




Laboratório de Engenharia de Software

# Revisões e Inspeções 1

Arndt von Staa  
Departamento de Informática  
PUC-Rio  
Março 2017

## Especificação



Laboratório de Engenharia de Software


- Objetivo da aula
  - apresentar técnicas de controle da qualidade baseados em leitura
- Justificativa
  - O controle da qualidade deve ser realizado continuamente
    - junto com o desenvolvimento
    - ainda **antes de se dispor do código**.
  - Algumas propriedades do código são **difíceis ou muito caras de testar**.
    - torna-se necessário o uso de técnicas de **controle da qualidade alternativas aos testes**.
  - **Revisões**, e **inspeções**, se bem realizadas, têm mostrado muito bons resultados.

• Pezzè, M.; Young, M.; *Teste e Análise de Software*; 2008; capítulo 18  
• Staa, A.v.; *Programação Modular*; Campus; 2000; capítulos 3, 10 e 12

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

Laboratório de Engenharia de Software

## Artefato, definição – recordação




- **Artefato** (*work product*):
  - é qualquer **resultado tangível** do desenvolvimento
    - está escrito, ou desenhado
    - está armazenado
  - e que teve a sua **qualidade controlada** de alguma forma
    - a forma de controlar pode ser desde bem simples
      - » ex. o artefato está registrado no repositório de controle de versões
    - até muito complexo
      - » ex. passou pela prova formal da corretude
    - uma forma intermediária baseia-se em leitura
- Como controlar a qualidade de um **artefato não executável**?

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

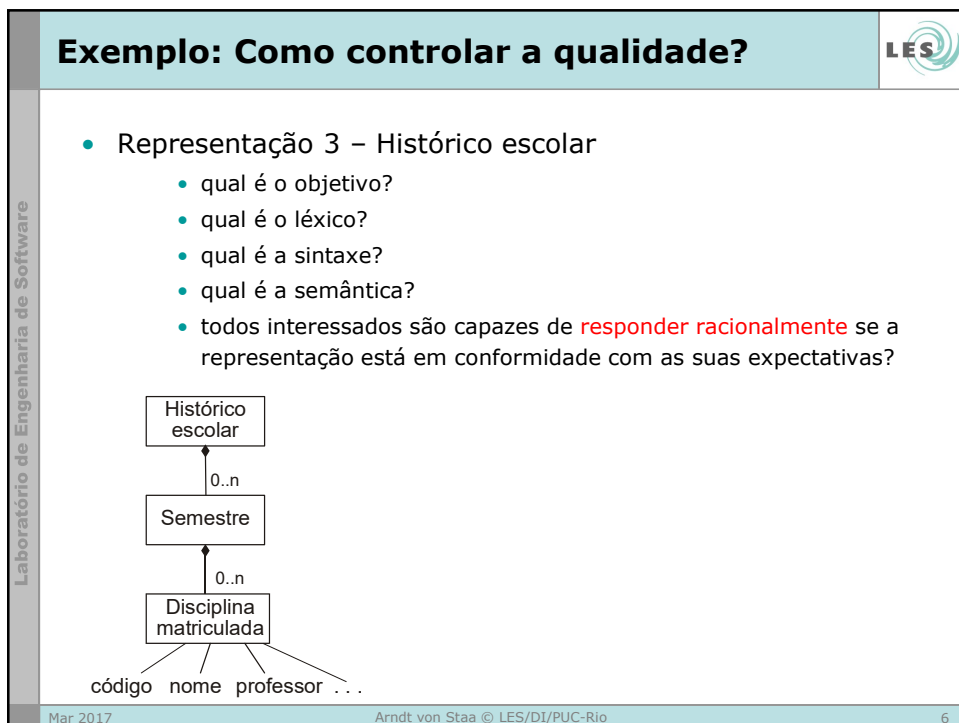
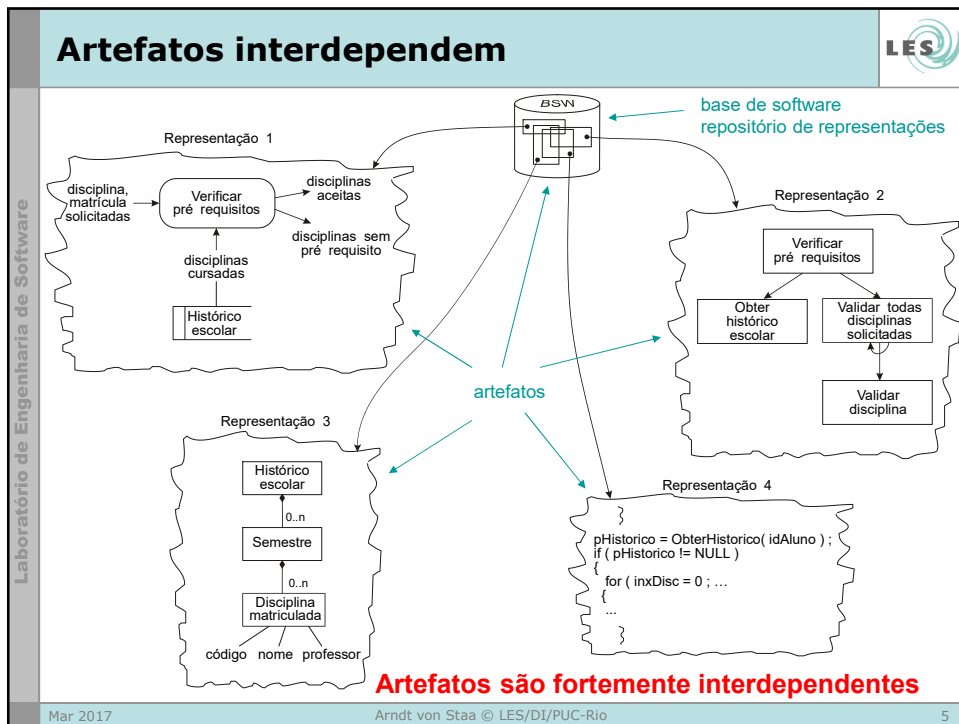
Laboratório de Engenharia de Software

