Laboratório de Engenharia de Software

Revisões e Inspeções 2

Arndt von Staa
Departamento de Informática
PUC-Rio
Março 2017

Laboratório de Engenharia de Software

Especificação

- Objetivo da aula
  - apresentar uma variedade de técnicas 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 com base em técnicas comprovadas essa redução é mais significativa, além de reduzir o risco de defeitos remanescentes

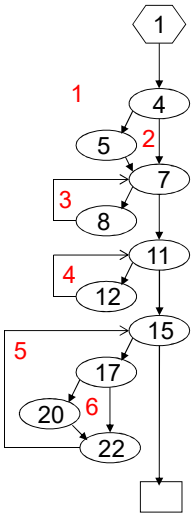
- Pezzè, M.; Young, M.; *Teste e Análise de Software*; 2008; capítulo 19
- Staa, A.v.; *Programação Modular*; Campus; 2000; capítulos 3, 10 e 12

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
    - ex. *bad smells*, anomalias
    - 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
    - *fan in*
    - *fan out*
    - god class
    - god method
    - ...

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

Exemplo: complexidade ciclomática

- Como reduzir a complexidade?
  - criar várias funções:

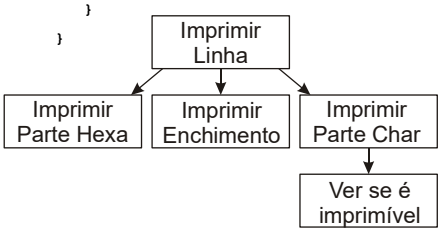
```
ImprimirLinha( linha , comp , DIM )
ImprimirParteHexa( linha , numCh )
ImprimirEnchimento( numCh , DIM )
ImprimirParteChar( linha , numCh )
bool EhImprimível( ch )
```

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

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

incompleto

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



Reorganização (*refactoring*)

- Diversos problemas existentes no código podem tornar mais custosa a manutenção
  - esses problemas são conhecidos por *bad smells*, ou anomalias de código
  - algumas dessas anomalias podem ser removidas por ferramentas de reorganização, ou *refactoring*
  - Eclipse fornece várias ferramentas de reorganização para Java
    - existem muitas outras
  - desenvolvimento ágil preconiza redação sem controle e, depois, a remoção das anomalias, verificando tudo com teste automatizado

Hafiz, M.; Overbey, J.; "Refactoring Myths"; *IEEE Software* 32(6); Los Alamitos, CA: IEEE Computer Society; 2015; pp39-43
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

Medição dinâmica

- Medição dinâmica
  - mede propriedades que dependem da execução:
    - hardware usado
    - plataforma usada (sistema operacional, bases de dados, ...)
    - organização e linguagem(ns) do programa usado
    - dados usados
- Exemplos comuns
  - medir cobertura do teste:
    - medir o número de arestas do fluxograma executadas
    - medir o número de vezes que um método é chamado
    - medir o número de vezes que cada retorno é executado (return, fim do método, exceção, exceção passada a diante)
    - identificar os estados e as transições executadas em uma máquina de estados

Medição dinâmica

- Desempenho teórico de um problema, notação O
- Exemplo: medição de desempenho real
  - suponha que o desempenho teórico de determinado algoritmo seja On^2
    - porém, considerando o algoritmo e dependendo dos dados utilizados pode ser On , On^2 , ou algo no meio.
  - torna-se interessante medir o desempenho médio
  - precisa-se executar o algoritmo com um número significativo de amostras estatisticamente representativas dos dados para determinar uma boa aproximação do desempenho médio

Medição dinâmica

- Medição de tempo é complicada
  - veja: https://en.wikipedia.org/wiki/System_time
  - a precisão do relógio depende do hardware, versão do sistema operacional e linguagem utilizada
  - a data inicial também varia em função do sistema operacional
- Precisão
  - nos primeiros PCs (PC-AT) era em torno de 10 ms
  - sistemas UNIX medem segundos a partir de 1/1/1970
    - existem funções que permitem chegar a ns
  - sistemas Windows de 2000 em diante medem 0,1 us a partir das 12:00 de 1/janeiro/1601 (UTC - Coordinated universal time)
  - sistemas MAC medem em ms a partir de 1/janeiro/2001

