




Laboratório de Engenharia de Software

# Qualidade de Software

## Conceitos

Arndt von Staa  
Departamento de Informática  
PUC-Rio  
Fevereiro 2017

## Especificação



Laboratório de Engenharia de Software

- Objetivo da aula
  - Definir o que é qualidade
  - Contextualizar qualidade de software

Leitura complementar: Pezzè – capítulos 1 e 2

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

Laboratório de Engenharia de Software

Engenharia de software

LES

Engenharia de Software visa desenvolver economicamente software adequado a todos os interessados possuindo qualidade assegurada e capaz de operar fidedignamente em ambientes reais

Engineering is the application of mathematics and scientific, economic, social, and practical knowledge in order to invent, innovate, design, build, maintain, research, and improve structures, machines, tools, systems, components, materials, processes, solutions, and organizations. (Wikipedia)

Fev 2017

Arndt von Staa © LES/DI/PUC-Rio

3

Laboratório de Engenharia de Software

Definição de qualidade

LES

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 todos os seus interessados

ABNT

Podem existir conflitos entre propriedades

Problema: necessidades implícitas, como saber quais são elas?

Fev 2017

Arndt von Staa © LES/DI/PUC-Rio

4

## Quem, ou o que, é o interessado?



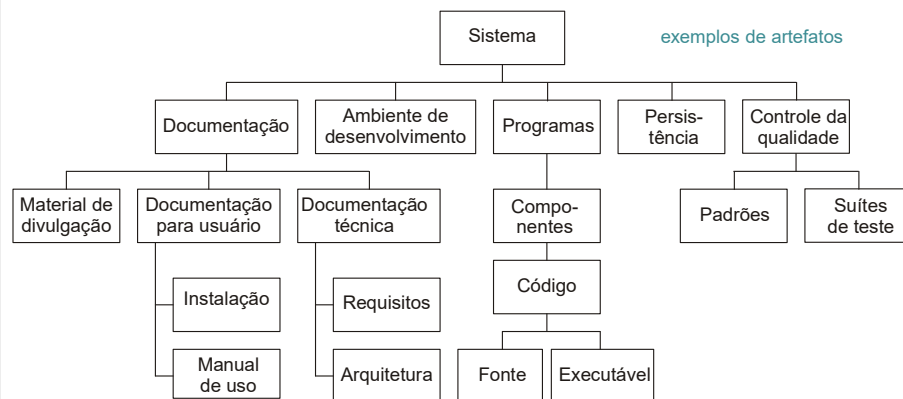
- Exemplos de **interessados**, ou **stakeholders**
  - papel desempenhado** por pessoa que interage com o artefato
    - usuário**
    - cliente
    - desenvolvedores
    - controladores externos da qualidade
    - mantenedores
    - operadores
    - auditores
    - ...
  - funcionalidade desempenhada pelo equipamento que interage com o artefato → **usuário** ?
    - sensores
    - atuadores
  - software de terceiros que interage com o artefato
    - componente, software como serviço (SOA)
    - ...

