminologia

- Caso de teste: define um cenário e um conjunto de dados que visam testar um determinado aspecto do artefato sob teste (AST)
- Cenário de teste: o estado que o contexto deve satisfazer para que o caso de teste possa ser executado
  - ex. banco de dados para teste está disponível e ativo
- Massa de teste (teste automatizado): um conjunto de casos de teste correspondente a uma execução do teste automatizado
- Suíte de teste: conjunto de todos os casos de teste a serem utilizados para testar o artefato sob teste
- Requisitos: aspectos não funcionais que devem ser respeitados pelo caso de teste

Exemplo de caso de teste formulário

IDENTIFICADOR DO CASO DE TESTE: Login001

DESCRIÇÃO: O usuário realiza um Login bem sucedido

PRE-CONDICOES:	AÇÕES DO USUÁRIO:	POS-CONDICOES:
• Sistema Sis está vinculado ao cadastro de usuários TesteSis.usuarios • O cadastro de usuários está criptografado com a senha XPTO### • A janela de login está aberta	• O usuário digita a identificação: *joaoSilva* • O usuário digita a senha: *joao####* • O usuário digita exatamente os *caracteres de controle* • O usuário clica "Login"	• A janela de login está fechada • O sistema Sis recebeu os dados *<autorizado, {a,b,c}>*

Requisitos:

- Segurança: todos os espaços que contenham dados do cadastro e/ou do usuário foram obliterados antes de retornar ao sistema Sis
- Segurança: a senha não foi exibida no monitor
- Segurança: os campos a preencher foram fornecidos em branco

Ações relacionadas com o teste

- Especificar e desenvolver instrumentação de apoio
- Formular as suítes de teste
- Efetuar o testes
- Verificar se o comportamento do artefato sob teste corresponde ao especificado e ao desejado
  - usualmente basta o especificado, porém pode ser que a especificação esteja errada
- Diagnosticar a causa das falhas observadas
- Depurar (*debug*) – remover as causas
  - sem introduzir novos defeitos
  - sem comprometer a qualidade de engenharia
- Assegurar alta eficácia e eficiência ao teste e à correção
- Repetir a realização dos testes com frequência
  - em especial durante a manutenção
- Coevoluir a suíte de teste e artefatos de apoio

Requisitos de um bom teste

- Sensitividade – se um teste exercita um defeito, o artefato deve falhar da mesma forma sempre que seja exercitado do mesmo modo
- Explicitude – se existe alguma restrição ou requisito a ser satisfeito, estes devem estar explicitados
- Modularidade – separar o todo em pedaços, cada qual podendo ser testado separadamente
  - o processo de testes testa artefatos, componentes, sistemas
- Visibilidade – o progresso dos testes deve ser observável
- Feedback – os processos de teste e de depuração devem identificar e evidenciar defeitos recorrentes
- Padronização – eliminar ou reduzir a chance de injetar defeitos recorrentes, ou de cometer enganos repetidos durante os testes

Princípios básicos de um teste

- Deve-se adotar uma postura destrutiva ao testar
  - o objetivo é descobrir em que condições **não** funciona
    - deseja-se reduzir o risco decorrente do uso do software
  - o objetivo não é ver se funciona

O que é um teste?

- Teste é um experimento controlado visando a identificação de problemas contidos em:
  - artefatos elementares
    - unidades
    - módulos
    - projetos a partir dos quais se gera código
  - componentes
    - conjuntos de módulos e/ou componentes, bibliotecas, dlls, dos
  - construtos
    - implementações parciais capazes de demonstrar o funcionamento
  - programas
  - sistemas

Problema: desvio do que é realizado com relação ao especificado ou desejado (funcional, não funcional, restrição). Pode envolver código, interface com o usuário, hardware, especificação, etc.

O que é um teste?

- Perguntas:
  - como conduzir os experimentos → quais casos de teste?
  - como selecionar os dados dos casos de teste?
  - como confrontar resultados esperados com os resultados observados?
  - como replicar os experimentos?
    - replicar para poder confirmar os resultados observados
    - replicar para poder diagnosticar os problemas
    - mostrar que os problemas observados foram totalmente sanados
  - como replicar, a partir dos relatos dos usuários, as falhas ocorridas em produção?
    - como induzir o usuário a informar correta e completamente a falha ou a solicitação de evolução?

