# DIM0436
# Qualidade de especificação

Richard Bonichon

20140724

# Resumo

1. Introdução
2. Qualidade de especificação
3. Verificação de programas

1 Introdução

2 Qualidade de especificação

3 Verificação de programas

# Conceitos básicos

- Usuário
- Requisitos funcionais / não funcionais
- Restrições
- Hipóteses responsabilidade do usuário

# O que é uma especificação?

## Definição (Especificação)

Um conjunto de requisitos a serem satisfeitos por um produto, desenho, ou serviço.

# O que é uma especificação?

## Definição (Especificação)

Um conjunto de requisitos a serem satisfeitos por um produto, desenho, ou serviço.

# O que é uma especificação?

## Definição (Especificação)

Um conjunto de requisitos a serem satisfeitos por um produto, desenho, ou serviço.


- Especificações em linguagem natural exigem que se assuma o significado de diverso termos
- Problemas de ambiguidade
- Problema muito estudado em Engenharia de Software, com métodos e formalismos para eliminá-/reduzi-lo.

# O que é uma especificação?

- Documento que serve de contrato entre o cliente e o projetista

# O que é uma especificação?

- Documento que serve de contrato entre o cliente e o projetista
- Para avaliar a correção de uma implementação

# O que é uma especificação?

- Documento que serve de contrato entre o cliente e o projetista
- Para avaliar a correção de uma implementação
- Refinamento continuado de especificações

# O que é uma especificação?

- Documento que serve de contrato entre o cliente e o projetista
- Para avaliar a correção de uma implementação
- Refinamento continuado de especificações
  - do mais abstrato ao mais concreto

# O que é uma especificação?

- Documento que serve de contrato entre o cliente e o projetista
- Para avaliar a correção de uma implementação
- Refinamento continuado de especificações
  - do mais abstrato ao mais concreto
  - o "como" de uma etapa pode ser o "que" da etapa seguinte, menos abstrata.

# Especificar programas


Programas são transformadores de estados.

# Especificação matemática

## Vantagens da formalidade

Usar um formalismo matemática permite:

- precisão
- verificação de consistência
- simulação
- verificação de programas
- uso de lógica matemática como linguagem de especificação

# Estilos de especificação

## Nível formal

Informal em linguagem natural

# Estilos de especificação

## Nível formal

**Informal** em linguagem natural

**Formal** sintaxe e semântica formalmente definidas.

# Estilos de especificação

## Nível formal

| | |
|---|---|
| Informal | em linguagem natural |
| Formal | sintaxe e semântica formalmente definidas. |
| Semi-formal | em geral apenas a sintaxe (e parte da semântica) formalmente definida. |

# Estilos de especificação

## Nível formal

**Informal** em linguagem natural

**Formal** sintaxe e semântica formalmente definidas.

**Semi-formal** em geral apenas a sintaxe (e parte da semântica) formalmente definida.

## Exemplo (Ferramentas formais)

Gramáticas (compiladores, linguagens formais) são exemplos de linguagens de especificação formal

# Estilos de especificação

## Tipo de detalhes

Operacional A especificação diz o que deve ser feito e como poderia ser feito.

# Estilos de especificação

## Tipo de detalhes

**Operacional** A especificação diz o que deve ser feito e como poderia ser feito.

**Descritivo** Apenas o resultado é descrito

1 Introdução

2 Qualidade de especificação

3 Verificação de programas

# Especificação operacional

- Seja $a$ um vetor com $n$ elementos.

# Especificação operacional

- Seja $a$ um vetor com $n$ elementos.
- O resultado da ordenação de $a$ é um vetor $b$ com $n$ elementos tal que:

# Especificação operacional

- Seja $a$ um vetor com $n$ elementos.
- O resultado da ordenação de $a$ é um vetor $b$ com $n$ elementos tal que:
  - o primeiro elemento de $b$ é o mínimo de $a$ (se vários elementos de $a$ têm o mesmo valor, qualquer um deles é aceitável),

