


Laboratório de Engenharia de Software

Princípios de Teste 2

Arndt von Staa
Departamento de Informática
PUC-Rio
Março 2017




Laboratório de Engenharia de Software

Especificação

- Objetivo da aula
 - apresentar e discutir os processos relacionados com testes
 - **processo**: modo de planejar, organizar e realizar um trabalho
- Justificativa
 - para realizar testes de forma confiável é necessário trabalhar de forma disciplinada
 - além de testar precisa-se depurar e disponibilizar o artefato corrigido
 - a disciplina é estabelecida através de processos definidos
- Texto
 - Pezzè, M.; Young, M.; Teste e Análise de Software; Porto Alegre, RS: Bookman; 2008, capítulo 4

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

Projeto do teste – eficiência e eficácia




Laboratório de Engenharia de Software

- Testes são
 - pouco eficientes
 - a cada execução de um conjunto de casos de teste são encontradas poucas falhas
 - pouco eficazes
 - é de conhecimento geral que sempre sobram defeitos remanescentes
 - isso decorre dos testes terem sido mal feitos?
 - ou é uma propriedade intrínseca?
 - demorados e caros
 - nós observamos que em torno de 35% a 50% do custo de desenvolvimento é gasto com criação de suítes e realização testes
 - custo é alto especialmente no caso de teste manual
 - usando técnicas formais leves e testes automatizados isso cai para entre 10% a 30%

O site: <http://www.softwaretestinghelp.com/software-test-metrics-and-measurements/> apresenta diversas métricas aplicadas a um exemplo específico

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

Projeto do teste – eficiência e eficácia




Laboratório de Engenharia de Software

- Entre as formas de melhorar o desempenho dos testes figuram
 - especificar cedo a **qualidade satisfatória** a alcançar
 - para todos os atributos de qualidade considerados relevantes
 - ouvindo todos os papéis de interessados
 - estabelecer um **processo disciplinado** de controle da qualidade
 - pode ser um processo ágil, por exemplo XP
 - **planejar** a sequência de testes
 - selecionar apropriadas **técnicas de realização** dos testes
 - dar preferência à **automação** da realização dos testes
 - usar ou desenvolver técnicas e ferramentas que permitam **gerar suítes** de teste a partir das especificações
 - . . .

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

Laboratório de Engenharia de Software

Especificação de um teste




- Cada artefato, independente do nível de abstração:
 - **recebe dados**, que devem satisfazer determinadas condições,
 - analogia com matemática: **domínio do artefato** (da função)
 - **produz resultados**, que devem satisfazer determinadas condições em função dos dados
 - o **contradomínio do artefato**
 - de maneira geral, os resultados constituem o objetivo do artefato
 - **satisfaz condições de qualidade**
 - requisitos não funcionais, inversos e de contrato
- Domínio e contradomínio podem conter diversas coisas,
 - por exemplo: dados elementares, estruturas de dados, bancos de dados, interação com o usuário, mensagens, ...

