


Laboratório de Engenharia de Software

Princípios de Teste 1

Arndt von Staa
Departamento de Informática
PUC-Rio
Março 2017



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

Mar 2017Arndt von Staa © LES/DI/PUC-Rio2

Laboratório de Engenharia de Software

O que é um teste?


- Teste é um **experimento controlado** visando **observar** e **registrar** todas as **falhas** geradas ao **executar** casos de teste
- O objetivo de registrar as falhas é **incentivar** a **remoção completa e correta dos defeitos** causadores
 - é a **remoção** que **conduz à melhoria** da qualidade do artefato



Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
3

Laboratório de Engenharia de Software


Exemplo de caso de teste formulário



IDENTIFICADOR DO CASO DE TESTE: Login001		
DESCRIÇÃO: O usuário realiza um Login bem sucedido		
PRE-CONDICOES (cenário): <ul style="list-style-type: none"> Componente vinculado corretamente ao cadastro de usuários para teste TesteSis.usuários usando a senha "BaseTeste" Janela de login está aberta 	AÇÕES DO USUÁRIO: <ul style="list-style-type: none"> O usuário digita a identificação: joaoSilva O usuário digita a senha: joao#### O usuário digita exatamente os caracteres do captcha O usuário clica "Login" 	POS-CONDICOES: <ul style="list-style-type: none"> A janela de login está fechada O sistema Sis recebeu os dados <autorizado, {a,b,c}> <p style="color: #00a0e3; margin-top: 10px;">Faltou alguma coisa?</p>
Requisitos: <ul style="list-style-type: none"> Segurança: o cadastro deve estar criptografado, teste com "BaseTeste" Segurança: todos os espaços de dados que contenham dados do cadastro e/ou do usuário devem ter sido obliterados antes de retornar ao sistema Sis Segurança: a senha não deve ser exibida no monitor Segurança: os campos a preencher devem ser fornecidos em branco 		

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
4

Alternativa, notação similar a DDTA (ATDD)




Laboratório de Engenharia de Software

- Dado** que o componente `login` está vinculado ao cadastro `TesteSis.usuarios` criptografado com a senha "`BaseTeste`"
- e dado que esse cadastro contém:
 - `<id: joaoSilva, senha: joao####, direitos: {a,b,c}>`
 - `<id: mariaSa, senha: #maria#, direitos: {c,d,e}>`
 - `<id: joseGomes, senha: joao####, direitos: {a,b,e}>`
- quando** o componente `login` for ativado por `Sis`
- o componente** abre a janela `login` contendo um `captcha` aleatório
- quando eu digitar `<id: joaoSilva, senha: joao####, e captcha >`
- e quando eu clicar `login`
- o componente fecha a janela `login`
- o componente retorna a `Sis` `<autorizado, direitos {a, b, c}>`

DDTA – desenvolvimento dirigido por testes de aceitação
 ATDD – *acceptance test driven development* (variante de BDD – behavior driven testing)

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
5

Limitação intrínseca



Laboratório de Engenharia de Software


- Teste somente sinaliza a ocorrência de falhas, **nunca a sua ausência.***

- E.W. Dijkstra

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
6

Laboratório de Engenharia de Software

Limitação intrínseca




- Por necessidade econômica a criação de suítes de testes é restrita a 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, **jámais o saberá**
 - Kaner, C.; Falk, J.; Nguyen, H.Q.
- É possível existir um **teste exaustivo**?
- Um conjunto de exemplos de uso, se nenhum deles encontra uma falha pode-se concluir que o artefato sob teste seja perfeito?
- É possível que as todas as propriedades de um conjunto contavelmente infinito possa ser estabelecido através de poucos **exemplos**?

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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
7

Laboratório de Engenharia de Software

Limitação intrínseca




- **Indecidibilidade**: não pode existir um programa T capaz de determinar: se para todas as possíveis 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 quaisquer E e P são equivalentes ou não.
 - **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 qualquer P é uma implementação exata de uma especificação E .