## Artefato, conceitos



- Cada artefato é composto por uma ou mais **representações**, cada qual escrita em uma **linguagem de representação**
- Linguagens de representação
  - podem ser
    - textuais
    - formulários ou tabulares
    - gráficas
  - possuem
    - léxico
      - dicionário de termos
    - sintaxe
    - semântica
    - nível de abstração

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



## O mais importante: **interfaces**

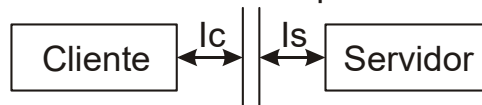


- Interfaces ocorrem em todo lugar
  - entre sistemas: ex. bancos de dados, mensagens
  - entre programas, processos e componentes: ex. arquivos, tabelas, mensagens, fluxos
  - entre módulos: ex. classe depende de classe
  - entre métodos: ex. parâmetros, dados persistentes
- O que interessa é a interface conceitual
  - tudo que é **recebido** e respectivas condições
  - tudo que é **retornado** e respectivas condições
  - nos diversos **contextos**: retorno normal, retorno por exceção
  - tudo sempre deve ser coerente

## Corretude da composição

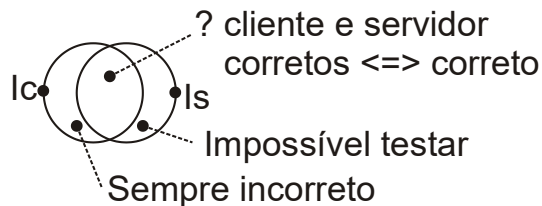


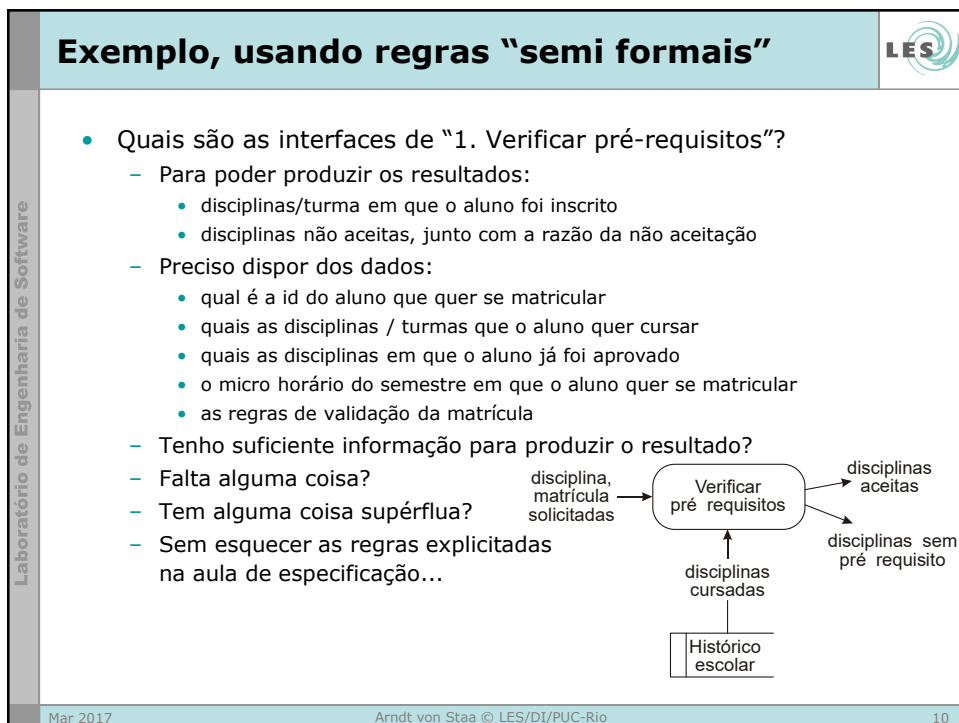
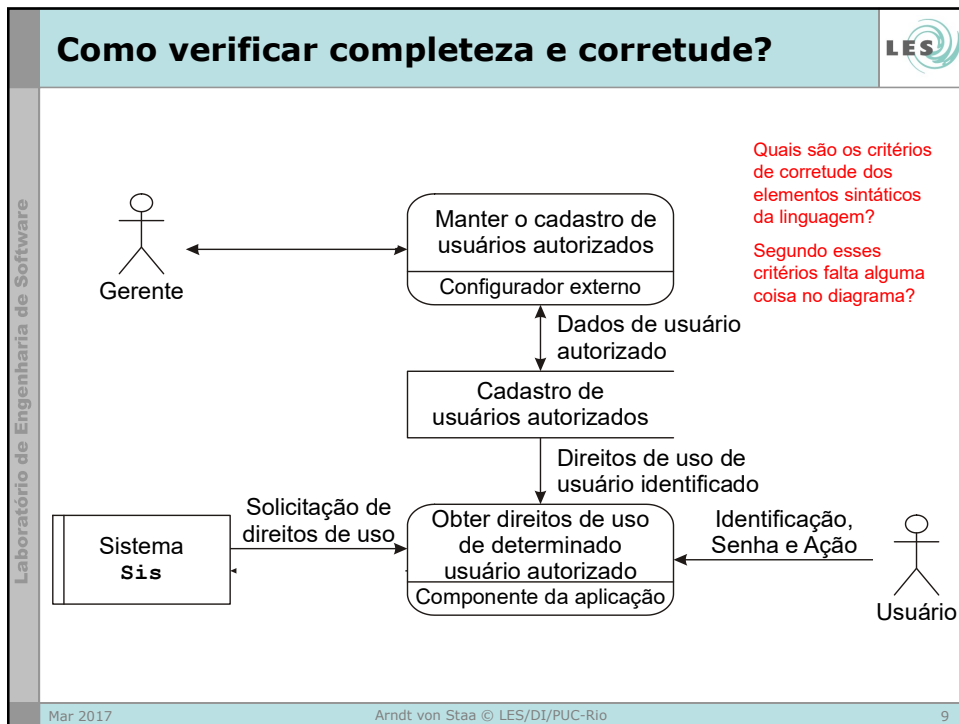
Controle realizado a partir do cliente



Ic - interface do ponto de vista do cliente

Is - interface do ponto de vista do servidor





## Como verificar o texto a seguir?



### Descrição da classe `RDT_ReadTestScript` do arcabouço de teste

Provê funções para a leitura e análise léxica dos comandos de teste.

Cada comando de teste está integralmente em uma linha.