Princípios básicos de um teste

- Para cada caso de teste é necessário verificar se os resultados obtidos correspondem a o que é esperado
  - implica a necessidade de se saber de antemão qual deveria ser o resultado a ser produzido para determinado conjunto de estímulos
    - implica a necessidade de se dispor de uma especificação
  - achismo ao testar é convite a problemas futuros
    - ex. fornece alguns dados, executa, observa o resultado, e acha que está, ou não, correto

É erro grosseiro pensar que testar é
somente exercitar o artefato com alguns dados,
sem preocupar-se de antemão com o resultado

Oráculo

- Oráculo: dado um resultado observado, o oráculo indica se este corresponde a o que foi especificado (ou é esperado)
- Mais formal
  - Seja *p* é um programa que implementa uma especificação (função) *f* operando sobre um domínio *D*
  - um oráculo é um predicado que retorna **`true`** se e somente se a execução de *p* para um dado $d \in D$ produz um resultado em conformidade com a especificação *f* para esse mesmo dado *D*

Dado um d **`∈`** D : *oráculo*(*p*(*d*) , *f*(*d*))
será **`true`** se executou como esperado
e **`false`** caso contrário

Wikipedia: Oráculos são seres humanos que fazem predições ou oferecem inspirações baseados em uma conexão com os deuses. (ver também Apolo, Delphi, pitonisa, sibila)

Oráculo

Laboratório de Engenharia de Software

- Todo teste requer um oráculo
  - dado um conjunto de casos de teste *T* precisamos sempre ser capazes de verificar se é verdadeiro:

$$\forall\, t \in T : \textbf{\textit{oráculo}}(\, p(\, t\,)\,,\, f(\, t\,))$$

  - a forma de avaliar o oráculo deve estar definido antes da criação da suíte de teste e antes do desenvolvimento do artefato sob teste
    - o oráculo determina como instrumentar o código para que possa ser avaliado
    - ferramentas de teste automatizado influenciam como desenvolver o programa sob teste, os módulos específicos de teste e a suíte de teste
    - alguns oráculos podem permanecer ativos na execução de produção
      - ex. assertivas

Problemas do oráculo

Laboratório de Engenharia de Software

- Nem sempre é possível construir um oráculo pouco complexo capaz de discernir se *p*(*t*) realmente corresponde a *f*(*t*)
  - pode ser impossível criar o oráculo
    - ex. o oráculo necessita de cálculos similares aos de *p*
    - ex. o oráculo necessita de serviços impossíveis de disponibilizar
  - o uso do oráculo pode ser inviável na prática
    - ex. o oráculo pode requerer tempo demais para processar
    - ex. o oráculo pode requerer recursos demasiados para executar
  - o custo do oráculo e da instrumentação é muito alto
    - ex. desenvolver o oráculo requer demasiados recursos
    - ex. desenvolver o oráculo requer conhecimentos muito especializados
  - o oráculo pode conter defeitos
    - quanto mais complexo, maior a probabilidade de conter defeitos
    - falsos positivos: o oráculo acusa problema que não existe
    - falsos negativos: o oráculo deixa de acusar problema que existe

Problemas do oráculo

- Mesmo tendo boas especificações, nem sempre é possível determinar se o resultado corresponde esperado
  - processamento não determinístico (assíncrono), exemplos:
    - multi-programação e/ou multi-processamento
    - interação baseada em mensagens de fato (ex. agentes)
      - chamar um método não é uma mensagem, é uma chamada convencional
  - difícil determinar o que de fato é esperado, exemplos
    - aprendizado, inteligência artificial
    - comportamento emergente
    - sistemas dependentes de dados aleatórios
  - depende de condições difíceis de provocar, exemplos:
    - esgotamento de recursos
    - desempenho
    - escalabilidade
    - sequências anômalas de estímulos
  - . . .