Fev 2017

Arndt von Staa © LES/DI/PUC-Rio

5

## O que é um artefato de software?

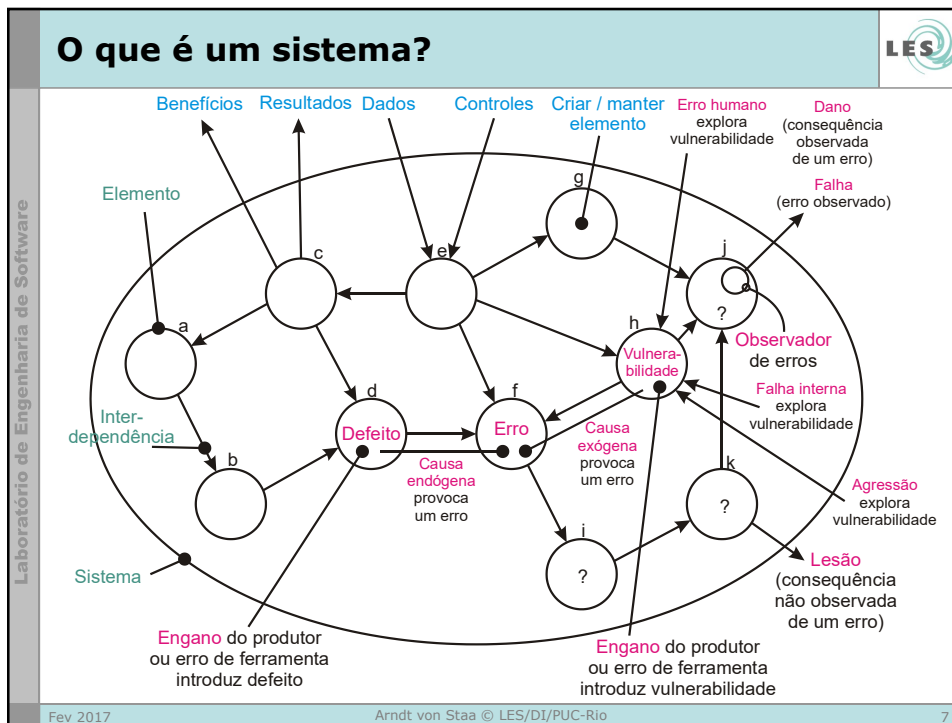


- Artefato é qualquer coisa **tangível** que possua **qualidade controlada** e que **componha** o sistema em questão
- Artefato é um conceito **recursivo**: artefatos podem ser compostos por artefatos
- Artefato forma uma **estrutura de relacionamentos**: artefato A depende de artefatos B, C, ...

Fev 2017

Arndt von Staa © LES/DI/PUC-Rio

6



## Definições

LES

- **Artefatos** são **resultados tangíveis** possuindo qualidade controlada e resultantes do desenvolvimento ou da manutenção
- **Defeito** é um fragmento de um artefato que, **se utilizado**, pode levar a um erro
- **Erro** é um **desvio** entre o que é desejado ou intencionado e o que é gerado ou derivado. Erro é causado por um defeito
- **Falha** é um erro **observado**
- **Latência do erro** é o **tempo decorrido** entre o momento em que o erro é **gerado** e o momento em que é **observado**
- **Dano** é a **consequência externa conhecida** (prejuízo observado) provocada por uma falha
- **Lesão** é a **consequência externa desconhecida** provocada por um erro não observado

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

## Definições



- **Vulnerabilidade** é um fragmento de um artefato, que, quando usado *em condições peculiares*, *pode* gerar um erro
  - ex. 1. proteção mal feita permite acesso a pessoas não autorizadas
  - 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 **vulnerabilidades são defeitos**
  - precisa-se procurar **explicitamente** por elas

## Definições



- **Inadequação** ex.
  - não atender às reais necessidades dos interessados
  - não satisfazer os requisitos não funcionais e inversos estabelecidos
- **Deficiência** ex.
  - interface humana ruim
  - induzir usuário a cometer erros de uso
  - documentação e auxílios não conformes com o implementado
- **Anomalia** (*bad smell*) ex.
  - o artefato está correto, do ponto de vista funcional e não funcional, mas é difícil de manter
  - engenharia ruim

## 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
- **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%)

## Risco e qualidade satisfatória



- É virtualmente impossível desenvolver um sistema perfeito
- **Risco** é um **evento** que tem uma **probabilidade de ocorrer**, **impactando negativamente** um interessado
- **Qualidade satisfatória**
  - Um artefato possui *qualidade satisfatória* caso satisfaça **plenamente** os anseios de todos os interessados, oferecendo riscos justificavelmente aceitáveis para cada propriedade
    - **fidedignidade**, ver apêndice
  - a noção de "satisfatório" varia
    - com a finalidade a que se destina o artefato
    - com a natureza do artefato
    - com o papel desempenhado pelo interessado
    - com o nível de treinamento / conhecimento do interessado
    - ...

