


Laboratório de Engenharia de Software

# Qualidade de Software

## Conceitos

Arndt von Staa  
Departamento de Informática  
PUC-Rio  
Fevereiro 2015



Laboratório de Engenharia de Software

## Especificação

- Objetivo da aula
  - A partir do ponto de vista do controle da qualidade, discutir as preocupações mais relevantes com relação à qualidade do software
    - enfatizar a necessidade da prevenção de defeitos já ao desenvolver
    - enfatizar que qualidade está fortemente relacionada com o serviço prestado ao usuário
      - o que o usuário explicitamente deseja → o que está especificado
      - o que o usuário implicitamente deseja → o que não foi especificado, mas é entendido como uma característica importante

Leitura complementar: Pezzè – capítulos 1 e 2

Fev 2015Arndt von Staa © LES/DI/PUC-Rio2

## O ponto de vista do usuário



- Um sistema correto é:
  - um que resolve o meu problema,
  - um que funciona sempre que dele preciso,
  - um que entendo como usar,
  - um de que eu possa justificavelmente depender.

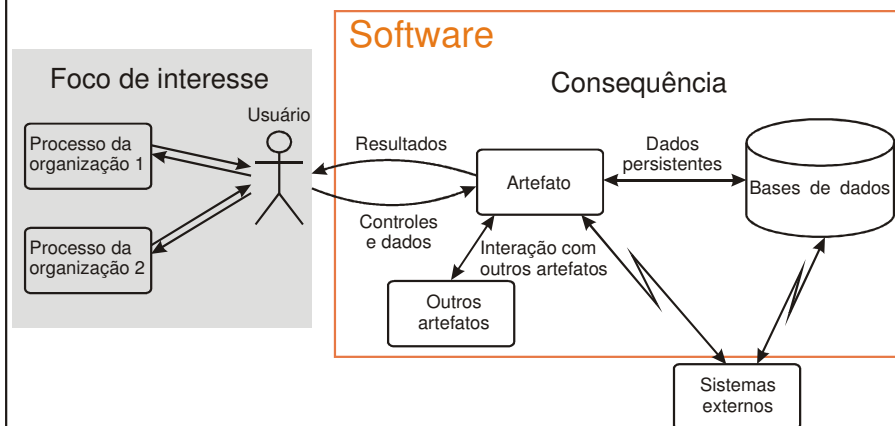
em 4 itens...

Mike Holcombe and Florentin Ipate  
Correct Systems - building a business process solution

## O ponto de vista do usuário




### Sistema intensivo em software



Laboratório de Engenharia de Software

## Não existe software perfeito



- Cerca de 40 a 50% dos programas **postos em uso** contêm defeitos não triviais
  - **defeito não trivial:**
    - demora para ser diagnosticado e removido
    - e/ou produz danos elevados
- Como reduzir este percentual?
  - procurar chegar perto do ideal **corretude por construção**
  - **realizar continuamente** controle da qualidade: ao especificar, arquitetar, projetar, codificar: **corretude por desenvolvimento**
- Como reduzir o tempo médio para recuperar?
  - projeto e programação orientada à **recuperabilidade**
- Como reduzir o tempo médio para corrigir?
  - projeto e programação orientada à **corrigibilidade**
- Como ...


} manuteni-  
bilidade

Boehm, B.W.; Basili, V.R.; "Software Defect Reduction Top 10 List"; IEEE Computer 34(1); Los Alamitos, CA: IEEE Computer Society; 2001; pages 135-137

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
5

Laboratório de Engenharia de Software

## Não existe software perfeito




- Custos estimados devidos a **testes inadequados**
  - estatísticas EUA (**2000**)
    - mercado total aproximadamente de US\$180 bilhões
    - mão de obra: cerca de 697.000 engenheiros de software mais cerca de 585.000 programadores
  - o **custo anual decorrente de uma infra estrutura inadequada para o teste** é estimada estar entre US\$ 22,2 (**12%**) e US\$ 59,5 (**33%**) bilhões
    - estatística baseada em *surveys* (pesquisas de opinião) envolvendo desenvolvedores e usuários
  - não há razão alguma para acreditar que a situação brasileira seja diferente, guardadas as proporções

NIST; *The Economic Impacts of Inadequate Infrastructure for Software Testing*; Planning Report 02-3; National Institute of Standards & Technology Program Office; Strategic Planning and Economic Analysis Group; May 2002