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

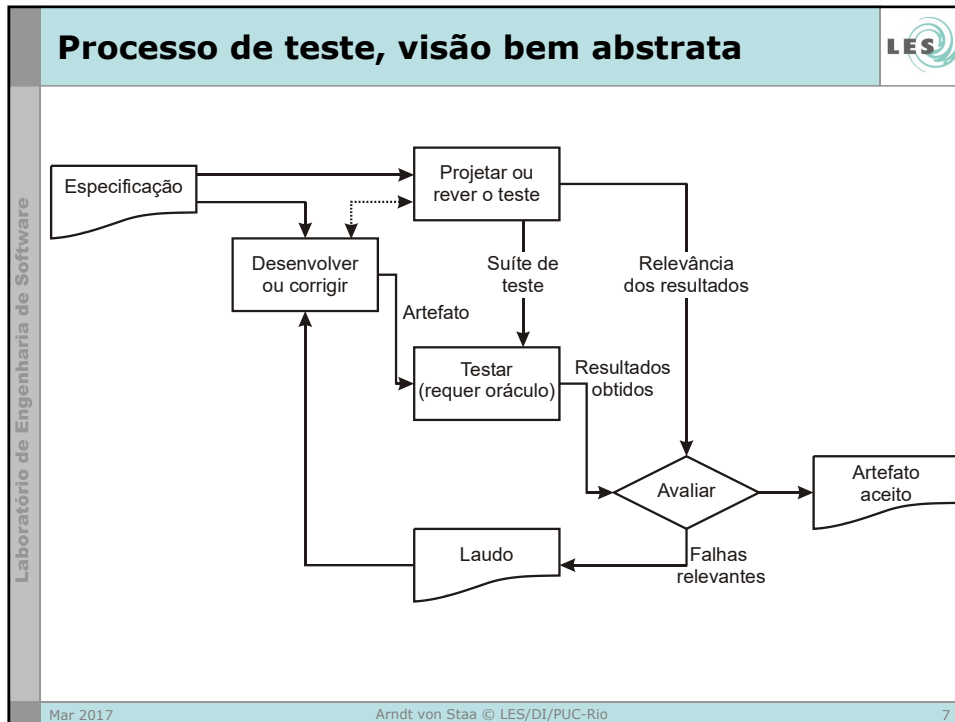
Laboratório de Engenharia de Software

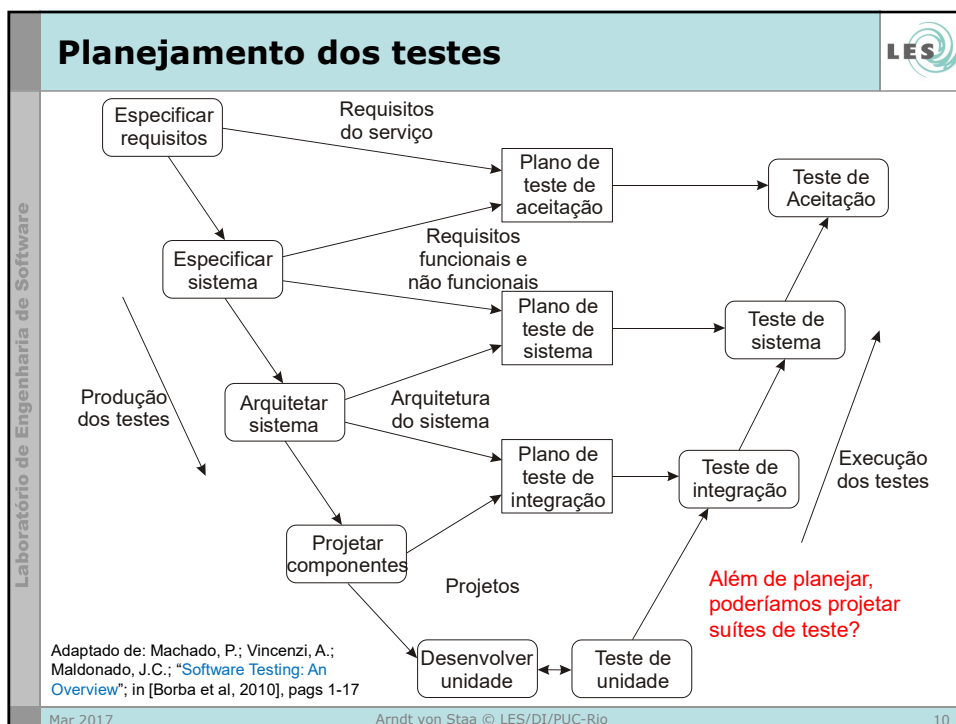
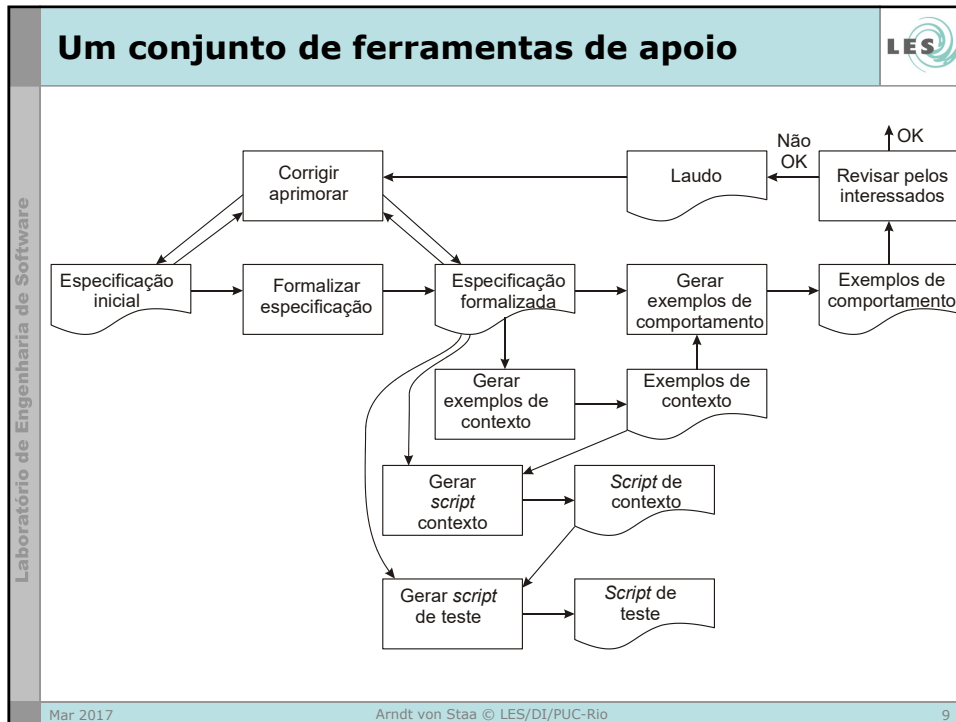
Especificação de um teste