# Especificação operacional

- Seja *a* um vetor com *n* elementos.
- O resultado da ordenação de *a* é um vetor *b* com *n* elementos tal que:
  - o primeiro elemento de *b* é o mínimo de *a* (se vários elementos de *a* têm o mesmo valor, qualquer um deles é aceitável),
  - o segundo elemento de *b* é o mínimo do vetor com n - 1 elementos obtido de *a* em se removendo o seu elemento mínimo;

# Especificação operacional

- Seja *a* um vetor com *n* elementos.
- O resultado da ordenação de *a* é um vetor *b* com *n* elementos tal que:
  - o primeiro elemento de *b* é o mínimo de *a* (se vários elementos de *a* têm o mesmo valor, qualquer um deles é aceitável),
  - o segundo elemento de *b* é o mínimo do vetor com n - 1 elementos obtido de *a* em se removendo o seu elemento mínimo;
  - e assim por diante até que todos os elementos de *a* tenham sido removidos.

# Especificação descritiva

- O resultado da ordenação de um vetor $a$ é um vetor $b$ que é uma permutação de $a$ e é ordenado.

# Especificação descritiva

- O resultado da ordenação de um vetor *a* é um vetor *b* que é uma permutação de *a* e é ordenado.
- Um vetor *b* é uma permutação de um vetor *a* se eles possuem exatamente os mesmos elementos, mas não obrigatoriamente na mesma ordem.

# Especificação descritiva

- O resultado da ordenação de um vetor *a* é um vetor *b* que é uma permutação de *a* e é ordenado.
- Um vetor *b* é uma permutação de um vetor *a* se eles possuem exatamente os mesmos elementos, mas não obrigatoriamente na mesma ordem.
- Um vetor é ordenado (em ordem crescente) se todo elemento do vetor é maior ou igual aos elementos encontrados nas posições precedentes desse vetor.

# Avaliar a qualidade da especificação

## Critérios

- Adequação apoio ao usuário

# Avaliar a qualidade da especificação

## Critérios

**Adequação** apoio ao usuário

**Explicitude** nada subentendido

# Avaliar a qualidade da especificação

## Critérios

**Adequação** apoio ao usuário

**Explicitude** nada subentendido

**Necessidade** só o necessário

# Avaliar a qualidade da especificação

## Critérios

| | |
|---|---|
| Adequação | apoio ao usuário |
| Explicitude | nada subentendido |
| Necessidade | só o necessário |
| Suficiência | tudo o necessário |

# Avaliar a qualidade da especificação

## Critérios

| | |
|---|---|
| Adequação | apoio ao usuário |
| Explicitude | nada subentendido |
| Necessidade | só o necessário |
| Suficiência | tudo o necessário |
| Consistência | sem contradição (internas, externas) |

# Avaliar a qualidade da especificação

## Critérios

| | |
|---|---|
| Adequação | apoio ao usuário |
| Explicitude | nada subentendido |
| Necessidade | só o necessário |
| Suficiência | tudo o necessário |
| Consistência | sem contradição (internas, externas) |
| Nivelamento | nível de abstração adequado |

# Avaliar a qualidade da especificação

## Critérios

| | |
|---|---|
| Adequação | apoio ao usuário |
| Explicitude | nada subentendido |
| Necessidade | só o necessário |
| Suficiência | tudo o necessário |
| Consistência | sem contradição (internas, externas) |
| Nivelamento | nível de abstração adequado |
| Compreensibilidade | tipo de leitores |