Cada comando de teste inicia com o caractere '=' seguido de um *string* que identifica o comando.

Cada comando pode requerer zero ou mais parâmetros que se encontram na mesma linha que o comando.

Parâmetros podem ser literais ou simbólicos.

Os parâmetros simbólicos precisam estar declarados antes de serem utilizados fornecendo <nome do símbolo, tipo, valor>.

A sintaxe dos literais contidos na linha de comando obedecem à sintaxe usada no código C/C++ (não o do `scanf`)

...

Adaptado do arcabouço de teste C++, módulo `ReadTest`

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

11

## Como verificar a interface da classe?



**`RDT_ReadTestScript`**( `SMT_SymbolTable * pTable` )

`~RDT_ReadTestScript`( )

`RDT_tpRetCode` `OpenTestScriptFile`( `char * FileNameParm` )

`STR_String *` `GetTestScriptFileName`( )

`int` `GetNumberLinesRead`( )

`int` `ReadTestScriptLine`( )

`int` `AnalyzeCommandLine`( `char * fieldTypes` , ... ) Obs: ... número variável de parâmetros

`bool` `AnalyzeCommand`( `char * Command` , `int dimCommand` )

`bool` `AnalyzeName`( `int * sizName` , `char * Name` , `int dimName` )

`bool` `AnalyzeChar`( `char * Char` )

`bool` `AnalyzeDouble`( `double * Double` )

`bool` `AnalyzeInt`( `long * Int` )

`bool` `AnalyzeBool`( `bool * Bool` )

`bool` `AnalyzeString`( `int * sizString` , `char * String` , `int dimString` )

- Basta isso para saber como implementar essa classe?
- Basta isso para saber como testar a classe?
  - Como você testaria `AnalyzeCommandLine`( `char * fieldTypes` , ... )?