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
8

Laboratório de Engenharia de Software

Terminologia: níveis de abstração dos testes




- Testes são organizados segundo níveis de abstração
 - teste de unidade: artefatos elementares
 - funções, métodos
 - teste de módulos: unidades de compilação
 - módulos, classes
 - representações executáveis e/ou a partir das quais se gera código
 - modelos executáveis
 - teste de componentes
 - conjuntos de módulos e/ou componentes, bibliotecas, dlls, dos
 - teste de construtos
 - construtos são implementações parciais que podem ser usadas para avaliar a sua adequação e corretude funcional e não funcional
 - podem ser implementações parciais úteis a interessados
 - teste de programas
 - teste de sistemas

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
9

Laboratório de Engenharia de Software


Terminologia: classes 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 (e.g. estruturas de dados)
- Teste funcional
 - teste caixa fechada
 - verifica se são satisfeitos os requisitos funcionais especificados
- Teste de requisitos não funcionais
 - verifica se são satisfeitos os requisitos não funcionais especificados
- Teste de regressão
 - verifica se tudo que não foi alterado continua operando como antes
- Teste de aceitação
 - verifica se o que foi desenvolvido se comporta conforme esperado
 - projetado pelos clientes e/ou usuários (ou seus representantes)
- Teste de liberação (disponibilização)
 - verifica se o que vai ser entregue e instalado nos equipamentos do cliente se comporta conforme esperado

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
10

Terminologia: casos de teste




Laboratório de Engenharia de Software

- Caso de teste:** define um *cenário*, um *conjunto de ações e dados* que visam testar um determinado aspecto do *artefato sob teste* (AST), e um *oráculo* capaz de observar falhas.
- Cenário de teste:** o estado (pré-condição) que o **contexto** deve satisfazer para que o caso de teste possa ser executado
 - ex. banco de dados para teste contém determinados dados e está ativo
- Suíte de teste:** conjunto de todos os casos de teste utilizados para testar o artefato sob teste
- Massa de teste** (script de teste): conjunto de casos de teste a serem executados em uma ativação de teste automatizado
- Roteiro de teste:** conjunto de casos de teste a serem executados manualmente
- Requisitos:** aspectos *não funcionais* que devem ser observados ao realizar os casos de teste

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
11

Como testar?



Laboratório de Engenharia de Software

- 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 as falhas reportadas
 - mostrar que as causas das falhas foram totalmente sanadas
- 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?
 - é possível instrumentar de modo que se possa diagnosticar sem precisar replicar?

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
12

Princípios básicos de um teste



- Deve-se adotar uma postura **destrutiva** ao testar. Por que?
 - o objetivo do teste **não é ver se funciona**
 - o objetivo é **descobrir em que condições não funciona**
 - deseja-se **maximizar o número de defeitos exercitados** de modo que provoquem falhas observáveis pelo teste
 - porém: não se sabe quais defeitos existem...
 - deseja-se reduzir o risco decorrente do **uso** do software
 - evitar a existência de **defeitos remanescentes**

Ações relacionadas com o teste



- **Selecionar** o **método** de teste a ser usado, **antes de iniciar** o desenvolvimento
- **Especificar** a **instrumentação** de apoio ao teste
- **Formular** o conjunto (**suíte**) de casos de teste, idealmente antes de iniciar o desenvolvimento
- **Efetuar** o teste do AST – Artefato **Sob Teste**
- **Observar** as falhas, i.e. **verificar** se o comportamento do AST corresponde ao **especificado** (ou **desejado**)
 - usualmente verifica-se somente o especificado,
 - precisa também verificar se a especificação está errada: requisitos explícitos e implícitos
- **Registrar** as falhas observadas

SÓ?

Ações relacionadas com o teste



- Teste é somente parte do problema, precisa-se ainda:
 - **diagnosticar**: identificar *correta e completamente* o defeito a partir da falha
 - procurar a **causa** da falha
 - pode ser: caso de teste errado, cenário errado, especificação errada, implementação errada, mistura disso, ...
 - **depurar**: (*debugging*) eliminar *completamente* o defeito, sem adicionar novos
 - **retestar**: verificar se a nova versão do artefato satisfaz a sua nova especificação e/ou sua correção
 - **teste de regressão**: verificar se tudo que não foi alterado continua operando corretamente
 - **(re)disponibilizar**: tornar a nova versão disponível para os interessados
 - **(re)instalar**: por a nova versão em operação

