


Laboratório de Engenharia de Software

Revisões e Inspeções 2

Arndt von Staa
Departamento de Informática
PUC-Rio
Março 2015

Especificação



Laboratório de Engenharia de Software

- Objetivo da aula
 - apresentar uma variedade de critérios de avaliação da qualidade vinculados às práticas de inspeção e revisão
- Justificativa
 - a inspeção já nos primeiros artefatos do desenvolvimento reduz o custo do desenvolvimento.
 - quando feito de uma forma sistemática e segundo critérios conhecidos essa redução é mais significativa
- Pezzè, M.; Young, M.; *Teste e Análise de Software*; 2008; capítulo 19
- Staa, A.v.; *Programação Modular*; Campus; 2000; capítulos 3, 10 e 12

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

Técnica de revisão: medição estática



- **Medições estáticas**

- **medem propriedades** do código ou de outros artefatos e indicam os fragmentos que têm chance de gerar problemas (*bad smells*, odores)
 - estilo de programação que tende a induzir defeitos (*fault proneness*)
 - estilo de programação difícil de manter
 - estilo de programação difícil de testar
- propriedades cujo **risco** foi avaliado através de experimentos
 - acoplamento
 - coesão
 - encapsulamento
 - complexidade ciclomática (McCabe) – número de decisões no fluxograma
 - *fan in*
 - *fan out*
 - ...

Mar 2015

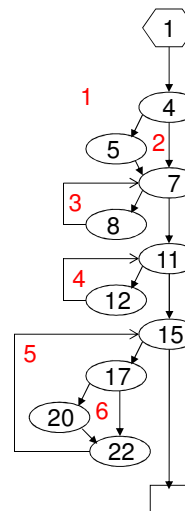
Arndt von Staa © LES/DI/PUC-Rio

3

Exemplo: complexidade ciclomática (McCabe)



```
2  pValor = ( char * ) pValorParm ;
3  numBytes = TamValor - Offset ;
4  if ( numBytes > DIM_LINHA ) {
5      numBytes = DIM_LINHA ;
6  } /* if */
7  for( i = 0 ; i < numBytes ; i ++ ) {
8      ValChar = *( pValor + Offset + i ) ;
9      fprintf( pArqLog , " %02X" , ValChar ) ;
10 } /* for */
11 for( ; i < DIM_LINHA ; i ++ ) {
12     fprintf( pArqLog , "   " ) ;
13 } /* for */
14 fprintf( pArqLog , "   " ) ;
15 for( i = 0 ; i < numBytes ; i ++ ) {
16     Ch = *( pValor + Offset + i ) ;
17     if ( ( Ch < 32 )
18         || ( Ch == 127 )
19         || ( Ch == 255 ) ) {
20         Ch = '.' ;
21     } /* if */
22     fprintf( pArqLog , "%c" , Ch ) ;
23 } /* for */
```



imprime, em hexadecimal e ASCII, até DIM_LINHA caracteres a partir do *offset* de um string

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

4

Exemplo: complexidade ciclomática



- Como reduzir?
 - criar várias funções:

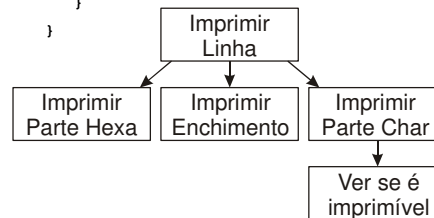
```
ImprimirLinha( linha , comp , DIM )
ImprimirParteHexa( linha , numCh )
ImprimirEnchimento( numCh , DIM )
ImprimirParteChar( linha , numCh )
bool EhImprimivel( ch )
```

```
ImprimirLinha( comp , linha , DIM ) {
    IniciarLinha( ) ;
    if ( comp <= 0 ) return ;
    numCh = comp ;
    if ( numCh > DIM ) numCh = DIM ;
    ImprimirParteHexa( linha )
    ImprimirEnchimento( DIM )
    ImprimirParteChar( linha )
}

ImprimirParteChar( linha ) {
    for ( int i = 0 ; i < numCh ; i++ ) {
        if ( EhImprimivel( linha[ i ] ) ) {
            ImprimirCh( linha[ i ] ) ;
        } else {
            ImprimirCh( '.' )
        }
    }
}
```

incompleto

- O código ao lado ficou mesmo mais legível?
- Mais manutenível?
- E o diagrama ao lado?
- Substituir as chamadas por pseudo-instruções funciona?