Isso complementa ou substitui o slide anterior?

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

12

## Especificação de AnalizeCommandLine



```
// Descrição
// Decodifica a linha contida no buffer de leitura, saltando o campo de comando
//
// Parâmetros
// tiposCampos - um string contendo os indicadores de tipo dos campos a serem lidos. O número
// de caracteres indicadores de tipo estabelece o número de campos a serem lidos
// Caracteres identificadores de tipo:
// b - booleano
// c - caractere
// d - double
// i - inteiro
// s - string, campos tipo string são lidos para dois parâmetros, o primeiro conterá o
// número de caracteres do string e o segundo conterá os caracteres do string
// com um zero adicionado ao final. O comprimento máximo é limitado a TAL_DIM_BUFFER
// caracteres.
// ... - lista de parâmetros, passados por referência, que receberão os valores dos campos lidos
//
// A sintaxe dos campos é idêntica à sintaxe dos literais c/c++. Por exemplo, um string
// deve estar contido entre aspas duplas.
// Ao ler um campo, primeiro é verificado se foi lido um nome. Caso tenha sido, verifica se
// se o nome está declarado como constante simbólica com tipo igual ao esperado
// Caso não seja um nome, é verificado se o tipo do literal lido é igual ao tipo esperado.
// strings podem conter qualquer tipo de caractere inclusive zeros (\0) e brancos
//
// Valor retornado
// Número de campos lidos. Em geral será igual ao número de caracteres contidos na lista de tipos.
...
```

## Exemplo de ReadLine



- Instrução contida na seção **FazAlgo** do executor do teste

```
numRead = AnalizeCommand( "is" ,
                           pInt, pLen, pString ) tudo por referência
```

- Comando contido no script de teste coerente com a instrução

```
=FazAlgo 15 "abcd" o resto é ignorado
```

- Resultado

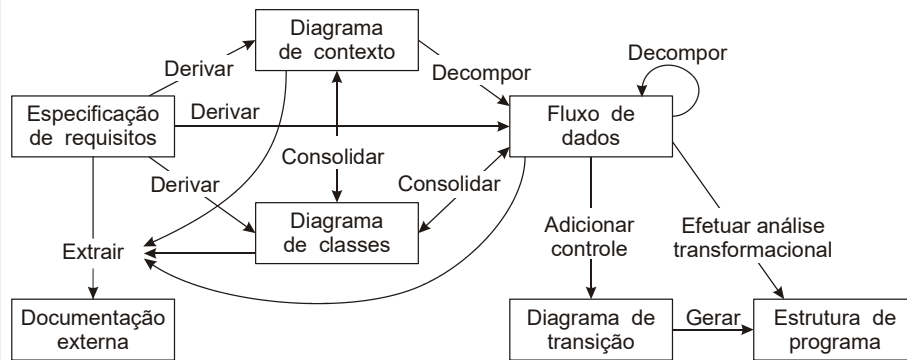
```
numRead == 2
*pInt    == 15
*pLen    == 4
*pString == "abcd"
```

O que é melhor, o texto anterior ou esse exemplo?

## Rede de linguagens de representação



Laboratório de Engenharia de Software



**Artefatos fazem parte de uma rede de transformações e dependências**

- **precisa-se assegurar continuamente a coerência do todo**

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

15

## Revisões e inspeções: definições



Laboratório de Engenharia de Software

- Uma **revisão** é uma **leitura crítica** do artefato e da documentação complementar, **anotando**:
  - os **defeitos** encontrados
  - as **omissões**
  - as **dúvidas** e dificuldades de compreensão
  - as possibilidades de **melhoria**
- Uma **inspeção** é uma **leitura crítica formalizada** do artefato e da documentação complementar
  - segundo um **plano** definido
  - obedecendo a **regras de leitura** definidas
  - envolvendo uma **pequena equipe**
  - anotando **defeitos, dúvidas e possibilidades de melhoria**

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

16



## Revisões e inspeções vs. testes



- Revisões e inspeções eliminam defeitos antes que se propaguem para outros artefatos ou para o serviço
  - um **defeito na especificação** provoca um **erro** ao criar a arquitetura (ou projeto), que se manifesta sob a forma de um **defeito na arquitetura**
  - um **defeito na arquitetura** provoca um **erro** ao criar o projeto, que se manifesta sob a forma de um **defeito no projeto**
  - um **defeito em um projeto** provoca um **erro** ao redigir o código, que se manifesta sob a forma de um **defeito no código** ou no **tratamento da interface**
  - a execução de um **defeito no código** pode causar um **erro execução**
    - lembre-se o erro **precisa ser observado** para tornar-se uma falha
  - um defeito no **tratamento ou definição de uma interface** pode estabelecer uma **vulnerabilidade**