# Critérios

Não ambiguidade só uma interpretação

# Critérios

**Não ambiguidade** só uma interpretação

**Exatidão** conformidade com o mundo real

# Critérios

**Não ambiguidade** só uma interpretação

**Exatidão** conformidade com o mundo real

**Concisão** o mínimo de palavras

# Critérios

**Não ambiguidade** só uma interpretação

**Exatidão** conformidade com o mundo real

**Concisão** o mínimo de palavras

**Viabilidade** possibilidade de implementação

# Critérios

**Não ambiguidade** só uma interpretação

**Exatidão** conformidade com o mundo real

**Concisão** o mínimo de palavras

**Viabilidade** possibilidade de implementação

**Verificabilidade/testabilidade** para cada item

# Critérios

**Não ambiguidade** só uma interpretação

**Exatidão** conformidade com o mundo real

**Concisão** o mínimo de palavras

**Viabilidade** possibilidade de implementação

**Verificabilidade/testabilidade** para cada item

**Rastreabilidade** mapeamento requisitos $\leftrightarrow$ implementação

# Critérios

**Não ambiguidade** só uma interpretação

**Exatidão** conformidade com o mundo real

**Concisão** o mínimo de palavras

**Viabilidade** possibilidade de implementação

**Verificabilidade/testabilidade** para cada item

**Rastreabilidade** mapeamento requisitos $\leftrightarrow$ implementação

**Prioridade** entre requisitos

# Critérios

Portabilidade independência da plataforma de implementação

# Critérios

Portabilidade independência da plataforma de implementação

Reutilizabilidade em diferentes contextos

# Critérios

**Portabilidade** independência da plataforma de implementação

**Reutilizabilidade** em diferentes contextos

**Formalidade** linguagem formal (ou estilo de redação)

# Propriedades de programas

## Tipos de propriedades

Pré-condições propriedades válidas sobre os dados de entrada

# Propriedades de programas

## Tipos de propriedades

Pré-condições propriedades válidas sobre os dados de entrada

Pós-condições propriedades válidas sobre os dados de saída

# Propriedades de programas

## Tipos de propriedades

**Pré-condições** propriedades válidas sobre os dados de entrada

**Pós-condições** propriedades válidas sobre os dados de saída

**Invariantes** propriedades sempre válidas durante a execução do programa

# Exemplo

## Fatorial

$$\begin{aligned} n! &= 1 \text{ se } n = 0 \vee n = 1 \\ &= n * (n-1)! \text{ se } n > 1 \end{aligned}$$

## Propriedades

Pré $n \geq 0$

# Exemplo

## Fatorial

$$\begin{aligned} n! &= 1 \text{ se } n = 0 \vee n = 1 \\ &= n * (n-1)! \text{ se } n > 1 \end{aligned}$$

## Propriedades

Pré $n \geq 0$

Pós $return = n! \wedge return > 0$ $

# Exemplo

## Fatorial

$$\begin{aligned} n! &= 1 \text{ se } n = 0 \vee n = 1 \\ &= n * (n-1)! \text{ se } n > 1 \end{aligned}$$

## Propriedades

Pré $n \geq 0$

Pós $return = n! \wedge return > 0$

Inv $return > 0$

# Uma especificação é definitiva?

- Em que momento se faz a especificação?
- O que pode acontecer durante a implementação do sistema?
- Requisitos?
- Valor da validação