## Graus de qualidade

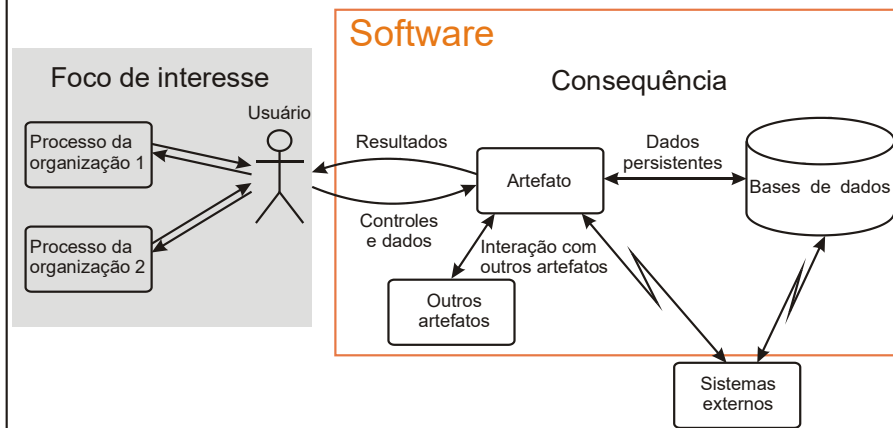


- **Qualidade por construção**
  - Um artefato possui *qualidade por construção* caso possua qualidade satisfatória, considerando todas as propriedades relevantes, **antes do primeiro teste**
    - um ideal ao qual nos devemos aproximar
- **Qualidade por desenvolvimento**
  - Um artefato possui *qualidade 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!

## Sistema intensivo em software



### Sistema intensivo em software



Em sistemas interativos, o usuário é parte do sistema intensivo em software

## Necessidade da especificação



- Não é possível atingir qualidade desejada, se
  - se os atributos de qualidade relevantes e respectivos graus a atingir **não forem definidos antes** de desenvolver
  - se graus a atingir não forem **observáveis**
    - mensurável
    - avaliável estatisticamente
    - avaliável por especialista devidamente treinado
    - **não vale achologia**: “acho que está correto” ou similar
- Exemplo
  - se tempo de resposta for um item relevante,
    - precisa ser dito qual o tempo de resposta a ser atingido
    - a exigência precisa ser mensurável
      - “deve ser rápido”, não é mensurável
      - 90 % das vezes deve ser menor do que 0,1s, 100% das vezes menor do que 10s.
        - » como medir?

## Especificação de baixíssimo nível



- Do ponto de vista do usuário (que tipo de usuário?) isso é uma boa especificação?

```
double f( double a )
{
    double x = a / 2.0 ;
    while ( x * x - a > 0.1E-5 )
    {
        x = ( x + a / x ) / 2.0 ;
    } /* while */
    return x ;
}
```

o que faz esse algoritmo?  
alguma coisa pode dar errado?
- E isso:
$$x_1 = a / 2 \rightarrow x_{n+1} = x_n - f(x_n) / f'(x_n)$$
- E finalmente isso produz definido como

```
sqrt( a ) =: x ::= ? a >= 0 -> 1 - ε < x**2 / a < 1 + ε
```



## Problema básico



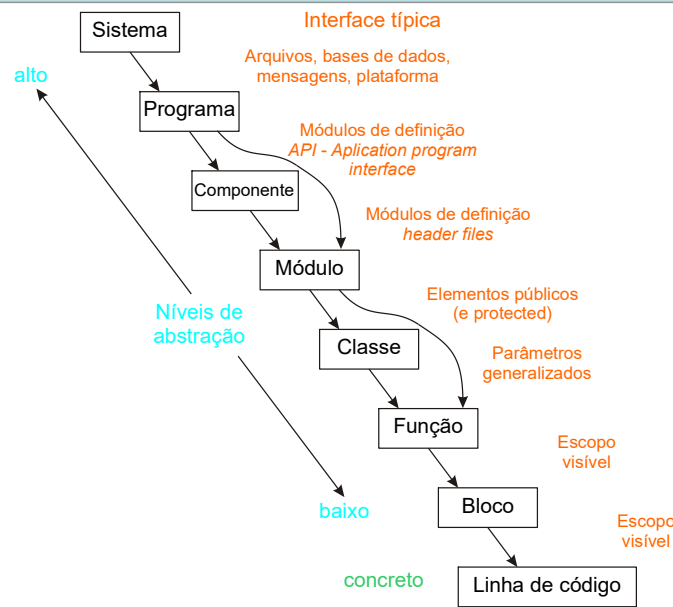
- O desenvolvimento de software é um **processo de aquisição e detalhamento de conhecimento**
  - no início o conhecimento é incompleto e abstrato
    - do problema a resolver
    - da solução proposta dada ao problema
  - à medida que vai sendo desenvolvido o conhecimento vai sendo completado
- Consequentemente muitas **solicitações de alteração ocorrem durante o desenvolvimento** em virtude de novo conhecimento adquirido
  - conhecimento sobre o **serviço** e a qualidade desse serviço
  - conhecimento sobre a **solução** e a qualidade da engenharia