Weyuker, E.J.; "On testing non-testable programs"; *The Computer Journal* 25(4); 1982; pp 465–470

Tipos de oráculo

- Comparação: resultados obtido e esperado são comparados, devendo:
  - ser iguais
  - ou diferir dentro de uma tolerância estabelecida
    - isso é usualmente o caso ao testar dados vírgula flutuante

Tipos de oráculo

- Inversa: verifica se, ao aplicar uma função inversa à função sob teste, são reproduzidos os dados de entrada, exemplos
  - `[M]*[M]`$^{-1}$ `= aprox [I]`
  - `arcsin( sin( alpha )) = aprox( alpha )`
  - controle de recursos usados (no caso de recursão)
    - antes da execução guarda-se a lista de recursos alocados: base
    - durante a execução: ||alocações|| >= ||exclusões|| ;
    - ao final da execução: (||alocações|| == ||exclusões||) && (base == conjunto de recursos ainda alocados)
    - restrição: durante o processamento não é permitido alterar recursos registrados na base

`[I]` – matriz identidade

|| x || em que x é um conjunto, é a cardinalidade (número de elementos) de x

Tipos de oráculo

- Assertiva: o resultado obtido deve satisfazer um conjunto de assertivas (condições)
  - também conhecidos por contratos
- Necessita de instrumentação:
  - pré-condições verificam se o processamento de *p* pode ser realizado
  - pós-condições verificam se o processamento *p* foi corretamente realizado
    - devem verificar se as alterações desejadas foram realizadas
  - assertivas pontuais verificam se as estruturas de dados satisfazem as suas restrições estruturais em determinado ponto
    - invariantes: devem valer antes e depois de executar *p*
  - assertivas estruturais verificam se estruturas de dados satisfazem integralmente as suas restrições estruturais

Tipos de oráculo

- Caminho – verifica se o caminho executado é idêntico ao caminho proposto no caso de teste
  - selecionam-se dados para percorrer exatamente um caminho a ser testado
- Caminho definição: a execução do teste percorre uma sequência de pontos conhecidos, exemplos
  - traço (*trace*) seletivo – pontos marcados de forma *ad hoc*
    - os *printf*'s (*watch*'es) inseridos para verificar corretude local
  - autômato – passagem por pontos correspondentes à entrada em estados de um autômato definido no projeto do artefato
  - diagrama de sequência – chamadas e retornos definidos no diagrama de sequência
  - grafo de chamada – os métodos são os vértices e chamadas as arestas, caminho é um percurso válido nesse grafo

Tipos de oráculo

- Contratos temporais: explora-se um *log* de eventos em que cada evento registra o *time stamp* **e** dados selecionados na forma <tag, valor> e verifica-se se a ordenação e a espera entre eventos contendo determinados tags está de acordo com o especificado, exemplo
  - após evento X o evento Y deve ocorrer em até no máximo 0,2s
  - após X não pode ocorrer Z antes de pelo menos 0,4s
- Contratos temporais são úteis em sistemas distribuídos
- Cada processo (*thread* / máquina) gera registros de eventos que são centralizados e ordenados pelo *time stamp*
  - procuram-se trechos do log iniciando e terminando em eventos caracterizados por determinados pares <tag, valor>

Rocha, Pedro G.S.; "Um mecanismo baseado em logs com meta-informações para a verificação de contratos em sistemas distribuídos"; *Dissertação de Mestrado, PUC-Rio*; 2014

Tipos de oráculo

- Visual: (manual) o resultado é examinado pelo testador para ver se confere com:
  - o indicado no roteiro de teste
    - suíte de teste descrita através de formulários
  - o intuitivamente esperado – isso é péssimo
  - o comportamento desejado (IHC)

- Aceitação: (usualmente manual) usa-se o artefato em contexto de, ou parecido com, produção e verifica-se se o construto sob teste corresponde às *atuais* necessidades e expectativas do usuário

Aspectos teóricos

- Seja *p* é um programa que implementa uma especificação *f* operando sobre um domínio *D*

  Um teste $T \subseteq D$ de *p* é confiável sse (se e somente se)

  $$?(\forall t \in T : oráculo(p(t) , f(t)) == \texttt{true}) \Rightarrow$$
  $$(\nexists d \in D : oráculo(p(d) , f(d)) == \texttt{false})$$

- Dito de outra maneira
  - Uma suíte de teste é confiável sse não sendo encontradas falhas ao testar, então não há mais falha remanescente.
- Confiabilidade não é mensurável, pois não se sabe quais são os defeitos remanescentes.