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

15

Princípios básicos de um teste



É erro grosseiro pensar que testar é
somente exercitar o artefato
com alguns dados,
sem preocupar-se de antemão com
a cobertura de propriedades e
o resultado a ser obtido

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

16

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 dispor de uma **especificação** "completa e correta"
 - implica saber como identificar e reportar **falhas**
 - implica saber como identificar e reportar **erros de uso**
 - **achismo ao testar** é convite a problemas futuros
 - exemplo de achismo: fornecer alguns dados, executar, observar o resultado, e **achar** que está ou não correto

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

17

Oráculo



- **Oráculo**: dado um **resultado observado**, o oráculo indica se este corresponde a o que foi **especificado** (é **esperado**)
- Mais formal
 - seja p um programa que implementa uma **especificação** f operando sobre um domínio D
 - D pode ser qualquer coisa, desde um dado singelo até uma coleção de muitos dados, gráficos, bases de dados etc.
 - um **oráculo** é um **predicado** O que retorna **true** se e somente se a execução de p , usando um $d \in D$, produz resultados em conformidade com a especificação $f(d)$

Dado um $d \in D$ então $O(p(d), f(d))$
será **true** **sse** $p(d)$ for idêntico a $f(d)$
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)

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

18

Tipos de oráculo




Laboratório de Engenharia de Software

- **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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
19

Tipos de oráculo




Laboratório de Engenharia de Software

- **Inversa:** verifica se, ao aplicar uma função inversa à função sob teste, são reproduzidos os dados de entrada, exemplos
 - $[M] * [M]^{-1} == \text{aprox } [I]$ [I] – matriz identidade
 - `arcsin(sin(alpha)) == aprox(alpha)`
 - controle de recursos usados
 - 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

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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
20

Tipos de oráculo




Laboratório de Engenharia de Software

- **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** corretamente realizado
 - **pós-condições** verificam se o processamento *p* **foi** corretamente realizado
 - **invariantes** devem valer *antes e depois* de executar *p* e *depende de dados alterados* durante a execução de *p*
 - **assertivas pontuais** verificam se a parte a ser manipulada das estruturas de dados satisfaz as *invariantes estruturais* no ponto em que o programa atuará (parâmetros)
 - **assertiva estrutural** verifica se toda a estrutura de dados satisfaz as suas invariantes estruturais

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
21

Tipos de oráculo



Laboratório de Engenharia de Software

- **Caminho** – verifica se o caminho executado é idêntico ao caminho proposto no caso de teste
 - **casos de teste estruturais** selecionam 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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
22

Tipos de oráculo



- **Estatístico** – verifica se a distribuição dos resultados é coerente com a distribuição esperada.
 - ex. 1: são tolerados até 30% de resultados incorretos
 - falsos positivos - informa uma condição ou evento que não ocorreu
 - falsos negativos - não informa uma condição ou evento que ocorreu
 - não produziu resultado, ex. erro de transmissão
 - ex. 2: os resultados devem corresponder a uma distribuição normal com média x e variância y

Tipos de oráculo




- **Contratos temporais**: dado um *log* de eventos em que cada evento registra um *time stamp* e *estados dos dados* na forma $\langle tag, valor \rangle$, verifica se a ordenação temporal e a espera entre eventos contendo determinados *tags* está de acordo com o especificado, exemplos
 - 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 log que são centralizados e ordenados pelo *time stamp*
 - procuram-se trechos do log iniciando e terminando em registros caracterizados por determinados pares $\langle tag, valor \rangle$

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

Laboratório de Engenharia de Software

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, ou BDD
 - o intuitivamente esperado – **isso é péssimo**
 - o comportamento desejado (IHC)

Mar 2017Arndt von Staa © LES/DI/PUC-Rio25

Laboratório de Engenharia de Software

Tipos de oráculo



- **Aprovação:** (usualmente manual) explora o artefato em um contexto de produção (ou parecido com ele) e verifica se o construto sob teste corresponde às **atuais** expectativas **explícitas** e **implícitas** do usuário e do cliente