Fev 2017

Arndt von Staa © LES/DI/PUC-Rio

17

## Evolução da abstração

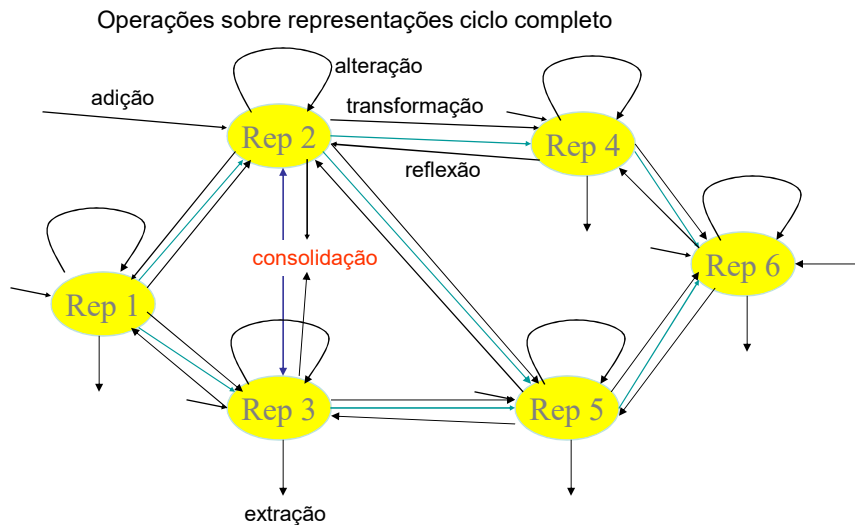


Fev 2017

Arndt von Staa © LES/DI/PUC-Rio

18

## Características do desenvolvimento



## Macro operações sobre representações



- Adição
- Extração
- Modificação
- Propagação (para frente)
- Reflexão (propagação reversa)
- Consolidação, negociação
- Pesquisa (procura)
- Reuso
  - *verbatim, as is*, assim como está; p.ex. através de referências
- Composição
- Decomposição
- Verificação
  - correto com relação à especificação e aos padrões do artefato
- Validação
  - correto com relação aos demais artefatos
- Aprovação
  - correto segundo os interesses do interessado

- *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.** silogismo
- *Sed in errore perseverare dementis*
  - enquanto que perseverar no erro é próprio do louco
  - **Corolário 2:** é sinal de incompetência e obstinação inútil **não aceitar que erramos**, não querer observar que erramos, não procurar formas de evitar os erros, não querer aprender com os erros nossos e de outros

- **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
  - isso implica a necessidade de avaliar a corretude (controlar a qualidade) também **durante a execução** do sistema
    - torna necessária a instrumentação do código

Laboratório de Engenharia de Software

## Não existe software perfeito

LES

- Defeitos podem ter causa
  - na especificação dos requisitos de serviço
  - na especificação de requisitos da solução
  - na organização do trabalho (processo)
  - na qualidade do trabalho realizado
  - na proficiência dos participantes do desenvolvimento
  - na fidedignidade dos componentes de terceiros e dos instrumentos utilizados
  - . . .

Fev 2017

Arndt von Staa © LES/DI/PUC-Rio

23

Laboratório de Engenharia de Software

## Não existe software perfeito

LES

- Perfeição não existe, densidade de defeitos [Bao, 2007]:
  - hardware (hardware é hard, ou contém software?)
    - 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])
  - software
    - 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 2017

Arndt von Staa © LES/DI/PUC-Rio

24

## Não existe software perfeito



- Mesmo sistemas perfeitos podem falhar
  - erros provocados por causas **exógenas**
    - mau uso, deliberado ou não
    - uso incorreto (possivelmente induzidos por interfaces ruins)
    - bases de dados poluídas
    - falhas de hardware
    - falhas da plataforma de software usada
    - falhas de componentes ou bibliotecas de terceiros
    - ...
- Consequência
  - sistemas precisam ser **clementes**
    - permitir a correção de erros
  - sistemas precisam conter **observadores dos erros gerados** durante a execução


## Propagação de defeitos



- Defeitos podem ser inseridos 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, corresponde a defeito(s) nesse(s) artefatos
  - exemplo de defeitos 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

Laboratório de Engenharia de Software

## Por que desenvolver visando qualidade?



- **Retrabalho inútil** – é trabalho que não precisaria ter sido realizado, se o trabalho original tivesse sido realizado **visando fidedignidade por construção**
- 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


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; pages 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; pages 135-137

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

Laboratório de Engenharia de Software

## Exemplos de retrabalho inútil?




- 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 (disciplina) de trabalho
    - processo, planejamento, padrões
- **Perfeccionismo** patológico
  - causa: melhorar, melhorar e melhorar mais ainda algo que já está satisfatório
- . . .