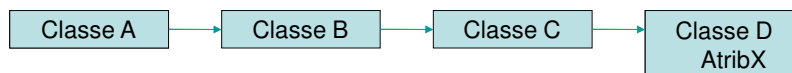
Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

5

Lei de Demeter

Deusa grega da agricultura (latim: Ceres)



- A métrica *número de classes requeridas* deve ser pequena
 - o que é pequeno?
 - devem ser evitados longas sequências de referências:


```
ObjA.metodoX( ){ return ObjB->ObjC->ObjD->atribX ; }
```
- Reorganização (refatoração) proposta:
 - procure realizar a operação na classe que define o atributo
 - substitua atributos públicos por métodos (*getters*) :
 - Na Classe A `metodoAX(){ return ObjB->metodoBX() ; }`
 - Na Classe B `metodoBX(){ return ObjC->metodoCX() ; }`
 - Na Classe C `metodoCX(){ return ObjD->metodoDX() ; }`
 - Na Classe D `metodoDX(){ return AtribX ; }`

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

6

Reorganização (*refactoring*)



- Diversos problemas existentes no código podem tornar mais custosa a manutenção
 - esses problemas são conhecidos por *bad smells*, odores ou anomalias de código
 - algumas dessas anomalias podem ser removidos por ferramentas de reorganização, ou *refactoring*
 - Eclipse fornece várias ferramentas de reorganização para Java
 - desenvolvimento ágil preconiza a remoção de odores após ter completado o teste automatizado

Fowler, M.; *Refactoring: Improving the Design of Existing Code*; Reading, Massachusetts: Addison-Wesley; 2000
Sjøberg, D.I.K.; Yamashita, A.; Anda, B.C.D.; Mockus, A.; Dybå, T.; "Quantifying the Effect of Code Smells on Maintenance Effort"; *IEEE Transactions on Software Engineering* 39(8); Los Alamitos, CA: IEEE Computer Society; 2013; pags 1144-1156

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

7

Técnica de revisão: análise estática



- **Análise estática**
 - examinam propriedades a partir da análise do código.
Exemplos
 - a partir do **grafo de chamadas** envolvendo o programa
 - que função ou método chama que outra função ou método?
 - que **catch** trata de um dado **throw**?
 - reorganização de componentes
 - existem arestas de corte no grafo que permitam separar componentes?
 - controle de **tipos semânticos**
 - ex. velocidade em m/s

tipo semântico – informa o significado e a dimensão do dado, não somente a codificação

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

8

Análise estática exemplo 1 / 3



- Calcular o tempo para o encontro de duas pessoas andando em direções contrárias ao redor de uma lagoa
 - velocidades: vp1 e vp2
 - perímetro: d
 - código proposto

```
double f( double d, double vp1, double vp2 )  
{  
    return d * ( vp1 / vp2 ) ;  
}
```

o código está
semanticamente errado
porém sintaticamente correto

- quais são os defeitos?
- como descobrir os defeitos através de análise estática?

A linguagem F# possui sintaxe para esse tipo de controle: ver `units of measure`

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

9

Análise estática exemplo 2 / 3



```
semantic_types double veloc , double dist , double time , double number ;  
rules  
{  
    veloc  = dist / time ;  
    number = veloc / veloc ;  
    time   = dist / veloc ;  
    dist   = dist * number ;  
} ;
```

o código ainda é vulnerável ao
uso de *dimensões* incompatíveis

```
time f( dist d, veloc vp1, veloc vp2 )  
{  
    return d * ( vp1 / vp2 ) ;  
}
```

este código viola as regras definidas.
Retorna distância e não tempo.

- Seria possível escrever um programa que acusa a existência de defeitos no código acima?
- E as vulnerabilidades de associação de dimensões erradas?
 - ex. passar km/h para um parâmetro esperando por m/s

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

10

Análise estática exemplo 3 / 3



```

dimensions vms , vkmh , km , m , s , h ;
rules
{
    vms = m / s ;
    vkmh = km / h ;
    vms = 0,27777778 * vkmh    constante de conversão km/h para m/s
} ;
semantic_types double veloc , double dist , double time , double number ;
operations
{
    veloc = dist / time ;
    veloc = veloc + veloc ;
    number = veloc / veloc ;
    time = dist / veloc ;
    dist = dist * number ;
} ;

time:s f( dist:m d, veloc:vms vp1, veloc:vkmh vp2 )
{
    return d / ( vp1 + vp2 ) ;
}

```

- Seria possível escrever um programa que acusa a existência de defeitos no código acima?

Mar 2015

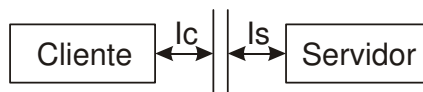
Arndt von Staa © LES/DI/PUC-Rio

11

Análise estática de interfaces

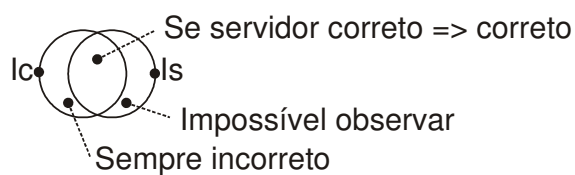


- Mesmo se não tiver uma ferramenta de controle de tipos
 - sempre especifique a semântica de todos os parâmetros
 - isso facilita muito a inspeção visual do uso das interfaces
 - uso incorreto de interfaces é uma das maiores causas de defeitos



lc - interface do ponto de vista do cliente

ls - interface do ponto de vista do servidor



Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

12

Análise estática de duplicação de código



- Copiar , colar e alterar código é nocivo
 - duplica código, agora existem dois para manter
 - se o código copiado contiver um defeito, as cópias também conterão
 - aumenta desnecessariamente o esforço de manutenção
 - duplicações também podem acontecer quando dois programadores se depararem com problemas similares
- Procure ferramentas de análise do código à busca de duplicações
 - elimine as duplicações encontradas
 - crie um método com parâmetros que implementa as diversas variantes do código base