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
6

## Não existe software perfeito




Laboratório de Engenharia de Software

- **Mesmo sistemas perfeitos podem falhar**
  - erros provocados por causas **exógenas**
    - erros provocados por mau uso, deliberado ou não
    - erros de uso induzidos por interfaces ruins
    - erros provocados por falhas de hardware
    - erros provocados por falhas da plataforma de software usada
- Perfeição não existe, densidade de defeitos [Bao, 2007]:
  - INTEL: no more than 80-90 defects in Pentium (em 2012: +-  $3 \cdot 10^9$  transistores no chip; existem GPU's com +-  $7 \cdot 10^9$  transistores [Wikipedia])
  - Standard Software: 25 defects / 1,000 lines of delivered code (kLOC)
  - Good Software: 2 defects / 1,000 lines
  - Space Shuttle Software: < 1 defect / 10,000 lines
  - Cellular Phone: 3 defects / 1,000 lines

Xinlong Bao; *Software Engineering Failures: A Survey*; School of EECS, Oregon State University, Corvallis, OR, U.S.A; apud Huckle, T.; Collection of Software Bugs; <http://www5.in.tum.de/~huckle/bugse.html>; last update October 5, 2007.

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
7

## Não existe software perfeito




Laboratório de Engenharia de Software

- **Não se pode esperar que sistemas não contenham defeitos**
  - caso um sistema não contenha defeitos, não o saberemos
    - algumas vezes podemos saber se módulos contêm defeitos ou não
  - implica a necessidade de medir o desempenho e avaliar a corretude **durante a execução** do sistema
    - torna necessária a instrumentação do código

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
8

Laboratório de Engenharia de Software

## Não existe software perfeito



- **Adicionar requisitos não funcionais a posteriori é usualmente muito caro**
  - as características necessárias precisam ser especificadas, arquitetadas, projetadas etc. junto com os requisitos funcionais
    - é necessário identificar a priori que características são necessárias
    - **isso é sempre possível?**
  - para atingir bons níveis de qualidade deve-se
    - prevenir a **injeção de defeitos**
    - controlar as potenciais falhas provocadas por causas exógenas
      - usuário
      - plataforma
      - bibliotecas


**injeção de defeito** – acidentalmente inserir um defeito, ou omitir algo relevante, na especificação, na arquitetura, no projeto, ou no código

Bass, L.; Clements, P.; Kazman, R. (2012); *Software Architecture in Practice* (3rd Edition); Addison-Wesley, Kindle Edition.

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
9

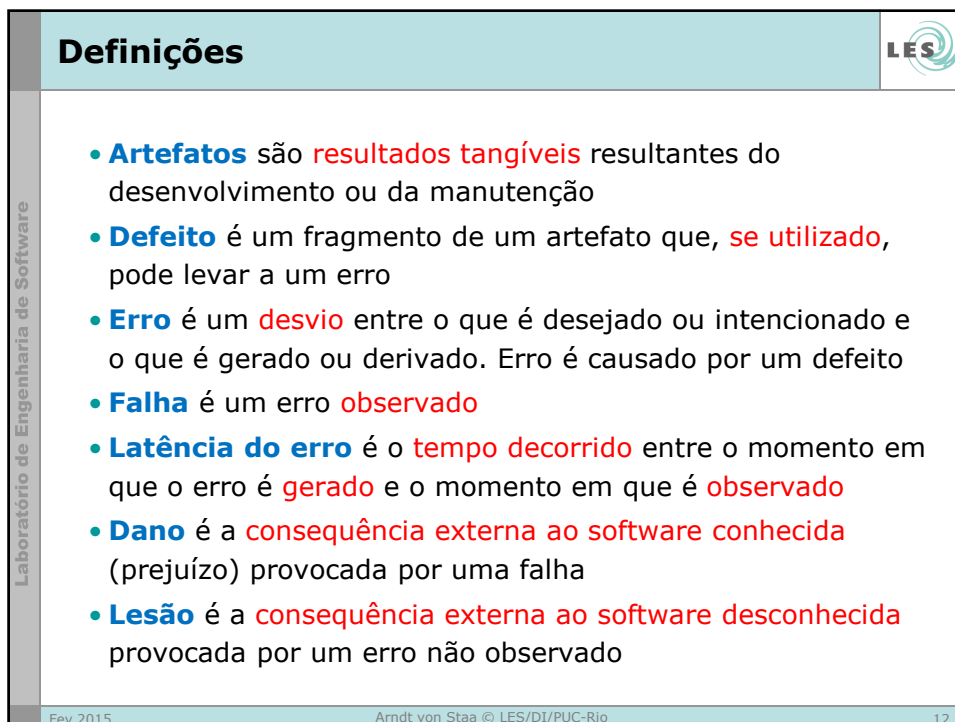
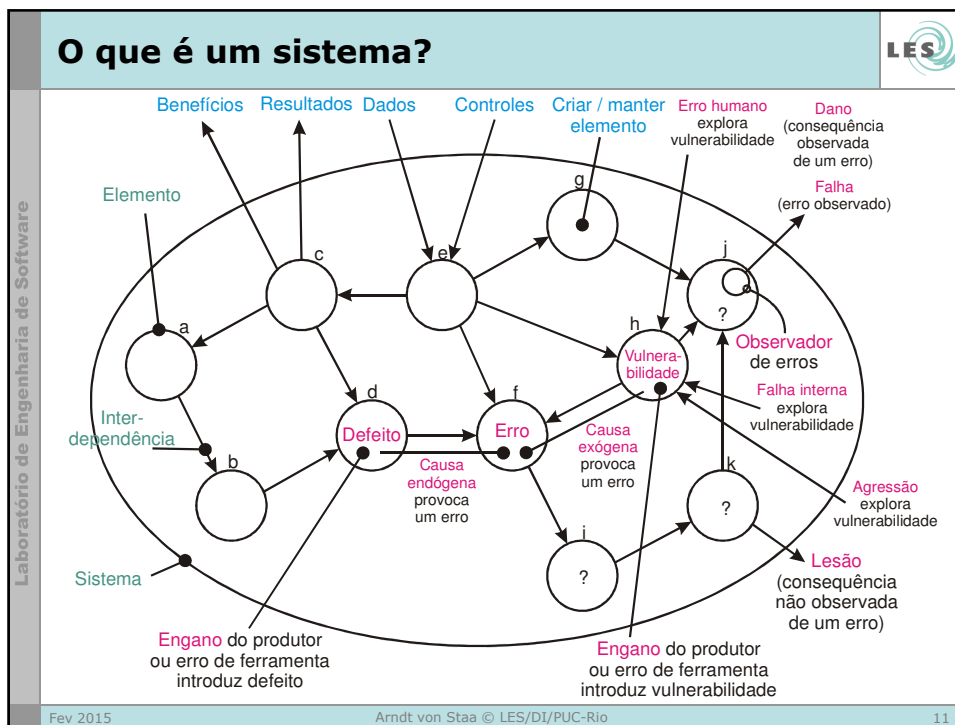
Laboratório de Engenharia de Software