Medição dinâmica

- Medição de tempo é complicada
- Como medir tempo depende do hardware, sistema operacional, linguagem, multi-programação e estilo de programação:
  - o desempenho medido vale somente nas condições em que foi medido
  - possivelmente somente nas condições específicas da rodada de medição, devido à multi-programação
  - não pode ser generalizado
- Para poder generalizar - dentro de limitações - ao invés de medir tempo, pode ser melhor medir o número de passagens por determinado ponto no algoritmo

Medição dinâmica

- Por que medir tempo real é importante
  - ações de tempo real: transmissão de voz e filmes
  - os pacotes precisam ser codificados, chegar ao destinatário e ser decodificados dentro do intervalo disponível
  - ações que precisam ser realizadas dentro de um tempo de resposta limitado
    - por exemplo, sistemas que dependem de uma amostragem realizada em intervalos regulares - ex. controle de sistemas elétricos, aparelhos auditivos
  - curiosidades:
    - Velocidade da luz no vácuo aproximadamente: 300.000.000 m/s, em fibra ótica aproximadamente 200.000.000 m/s
      - 1 nano segundo é 10^-9 s, distância percorrida no vácuo: 30cm

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`es tratam de um dado `throw`?
    - reorganização de componentes
      - existem arestas de corte no grafo de chamadas que permitam separar componentes?
    - controle de tipos semânticos
      - ex. velocidade em m/s, nome de pessoa em 20 chars
  - existem várias ferramentas que fazem análise estática, ver Wikipedia "*List of tools for static code analysis*"

aresta de corte – é um conjunto pequeno (<= 3?) de um grafo conexo que, quando retiradas transforma o grafo em duas partes desconexas

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

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`**

Análise estática exemplo 2 / 3


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

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

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

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

Análise estática exemplo 3 / 3

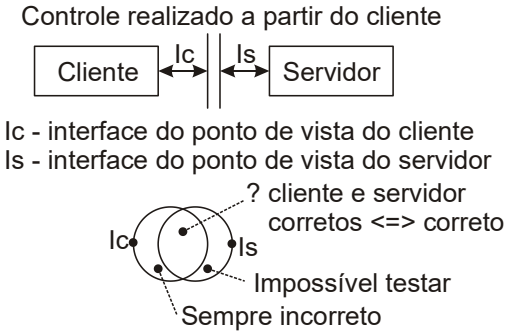

```
dimensions vms , vkmh , km , m , s, h ;
dimension_rules
{
   vms  =  m / s ;
   vkmh = km / h ;
   vms  = 0,27777778 * vkmh        constante de conversão km/h para m/s
} ;
semantic_types double veloc , double dist , double time , double number ;
operations
{
   veloc  = dist  / time  ;
   veloc 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

Análise estática de interfaces

- Mesmo se não dispuser de uma ferramenta de controle de tipos semânticos
  - sempre especifique, em comentários, a semântica de todos os parâmetros
  - isso facilita muito a inspeção visual do uso das interfaces
  - uso incorreto de interfaces é uma das frequentes causas de defeitos



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
    - aumenta o risco de manutenção incorreta
  - duplicações também podem acontecer quando dois programadores se depararem com problemas similares
- Procure ferramentas de análise do código que procuram por duplicações (servem também como controladores de plágio)
  - elimine as duplicações encontradas
    - crie um método com parâmetros que implementa as diversas variantes do código base

Uso de análise estática

- 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