## Revisões e inspeções vs. testes



- Revisões **devem** ser praticadas **a partir do primeiro momento** do desenvolvimento
  - **reduzem** significativamente o **retrabalho inútil**
    - quanto maior a **latência do erro**, maior a poluição provocada pelo erro e mais cara e demorada fica a depuração
  - **porém não eliminam** a possibilidade de defeitos passarem despercebidos
    - testes serão sempre necessários

Laboratório de Engenharia de Software

## Técnicas de revisar ou inspecionar




- **Leitura simples**
  - rever antes de prosseguir
    - habitue-se a **fazer isso sempre**
  - procure sempre ser rigoroso ao rever
    - **ler para encontrar defeitos**
- **Leitura dirigida segundo critérios estabelecidos**
  - padrões de uso das linguagens de representação
  - manual de critérios
  - pode envolver técnicas formais
    - argumentação da correitude
- **Leitura baseada em pontos de vista**
  - segundo papéis desempenhados ou simulados pelo revisor

• Correção de trabalhos é um exemplo de revisão?

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

Laboratório de Engenharia de Software

## Check list do ponto de vista do usuário




- Marque todos os elementos cuja especificação não deixe clara a sua intenção
- Marque todos os elementos com **especificação ambígua**
  - diferentes leitores podem entender coisas diferentes
- Marque todos os elementos com **especificação imprecisa**
  - ex. “deve ter bom desempenho”
- Marque todas as **redundâncias de especificação**
  - diferentes elementos especificam a mesma coisa
- Marque todos os elementos que possam ser excluídos sem comprometer as necessidades e expectativas do serviço
- Liste todos os elementos relevantes e que não aparecem na especificação
- . . .

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

Laboratório de Engenharia de Software

## Modos de revisar




- **Revisão pelo próprio autor** (*desktop checking*)
  - o autor lê e anota todos os problemas encontrados para, **depois**, removê-los
  - inconvenientes intrínsecos:
    - o autor tende a “ler o que acha que está escrito e não o que está de fato escrito”
    - erros de entendimento por parte do autor não são observáveis
      - o autor dirá que está correto segundo o que entendeu de forma incorreta
    - sujeito à síndrome da “ideia fixa” → insistir no erro
- O inconveniente pode ser atenuado utilizando **técnicas formais** (leves) para dirigir a leitura
  - a técnica formal induz uma **forma alternativa** (redundância útil) a ser usada para observar o mesmo artefato

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

Laboratório de Engenharia de Software

## Modos de revisar




- **Desenvolvimento em pares** (duplas)
  - *pair programming* proposto por XP – eXtreme Programming
  - o desenvolvimento de um artefato é realizado sempre por **duas pessoas** trabalhando em conjunto à frente de **um mesmo computador**
    - um digita, i.e. escreve o código
    - outro monitora o que está sendo escrito
      - observa erros de digitação
      - observa erros de uso dos elementos do programa
      - observa desvios com relação à especificação ou ao projeto
      - observa desatenção com relação a boas práticas, padrões e normas
      - propõe soluções alternativas melhores
    - tem sido utilizado com sucesso ao desenvolver código
  - trabalho em grupo é uma forma de desenvolvimento em “pares”?
    - em geral utilizado para especificar e arquitetar sistemas

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

Laboratório de Engenharia de Software

## Modos de revisar




- **Revisão por parceiros** (*peer review*)
  - um dos parceiros (colega) do autor **lê e anota** todos os defeitos e demais problemas encontrados
    - defeitos e outros problemas devem ser anotados em um documento (formulário)
      - de preferência à parte → laudo explícito serve como controle do processo de garantia da qualidade
    - pequenas melhorias podem ser anotadas no próprio artefato
      - seria bom se existisse editor capaz de registrar alterações e anotações
        - » exemplo: registro de alterações em Word
    - ninguém deve ficar ofendido com as anotações, o que está em jogo é a qualidade do artefato!
    - ninguém deve deixar de anotar defeitos em virtude de algum receio de melindrar o autor
      - o que não impede de ser bem educado