## Dizer romano há bem mais de 2000 anos



- **Errare humanum est**
  - **errar é inerente ao ser humano**
  - **Corolário 1:** como humanos são falíveis e o desenvolvimento de software é intensivo em esforço humano, é utópico esperar que software não contenha defeitos.
- **Sed in errore perseverare dementis**
  - **mas perseverar no erro é próprio do louco**
  - **Corolário 2:** Devemos ser modestos, pois é sinal de incompetência não aceitar que erramos, não querer observar que erramos, não procurar formas de corrigir os erros, não querer aprender com os erros nossos e de outros

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
10




**Definições**


Laboratório de Engenharia de Software

- **Vulnerabilidade** é um fragmento de um artefato que, quando elaborado ou usado em condições peculiares, pode gerar um erro
  - ex. 1. proteção mal feita permite acesso a pessoa não autorizada
  - ex. 2. interface do usuário mal organizada induz erros humanos
  - ex. 3. dados confidenciais armazenados de forma legível por terceiros permitem não autorizados utilizar dados críticos
- na realidade uma vulnerabilidade é um defeito
  - precisa-se procurar explicitamente por elas

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
13

**Definições**



Laboratório de Engenharia de Software

- **Inadequação** ex.
  - não atende às necessidades de usuários ou de outros artefatos
  - não satisfaz requisitos não funcionais necessários
- **Deficiência** ex.
  - induzir usuário a cometer erros de uso
- **Anomalia** (*bad smell*) ex.
  - o artefato está correto, do ponto de vista funcional e não funcional, mas é difícil de manter

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
14

Laboratório de Engenharia de Software

## Definições




- **MTBF** – *mean time between failures*
  - tempo médio entre falhas sucessivas
- **MTTRc** – *mean time to recover*
  - tempo médio requerido para repor o sistema em funcionamento correto (recuperar) após uma falha que cancele a execução
- **MTTRp** – *mean time to repair*
  - tempo médio requerido para corrigir o defeito causador da falha e por a nova versão do sistema em uso após o registro da correspondente falha
- **MTTEv** – *mean time to evolve*
  - tempo médio requerido para evoluir ou adaptar o software após o registro de uma solicitação de alteração

Na realidade o que interessa mesmo é a distribuição dos tempos, uma vez que tempos médios tendem a esconder potenciais problemas sérios -> mínimo , máximo , percentis (25, 50, 75 e 100%)

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
15

Laboratório de Engenharia de Software

## Propagação de defeitos



- Defeitos podem ocorrer na especificação, na arquitetura, nos projetos, no código, nas suítes de teste, ...
- Ex. um defeito em um **artefato antecedente** leva a erro(s) no(s) **artefato(s) consequente(s)**, o que, em última análise, são defeito(s) nesse(s) artefatos
  - exemplo de defeits na especificação:
    - esquecer requisitos relevantes
      - ex. esquecer segurança
  - exemplos de defeito propagado para a arquitetura
    - software não prevê proteção dos dados do usuário contra uso indevido

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
16



Laboratório de Engenharia de Software

## O que queremos?

Ter a **certeza** de que estamos desenvolvendo **economicamente** software possuindo **qualidade de serviço assegurada** e **capaz de operar em ambientes reais**

O que é **Engenharia de Software**?