Aspectos teóricos

- Evidentemente confiabilidade é um ideal
  - é um ideal, pois implica que o teste deve ser capaz de observar todos os defeitos porventura existentes
    - quase sempre impossível
    - além do conjunto de defeitos ser inerentemente desconhecido
  - o real é sobrarem 0 ou mais defeitos remanescentes
- Eficácia do teste – é o percentual dos defeitos existentes que foi identificado pelo teste

  (defeitos_observados / defeitos_existentes) * 100
  - problema: o conjunto de defeitos existentes é desconhecido
    - usando mutantes pode-se obter uma aproximação da eficácia

Mutante – é uma versão contendo um ou mais defeitos propositalmente introduzidos. Obviamente esses defeitos são conhecidos e um bom teste deve ser capaz de identificá-los.

Projeto do teste – eficiência e eficácia

- Testes são
  - pouco eficientes
    - a cada execução do teste são encontradas poucas falhas
  - pouco eficazes
    - estatísticas (antigas) mostram que, de maneira geral, testes encontram de 65% a 75% de todos os problemas identificados
      - isso decorre dos testes terem sido mal feitos?
      - ou é uma propriedade intrínseca?
  - adicionalmente podem ser demorados e caros
    - especialmente no caso de teste manual

Projeto do teste – eficiência e eficácia

- Como melhorar a eficácia e a eficiência e reduzir a demora?
  - planejar e projetar cuidadosamente como será realizado o teste
    - estabelecer uma estratégia de teste
    - utilizar apropriados critérios de seleção de casos de teste
  - automatizar a realização dos testes
    - melhora a eficiência
    - reduz a demora
  - automatizar a geração das suítes de teste
    - melhora a eficácia

Projeto do teste – eficiência e eficácia

- A eficácia do teste depende do critério de seleção dos casos de teste
  - eficácia está relacionada com a cobertura do código durante os testes
  - à medida que os artefatos se tornam mais abrangentes (nível de abstração mais alto) menor será a cobertura
- Um critério de seleção é um método usado para a escolher os casos de teste que comporão uma suíte de teste
  - se obedecido, gera um conjunto de casos de teste capaz de identificar falhas causadas por uma determinada classe de defeitos
  - existem diversos critérios de seleção de casos de teste
    - cada um tem um domínio de efetividade

Categorias de teste

- Teste caixa fechada, ou caixa preta
  - utiliza estritamente a especificação para gerar os casos de teste
- Teste caixa aberta, ou caixa branca, ou caixa de vidro
  - utiliza a estrutura do código para gerar os casos de teste e a especificação para determinar os resultados esperados
- Teste caixa entreaberta, ou caixa cinza
  - utiliza a organização dos dados ou aspectos da estrutura do código para gerar os casos de teste e a especificação para determinar os resultados esperados
- Teste aleatório
  - gera os casos de teste aleatoriamente segundo algum conjunto de regras e verifica os resultados através de medições, e/ou assertivas executáveis, e/ou oráculos dinâmicos

Modos de teste

- Teste estrutural
  - teste caixa aberta
    - baseado na especificação e na totalidade do código
  - teste caixa entreaberta
    - baseado na especificação e em aspectos do código
- Teste funcional
  - teste caixa fechada
    - idealmente projetado pelos desenvolvedores (?) em conjunto com os (representantes dos - *proxy*) clientes e/ou usuários
- Teste de requisitos não funcionais
  - verifica se determinados requisitos de qualidade são satisfeitos
- Teste de regressão
  - verifica se o que não foi alterado continua operando como antes
- Teste de aceitação
  - teste caixa fechada
    - projetado pelos (representantes dos) clientes e/ou usuários
- Teste de liberação (disponibilização)
  - verifica se o que vai ser entregue opera conforme esperado