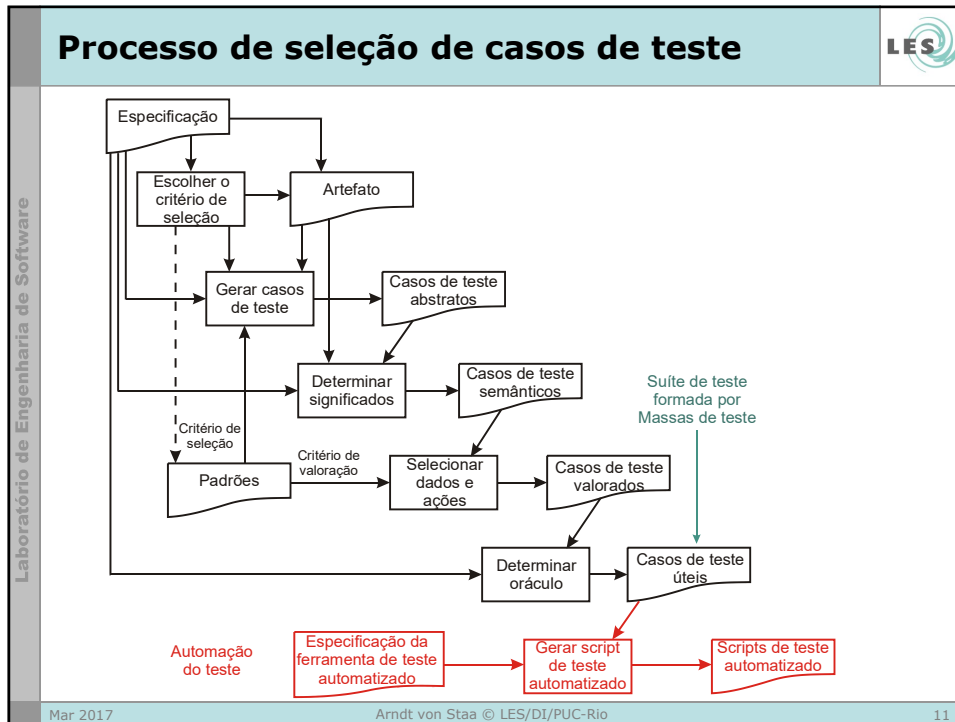


- Uma suíte de teste deve
 - **utilizar contexto e dados que valem** e confirmar se o resultado e a qualidade requerida são satisfeitos
 - **utilizar contexto ou dados que não valem** e verificar se a resposta "faz sentido" propositalmente ambíguo – frequentemente não está especificado
 - obs. projetando bem a interface-com-o-usuário torna impossível a escolha de dados que não valem
 - é quase sempre possível escrever suítes de **teste caixa fechada** antes de dispor do artefato
 - inicialmente realiza-se um "**teste de desbaste**"
 - testa os requisitos funcionais mais relevantes
 - não se preocupa com elevada cobertura
 - podem ser usadas **especificações através de exemplos**
 - após dispor do artefato realiza-se um "**teste completo**"
 - testa todos os requisitos segundo os critérios de teste escolhidos
 - o teste completo passa a ser também o **teste de regressão**

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







Processo de seleção de casos de teste

```

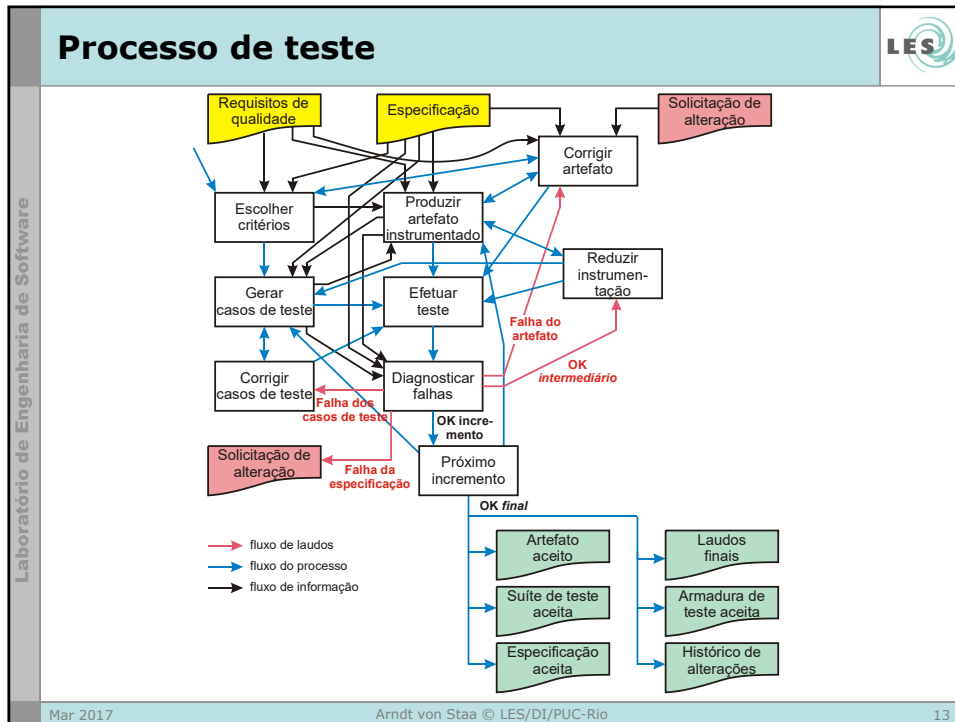
while ( ! feof( pArq ))
{
    ...
    printf( "%d" , strlen( Buffer ) ) ;
    ...
}
  