Uso de análise estática



- Procure ferramentas de análise estática, exemplos
 - a primeira delas: *lint* para UNIX
 - findbugs (java)
 - cpplint (google c++)
 - existem inúmeras outras para as mais variadas plataformas

Técnica de revisão: análise dinâmica



- **Análise dinâmica** (incluído aqui para confronto com análise estática)
 - apoia revisões **após** o desenvolvimento
 - verifica propriedades relevantes durante a execução
 - é mais para teste do que para revisão ou inspeção
 - exemplos
 - satisfação de condições
 - assertivas executáveis, verificadores estruturais
 - controle dinâmico de tipos
 - controle vazamento de recursos
 - avaliação do desempenho
 - algumas ferramentas
 - **valgrind** para C++ / UNIX; **Insure++**
 - ferramentas de medição de desempenho
 - ...

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

15

Técnica de revisão: uso de padrões



- **Estude e use** padrões e técnicas de revisão ou inspeção
 - melhora a eficácia das revisões e inspeções reduz a frequência de enganos → reduz a inserção de defeitos
 - mais próximo do ideal correto por construção
 - **com ganho** de produtividade
 - **redução de custos** em virtude da redução do retrabalho inútil
 - treinamento deve ser adequado às características da organização e do projeto
 - os critérios usados devem **amoldar-se** à natureza do problema, à cultura da organização e à qualidade requerida
- Precisa mesmo?
 - O treinamento em padrões de projeto e de programação **é requerido como parte da formação** de bons programadores

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

16

Laboratório de Engenharia de Software

Técnica de revisão: técnicas formais leves




- Utilize **técnicas formais leves**
 - também faz parte da formação de bons programadores
 - **assertivas** e **argumentação** da corretude
 - especificar, projetar e programar utilizando técnicas e modelos **suficientemente formais**
 - procure e utilize ferramentas
 - **verificadores de modelos** (*model checkers*)
 - **transformadores** que convertem modelos em outros artefatos de nível de abstração mais baixo, ou então redigidos em outra linguagem de representação
 - » a grande maioria dos transformadores converte para **esqueletos de código** que precisam ser preenchidos pelo desenvolvedor
 - » a geração de esqueletos dificulta a manutenção
 - **geradores** que convertem modelos em código compilável

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

Laboratório de Engenharia de Software

Critérios de qualidade para casos de uso




- Casos de uso estabelecem um **diálogo** entre **usuário** e o **sistema** a ser desenvolvido (ou existente)
 - possivelmente envolvem vários interessados
 - devem conter ações relacionadas com todos os interessados
- Casos de uso devem evitar detalhes de implementação
 - deve-se evitar descrever a **interface física** com o usuário
- Deve-se evitar ações excessivamente detalhadas
 - ex. Usuário fornece largura do produto (fornece as dimensões)
- Deve-se evitar ações excessivamente vagas
 - ex. Usuário fornece dados
- As ações devem ser coerentes com o objetivo do caso de uso
 - o objetivo do caso de uso deve estar refletido no título

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

Laboratório de Engenharia de Software

Critérios de qualidade para casos de uso




- Casos de uso devem ser redigidos para que **leitores leigos** em computação possam ler e entender
 - evite jargão “computês”
 - procure manter-se no domínio do problema e não focar a implementação
- Casos de uso devem ser redigidos com sintaxe e ortografia correta
 - evite palavras complicadas ou fora de uso
 - evite adjetivos e advérbios que pouco ou nada contribuam para o entendimento
 - use uma linguagem polida (bem educada)
- Casos de uso podem levar ou a sucesso ou a falha,
 - o texto deve deixar clara qual das duas terminações ocorrem
 - e o que fazer quando ocorrem erros de uso

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

Laboratório de Engenharia de Software

Critérios de qualidade para casos de uso




Caso de uso	<ul style="list-style-type: none"> • O nome do caso de uso descreve o objetivo a alcançar? ex. Efetu<u>ar</u> login no sistema específico
Resumo	<ul style="list-style-type: none"> • A descrição é resumida? • Explica bem o objetivo principal?
Escopo	<ul style="list-style-type: none"> • A natureza do caso de uso é “caixa preta”? <small>Obs. um caso de uso deve ser uma característica (<i>feature</i>, funcionalidade)</small> • O escopo contém tudo o que o caso de uso abordará? • O escopo contém coisas que o caso de uso não abordará?
Ator principal	<ul style="list-style-type: none"> • O ator principal aciona o caso de uso?
Interessados	<ul style="list-style-type: none"> • Cada interessado figura em pelo menos uma ação e/ou regra de negócio? • Todas as interações externas referem a algum interessado? • Os interesses (metas) de cada interessado estão definidas?