Nível de abstração do teste: unidade

Laboratório de Engenharia de Software

- Teste de unidade
  - Foco: exame minucioso do código e da interface disponibilizada pela unidade
  - Testa uma unidade independente das outras
  - Exemplos de unidades
    - funções
    - classes
    - módulos
    - componentes ?
    - características (*features*) ?
    - agentes ?
    - aspectos ?
    - páginas *Web*
    - *widgets*
    - *applets*
      - um fragmento que é executado no contexto de outro programa, usualmente um *browser* ex. JavaScript
    - *servlets*
      - recurso adicionado a um servidor

Nível de abstração do teste: componentes

Laboratório de Engenharia de Software

- Teste de integração
  - Testa a composição de unidades previamente testadas
  - Foco 1: exame minucioso do uso das interfaces entre os componentes integrados
    - parâmetros
    - variáveis globais
    - mensagens
    - exceções
    - estados
      - corotinas
    - sincronização
    - protocolos
    - recuperação (*roll back*)
  - Foco 2: testar um artefato composto como se fosse uma unidade

Problema: explosão combinatória, à medida que cresce o número de decisões, muito mais casos de teste são necessários para um teste abrangente ➔ isso pode tornar o teste inviável na prática

Nível de abstração do teste: sistema

- Teste alfa
  - teste do sistema usando o ponto de vista do usuário, realizado pelo desenvolvedor nas instalações do desenvolvedor
- Teste beta
  - teste do sistema usando o ponto de vista do usuário, realizado pelo usuário nas instalações do usuário
  - teste realizado antes do sistema ter sido dado por concluído
    - adequação, usabilidade, desempenho, ...
    - identificações de pontos fracos e fortes
      - pode gerar solicitações de melhoria
      - ainda há tempo de resolver diversos pontos fracos
    - interessa ao desenvolvedor, pois usuários tendem a exercitar usos não necessariamente testados nos testes realizados no contexto dos desenvolvedores
    - interessa aos usuários, pois permite integrar com artefatos do usuário antes do sistema estar disponível

Teste ao utilizar o artefato

- Teste da utilidade / adequação
  - o que faz?
  - preciso do que faz?
  - resolve de fato o meu problema?
    - completamente?
- Teste do processo de trabalho do usuário
  - dados podem ser fornecidos na ordem que o usuário deseja e não na ordem que o sistema espera?
  - o usuário pode parar e retomar a qualquer momento?
  - o processo é clemente?
    - perdoa sequências de trabalho não convencionais
    - permite alterar as seleções de ação

Teste ao utilizar o artefato

- Teste da utilizabilidade – (interface humana) verifica se o artefato é fácil de usar e de aprender a utilizar por usuários com o treinamento esperado
  - ponto de vista usuário "normal"
  - ponto de vista usuário com deficiências físicas
- Teste de erros de uso – verifica se o artefato trata adequadamente os erros de uso acidentais e as falhas exógenas usuais
  - mensagens fazem sentido e dão ao usuário indicações de como corrigir ou fazer certo na próxima tentativa?
  - uso incorreto provoca lesões? Dá para testar isso?

Teste ao utilizar o artefato

- Utilizabilidade, alguns critérios:
  - sem entraves
    - bloqueios, cancelamentos
  - clemente
    - perdoa erros de uso
      - permite voltar atrás ou desfazer (*undo*)
      - permite apagar o resultado de uma transação indesejada
    - facilita a alteração dos dados, mesmo se tardia
  - ágil
    - não provoca esperas demasiadas
  - fácil de aprender a usar corretamente
    - fácil encontrar o início das diferentes ações disponíveis
  - uniformidade do "*look and feel*"
  - aspecto bonito
    - o que é "bonito" ?