```

- Casos de teste:

casos de teste úteis

Abstratos	Semânticos	Valorados	Resultado esperado
Executa 0 vezes	Arquivo vazio	{ }	<i>nada</i>
Executa 1 vez	Arquivo com 1 registro	{ aaa }	3
Executa 3 vezes	Arquivo com 3 registros	{ aaa , bb , c }	3 2 1


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



Resultado final

- Artefato e suíte de teste **aceitos**
 - estarão **aceitos segundo o teste**
 - portanto a qualidade assegurada pelo teste depende do **rigor e da abrangência** da suíte de teste
 - o tamanho da suíte não implica rigor nem abrangência
 - rigor e abrangência dependem dos **critérios de seleção de casos de teste** utilizados para criar as suítes de teste
 - rigor depende do grau de formalidade da especificação
 - um dos objetivos da criação de suítes de teste é criar suítes rigorosas e com o menor número de casos de teste capaz de assegurar a qualidade requerida


Mar 2017 Arndt von Staa © LES/DI/PUC-Rio 14

Resultado final


Laboratório de Engenharia de Software

- Qualquer teste percorre um determinado **caminho** no artefato sob teste
 - dado um mesmo conjunto de dados de entrada o caminho percorrido é, em geral, o mesmo
 - não será o mesmo se existirem dependências temporais
 - por exemplo sincronização entre processos
 - por exemplo uso de variáveis não inicializadas

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
15

Resultado final


Laboratório de Engenharia de Software


- Limitação da suíte de teste
 - dados que correspondem a um mesmo **caso de teste abstrato** **tendem a percorrer os mesmos caminhos**
 - portanto a suíte pode nunca percorrer o AST de forma que exercite determinados caminhos, mesmo que seja repetido o teste
 - caso esses caminhos contenham defeitos, estes permanecerão desconhecidos, independentemente do número de vezes que o artefato seja retestado

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
16

Laboratório de Engenharia de Software

Fatores de qualidade relevantes

- **detectabilidade**: facilidade de observar que ocorreu um erro reportando a correspondente falha
 - erros ocorrem
 - **endógeno** – por causas internas
 - **exógeno** – por causas externas. ex. erros de uso, outros sistemas
 - requer **redundâncias** contidas no sistema (código), exemplos
 - **instrumentação**, exemplos
 - **assertivas**
 - oráculos automatizados
 - geradores e verificadores de **logs**
 - **comparar n réplicas (resultados)** e determinar o possível responsável pelas discrepâncias observadas:
 - uso de diferentes especificações para calcular o resultado
 - implementar n versões de uma mesma especificação
 - distribuir v ≥ 1 versões sobre n ≥ 3 máquinas independentes
 - ...




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

Laboratório de Engenharia de Software

Fatores de qualidade relevantes

- **diagnosticabilidade**: facilidade de determinar qual é o defeito a partir do relatório de ocorrência de uma falha
 - apoio à **localização do defeito** a partir do relatório da falha
 - relatório da falha acompanhada de **informação complementar** (ex. **logs**) capaz de descrever o estado da execução no **ponto em que foi observado o erro**, exemplos:
 - pilha de execução
 - estado das variáveis relevantes
 - ações e dados fornecidos pelo usuário
 - estado dos recursos funcionais, ex. sockets, semáforos, ...
 - estado dos sensores e atuadores
 - pequena **latência do erro** implica que o ponto de observação **deve estar próximo do defeito causador**
 - isso implica que a detecção deve ser realizada por instrumentação
 - o ser humano não tem condições para sinalizar falhas com a granularidade e rapidez necessária