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

Laboratório de Engenharia de Software

## Como eliminar causas de retrabalho inútil?



- O que fazer para **reduzir ou mitigar** de vez esse **risco**?
  - **organizar e disciplinar** (planejar) o trabalho
  - utilizar sistematicamente **boas práticas** ao desenvolver
  - saber como será controlada a qualidade **antes de iniciar**
  - controlar **continuamente** a qualidade de todos os artefatos
  - produzir uma **boa especificação** do que se quer que seja feito
    - evitar especificações erradas, incompletas ou inexistentes
    - procurar usar técnicas formais leves
  - 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*)


Fev 2017

Arndt von Staa © LES/DI/PUC-Rio

29

Laboratório de Engenharia de Software

## Controle da qualidade versus evitar defeitos



- Controle da qualidade **não leva a sistemas possuindo a qualidade desejada**
  - somente produz um laudo
  - a melhoria da qualidade advém da **eliminação dos defeitos observados**
    - provoca retrabalho inútil → mais custos
    - se não feito, gera dívida técnica
- Ao invés de fiar-se somente no controle da qualidade, por que não desenvolver de modo que se **tenha a (quase) certeza de não ter introduzido defeitos**?
  - o mais próximo possível de **correto por construção**

Fev 2017

Arndt von Staa © LES/DI/PUC-Rio

30

## Qualidade dos artefatos



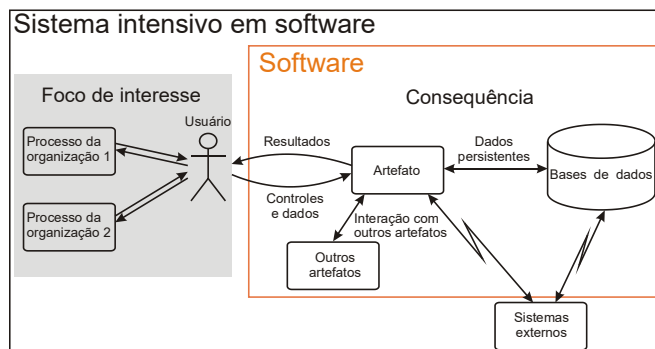
Duas visões da qualidade:

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

## Qualidade do serviço, qualidade externa

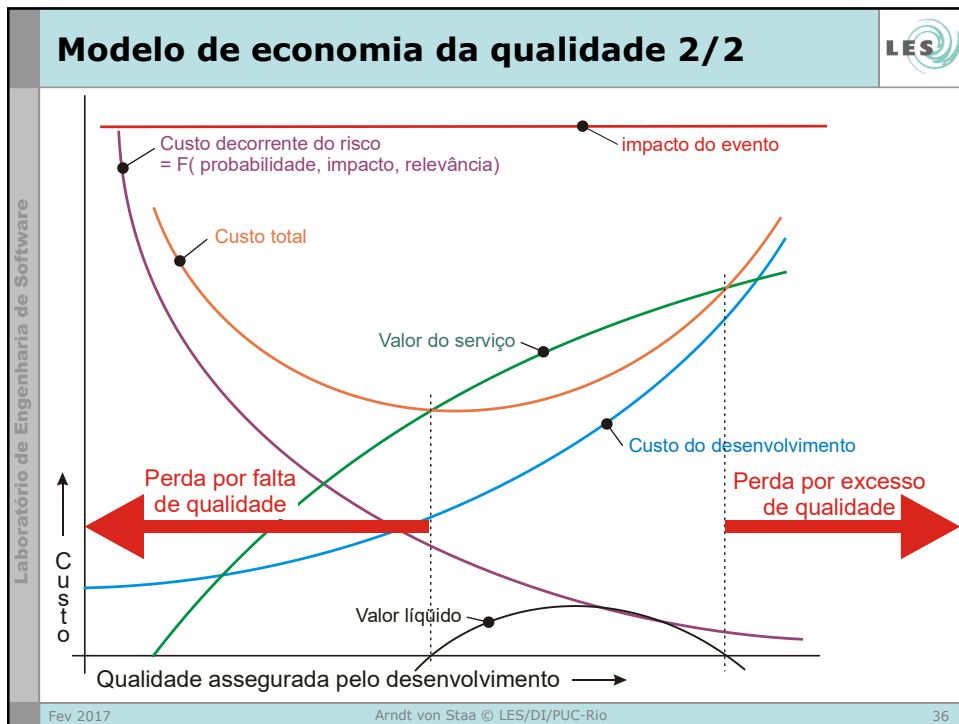
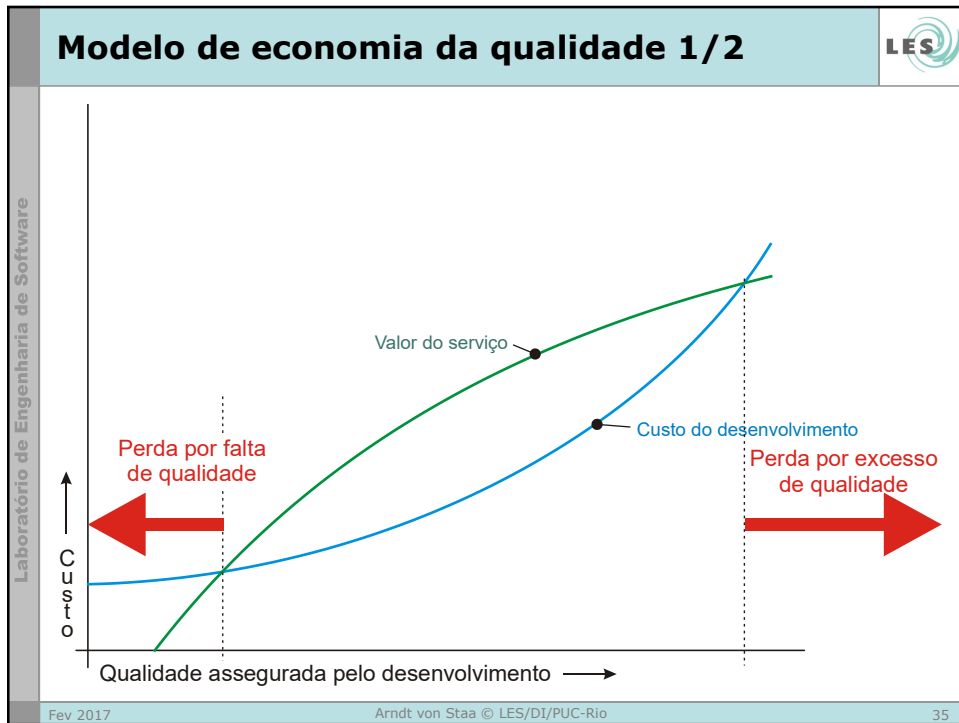


- O foco de interesse são as **tarefas** que o **usuário realiza** no contexto da organização em que atua
  - o usuário **não quer** meramente **usar um artefato** (sistema)
  - o usuário **quer** realizar **adequada e facilmente** tarefas **com o apoio do artefato**
- COBIT → sistemas de informação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









Laboratório de Engenharia de Software

# Apêndice

LES

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

Laboratório de Engenharia de Software

## Fidedignidade de software

LES

Um sistema **intensivo em software** é **fidedigno** caso atenda **satisfatoriamente** um **conjunto de propriedades** de modo que se possa **justificavelmente** **depende** 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 2017
Arndt von Staa © LES/DI/PUC-Rio
38

Fatores da fidedignidade, básicas		LES
Adequação:	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; ...	<p>As características são adaptadas de (Avizienis, 2004) e de outros autores, são mais abrangentes do que as do autor citado</p>
Confiabilidade:	habilidade de, sempre que solicitado, prestar <b>serviço fidedigno</b>	
Disponibilidade:	estar pronto para prestar serviço fidedigno <b>sempre que necessitado</b>	
Utilizabilidade:	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	
Clemência:	habilidade do software <b>perdoar erros de uso</b>	

Laboratório de Engenharia de Software  
 Fev 2017  
 Arndt von Staa © LES/DI/PUC-Rio  
 39