  - apoia revisões após o desenvolvimento
    - verifica propriedades relevantes durante a execução
    - é mais para teste do que para revisão ou inspeção
  - exemplos
    - logs e exploradores de logs
    - controle da satisfação de condições
      - assertivas executáveis, verificadores estruturais
    - controle dinâmico de tipos
    - controle de vazamento de recursos
  - algumas ferramentas
    - **`valgrind`** para C++ / UNIX; **`Insure++`**
    - **...**

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
  - o apêndice contém exemplos de padrões de revisão de casos de uso e de estruturas de código
- Precisa mesmo?
  - O treinamento em padrões de projeto e de programação é requerido como parte da formação de bons programadores

Técnica de revisão: técnicas formais leves

- Sempre que puder utilize técnicas formais leves
  - também faz parte da formação de bons programadores
  - assertivas, argumentação da corretude e instrumentação
    - será visto na segunda parte da disciplina
  - 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

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
- Use um vocabulário relacionado com o domínio do problema
  - adequado aos cliente e usuário
  - adequado aos desenvolvedores
- Evite incluir ou discutir detalhes de implementação
  - deve-se evitar descrever a interface física com o usuário
  - ao desenvolver associe termos de especificação com os termos de codificação (nomes de classes, métodos, atributos, ...)
- Detalhes estão no apêndice

Critérios de qualidade para casos de uso

- Redija para que leitores leigos em computação possam ler e entender
  - evite jargão "computês"
  - enfoque o domínio do problema e não a implementação
  - evite adjetivos e advérbios que pouco ou nada contribuam para o entendimento
  - use uma linguagem polida (bem educada)
  - sugestão crie um dicionário de dados / termos técnicos
- Redija usando sintaxe e ortografia corretas
- Trate todas as sequências de sucesso e todas as de falha
  - o texto deve deixar claro o que acontece se
    - a sequência (cenário) de ações for bem sucedida
    - a sequência de ações for mal sucedida e o porquê disso
    - o que fazer quando ocorrerem erros de uso

Critérios gerais de avaliação da qualidade de especificações

- 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, considerando o nível de abstração do artefato
- Suficiência
  - a especificação não omite aspectos relevantes
    - exemplo: a especificação de uma classe relaciona os atributos e métodos públicos e protegidos, e as dependências de outras classes, mas não menciona os elementos privados
- Consistência
  - a especificação não contém contradições internas, ou com outros documentos

Critérios gerais de avaliação da qualidade de especificações

- 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

Critérios gerais de avaliação da qualidade de especificações

- 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

Critérios gerais de avaliação da qualidade de especificações

- 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

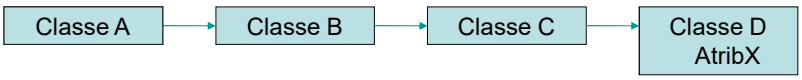
Outros critérios de qualidade de especificações

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

Apêndice

Lei de Demeter

Deusa grega da agricultura (latim: Ceres)



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