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

Fatores de qualidade relevantes



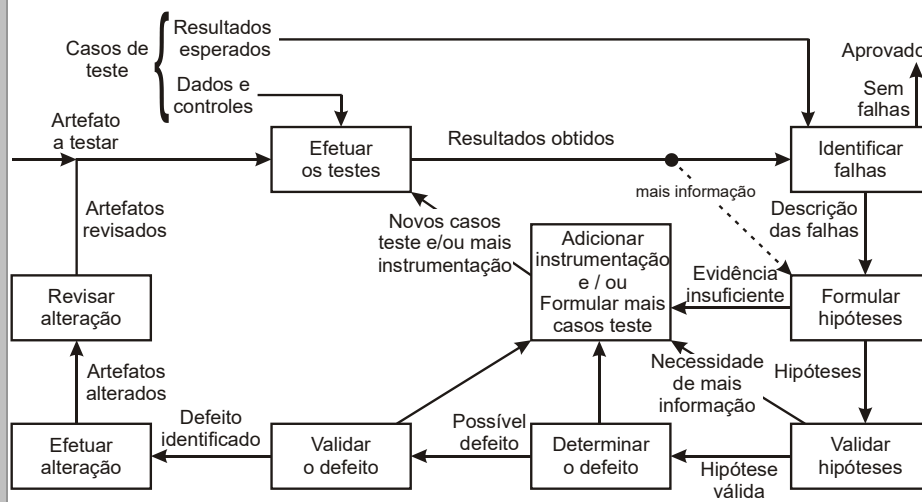
- **depurabilidade**: facilidade de eliminar completa e corretamente o defeito
 - artefato com boa qualidade de engenharia
 - boa arquitetura e boa modularidade
 - especificação, arquitetura e projeto – correspondem exatamente a o que está implementado
 - documentação técnica suficiente para o mantenedor localizar e entender como alterar corretamente o artefato
 - ambiente utilizado para desenvolver existe
 - sub-sistema de apoio à manutenção existe
 - scripts de teste automatizado
 - roteiros de teste
 - scripts de recompilação e integração
 - documentação técnica útil

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

21

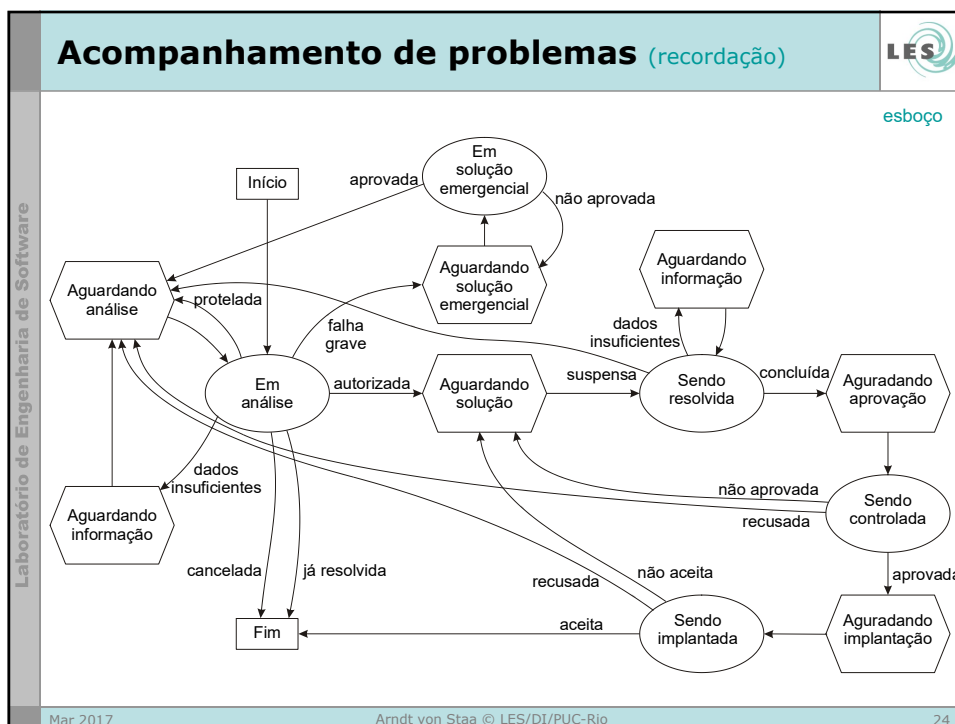
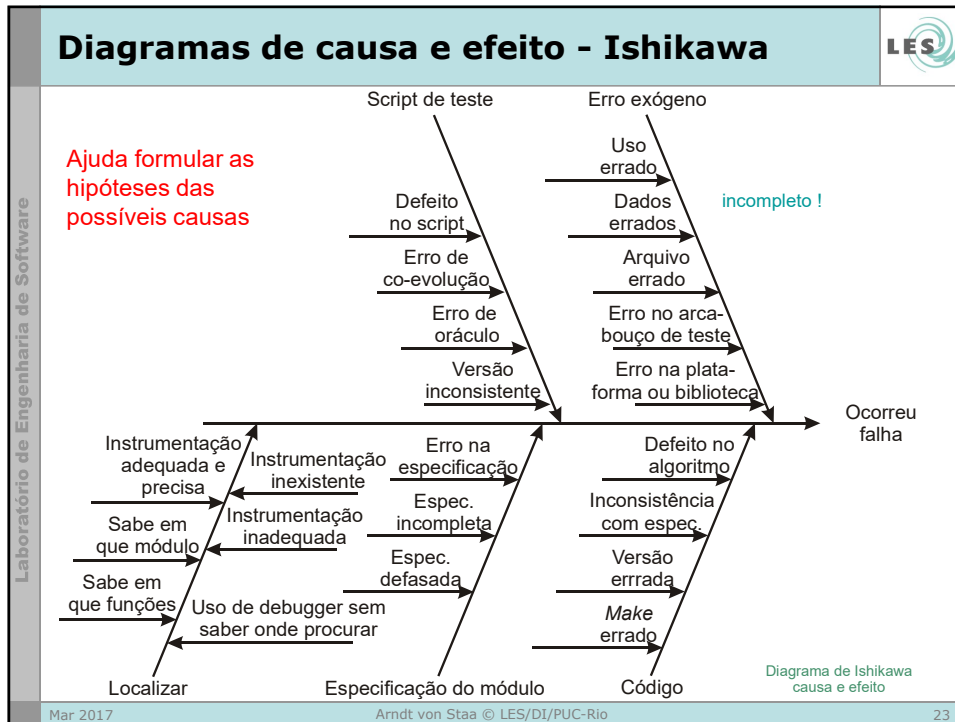
Processo de depuração