- qualidade de serviço - qualidade **observada** pelos usuários
- qualidade de engenharia - qualidade **requerida** para assegurar a qualidade de serviço

LES

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
17

Laboratório de Engenharia de Software

## Definição de qualidade de artefato

A **qualidade de um artefato** é um conjunto de **propriedades** a serem satisfeitas em **determinado grau**, de modo que o artefato **satisfaça** as **necessidades explícitas e implícitas** de seus **usuários** e demais **interessados**

**Interessado** é qualquer pessoa, organização ou componente que pode ser afetada pelo artefato (*stakeholder*)


**Problema:** necessidades **implícitas**, como saber quais são elas?

LES

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
18

Laboratório de Engenharia de Software

## Fidedignidade de software



Um sistema **intensivo em software** é **fidedigno** caso atenda **satisfatoriamente** um **conjunto de propriedades** de modo que se possa **justificavelmente depender** dele, **assumindo riscos compatíveis** com o **serviço** por ele prestado.

Qual é a diferença entre qualidade e fidedignidade?

O que é qualidade satisfatória?

O que é risco?

- **fidedigno** (Adjetivo) 1. Digno de fé; **merecedor de crédito** (Aurélio)


Avizienis, A.; Laprie, J.-C.; Randell, B.; "Dependability and Its Threats: A Taxonomy"; in: Jacquart, R., eds.; *Proceedings of the IFIP 18th World Computer Congress: Building the Information Society*; Dordrecht: Kluwer; 2004; pages 91-120

Weinstock, C.B.; Goodenough, J.B.; Hudak, J.J.; *Dependability Cases*; CMU/SEI -2004-TN-016, Software Engineering Institute, Carnegie Mellon University; 2004

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
19

Laboratório de Engenharia de Software

## Fatores da fidedignidade, básicas



<b>Adequação:</b>	prestar o serviço que <b>interessa ao usuário</b> <b>usuário</b> pode ser: pessoa; outro artefato; outro sistema; sensores e/ou atuadores; mantenedores; operadores; programadores clientes; ...
<b>Confiabilidade:</b>	habilidade de, sempre que solicitado, prestar <b>serviço fidedigno</b>
<b>Disponibilidade:</b>	estar pronto para prestar serviço fidedigno <b>sempre que necessitado</b>
<b>Utilizabilidade:</b>	habilidade de <b>interagir com o usuário</b> sem induzi-lo a erro, nem permitir que erros de uso (observáveis) fiquem despercebidos; dito de outra forma: habilidade do software poder ser entendido, aprendido, corretamente utilizado e ser atraente ao usuário
<b>Clemência:</b>	habilidade do software <b>perdoar erros de uso</b>

As características são adaptadas de (Avizienis, 2004) e de outros autores, são mais abrangentes do que as do autor citado

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
20

Fatores da fidedignidade, básicas		LES
Laboratório de Engenharia de Software	<b>Interoperabilidade:</b>	habilidade do software poder ser corretamente <b>conectado</b> com outros sistemas
	<b>Escalabilidade:</b>	habilidade da <b>capacidade de processamento</b> do software <b>crescer junto</b> com o crescimento da demanda
	<b>Durabilidade:</b>	habilidade do software operar fidedignamente por <b>períodos de duração indefinida</b> <ul style="list-style-type: none"> <li>– longa duração, e.g. 24/7</li> </ul>
	<b>Economia:</b>	habilidade de produzir resultados necessitando de <b>poucos recursos</b> <ul style="list-style-type: none"> <li>– ex. computacionais, humanos, financeiros</li> </ul>
	<b>Desempenho:</b>	habilidade de <b>atender à demanda</b> consumindo recursos (ex. tempo, memória) dentro do limite estipulado
Fev 2015		Arndt von Staa © LES/DI/PUC-Rio

21

Fatores da fidedignidade, segurança		LES
Laboratório de Engenharia de Software	<b>Segurança:</b>	habilidade de <b>evitar consequências catastróficas</b> aos usuários, à organização, ou ao ambiente <ul style="list-style-type: none"> <li>– <i>safety</i>, risco baixo</li> </ul>
	<b>Proteção:</b>	habilidade de <b>evitar o sucesso</b> de tentativas de agressão
	<b>Privacidade:</b>	habilidade de <b>proteger dados e código contra acesso</b> (uso) indevido accidental ou deliberado
	<b>Integridade:</b>	habilidade de <b>evitar a corrupção</b> (adulteração) intencional ou accidental de elementos
Fev 2015		Arndt von Staa © LES/DI/PUC-Rio