Exato:

- necessário – contém **nada** que possa ser eliminado
- suficiente – contém **tudo** o que é necessário

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

Critérios de qualidade para casos de uso




Laboratório de Engenharia de Software

Invariante	<ul style="list-style-type: none"> Foi redigida como um conjunto de condições? Todas as condições são relevantes para o caso de uso? Falta alguma condição relevante para o caso de uso? Todas as condições precisam valer antes e também após qualquer forma de <i>término normal</i> do caso de uso?
Pré condições	<ul style="list-style-type: none"> Foram redigidas como um conjunto de condições? Invariante + pré-condições estabelecem todas as condições necessárias para poder iniciar o caso de uso? <ul style="list-style-type: none"> falta alguma condição necessária para iniciar o caso de uso? Todas as pré-condições são diferentes das condições invariantes?
Acionamento	<ul style="list-style-type: none"> O evento que aciona o caso de uso é disparado pelo ator principal? Se não for, o caso de uso é parte de uma máquina de estado?

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
21

Critérios de qualidade para casos de uso




Laboratório de Engenharia de Software

Fluxo principal (cenário de sucesso)	<ul style="list-style-type: none"> Possui entre 3 e 9 ações? Podem ocorrer casos que violem essa regra As ações estão redigidas segundo o padrão de redação? As ações do fluxo principal correspondem ao objetivo principal e não consideram anomalias de execução? Basta a satisfação da invariante e das pré-condições para poder disparar o fluxo principal? O fluxo principal pode ser disparado pelo ator principal? O fluxo principal resulta em algo que interesse ao ator principal? O fluxo principal é viável? O fluxo principal é capaz de assegurar a invariante ao terminar? O fluxo principal é capaz de assegurar a pós-condição ao terminar? <p style="color: #00bcd4; margin-top: 5px;">Obs. podem ser usados os critérios de qualidade de especificações.</p>
--	---

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
22

Critérios de qualidade para casos de uso		LES
Laboratório de Engenharia de Software	<p>Fluxos alternativos (cenário alternativo)</p> <ul style="list-style-type: none"> • Cada fluxo alternativo é disparado por um evento ou condição claramente definida? • Cada fluxo alternativo ou inicia em uma ação definida, ou é genérico? • A condição de disparo é não ambígua com relação aos demais fluxos alternativos? • A condição de disparo do fluxo alternativo pode ocorrer na prática? • Todas as condições anormais (exceções) foram consideradas? <ul style="list-style-type: none"> – Considere uma a uma as ações do fluxo principal e dos fluxos alternativos. • As ações estão redigidas segundo o padrão de redação? • O fluxo alternativo com término anormal é capaz de assegurar a garantia mínima? • Está claro qual será o resultado da execução do fluxo alternativo? (poderia estar nas pós-condições) <p>Obs. podem ser usados os critérios de qualidade de especificações.</p>	
Mar 2015		Arndt von Staa © LES/DI/PUC-Rio 23


Critérios de qualidade para casos de uso		LES
Laboratório de Engenharia de Software	<p>Pós condições</p> <ul style="list-style-type: none"> • Foram redigidas como um conjunto de condições? • A invariante + pós-condições estabelecem completamente o resultado ao término do fluxo normal? <ul style="list-style-type: none"> • falta alguma condição de saída de interesse para o ator principal ou outros interessados? • Todas as pós-condições são diferentes das condições invariantes? • As pós condições podem ser utilizadas como oráculo de teste? <p>Garantia mínima</p> <ul style="list-style-type: none"> • Foi redigida como um conjunto de condições? • A garantia mínima estabelece completamente o resultado ao término (anormal) de um <i>fluxo alternativo</i>? • Falta alguma condição de garantia mínima? • As condições podem ser utilizadas como oráculo de teste? 	
Mar 2015		Arndt von Staa © LES/DI/PUC-Rio 24

Critérios de qualidade para casos de uso


Laboratório de Engenharia de Software

Requisitos	<ul style="list-style-type: none"> • Cada requisito corresponde a um requisito não funcional? • Foram considerados todos os requisitos não funcionais relevantes? • Todos os requisitos relacionados são relevantes para o caso de uso? • Os requisitos são viáveis? • Todos os requisitos são verificáveis?
Regras de negócio	<ul style="list-style-type: none"> • Todas as regras de negócio explicitamente ou implicitamente requeridas pelas diversas ações estão definidas? • Todas as regras de negócio estão relacionadas a pelo menos uma ação? • As regras de negócio são compatíveis com as condições válidas nas ações a que se relacionam?