Mar 2017


Arndt von Staa © LES/DI/PUC-Rio

22



Laboratório de Engenharia de Software

Relato de falhas




- O melhor testador
 - não é aquele que encontra o maior número de falhas
 - é aquele que consegue **promover a eliminação do maior número dos defeitos** correspondentes às falhas por ele encontradas
- O relato de falha (laudo, FAP) é o instrumento que se utiliza para **vender a falha** ao programador responsável por eliminar o defeito causador
 - um exemplo de modelo para relatar falhas:
<https://bugs.opera.com/wizard/>

Kaner, C.; *Bug Advocacy: How to Win Friends and Stomp BUGs*; 2000; Buscado em: 2008; URL:
<http://www.kaner.com/pdfs/bugadvoc.pdf>

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
25

Laboratório de Engenharia de Software

Manutenção




- Manutenção ocorre **durante o desenvolvimento**
 - atender a **solicitações de mudança** de requisitos
 - neste caso o “defeito” é a mudança solicitada
 - o desenvolvimento de software é um **processo de aprendizado**
 - aprende-se sobre o problema a resolver
 - aprende-se sobre a tecnologia utilizada para desenvolver
- Manutenção ocorre **após entrega** óbvio
 - criar, alterar, remover e co-evoluir artefatos do sistema
 - cerca de 70% do custo técnico corresponde a manutenção após a entrega
 - custo técnico: desenvolvimento, manutenção, evolução, dano provocado por falhas

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
26

Laboratório de Engenharia de Software

Manutenção




- Tipos de manutenção após entrega
 - Manutenção **adaptativa** [Nosek & Palvia, 1990]
 - modifica o artefato sem afetar a sua funcionalidade
 - velho 20% novo 18%
 - Manutenção **perfectiva** e **evolução**
 - modificam a funcionalidade do artefato
 - velho 55% novo 65%
 - Manutenção **corretiva**
 - remove defeitos do artefato
 - velho 25% novo 17%
 - Manutenção **preventiva**
 - melhora a engenharia do artefato

- Schach, S.; Jin, B.; Yu, L.; Heller, G.Z.; "Determining the Distribution of Maintenance Categories: Survey versus Measurement"; *Empirical Software Engineering* 8; Kluwer; 2003; pp 351–365
- Nosek, J.T.; Palvia, P.; "Software Maintenance Management: changes in the last decade"; *Software Maintenance: Research and Practice* 2(3); 1990; pp 157-174