22

Fatores da fidedignidade, tolerância		LES
Laboratório de Engenharia de Software	<b>Robustez:</b>	<p>habilidade de, <b>em tempo de execução</b>, <b>detectar erros</b> (<b>reportar falhas</b>) de modo que os possíveis danos possam ser mantidos em um patamar aceitável</p> <ul style="list-style-type: none"> <li>– um software robusto não provoca lesões <ul style="list-style-type: none"> <li>• <b>lesão</b>: “prejuízo” desconhecido causado por erro não observado</li> </ul> </li> <li>– pode gerar danos, desde que controlados <ul style="list-style-type: none"> <li>• <b>dano</b>: “prejuízo” causado por erro observado</li> </ul> </li> </ul>
	<b>Recuperabilidade:</b>	<p>habilidade de ser rapidamente <b>reposto em operação</b> fidedigna, preventivamente ou após a ocorrência de uma falha</p>
	<b>Corrigibilidade:</b>	<p>habilidade de ser <b>fácil e rapidamente corrigido</b> quando da ocorrência de uma falha</p>
	<b>Resiliência:</b>	<p>habilidade de <b>amoldar-se a condições anormais de funcionamento</b> sem comprometer a fidedignidade</p>
	<div> Fev 2015 Arndt von Staa © LES/DI/PUC-Rio 23 </div>	

Fatores da fidedignidade, evolução		LES
Laboratório de Engenharia de Software	<b>Manutenibilidade:</b>	<p>habilidade de poder ser <b>modificado ou corrigido</b> com facilidade e sem que novos defeitos sejam inseridos</p> <ul style="list-style-type: none"> <li>– <b>manutenção preventiva</b> – <i>refactoring</i></li> <li>– <b>correção</b> – <b>corrigibilidade</b> (<i>argh!</i>)</li> <li>– <b>melhorias, adaptação e evolução</b> – <b>evolutibilidade</b></li> </ul>
	<b>Longevidade:</b>	<p>habilidade do software e dos dados persistentes por ele utilizados terem <b>vida longa</b>, evoluindo junto com as necessidades do usuário, com a plataforma, ou com a tecnologia utilizada para a sua implementação</p> <ul style="list-style-type: none"> <li>– engenharia reversa</li> <li>– reengenharia</li> <li>– rejuvenescimento</li> </ul>
	<b>Co-evolutibilidade:</b>	<p>facilidade de <b>manter a coerência</b> entre todos os artefatos que constituem o software.</p>
	<b>Disponibilizabilidade:</b>	<p>facilidade de <b>distribuir e por em uso correto</b> as novas versões.</p>
	<div> Fev 2015 Arndt von Staa © LES/DI/PUC-Rio 24 </div>	

Fatores da fidedignidade, controle		LES
Laboratório de Engenharia de Software	<p><b>Controlabilidade</b> (Verificabilidade , Validabilidade , Aprovabilidade):            habilidade de ter sua <b>qualidade controlada</b> com facilidade, baixo custo e suficiente rigor <b>sempre que desejado</b></p> <p><b>Testabilidade</b>: habilidade de ser <b>testado com o rigor</b> necessário, a um custo baixo e sempre que desejado            – uma forma de realizar (parcialmente) as três características anteriores</p> <p><b>Mensurabilidade</b>: habilidade do software <b>medir seu desempenho</b></p> <p><b>Detectabilidade</b>: habilidade do <b>software em execução observar erros</b>, iniciando alguma operação de recuperação ou de prevenção de danos</p> <p><b>Diagnosticabilidade</b>: facilidade de <b>determinar a causa</b> de uma falha ou identificar os pontos de alteração</p> <p><b>Depurabilidade</b>: facilidade de <b>remover correta e completamente</b> os defeitos diagnosticados, ou de realizar a alteração</p>	
	Fev 2015	Arndt von Staa © LES/DI/PUC-Rio
		25


Fatores do controle da qualidade		LES
Laboratório de Engenharia de Software	<p><b>Sensitividade</b> se um controle da qualidade (e.g. caso de teste) observa uma falha, esta será <b>sempre</b> observada ao repetir o controle enquanto o artefato não for alterado</p> <p><b>Redundância</b> as diferentes formas de <b>dizer ou observar a mesma coisa</b> são coerentes</p> <p><b>Particionamento</b> (modularidade) quebrar um problema em <math>n &gt; 1</math> subproblemas, cada qual bem definido, bem delimitado e autocontido</p> <ul style="list-style-type: none"> <li>• cada módulo, componente ou programa visa um <b>único propósito</b></li> <li>• o propósito pode ser estabelecido em níveis de abstração elevados (programas, componentes), ou baixos (módulos)</li> </ul> <p><b>Visibilidade</b> progresso, especificações e design são observáveis</p> <p><b>Feedback</b> (retroalimentação) sempre manter todos os interessados informados</p>	
	Fev 2015	Arndt von Staa © LES/DI/PUC-Rio
		26