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
25


Critérios de qualidade para casos de uso


Laboratório de Engenharia de Software

Casos de uso correlatos	<ul style="list-style-type: none"> • Todos os casos de uso que abordem aspectos ou características (<i>features</i>) correlatas estão referenciados? • Algum caso de uso referenciado não aborda aspectos ou características correlatas?
Artefatos correlatos	<ul style="list-style-type: none"> • Todos os documentos, artigos, mensagens, fontes de informação, etc. que contêm alguma informação relevante para o caso de uso estão relacionados? • Somente estão relacionados os artefatos que contêm alguma informação relevante para o caso de uso?

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
26

CrITÉRIOS da qualidade de especificações




Laboratório de Engenharia de Software

- **Explicitude** (explícito + -(t)ude)
 - todos os itens da especificação estão explicitamente definidos e documentados
- **Necessidade**
 - a especificação contém somente os itens cuja presença possa ser justificada
- **Suficiência**
 - a especificação não omite aspectos relevantes
 - exemplo: a especificação de uma classe relaciona os atributos e métodos públicos e protegidos, e todas as dependências de outras classes
- **Consistência**
 - a especificação não contém contradições internas, ou com outros documentos

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

CrITÉRIOS de qualidade de especificações




Laboratório de Engenharia de Software

- **Ausência de redundância**
 - Redundância ocorre quando um mesmo item é especificado em vários lugares (duplicação, repetição), exemplos
 - descrever a interface com o usuário no cabeçalho do módulo e também na função que implementa esta interface
 - especificar a parte externada tanto no módulo de definição como no módulo de implementação
- **Verificabilidade** (isso é obrigatório)
 - deve ser possível **determinar objetivamente a satisfação** de cada item da especificação
- **Testabilidade** (isso é desejável)
 - é uma forma de verificabilidade em que a satisfação dos itens de especificação é verificada através de casos de teste

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

Critérios de qualidade de especificações




Laboratório de Engenharia de Software

- **Concisão**
 - a especificação é redigida com poucas palavras
 - evite circunlóquios, adjetivos e advérbios que nada acrescentem à especificação
- **Compreensibilidade**
 - a especificação deve ser compreensível pelos diversos leitores
 - leitores são possivelmente leigos em computação
- **Não-ambiguidade** (inequívoco)
 - todos os leitores devem entender o item exatamente da mesma maneira
- **Prioridade**
 - está claro o que é efetivamente requerido – essencial – e o que é apenas desejado

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
29

Critérios de qualidade de especificações



Laboratório de Engenharia de Software

- **Nivelamento**
 - a especificação está no nível de abstração do artefato sendo especificado. Exemplos
 - para a arquitetura do sistema, são relevantes os modelos conceituais das bases de dados utilizadas
 - para um usar um sistema não interessa saber como as bases de dados são organizadas
 - mas é necessário saber que serviços o sistema presta
 - interface **abstrata** com o usuário
- **Viabilidade**
 - cada item da especificação pode ser implementado pela equipe disponível
 - é teoricamente possível implementar com o desempenho desejado
 - computabilidade
 - complexidade teórica
 - análise de algoritmos
 - a equipe tem competência para implementar

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
30

Laboratório de Engenharia de Software

Outros critérios de qualidade de especificações

LES

- Adequação
- Acurácia ou Exatidão
- Precisão
- Desempenho
- Portatilidade
- Proteção
- Segurança contra desastres
 - *safety*
- Segurança contra mau uso
 - *security*
- Recuperabilidade
- Tolerância a falhas
- Não cancelamento
- . . .