Mar 2017Arndt von Staa © LES/DI/PUC-Rio26

Oráculo e testes



- Todo caso de teste requer um oráculo
- Dado um conjunto de casos de teste $T_O \subseteq D$ baseado em um **tipo de oráculo** O precisamos ser capazes de verificar se é verdadeiro:

$$\forall t \in T_O : O(p(t), f(t))$$

- Simbologia
 - T_O – suíte de teste T baseada no **tipo** (modo de calcular) do oráculo O
 - $:$ – vale

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

27

Oráculo e testes




- O oráculo **faz parte** da especificação da suíte de teste
- Idealmente o oráculo deve ser definido antes do desenvolvimento do **artefato sob teste**
 - o oráculo determina como **instrumentar** o código para que possa ser avaliado
 - ex. geração de log
 - ex. verificação de assertivas
 - 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 em tempo de uso produtivo
 - ex. assertivas

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio


28

Problemas do oráculo


Laboratório de Engenharia de Software

- O oráculo **pode conter defeitos**
 - quanto mais complexo, maior a probabilidade de o oráculo conter defeitos
 - **falsos positivos**: o oráculo acusa problema que não existe
 - **falsos negativos**: o oráculo deixa de acusar problema que existe
 - implica a necessidade de corrigir
 - a suíte de teste
 - e/ou o oráculo

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
29

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 $f(t)$
 - quanto mais complexo o oráculo, maior será o risco de estar errado
 - 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 para realizar os testes
 - o uso do oráculo pode ser inviável na prática
 - ex. o oráculo pode requerer tempo demais para processar
 - ex. o oráculo pode requerer recursos demais para executar
 - o custo do oráculo ou da instrumentação pode ser muito alto
 - ex. desenvolver o oráculo requer demasiados recursos
 - ex. desenvolver o oráculo requer conhecimentos muito especializados

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
30

Problemas do oráculo



- Mesmo tendo boas especificações, nem sempre é possível determinar se o resultado corresponde ao 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 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

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

31

Aspectos teóricos



- Seja p 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)
$$?(\forall t \in T : \text{oráculo}(p(t), f(t)) == \text{true}) \Rightarrow$$
$$(!\exists d \in D : \text{oráculo}(p(d), f(d)) == \text{false})$$
- Dito de outra maneira
 - Uma suíte de teste é confiável, se **permite assegurar** que não há defeito remanescente caso não sejam encontradas falhas ao testar.
- Confiabilidade **não é mensurável**, pois é impossível saber quais são todos os defeitos remanescentes.

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

32

Aspectos teóricos



- Evidentemente confiabilidade é um **ideal**
 - é um **ideal**, pois implica que o teste deveria ser capaz de observar **todos** os defeitos porventura existentes
 - quase sempre impossível, pois o conjunto de defeitos é inerentemente desconhecido
 - o **real** é sobramente **defeitos remanescentes desconhecidos**
- **Eficácia do teste** – é o percentual dos **defeitos existentes** que foi identificado pelo teste
$$(\text{defeitos_observados} / \text{defeitos_existentes}) * 100$$
 - problema: o conjunto de *defeitos existentes* é **desconhecido**
 - usando **mutantes** pode-se obter uma **aproximação da eficácia**


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

Apêndice



Laboratório de Engenharia de Software

Terminologia: 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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
35

Laboratório de Engenharia de Software

Nível de abstração do teste: unidade




- **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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
36

Laboratório de Engenharia de Software

Nível de abstração do teste: componentes




- **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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
37

Laboratório de Engenharia de Software


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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
38

Teste ao utilizar o artefato




Laboratório de Engenharia de Software

- 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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
39

Teste ao utilizar o artefato




Laboratório de Engenharia de Software

- 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?](#)

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
40

Teste ao utilizar o artefato




Laboratório de Engenharia de Software

- **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” ?

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
41

Tipos de teste – especificação



Laboratório de Engenharia de Software


- **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]

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
42

Laboratório de Engenharia de Software

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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
43

Laboratório de Engenharia de Software

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*

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
44

Laboratório de Engenharia de Software

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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
45

Laboratório de Engenharia de Software


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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
46

Mais tipos de teste



Laboratório de Engenharia de Software

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

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
47

Bibliografia complementar



Laboratório de Engenharia de Software

- 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], pages 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; pages 70-79

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
48

Laboratório de Engenharia de Software

LES

FIM

Mar 2017Arndt von Staa © LES/DI/PUC-Rio49