```
ObjA.metodoX( ){ return ObjB->ObjC->ObjD->atribX ; }
```

  - módulos ou classes devem depender exclusivamente de seus vizinhos imediatos, considerando a relação de **dependência de classe**
- 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; }`

Critérios de qualidade para casos de uso

- Evite ações excessivamente detalhadas, ex.
  - Usuário fornece largura do produto
  - Usuário fornece altura do produto

  } Usuário fornece as dimensões
- Evite ações excessivamente vagas
  - ex. Usuário fornece dados
- Use regras de negócio para descrever as composições de dados, ex.
  - dimensões são: altura, largura, profundidade, peso
- Assegure que as ações sejam coerentes com o objetivo do caso de uso
  - o objetivo do caso de uso deve estar refletido no título

Critérios de qualidade para casos de uso

Caso de uso	• O nome do caso de uso descreve o objetivo a alcançar? ex. Efetuar login no sistema específico
Resumo	• A descrição é resumida? • Explica bem o objetivo principal?
Escopo	• A natureza do caso de uso é "caixa preta"? Obs. um caso de uso deve ser uma característica (*feature*, funcionalidade) • O escopo contém tudo o que o caso de uso abordará? • O escopo contém coisas que o caso de uso não abordará?
Ator principal	• O ator principal aciona o caso de uso?
Interessados	• 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

Critérios de qualidade para casos de uso

Laboratório de Engenharia de Software

Invariante	• 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 *término normal* do caso de uso?
Pré condições	• Foram redigidas como um conjunto de condições? • Invariante + pré-condições estabelecem todas as condições necessárias para poder iniciar o caso de uso? • falta alguma condição necessária para iniciar o caso de uso? • Todas as pré-condições são diferentes das condições invariantes?
Acionamento	• 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?

Critérios de qualidade para casos de uso

Laboratório de Engenharia de Software

Fluxo principal (cenário de sucesso)	• 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? Obs. podem ser usados os critérios de qualidade de especificações.

Critérios de qualidade para casos de uso

Laboratório de Engenharia de Software

Fluxos alternativos (cenário alternativo)	• 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? – 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) Obs. podem ser usados os critérios de qualidade de especificações.

Critérios de qualidade para casos de uso

Laboratório de Engenharia de Software

Pós condições	• Foram redigidas como um conjunto de condições? • A invariante + pós-condições estabelecem completamente o resultado ao término do fluxo normal? • 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?
Garantia mínima	• Foi redigida como um conjunto de condições? • A garantia mínima estabelece completamente o resultado ao término (anormal) de um *fluxo alternativo*? • Falta alguma condição de garantia mínima? • As condições podem ser utilizadas como oráculo de teste?

Critérios de qualidade para casos de uso

Laboratório de Engenharia de Software

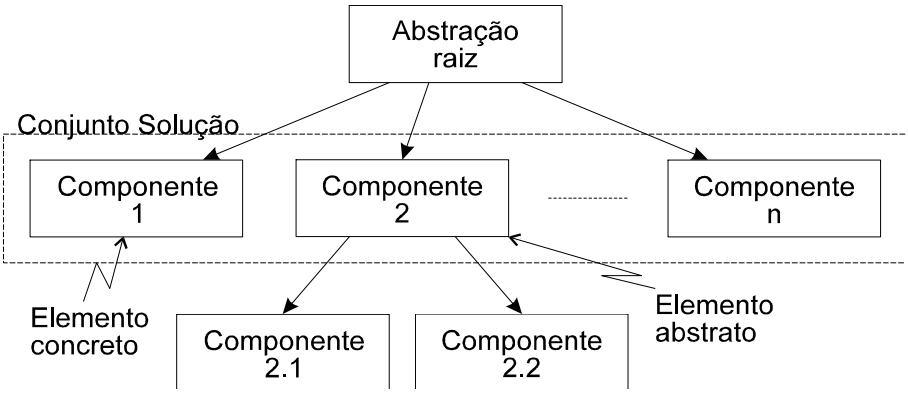
Requisitos	• Cada requisito corresponde a um requisito não funcional? • Foram considerados todos os requisitos não funcionais relevantes? • Todos os requisitos relacionados são relevantes para o caso de uso? • Os requisitos são viáveis? • Todos os requisitos são verificáveis?
Regras de negócio	• Todas 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?

Critérios de qualidade para casos de uso

Laboratório de Engenharia de Software

Casos de uso correlatos	• Todos os casos de uso que abordem aspectos ou características (*features*) correlatas estão referenciados? • Algum caso de uso referenciado não aborda aspectos ou características correlatas?
Artefatos correlatos	• 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?

Critérios de qualidade de projetos



Critérios de qualidade de decomposições

Elementos

- Definição ou especificação dos requisitos do elemento
- Adequação
- Nivelamento
- Necessidade
- Suficiência
- Viabilidade
- Minimalidade
- Reutilizabilidade
- Ortogonalidade Completeza da interface
- Integrabilidade
- Flexibilidade

Itens de interface

- Tipo do item
- Necessidade do item
- Suficiência de itens
- Minimalidade de itens
- Ortogonalidade de itens
- Encapsulamento

Critérios de qualidade de decomposições

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

Critérios de qualidade de decomposições

- 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

Critérios de qualidade de decomposições

- 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

Critérios de qualidade de decomposições

- 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

Critérios de qualidade de decomposições

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

Critérios de qualidade de decomposições

- 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

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

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

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

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

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

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.

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

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.

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]

Ex. 2: Uniformidade de terminologia 2 / 5

- Elementos utilizados
  - nomes (verbete)
    - simples
    - compostos
  - significados (acepção)
    - atribuídos aos nomes
    - para os quais deveriam existir nomes

expressivo — verbete
[De *expresso* + *-ivo*.] — etimologia
Adjetivo — classe léxica
1.Que exprime; significativo: — acepção
Calou-se, magoado, num silêncio ***expressivo***. — exemplo
extraído de [Aurélio]

- 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

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

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.

Referências

- 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; pags 79-86
- Staa, A.v.; *Programação Modular*; Rio de Janeiro: Campus; 2000

FIM