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

31

Laboratório de Engenharia de Software

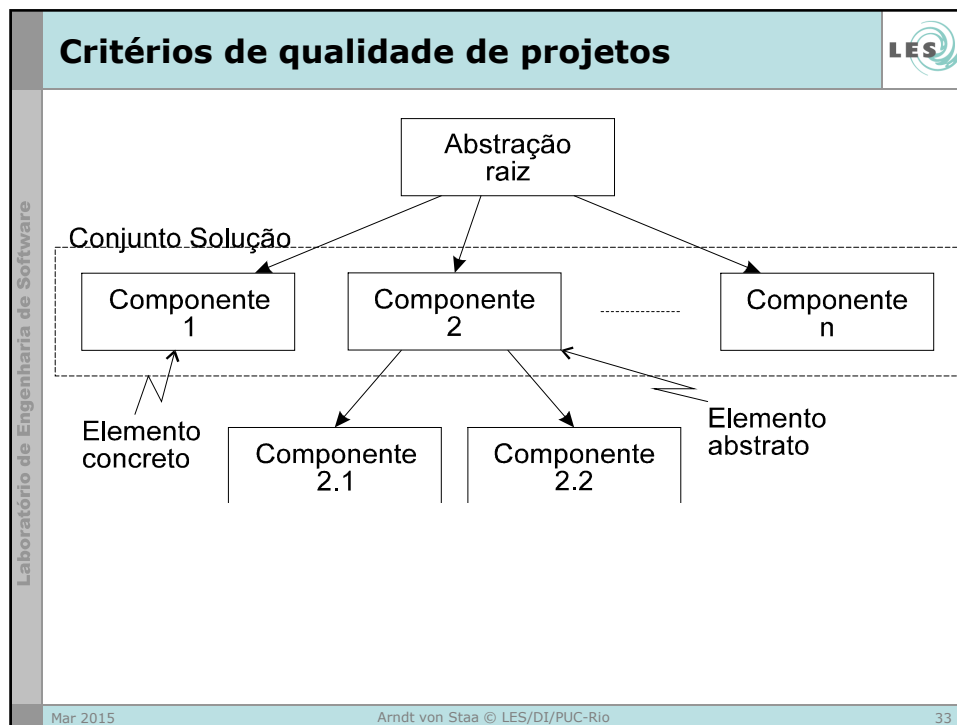
LES

Apêndice


Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

32




Critérios de qualidade de decomposições



Elementos	Itens de interface
<ul style="list-style-type: none"> Definição ou especificação dos requisitos do elemento Adequação Nivelamento Necessidade Suficiência Viabilidade Minimalidade Reutilizabilidade Ortogonalidade Completeza da interface Integrabilidade Flexibilidade 	<ul style="list-style-type: none"> Tipo do item Necessidade do item Suficiência de itens Minimalidade de itens Ortogonalidade de itens Encapsulamento

Mar 2015 Arndt von Staa © LES/DI/PUC-Rio 34

Critérios de qualidade de decomposições



Laboratório de Engenharia de Software


Especificação dos requisitos do elemento

- avaliada segundo os critérios de *Especificação de Requisitos*
- nem todos os critérios precisam ser abordados em cada um dos elementos. Ressaltamos os seguintes:

- **Definição**
 - deve estar claramente definida a intenção do elemento
- **Adequação**
 - compatibilidade entre o serviço a ser prestado pelo elemento e as necessidades (essência / precisa) e expectativas (deseja) do usuário
- **Nivelamento**
 - todos os elementos do conjunto de solução estão no mesmo nível de abstração

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
35

Critérios de qualidade de decomposições




Laboratório de Engenharia de Software

- **Necessidade**
 - cada elemento do conjunto de solução efetivamente **contribui** para a sua solução
- **Suficiência**
 - o conjunto solução **resolve completamente** a abstração raiz
- **Viabilidade**
 - o elemento ou já é uma solução concreta, ou admite uma solução satisfatória

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
36

Critérios de qualidade de decomposições




Laboratório de Engenharia de Software

- **Minimalidade**
 - não existe **subconjunto** do conjunto de solução que por si só represente um conceito bem definido
- **Ortogonalidade**
 - cada elemento resolve uma **parte da abstração raiz** que não é resolvida por qualquer outro elemento
 - ideal: cada parte da abstração raiz é resolvida por exatamente um elemento
- **Reusabilidade**
 - o elemento pode ser utilizado de forma verbatim em uma variedade de contextos
 - quanto mais bem delimitado for o elemento
 - maior a chance de poder ser reutilizado – reuso por acaso
 - mais reutilizável será – reuso por projeto

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
37

Critérios de qualidade de decomposições




Laboratório de Engenharia de Software

- **Completeza de interface**
 - a especificação da interface do elemento está completa e em conformidade com os requisitos do elemento
 - é na realidade um **fator de qualidade** (um conjunto de critérios), a ser visto mais adiante
- **Integrabilidade**
 - os elementos do conjunto de solução **podem ser integrados** sem necessitarem de adaptações
- **Flexibilidade**
 - o elemento pode ser utilizado em diferentes contextos através da seleção de parâmetros existentes na interface
 - através de parâmetros, ou redefinições de métodos pode-se criar elementos genéricos
 - a flexibilidade deve ser a **mínima necessária**

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
38

Critérios de qualidade de decomposições




Laboratório de Engenharia de Software

A interface de um elemento é composta por **itens da interface**

- **Tipo do item**
 - deve estar claramente definida a semântica do item e a sua escala de medição, exemplos:
 - velocidade em m/s
 - símbolo é um string ASCII terminado em zero
- **Necessidade do item**
 - o item da interface é necessário para poder corretamente utilizar o elemento
- **Suficiência de itens**
 - o conjunto de todos os itens da interface permite corretamente utilizar o elemento

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

Critérios de qualidade de decomposições




Laboratório de Engenharia de Software

- **Minimalidade de itens**
 - não existe um subconjunto do conjunto de itens e que represente um conceito bem definido, i.e. um item composto
- **Ortogonalidade de itens**
 - cada item se refere a um requisito do elemento que nenhum outro item considera
- **Encapsulamento**
 - são tornadas visíveis na interface física somente as propriedades de implementação efetivamente necessárias

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

Laboratório de Engenharia de Software

Manual de critérios




- **Crie e mantenha** um *Manual de Critérios de Qualidade* para a sua organização
 - componentes básicos
 - padrões de arquitetura e de projeto
 - *design patterns*
 - padrões de programação
 - defina com precisão as **restrições de uso** das várias linguagens de representação utilizadas
 - crie **padrões técnicos** ajustando as linguagens de representação às necessidades da organização