Conceitos básicos


Laboratório de Engenharia de Software

- **Qualidade satisfatória**
  - Um artefato possui *qualidade satisfatória* caso satisfaça **plenamente** as necessidades e anseios de clientes e usuários, oferecendo riscos de uso justificavelmente aceitáveis
    - **fidedignidade**
  - a noção de “satisfatório” varia
    - com a finalidade a que se destina o artefato
    - com a natureza do artefato
    - com o papel desempenhado pelo usuário
    - com o nível de treinamento / conhecimento do usuário

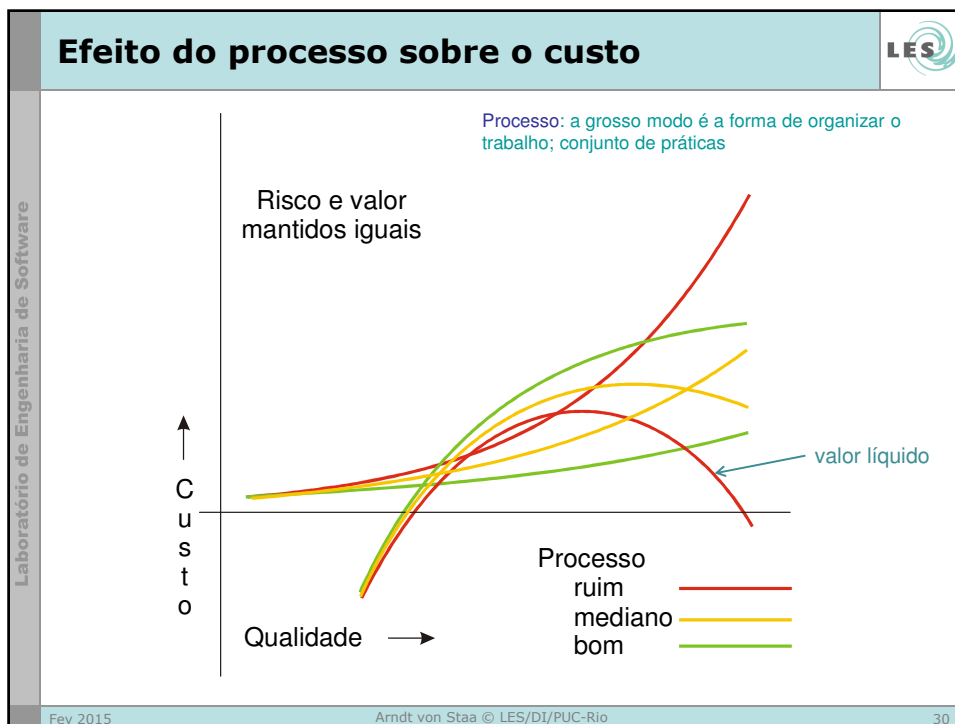
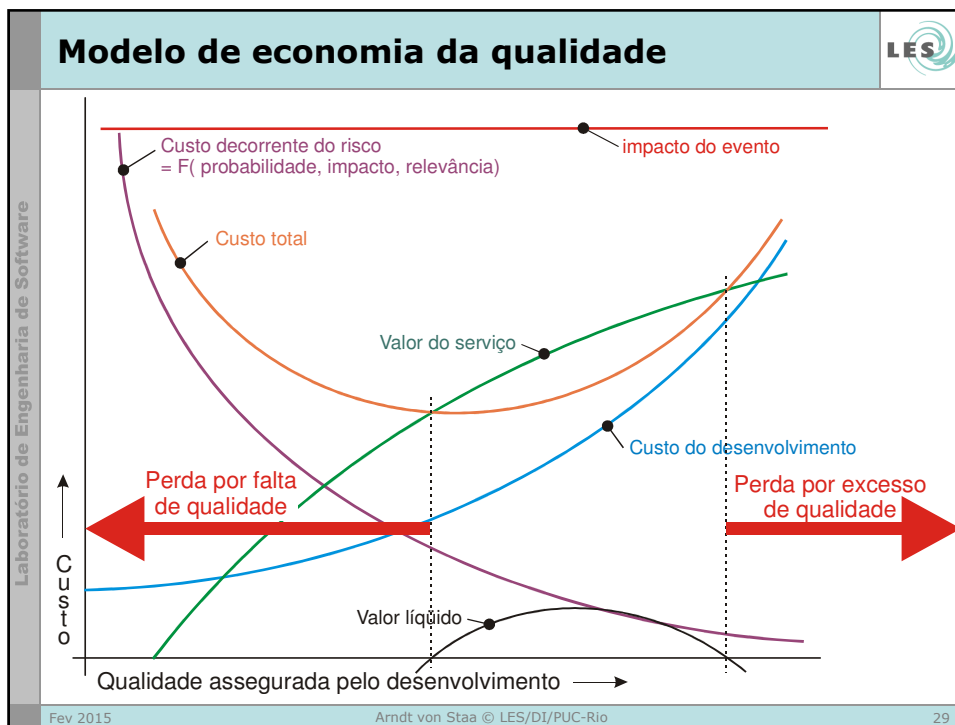
Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
27