Fatores da fidedignidade, básicas		LES
Interoperabilidade:	habilidade do software poder ser corretamente <b>conectado</b> com outros sistemas	<p>As características são adaptadas de (Avizienis, 2004) e de outros autores, são mais abrangentes do que as do autor citado</p>
Escalabilidade:	habilidade da <b>capacidade de processamento</b> do software <b>crescer junto</b> com o crescimento da demanda	
Durabilidade:	habilidade do software operar fidedignamente por <b>períodos de duração indefinida</b> – longa duração, e.g. 24/7	
Economia:	habilidade de produzir resultados necessitando de <b>poucos recursos</b> – ex. computacionais, humanos, financeiros	
Desempenho:	habilidade de <b>atender à demanda</b> consumindo recursos (ex. tempo, memória) dentro do limite estipulado	

Laboratório de Engenharia de Software  
 Fev 2017  
 Arndt von Staa © LES/DI/PUC-Rio  
 40

Fatores da fidedignidade, segurança		LES
Laboratório de Engenharia de Software	<b>Segurança:</b>	<p>habilidade de <b>evitar consequências catastróficas</b> aos usuários, à organização, ou ao ambiente</p> <ul style="list-style-type: none"> <li>– <i>safety</i>, risco baixo</li> </ul>
	<b>Proteção:</b>	<p>habilidade de <b>evitar o sucesso</b> de tentativas de agressão</p>
	<b>Privacidade:</b>	<p>habilidade de <b>proteger dados e código contra acesso</b> (uso) indevido acidental ou deliberado</p>
	<b>Integridade:</b>	<p>habilidade de <b>evitar a corrupção</b> (adulteração) intencional ou acidental de elementos</p>
Fev 2017		Arndt von Staa © LES/DI/PUC-Rio 41

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>• <i>lesão: "prejuízo" desconhecido causado por erro não observado</i></li> </ul> </li> <li>– pode gerar danos, desde que controlados <ul style="list-style-type: none"> <li>• <i>dano: "prejuízo" causado por erro observado</i></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>
	Fev 2017	

Laboratório de Engenharia de Software	<b>Fatores da fidedignidade, evolução</b>		
	<b>Manutenibilidade:</b>	habilidade de poder ser <b>modificado ou corrigido</b> com facilidade e sem que novos defeitos sejam inseridos <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>	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 <ul style="list-style-type: none"> <li>– engenharia reversa</li> <li>– reengenharia</li> <li>– rejuvenescimento</li> </ul>	
	<b>Co-evolutibilidade:</b>	facilidade de <b>manter a coerência</b> entre todos os artefatos que constituem o software.	
	<b>Disponibilizabilidade:</b>	facilidade de <b>distribuir e por em uso correto</b> as novas versões.	
	Fev 2017	Arndt von Staa © LES/DI/PUC-Rio	43

Laboratório de Engenharia de Software	<b>Fatores da fidedignidade, controle</b>		
	<b>Controlabilidade (Verificabilidade , Validabilidade , Aprovabilidade):</b>	habilidade de ter sua <b>qualidade controlada</b> com facilidade, baixo custo e suficiente rigor <b>sempre que desejado</b>	
	<b>Testabilidade:</b>	habilidade de ser <b>testado com o rigor</b> necessário, a um custo baixo e sempre que desejado <ul style="list-style-type: none"> <li>– uma forma de realizar (parcialmente) as três características anteriores</li> </ul>	
	<b>Mensurabilidade:</b>	habilidade do software <b>medir seu desempenho</b>	
	<b>Detectabilidade:</b>	habilidade do <b>software em execução observar erros</b> , iniciando alguma operação de recuperação ou de prevenção de danos	
	<b>Diagnosticabilidade:</b>	facilidade de <b>determinar a causa</b> de uma falha ou identificar os pontos de alteração	
	<b>Depurabilidade:</b>	facilidade de <b>remover correta e completamente</b> os defeitos diagnosticados, ou de realizar a alteração	
	Fev 2017	Arndt von Staa © LES/DI/PUC-Rio	44

Fatores do controle da qualidade		LES
Laboratório de Engenharia de Software	Sensitividade	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
	Redundância	as diferentes formas de <b>dizer ou observar a mesma coisa</b> são coerentes
	Modularidade	quebrar um problema em $n > 1$ subproblemas perfeitamente integráveis, cada qual bem definido, bem delimitado e autocontido <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>
	Visibilidade Feedback	progresso, especificações e design são observáveis (retroalimentação) sempre manter todos os interessados informados
Fev 2017		Arndt von Staa © LES/DI/PUC-Rio 45

		LES
Laboratório de Engenharia de Software	FIM	
Fev 2017		Arndt von Staa © LES/DI/PUC-Rio 46