Organização aqui significa: empresa, divisão de uma empresa, ou grupo dentro de uma empresa que se dedica ao desenvolvimento de sistemas intensivos em software

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
41

Laboratório de Engenharia de Software

Manual de critérios



- Verifique se cada um dos critérios do manual
 - é coerente com os demais critérios
 - corresponde a alguma **coisa relevante**
 - ou **elimina riscos** de defeitos ou vulnerabilidades
 - ou contribui para a **satisfação** de algum requisito não funcional
 - manutenibilidade
 - testabilidade
 - segurança
 - ...
 - evite **achologia** – conduza experimentos, ou procure evidências na Web

achologia – “lógica” baseada em opinião ou intuição e não em fatos. Ex. eu acho que x é verdade

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
42

Manual de critérios



- A publicação e o uso de um *Manual de Critérios de Qualidade* é, por si só, um fator significativo para o aumento da qualidade sem que isto implique em mais custos ou perda de produtividade
 - desenvolver com base em critérios conhecidos reduz o número de defeitos e vulnerabilidades inseridos sem requerer mais esforço [Weinberg, 1998]
 - menos defeitos inseridos reduz o retrabalho inútil
 - retrabalho inútil é esforço perdido, ou seja é produtividade diminuída e custo aumentado
 - menos retrabalho no mínimo mantém produtividade e custo, na prática melhora sensivelmente esses dois fatores

Weinberg, G.M.; *The Psychology of Computer Programming*; 2nd edition; New York: Dorset House; 1998

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

43

Como especificar um critério de avaliação



- **Objetivo**
 - descreve a finalidade do critério
 - para que serve o critério? O que avalia?
- **Porque é relevante**
 - justifica a relevância de se atingir o objetivo
 - por que interessa atingir o objetivo?
- **Artefatos a que se aplica**
 - lista os artefatos que estarão sujeitos à avaliação
- **Elementos utilizados**
 - discrimina os elementos do artefato utilizados durante a avaliação segundo o critério
- **Propriedades envolvidas**
 - discrimina as métricas das propriedades utilizadas no critério


Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

44

Laboratório de Engenharia de Software

Como especificar um critério de avaliação



- **O que é avaliado**
 - identifica as propriedades avaliadas ou medidas
- **Como é avaliado**
 - descreve o processo de avaliação
- **Escalas**
 - descreve como é apresentado o resultado da avaliação
- **Tratamento dos resultados**
 - descreve o que deve ser feito em resposta à avaliação
 - tratamento das não conformidades observadas


Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

45

Laboratório de Engenharia de Software

Exemplo 1 : Completeza da interface 1 / 6



- **Objetivos da avaliação**
 - Assegurar que o artefato produza exatamente (todos e somente) os resultados especificados nos requisitos
 - Assegurar que o artefato receba exatamente os dados necessários para satisfazer todos os seus requisitos
- **Porque é relevante avaliar**
 - A inconsistência, o excesso, ou a falta de completeza dos resultados e dados necessários face aos requisitos de um artefato são causas de erro de projeto, implementação e de uso (chamada) do artefato.
- **A que artefatos se aplica**
 - Às especificações e ao código de programas, componentes, módulos, classes e funções.

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

46

Exemplo 1 : Completeza da interface 2 / 6



- Elementos a que se aplica
 - Requisitos:
 - especificações das condições, atributos, propriedades ou características, funcionais ou não, a serem satisfeitas pelo artefato
 - Dados:
 - todos itens de interface que transferem dados do cliente para o artefato
 - **item de interface**: parâmetro, valor retornado, atributo de objeto, atributo de classe, variável global, arquivo, base de dados, mensagem, tipos e classes definidos pelo usuário usados para especificar dados da interface
 - Resultados:
 - todos itens de interface que transferem do artefato para o cliente

Exemplo 1 : Completeza da interface 3 / 6



- O que é avaliado
 - A correspondência exata entre os objetivos e os requisitos
 - A correspondência exata entre os resultados produzidos pelo artefato e os seus requisitos
 - A correspondência exata entre os dados recebidos e os resultados produzidos

Exemplo 1 : Completeza da interface 4 / 6



- Como é avaliado
 - Para cada requisito é verificado a sua necessidade com relação aos objetivos do artefato
 - Para cada objetivo é verificado se os requisitos garantem a respectiva plena satisfação
 - Para cada resultado a ser produzido, é verificado se existe algum requisito que torne necessário este resultado
 - Para cada requisito, é verificado se existem suficientes resultados para que este requisito possa ser satisfeito
 - Para cada dado, é verificado se existe algum resultado que dependa deste dado
 - Para cada resultado, é verificado se existem suficientes dados para poder produzir este resultado