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
23

Laboratório de Engenharia de Software

## Críticas às revisões




- Prós
  - simplicidade
  - eficácia
    - apesar de informais, revisões tendem a encontrar um número significativo de defeitos
    - se feitas por pessoas treinadas em aspectos formais (argumentação da correção), tornam-se menos suscetíveis a fatores humanos
  - eficiência
    - em uma única revisão identifica-se uma quantidade grande de defeitos
    - cada execução de um teste identifica um ou poucos defeitos
  - baixo custo
    - o custo da revisão é amplamente compensado pela redução do retrabalho inútil
    - muitas coisas podem ser automatizadas
      - **análise estática**: *lint*, problemas potenciais, defeitos, inconsistências

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
24

Laboratório de Engenharia de Software

## Críticas às revisões

- **Contras**
  - a eficácia da revisão depende excessivamente da habilidade dos revisores
    - proficiência
    - cultura
  - a eficácia depende do rigor adotado pelo revisor
    - a falta de treinamento dos revisores amplifica os problemas decorrentes da informalidade
  - frequentemente revisões não são sensitivas nem consistentes
    - diversas classes de defeitos não são observadas
    - revisor muda de opinião
    - diferentes revisores têm diferentes opiniões




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

Laboratório de Engenharia de Software

## Inspeções, plano


- **Plano de inspeção, etapas**
  - **Produzir sinopse** (resumo)
    - autor redige uma descrição resumida do artefato a ser inspecionado
  - **Organizar a reunião de inspeção**
    - o autor seleciona a equipe de inspeção
    - o autor marca a data e horário da reunião de inspeção
  - **Realizar a leitura individual pelos inspetores**
    - o autor distribui aos revisores o material a inspecionar e o de apoio
    - cada um dos revisores (pares) lê e anota os problemas observados
  - **Coletar e filtrar as observações em reunião de inspeção**
    - todos que concordaram devem comparecer e ser **pontuais**
    - resulta no **laudo da inspeção**
  - **Corrigir segundo o laudo**
    - o autor faz as correções segundo o laudo
  - **Acompanhar o progresso**
    - são registrados os eventos: distribuição, reunião, conclusão
    - é acompanhada a resolução dos problemas
    - se solicitada, é realizada nova inspeção após a correção



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

Laboratório de Engenharia de Software

## Inspeções, execução




- Antes da reunião
  - o autor
    - produz uma sinopse (resumo) do artefato a ser inspecionado
      - contexto
      - especificação
      - esboço da solução
    - seleciona a equipe de inspeção
    - marca data, hora e local da inspeção em concordância com todos os membros da equipe
  - os revisores
    - concordam revisar
    - concordam com a data e hora da reunião
    - leem individualmente o artefato a ser inspecionado e documentos correlatos
    - anotam dúvidas, problemas encontrados e sugestões de melhoria

Mar 2017Arndt von Staa © LES/DI/PUC-Rio27

Laboratório de Engenharia de Software

## Inspeções, execução



- Durante a reunião
  - o autor narra o comportamento do artefato
    - *walkthrough*
  - os revisores, que já devem ter lido a documentação fornecida
    - solicitam explicações
    - identificam e avaliam riscos
    - indicam a existência de problemas
      - conforme as anotações deles
      - ou os identificados durante a apresentação
    - sugerem melhorias
  - é produzido um laudo
    - com todos os problemas e sugestões observados
    - com uma avaliação final, aprovado, correções simples, refazer a inspeção

Mar 2017Arndt von Staa © LES/DI/PUC-Rio28

- Após a reunião
  - autor faz as correções
  - possivelmente modifica coisas que não haviam sido anotadas
  - dependendo da gravidade das observações
    - repete a inspeção
    - libera o artefato para a equipe

## Apêndice

## Modos de revisar



- **Revisões progressivas** (*round-robin*)
  - seleciona-se um conjunto de parceiros que farão a revisão
    - cada um com uma determinada especialidade
    - cada um examinará a partir de um ponto de vista específico, exemplos de pontos de vista:
      - como se pode testar isso?
      - está de acordo com os padrões de projeto e os de programação?
      - está de acordo com os padrões das bases de dados da organização?
      - qual é o esforço para implementar isso?
      - o usuário precisa e entende isso?
      - ...
  - cada parceiro lê e anota todos os problemas encontrados dentro de sua especialidade
  - a seguir passa a diante para o próximo parceiro da lista

## Revisões e inspeções são eficazes




- Parcela significativa dos defeitos existentes em um artefato é encontrada através de revisões ou inspeções
  - segundo vários autores a inspeção, quando for praticada, encontra de **60% a 80% do total** dos defeitos encontrados
    - não é de se esperar?
      - se o controle é feito antes dos testes, então sobram menos defeitos a serem encontrados através dos testes
      - também sobram menos defeitos **remanescentes** entregues ao usuário
  - alguns autores mencionam que se **economiza perto de 40% do custo total** de desenvolvimento quando se praticam inspeções
    - não é de se esperar?
    - afinal a inspeção reduz o custo decorrente do retrabalho inútil



Laboratório de Engenharia de Software

## Revisões e inspeções: aplicam-se a o que?



- Revisões e inspeções podem ser utilizadas para controlar a qualidade de **qualquer artefato** em **qualquer linguagem de representação** destinada a humanos
  - especificações
  - modelos
  - projetos
  - código
  - scripts de teste
  - processos de desenvolvimento definidos
  - planos
  - padrões
  - auxílio (help) para o usuário
  - documentação para o usuário
  - teses, dissertações, trabalhos de fim de curso
  - . . .


Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

33

Laboratório de Engenharia de Software

## Revisões e inspeções são eficientes



- Revisões e inspeções informam **diretamente** o defeito
  - em geral o custo é baixo para **localizar completamente** o defeito
  - podem ser realizadas com relação a artefatos não executáveis ou ainda incompletos


Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

34

Laboratório de Engenharia de Software

## Revisões e inspeções vs. testes




- Contrastando, testes informam o sintoma (falha) obrigando diagnosticar a causa (defeito)
  - custo alto para **identificar a causa exata** e **localizar correta e completamente** o defeito
  - testes somente podem ser realizados com relação a artefatos executáveis
  - teste ocorre tarde no desenvolvimento
    - depois de já terem sido comprometidos muitos recursos

Mar 2017Arndt von Staa © LES/DI/PUC-Rio35

Laboratório de Engenharia de Software

## Inspeções




- Uma inspeção é uma **leitura crítica** do artefato e da documentação complementar
  - segundo um plano definido
  - obedecendo a regras de leitura definidas
  - envolvendo uma pequena equipe
- Inspeções diferenciam-se das revisões por serem mais formalizadas e serem mais eficazes
  - são também (bem?) mais caras

Mar 2017Arndt von Staa © LES/DI/PUC-Rio36

Laboratório de Engenharia de Software

## Inspeções, equipe




- Papéis desempenhados
  - **autor**
    - tem treinamento nos critérios a serem utilizados na avaliação
      - critérios visam aproximar-se do desenvolvimento correto por construção
    - desenvolve o artefato
      - idealmente segundo um processo definido e visando os critérios
    - escolhe a equipe de inspeção específica para o artefato e marca uma data para a reunião de inspeção
    - distribui com suficiente antecedência aos membros do comitê o material relativo ao artefato e as informações complementares
  - **inspetores**, dois ou três
    - têm treinamento nas técnicas e critérios utilizados na avaliação
    - recebem com antecedência o material a ser inspecionado
    - lêem e anotam
      - os defeitos encontrados
      - dúvidas e dificuldades de compreensão
      - possibilidades de melhorias

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

Laboratório de Engenharia de Software

## Inspeções, equipe




- Papéis desempenhados, cont.
  - **moderador**
    - verifica se estão satisfeitas as condições para realizar a reunião
    - conduz a reunião
    - procura manter o foco da discussão
    - produz um laudo com as observações (se não tiver secretário)
    - deve ser um desenvolvedor sênior, mas não gerente
  - opcionalmente um **secretário**
    - produz um laudo (relatório) com as observações
  - opcionalmente um **controlador da qualidade**
    - verifica se os quesitos de qualidade foram devidamente abordados

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

Laboratório de Engenharia de Software

## Inspeções, equipe




- Observações
  - O objetivo de uma inspeção é **examinar a qualidade do artefato**
  - Inspeções não se destinam a participantes mostrarem que sabem muito
  - Inspeções **jamaís devem ser usadas para avaliar os autores** dos artefatos inspecionados
    - corolário: gerentes não deveriam participar das reuniões de inspeção

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

Laboratório de Engenharia de Software

## Pontos de vista, exemplos




- Do ponto de vista do **testador**
  - para cada requisito discriminado na especificação: quais seriam os casos de teste que você utilizaria para testar o requisito?
  - está claro como determinar o resultado esperado?
- Do ponto de vista do **especificador**
  - existem potenciais conflitos entre os requisitos?
  - faltam requisitos, o serviço está completamente descrito?
  - além dos requisitos funcionais foram considerados os não funcionais relevantes para a aplicação?
- Do ponto de vista do **implementador**
  - como estimar o esforço para implementar cada requisito?
  - está claro o que é desejado?
  - os casos de uso têm dimensão moderada?
    - requerem no máximo 3 dias para serem implementados

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

Laboratório de Engenharia de Software

## Pontos de vista, exemplos




- Do ponto de vista do **usuário**
  - preciso realmente deste requisito?
  - falta algum requisito (funcional ou não) de que necessito?
  - entendi o que o requisito se propõe a fazer?
  - está de acordo com o que espero do serviço?
  - está de acordo com a legislação?
  - o tratamento de erros de uso faz sentido?
- Do ponto de vista da **prevenção de riscos**
  - o artefato oferece riscos (*design bad smells*) de levar a código contendo defeitos, deficiências ou vulnerabilidades?
  - o **sistema** pode causar elevados impactos negativos (prejuízos) caso venha a falhar ou ser utilizado de forma inesperada?