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
27

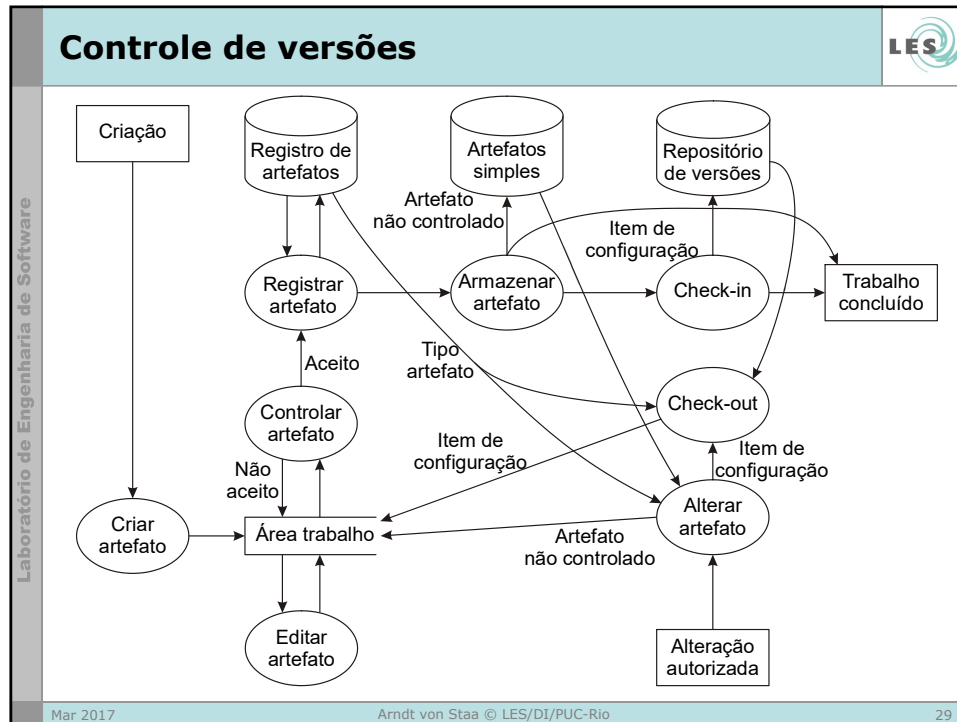
Laboratório de Engenharia de Software

Problema da manutenção



- Um **sistema é um artefato composto** por vários outros, alguns exemplos de artefatos componentes:
 - especificação (em vários níveis de abstração)
 - código
 - scripts de teste
 - documentação técnica
- Quando um artefato é modificado pode tornar-se necessário modificar vários outros para assegurar coerência do todo
 - **co-evolução**
- Quanto menos trabalhoso e menos propenso a enganos for a co-evolução mais **manutenível** é o sistema

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
28




Apêndice

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

Laboratório de Engenharia de Software

Ferramentas recomendadas



Huizinga, D.; Kolawa, A.; *Automated Defect Prevention: Best Practices in Software Management*; Hoboken, New Jersey: John Wiley & Sons; 2007


Nível mínimo:

- **Ambiente de desenvolvimento integrado (IDE – Integrated Development Environment)**
 - Eclipse, Visual Studio, ...
- **Sistema de controle de versões**
 - SVS, Subversion, GIT, ...
- **Sistema automatizado para a reconstrução de programas**
 - make, gmake, ant, maven, hudson ...
- **Sistema de acompanhamento de problemas**
 - Bugzilla, Jira, ...
- **Repositório compartilhado de artefatos aceitos**
- **Repositório de padrões, diretrizes, e convenções**
- **Repositório de documentação técnica (JavaDoc ou similar)**

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
31

Laboratório de Engenharia de Software

Ferramentas recomendadas




Nível intermediário

- Todas as do nível mínimo
- **Sistema de gerência de requisitos**
 - acompanha a alteração dos requisitos
 - permite determinar o impacto de alterações
 - rastreia os requisitos nos artefatos desenvolvidos
- **Sistema de teste automatizado básico**
 - realiza o teste de unidades
 - funções, classes, módulos
 - realiza o teste de regressão das unidades

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
32

Laboratório de Engenharia de Software

Ferramentas recomendadas



Nível avançado


- Todos do nível intermediário
- **Transformadores**
 - geram esqueletos a partir de diagramas
 - geram esqueletos de módulos de teste
 - geram outros artefatos intermediários
- **Geradores**
 - geram artefatos prontos para serem utilizados (compilados)
 - geram módulos de teste
 - geram casos de teste úteis *idealmente scripts de teste*
- **Analisadores estáticos**
 - verificam propriedades de artefatos segundo regras definidas
- **Teste automatizado avançado**
 - junit, dbUnit, jBehave, selenium, concordion, ...