Tipos de teste – especificação

- Teste da corretude – procura encontrar diferenças entre o especificado e o implementado sem preocupar-se com a adequação
- Teste da interface entre artefatos – verifica se a interface do artefato permite a construção de sistemas que utilizem o artefato sob teste de forma *verbatim*
  - sem requerer quaisquer alterações, adaptações ou interfaces de conversão
    - *wrappers*
    - bacalhaus ☺

verbatim: literalmente, assim como foi escrito [Aurélio]

Tipos de teste – especificação

- Teste da documentação – verifica se a documentação técnica e a fornecida ao usuário, inclusive *help* e tutoriais, são coerentes com o artefato tal como disponibilizado
  - verificar se o documento corresponde exatamente a o que o artefato faz
    - ajustar um dos dois, ou ambos, caso haja discrepâncias
  - procure escrever um esboço da documentação antes de iniciar o desenvolvimento
    - *storyboard*
    - casos de uso com esboços das interfaces
  - sempre que possível fazer a documentação exercitar o artefato
    - a documentação vira uma coletânea de casos de teste

exatamente quer dizer: nem mais, nem menos

Tipos de teste – segurança

- Teste da robustez – verifica se o artefato resiste a agressões intencionais ou fortuitas
  - uso incorreto acidental
  - uso pelo "gato" ou pelo "macaco" ☺
  - "*fail safe*"
  - "*graceful degradation*"
- Teste da segurança – procura encontrar brechas de segurança que permitam pessoas azaradas ou mal intencionadas a causar danos
- Teste de invasão – similar a teste de segurança, mas praticado para invadir mesmo, usando uma postura de *cracker*

Tipos de teste – por em uso

- Teste da instalação – verifica se a distribuição e instalação operam corretamente
- Teste da implantação – verifica se o artefato pode ser colocado em correto funcionamento nas plataformas do usuário
- Teste de durabilidade – verifica se o artefato pode ser utilizado intensivamente por longos períodos (ex. 24 x 7) sem se deteriorar

Tipos de teste – capacidade

- Teste de desempenho – verifica se o desempenho satisfaz os requisitos estabelecidos, ou, caso estes não tenham sido especificados, se o desempenho é razoável considerando o artefato em questão
- Teste de capacidade – verifica se o artefato atende a demanda de determinado recurso, considerando a demanda máxima estimada
- Teste de limite – (exaustão) procura determinar os limites de capacidade a partir dos quais o artefato entra em colapso por excesso de demanda
- Teste de sobrecarga – ("*stress test*") verifica o comportamento quando recursos forem excedidos (ex. falta memória, ultrapassa a demanda limite)
- Teste de escalabilidade – verifica se é possível fazer crescer a capacidade à medida que cresce a demanda

Mais tipos de teste

- Teste de compatibilidade (interoperabilidade) com outros sistemas
- Teste de conversão
- Teste do backup
- Teste da recuperação
- . . .

Bibliografia complementar

- Bastos, A. 2; Rios, E.; Cristalli, R.; Moreira, T.; *Base de Conhecimento em Teste de Software*; São Paulo: Martins; 2007
- Delamaro, M.E.; Maldonado, J.C.; Jino, M.; *Introdução ao Teste de Software*; Rio de Janeiro, RJ: Elsevier / Campus; 2007
- Kaner, C.; Falk, J.; Nguyen, H.Q.; *Testing Computer Software*; International Thompson Computer Press; 1993
- Machado, P.; Vincenzi, A.; Maldonado, J.C.; "Software Testing: An Overview"; in [Borba, P.; Cavalcanti, A.; Sampaio, A.; Woodcock, J.; eds.; *Testing Techniques in Software Engineering*; Berlin: Springer, Lecture Notes in Computer Science; LNCS 6153; 2010], pags 1-17
- Pezzè, M.; Young, M.; Teste e Análise de Software; Porto Alegre, RS: Bookman; 2008
- Staa, A.v.; Programação Modular; Rio de Janeiro: Campus; 2000
- Whittaker, J.A.; "What is software testing? And why is it so hard?"; *IEEE Software* 17(1); Jan 2000; pags 70-79

FIM