# V & V

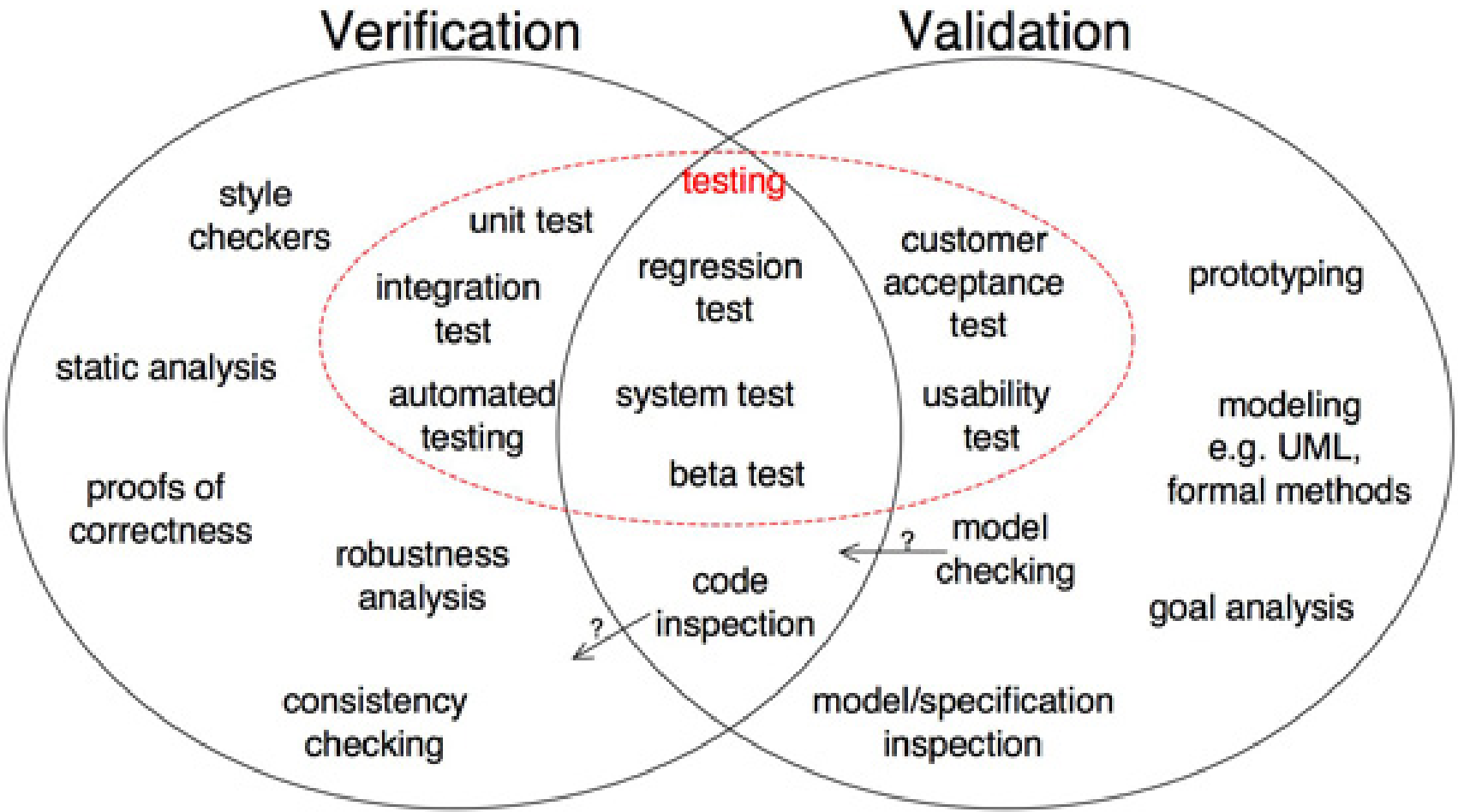
## Definição (Validação)

*Are we building the right thing?*

## Definição (Verificação)

*Are we building the thing right?*

# V & V em imagem


Verification
Validation
testing
style checkers
unit test
integration test
automated testing
regression test
system test
beta test
customer acceptance test
usability test
prototyping
modeling e.g. UML, formal methods
goal analysis
static analysis
proofs of correctness
robustness analysis
consistency checking
code inspection
?
model checking
?
model/specification inspection

1 Introdução

2 Qualidade de especificação

3 Verificação de programas

# Porque verificar programas?

## Observações

- Erros são comuns em programação
- O funcionamento correto de um programa depende dos dados fornecidos

# Porque verificar programas?

## Observações

- Erros são comuns em programação
- O funcionamento correto de um programa depende dos dados fornecidos

## Perguntas

- Come verificar se o programa desenvolvido atende à especificação?
- Como corrigi-lo se for o caso?

# Exemplos

## Especificação

A partir de duas sequências ordenadas, `merge` produz uma nova sequência ordenada que entrelace os números das duas sequências.

## Exemplo (Merge)

Entradas $v_1 = \{1, 3, 6, 9\}$, $v_2 = \{2, 3, 3, 7, 10\}$

Saída $v_3 = \{1, 2, 3, 3, 3, 4, 6, 7, 9, 10\}$

# merge

## Implementação

```
1  void merge(int v1[], int v2[], int v3[],
2             int t1, int t2) {
3    int i1 = 0, i2 = 0, i3 = 0;
4    int t3 = t1 + t2;
5
6    while (t3 > 0) {
7      if (v1[i1] > v2[i2]) {
8        v3[i3] = v2[i2];
9        i2 = i2 + 1;
10     }
11     else {
12       v3[i3] = v1[i1];
13       i1 = i1 + 1;
14     }
15     i3 = i3 + 1;
16     t3 = t3 - 1;
17   }
18 }
```