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
33

Laboratório de Engenharia de Software


Relato de falhas


FAP – Ficha de acompanhamento de problemas





- Conteúdo do **relato de falha (laudo)** [Kaner, 1993]:
 - data
 - quem relatou
 - em que programa ocorreu?
 - versão
 - se tiver, o **número do build**
 - resumo da falha
 - um parágrafo (título) de no máximo 2 linhas
 - tipo da falha (ver a seguir)
 - severidade da falha (ver a seguir)
 - descrição
 - o que ocorreu e o que deveria ter acontecido?
 - o que você estava fazendo quando ocorreu a falha?
 - você pode fornecer os dados que estava usando?
 - cenário de uso ao observar a falha
 - como replicar a falha?
 - opcionalmente, sugestão de solução
 - anexos

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
34


Laboratório de Engenharia de Software	Relato de falhas FAP – Ficha de acompanhamento de problemas	
	<p>Tipo da falha</p> <ul style="list-style-type: none"> • erro de processamento • erro de projeto / especificação • erro de documentação • erro de biblioteca • erro de plataforma de software • erro de hardware • dúvida • . . . 	
Mar 2017		Arndt von Staa © LES/DI/PUC-Rio

Laboratório de Engenharia de Software	Relato de falhas FAP – Ficha de acompanhamento de problemas	
	<p>Severidade</p> <ul style="list-style-type: none"> • problema menor <ul style="list-style-type: none"> – dá para ser contornado – é uma inconveniência • grave <ul style="list-style-type: none"> – não dá para continuar usando o artefato • infeccioso <ul style="list-style-type: none"> – propaga erros para outros programas ou sistemas • catástrofe <ul style="list-style-type: none"> – destrói dados persistentes – provoca grandes danos – provoca quebra de equipamento – provoca mortes, ruína de empresas, desastres ecológicos 	
Mar 2017		Arndt von Staa © LES/DI/PUC-Rio

Laboratório de Engenharia de Software	Relato de falhas – solução dada FAP – Ficha de acompanhamento de problemas	
	<ul style="list-style-type: none"> • Conteúdo do relato da solução dada <ul style="list-style-type: none"> – Estados e datas: aberto, em resolução, em implantação, resolvido – Responsável por realizar (gerenciar) a alteração – Natureza da decisão <ul style="list-style-type: none"> • pendente de decisão, não reproduzível, resolvido, concluído, não é falha, cancelado pelo informante, requer mais informação, duplicata, cancelada, postergada, ... – Descrição da solução – Artefatos alterados e versão da alteração <ul style="list-style-type: none"> • natureza da alteração, por artefato <ul style="list-style-type: none"> – código simples, código complexo, falta de controle, reestruturação, projeto, arquitetura, especificação – Responsável por aceitar a solução 	
	Mar 2017	Arndt von Staa © LES/DI/PUC-Rio

Laboratório de Engenharia de Software	Projeto do teste – eficiência e eficácia	
	<ul style="list-style-type: none"> • A eficácia do teste depende do critério de seleção dos casos de teste <ul style="list-style-type: none"> – o critério de teste está relacionado com cobertura de alguma propriedade, ex. <ul style="list-style-type: none"> • linhas de código • chamadas e retornos, inclusive exceções • assertivas, ... – à medida que os artefatos sobem de nível de abstração, necessariamente menor será a granularidade possível de testar, ex. <ul style="list-style-type: none"> • linhas de código – pequena granularidade • chamadas e retornos – média granularidade • funcionalidades do usuário – grande granularidade 	
	Mar 2017	Arndt von Staa © LES/DI/PUC-Rio

Projeto do teste – eficiência e eficácia




Laboratório de Engenharia de Software

- Um **critério de seleção** é um método usado para a escolher os casos de teste que comporão uma suíte de teste
 - tende a gerar uma suíte de teste capaz de identificar falhas causadas por uma **determinada classe** de defeitos
 - existe uma grande variedade de critérios de seleção de casos de teste

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

Bibliografia complementar



Laboratório de Engenharia de Software

- Huizinga, D.; Kolawa, A.; *Automated Defect Prevention: Best Practices in Software Management*; Hoboken, New Jersey: John Wiley & Sons; 2007
- Kaner, C.; Falk, J.; Nguyen, H.Q.; *Testing Computer Software*; International Thompson Computer Press; 1993
- Kaner, C.; *Bug Advocacy: How to Win Friends and Stomp BUGs*; 2000; www.kaner.com (testing website) www.badsoftware.com (legal website)
- Kemerer, C.F.; Slaughter, S.; "An Empirical Approach to Studying Software Evolution"; *IEEE Transactions on Software Engineering* 28(4); 1999; pages 493-509
- Lewis, W.E.; *Software Testing and Continuous Quality Improvement*; Boca Raton: Auerbach; 2000
- Skwire, D.; Cline, R.; Skwire, N.; *First Fault Software Problem Solving: A Guide for Engineers, Managers and Users*; Kindle edition; Ireland: Opentask; 2009
- Wong, W.E.; Gao, R.; Li, Y.; Abreu, R.; Wotawa, F.; "A Survey on Software Fault Localization"; *IEEE Transactions on Software Engineering* 42(8); Los Alamitos, CA: IEEE Computer Society; 2016; pages 707-740

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

Laboratório de Engenharia de Software

LES

FIM

Mar 2017Arndt von Staa © LES/DI/PUC-Rio41