Exemplo 1 : Completeza da interface 5 / 6



- Escalas (unidades de medida)
 - contagem e registro de todas as não conformidades:
 - lista dos problemas de entendimento da especificação
 - lista de defeitos nos objetivos
 - lista dos requisitos desnecessários
 - lista dos requisitos em falta (insuficientes)
 - lista dos dados usados e não registrados na interface;
 - lista dos dados registrados como recebidos, mas não usados;
 - lista dos dados registrados como requeridos, mas não recebidos;
 - lista dos resultados produzidos e não registrados na interface;
 - lista dos resultados registrados como produzidos, mas não requeridos
 - lista dos resultados registrados como requeridos, mas não produzidos

Laboratório de Engenharia de Software

Exemplo 1 : Completeza da interface 6 / 6




- Tratamento dos resultados
 - a especificação da função deve ser corrigida repetidas vezes até que não existam mais
 - defeitos observados
 - problemas de entendimento
 - sugestões de melhorias
 - após cada correção deve ser **refeita** a revisão
 - a especificação será aceita somente se não existir defeito observado nem problema de entendimento.

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
51

Laboratório de Engenharia de Software

Ex. 2: Uniformidade de terminologia 1 / 5




- Objetivo
 - Eliminar erros de comunicação devidos ao uso de significados diferentes para um mesmo nome e/ou devido à escolha de um nome não **expressivo**
- Porquê é relevante
 - Uma das principais fontes de erros de desenvolvimento e/ou de manutenção é o entendimento incorreto das especificações
- A que artefatos se aplica
 - A todos os artefatos

Expressivo: 1. Que exprime; significativo; [Aurélio]

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
52

Laboratório de Engenharia de Software

Ex. 2: Uniformidade de terminologia 2 / 5



- Elementos utilizados
 - nomes (verbetes)
 - simples
 - compostos
 - significados (acepção)
 - atribuídos aos nomes
 - para os quais deveriam existir nomes
- O que é avaliado
 - Para cada significado existe um único nome
 - Para cada nome existe um único significado
 - O nome induz o leitor a saber qual o significado

expressivo
[De *expresso* + -ivo.]
Adjetivo

1. Que exprime; significativo:
Calou-se, magoado, num silêncio expressivo.


verbetes
etimologia
classe léxica
acepção

exemplo
extraído de [Aurélio]

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
53

Laboratório de Engenharia de Software

Ex. 2: Uniformidade de terminologia 3 / 5



- Como é avaliado
 - existe um **glossário** fornecendo todos os pares <nome, significado> conhecidos
 - todos os problemas têm sua posição marcada no artefato
 - para cada ocorrência de um nome no artefato
 - verifique se o seu significado está explícito. Não estando:
 - registre problema: nome sem significado explícito
 - verifique se é expressivo, não sendo
 - registre problema: falta de expressividade e o porquê dela
 - verifique se o correspondente significado é igual ao definido no glossário. Não sendo:
 - registre problema: o nome com vários significados e quais são eles
 - para cada ocorrência de um significado no artefato
 - verifique se existe um nome no glossário. Não existindo:
 - registre problema: significado sem nome explícito
 - verifique se existem outros nomes no glossário. Existindo:
 - registre problema: significado com vários nomes

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
54

Ex. 2: Uniformidade de terminologia 4 / 5



- Unidades de medida
 - contagem dos nomes sem significados no glossário
 - contagem de nomes não expressivos
 - contagem de significados com vários nomes
 - contagem dos nomes utilizados com significado diferente do registrado no glossário
 - contagens de significados sem nomes explícitos
- Problemas:
 - o que é um significado?
 - como se identifica um?
 1. Quando determinado usuário ativa o editor de folha de tempo, será buscada a folha
 2. de tempo correspondente ao dia atual conforme o relógio do computador. Caso já
 3. existam 1 ou mais tarefas registradas relativas ao dia atual, a folha é inicializada
 4. com esta lista de tarefas. Caso contrário a folha será deixada em branco.

Ex. 2: Uniformidade de terminologia 5 / 5



- Tratamento dos resultados
 - substitua cada nome não expressivo por um que seja
 - escolha use um dos vários nomes de um dado significado
 - para cada discrepância
 - gere um par < nome , significado > temporário
 - para cada par temporário verifique se já existe um nome com significado semelhante ou igual
 - o nome sendo novo e não existindo significado semelhante ou igual, efetive o par temporário no glossário
 - existindo nome com significado igual, todos os usos do novo nome devem ser substituídos de modo a referenciar o nome já existente
 - existindo um nome com significado semelhante, deve ser justificada a necessidade da diferença de significados
 - sendo possível justificar, efetive o novo par no glossário
 - não sendo possível justificar, corrija o artefato de modo a usar o nome e o significado já definidos no glossário.

Laboratório de Engenharia de Software

Referências

LES

- Cockburn, A.; *Escrevendo Casos de Uso Eficazes - Um Guia para Desenvolvedores de Software*; São Paulo, SP: Bookman; 2005
- 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
- Staa, A.v.; *Programação Modular*; Rio de Janeiro: Campus; 2000

Mar 2015Arndt von Staa © LES/DI/PUC-Rio57

Laboratório de Engenharia de Software

FIM

LES

Mar 2015Arndt von Staa © LES/DI/PUC-Rio58