# merge

## Implementação

```
1  void merge(int v1[], int v2[], int v3[],
2             int t1, int t2) {
3    int i1 = 0, i2 = 0, i3 = 0;
4    int t3 = t1 + t2;
5
6    while (t3 > 0) {
7      if (v1[i1] > v2[i2]) {
8        v3[i3] = v2[i2];
9        i2 = i2 + 1;
10     }
11     else {
12       v3[i3] = v1[i1];
13       i1 = i1 + 1;
14     }
15     i3 = i3 + 1;
16     t3 = t3 - 1;
17   }
18 }
```

## Observações

- Se uma das sequências for vazia, há uma comparação indevida.
  `if (v1[i1] > v2[i2])}`

# merge (segunda versão)

## Implementação

```
1  void merge(int v1[], int v2[], int v3[],
2             int t1, int t2) {
3    int i1 = 0, i2 = 0, i3 = 0;
4
5    while (t1 > 0 && t2 > 0) {
6      if (v1[i1] > v2[i2]) {
7        v3[i3] = v2[i2];
8        i2 = i2 + 1;
9        t2 = t2 - 1;
10     }
11     else {
12       v3[i3] = v1[i1];
13       i1 = i1 + 1;
14       t1 = t1 - 1;
15     }
16     i3 = i3 + 1;
17   }
18 }
```

# merge (segunda versão)

## Implementação

```
1  void merge(int v1[], int v2[], int v3[],
2             int t1, int t2) {
3    int i1 = 0, i2 = 0, i3 = 0;
4
5    while (t1 > 0 && t2 > 0) {
6      if (v1[i1] > v2[i2]) {
7        v3[i3] = v2[i2];
8        i2 = i2 + 1;
9        t2 = t2 - 1;
10     }
11     else {
12       v3[i3] = v1[i1];
13       i1 = i1 + 1;
14       t1 = t1 - 1;
15     }
16     i3 = i3 + 1;
17   }
18 }
```

## Observações

- Comparação ocorre só se houver elementos nos dois vetores
- Mas deixa-se de colocar os elementos restantes do vetor no vetor $v_3$

# merge (terceira versão)

## Implementação

```
1  void merge(int v1[], int v2[], int v3[],
2             int t1, int t2) {
3    int i1 = 0, i2 = 0, i3 = 0;
4    while (t1 > 0 && t2 > 0) {
5      if (v1[i1] > v2[i2]) { v3[i3] = v2[i2];
6        i2++; t2--; }
7      else { v3[i3] = v1[i1]; i1++; t1--; }
8      i3--; }
9    while (t2 > 0) {
10     v3[i3] = v2[i2];
11     i2++; t2--; i3++;
12   }
13   while (t1 > 0) {
14     v3[i3] = v1[i1];
15     i1++; t1--; i3++;
16   }
17 }
```

# merge (terceira versão)

## Implementação

```
1  void merge(int v1[], int v2[], int v3[],
2             int t1, int t2) {
3    int i1 = 0, i2 = 0, i3 = 0;
4    while (t1 > 0 && t2 > 0) {
5      if (v1[i1] > v2[i2]) { v3[i3] = v2[i2];
6        i2++; t2--; }
7      else { v3[i3] = v1[i1]; i1++; t1--; }
8      i3--; }
9    while (t2 > 0) {
10     v3[i3] = v2[i2];
11     i2++; t2--; i3++;
12   }
13   while (t1 > 0) {
14     v3[i3] = v1[i1];
15     i1++; t1--; i3++;
16   }
17 }
```

## Observações

- É muito fácil introduzir erros num programa simples
- Como fazer para programas complexos?
- Como determinar se um programar está correto?
- Minimizar os erros, por meio de alguns procedimentos.

# Meios de verificação

## Inspeção

Seguir a lógica do programa em busca de situações incorretas/não previstas

## Testes

Casos de testes são selecionados, com dados correspondentes, com quais o programa é executado e os resultados verificados.

> *Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.*
> *– Edgar W. Dijkstra, The Humble Programmer, ACM Turing Lecture 1972*