Conceitos básicos


Laboratório de Engenharia de Software


- **Qualidade por construção**
  - Um artefato possui *qualidade por construção* caso possua qualidade satisfatória, considerando todas as propriedades relevantes, **antes do primeiro teste**
    - um ideal ao qual devemos-nos aproximar
- **Qualidade por desenvolvimento**
  - Um artefato possui *qualidade por desenvolvimento* caso possua qualidade satisfatória, considerando todas as propriedades relevantes, **antes de ser posto em uso**
    - podem sobrar defeitos não conhecidos!
- **Qualidade por manutenção**
  - Um artefato possui *qualidade por manutenção* caso possua qualidade satisfatória, considerando todas as propriedades relevantes, **antes de ser repostado em uso**
    - podem ter sido adicionados defeitos não conhecidos!

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
28



Laboratório de Engenharia de Software

## Por que desenvolver com qualidade?




- Custo do mau funcionamento, exemplos
  - aborrecimento
  - pequenas perdas financeiras e/ou de material
  - grandes perdas financeiras e/ou de material
  - falência de empresas
  - acidentes graves
  - danos ecológicos consideráveis
  - perda de vidas
- Custo da correção
  - retrabalho inútil

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
31

Laboratório de Engenharia de Software

## Por que desenvolver com qualidade?



- Uma das principais causas do excessivo tempo gasto (custo) ao desenvolver software é o retrabalho inútil
  - é responsável, em média, por cerca de 50% do custo de desenvolvimento
- Retrabalho inútil – é trabalho que não precisaria ter sido realizado, se o trabalho anterior tivesse sido realizado visando fidedignidade, ou seja pelo menos de forma correta, completa, consistente, e adequada aos usuários

existe retrabalho útil?


- Fairley, R.E.; Willshire, M.J.; "Iterative Rework: The Good, the Bad, and the Ugly"; *IEEE Computer* 38(9); Los Alamitos, CA: IEEE Computer Society; 2005; pags 34-41
- Boehm, B.W.; Basili, V.R.; "Software Defect Reduction Top 10 List"; *IEEE Computer* 34(1); Los Alamitos, CA: IEEE Computer Society; 2001; pags 135-137

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
32



Laboratório de Engenharia de Software

## O que é retrabalho inútil?



- Exemplos:
  - desenvolver algo e **descobrir que não era bem isso** que se queria ou que se precisava → inadequação
    - causa: especificação inexistente ou mal formulada
  - desenvolver algo e descobrir que **está eivado de defeitos**
    - causa: falta de disciplina
    - causa: falta de conhecimento de como raciocinar sobre programas, componentes, módulos e código
  - trabalhar **sem foco**
    - causa: falta de método de trabalho
      - processo, planejamento
  - **perfeccionismo** patológico
    - causa: melhorar, melhorar e melhorar mais ainda algo que já está satisfatório
  - . . .


Fev 2015

Arndt von Staa © LES/DI/PUC-Rio

33

Laboratório de Engenharia de Software

## Como eliminar causas de retrabalho inútil?



- O que fazer para **reduzir ou evitar** de vez esse **risco**?
  - **organizar e disciplinar** (planejar) o trabalho
  - utilizar sistematicamente **boas práticas** ao desenvolver
  - controlar **imediatamente** a qualidade de todos os artefatos
  - produzir uma **boa especificação** do que se quer que seja feito
    - evitar especificações erradas, incompletas ou inexistentes
  - produzir uma **arquitetura** – organização da solução – adequada ao problema a resolver
    - modelar o problema a resolver → **modelagem conceitual**
    - modelar a solução → **modelagem física**
    - desenvolver testes junto com a criação dos modelos → **desenvolvimento dirigido por testes** (*test driven development*)
  - **especificar** detalhes da **implementação**
    - especificação de funções,
    - assertivas – desenvolvimento dirigido por argumentação, *design by contract*

Fev 2015

Arndt von Staa © LES/DI/PUC-Rio

34

## Qualidade dos artefatos



Duas visões da qualidade:

- **qualidade do serviço**
  - é a **qualidade** do artefato tal como **observada pelo usuário**
    - usuários – interessados (*stakeholders*), exemplos
      - pessoas – usuário propriamente dito
      - outros artefatos
      - desenvolvedores cliente
- **qualidade da engenharia**
  - é a **qualidade requerida pela implementação** do artefato, necessária para atingir a qualidade de serviço desejada
  - é **observada pelos desenvolvedores**
    - usuários, exemplos
      - desenvolvedores do artefato
      - mantenedores
      - testadores

Fev 2015

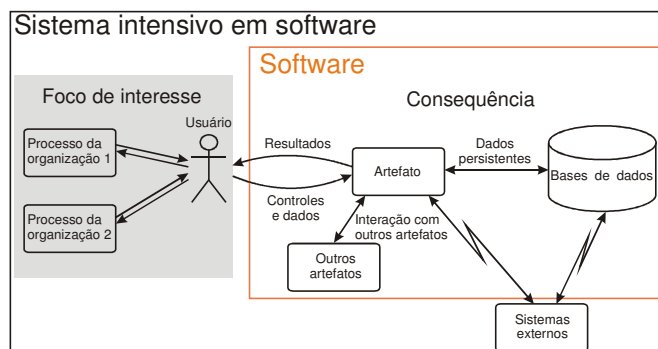
Arndt von Staa © LES/DI/PUC-Rio

35

## O que realmente interessa



- O foco de interesse são as **tarefas** que o **usuário realiza** no contexto da organização em que atua
  - o usuário **não quer** meramente **usar um artefato** (sistema)
  - o usuário **quer** realizar **adequada e facilmente** tarefas **com o apoio do artefato**
- COBIT → sistemas de informação são parte de um serviço, não são produtos




Fev 2015

Arndt von Staa © LES/DI/PUC-Rio

36

## Qualidade dos artefatos



Laboratório de Engenharia de Software


- Alcança-se qualidade de serviço **a partir da** qualidade de engenharia
  - O **objetivo primário** é a qualidade do serviço
  - Qualidade de engenharia é **objetivo secundário**
    - é incorporada na medida do necessário

Fev 2015

Arndt von Staa © LES/DI/PUC-Rio

37

## Garantia da qualidade



Laboratório de Engenharia de Software

- É o conjunto de **atividades sistemáticas** visando **assegurar a adequação ao uso** do artefato como um todo


**adequação ao uso:** é alta a **probabilidade** de que o software satisfaça plenamente as necessidades e anseios explícitos e implícitos do usuário e do cliente (fidedignidade)

Fev 2015

Arndt von Staa © LES/DI/PUC-Rio

38

**Atividades típicas 1/4**




Laboratório de Engenharia de Software

- Verificar se **são obedecidas** todas as normas, padrões, práticas e convenções estipuladas
- Verificar se as **documentações técnica** e de uso
  - são produzidas
  - são utilizadas para desenvolver
  - são mantidas atualizadas – **co-evoluídas**
  - estão sempre disponíveis
  - são inteligíveis por todos os leitores a que se destinam
  - contêm tudo o que deveriam conter

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
39

**Atividades típicas 2/4**




Laboratório de Engenharia de Software

- Verificar se o **controle da qualidade** é realizado conforme determinado
  - critérios de aceitação estabelecidos previamente
- Verificar se os problemas identificados pelo controle da qualidade são **registrados** e **acompanhados** até a sua completa resolução
  - acompanhamento de demandas (*issue tracking*)
  - gerência da configuração
  - gerência de requisitos
    - gerência da evolução e da adaptação

Fev 2015
Arndt von Staa © LES/DI/PUC-Rio
40

Laboratório de Engenharia de Software

## Atividades típicas 3/4



- Verificar se medições são coletadas e **usadas para melhorar**:
  - o processo de desenvolvimento
  - padrões de uso das linguagens de representação
  - padrões de qualidade de representações
- Verificar se métodos, técnicas e ferramentas são **adequadas e utilizadas** por todos os participantes
- Verificar se todos utilizam as **mesmas versões das ferramentas** e os mesmos parâmetros de configuração
- Verificar se os artefatos aprovados são **armazenados em bibliotecas controladas**
  - controle de versões


Fev 2015

Arndt von Staa © LES/DI/PUC-Rio

41

Laboratório de Engenharia de Software

## Atividades típicas 4/4



- Verificar se existe um sistema de **backup** adequado
  - assegurar que este sistema funciona corretamente
- Verificar se o **pessoal** envolvido no projeto
  - é **disciplinado**
  - possui **proficiência** suficiente
  - está **habilitado** a usar as ferramentas, padrões e processos disponíveis
  - dispõe de oportunidades para obter o **treinamento necessário**
    - evolução profissional contínua

Fev 2015

Arndt von Staa © LES/DI/PUC-Rio

42

Laboratório de Engenharia de Software

LES

FIM

Fev 2015

Arndt von Staa © LES/DI/PUC-Rio

43