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

Laboratório de Engenharia de Software

## Propriedades de artefatos **anteriores**



- Uma especificação, arquitetura, ou projeto deve:
  - deixar claro o que deve ser implementado
  - apoiar o controle da qualidade
    - do próprio artefato, ex.
      - sintaxe
      - semântica
      - padrões de uso
    - do artefato consequente, ex.
      - rastreamento da transformação
    - do resultado da implementação, ex.
      - através de suítes de teste derivadas a partir do artefato antecedente

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

Laboratório de Engenharia de Software

## Leitura dirigida

- **Leitura baseada em pontos de vista** encontram mais defeitos do que leitura não dirigida
  - leitura baseada em **cenários**
    - como se comporta o sistema nas condições A, B, C, ... ?
  - leitura baseada no **papel** desempenhado pelo observador
    - ponto de vista do testador
      - como vou testar isso?
    - ponto de vista do mantenedor
      - é fácil manter isso?
      - as características (*features*) estão bem delimitadas?
    - ...

LES

Boehm, B.W.; Basili, V.R.; "Software Defect Reduction Top 10 List"; *IEEE Computer* 34(1); 2001; pags 135-137 – dizem que leitura dirigida captura 35% mais defeitos do que leitura normal

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

Laboratório de Engenharia de Software

## Principais tipos de defeitos em **projetos**


Defeito	Descrição	Aplicado a projeto
Omissão	falta informação necessária no artefato	um ou mais requisitos ou características não são abordados no projeto
Incorreção	alguma informação contida no artefato conflita com o domínio do problema, ou com o serviço a ser prestado	o projeto contém uma reificação errônea de um requisito
Inconsistência	alguma informação contida no artefato contradiz o que se encontra em outros artefatos ou mesmo neste artefato	um conceito registrado no projeto está em desacordo com o que se encontra em outros artefatos

**Reificação:** No processo de alienação, o momento em que a característica de ser uma "coisa" se torna típica da realidade objetiva. [Aurélio eletrônico] → determina como resolver um requisito ou uma abstração

LES

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

## Principais tipos de defeitos em projeto



Laboratório de Engenharia de Software


Defeito	Descrição	Aplicado a projeto
Ambiguidade	Informação contida no artefato admite uma variedade de interpretações	Um conceito contido no projeto favorece dúvidas quanto ao seu significado, possibilitando um erro de compreensão
Desnecessário	Informação contida no artefato não é utilizada, ou pertence a outro nível de abstração	O projeto contém informação que, mesmo se relevante, não deveria se encontrar nesse artefato
Duplicata	Um mesmo conceito é definido ou especificado em dois ou mais artefatos ou em diferentes locais de um mesmo artefato	O projeto especifica de novo um conceito já existente em um ou mais outros artefatos <b>risco: inconsistência</b>

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

45

## Referências bibliográficas



Laboratório de Engenharia de Software

- Cockburn, A.; Williams, L.; "The costs and benefits of pair programming"; in Succi, G.; Marchesi, M.; eds.; *Extreme Programming Examined*; Boston, MA: Addison-Wesley Longman; ISBN:0-201-71040-4; 2001; pp 223-243
- Denger, C.; Shull, F.; "A Practical Approach for Quality-Driven Inspections"; *IEEE Software* 24(2); Los Alamitos, CA: IEEE Computer Society; 2007; pages 79-86
- Laitenberger, O.; "A Survey of Software Inspection Technologies"; in Chang, S.K. ed.; *Handbook on Software Engineering, vol. 2*; 2002; pages 517-556
- Pezzè, M.; Young, M.; *Teste e Análise de Software*; 2008;
- Shull, F.; Rus, I.; Basili, V.R.; "How Perspective-Based Reading Can Improve Requirements Inspections"; *IEEE Computer* 33(7); 2000; pages 73-79
- Staa, A.v.; *Programação Modular*; Campus; 2000;
- Travassos, G.H.; Shull, F.; Carver, J.; Basili, V.R.; *Reading Techniques for OO Design Inspections*; PESC-COPPE; Technical Report 575/02, Rio de Janeiro: COPPE/UFRJ; 2002
- Younessi, H.; *Object-Oriented Defect Management of Software*; Upper Saddle River, NJ: Prentice Hall; 2002

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

46

Laboratório de Engenharia de Software

LES

FIM

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

47