


Laboratório de Engenharia de Software

Critério de valoração de casos de teste

Arndt von Staa
Departamento de Informática
PUC-Rio
Março 2017

Especificação



Laboratório de Engenharia de Software

- Objetivo desse módulo
 - apresentar os conceitos relacionados com critérios de valoração de casos de teste
 - apresentar e exemplificar o uso do critério de valoração.
- Justificativa
 - a escolha de condições de teste e de valores para os casos de teste semânticos deve enfatizar valores que tenham maior probabilidade de encontrar defeitos
- Texto
 - Pezzè, M.; Young, M.; Teste e Análise de Software; Porto Alegre, RS: Bookman; 2008, capítulo 9
 - Staa, A.v.; Programação Modular; Campus; 2000
 - Capítulo 15

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

2

Quais são os erros? São sempre detectáveis?



```
char meuString[ 6 ] ;
strcpy( meuString , "123456" ) ;

strncpy( meuString , "123456" , sizeof( meuString ) ) ;

strncpy( meuString , "123456" , strlen( meuString ) ) ; quais os problemas a mais?

void Copia( int dimStrDest , char * StrDest , char * StrOrg ) ;
. . .
char meuString[ 6 ] ;
Copia( sizeof( meuString ) , meuString , "123456" ) ;
. . .
void Copia( int dimStrDest , char * strDest , char * strOrg )
{
    assert( dimStrDest > strlen( strOrg ) ) ;      quais os problemas inerentes a essa linha de código?
    memcpy( strDest , strOrg , strlen( strOrg ) + 1 ) ;
}
                                           Esse código está correto? Como testá-lo?
```

Mar 2017

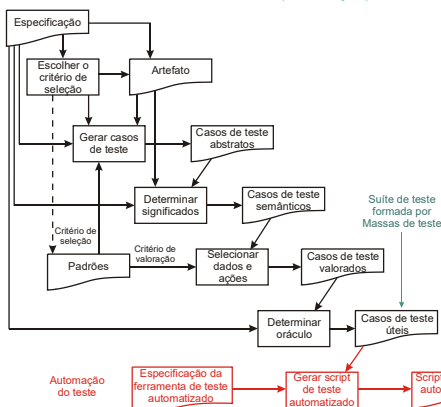
Arndt von Staa © LES/DI/PUC-Rio

3

Critério de seleção de casos de teste



Processo (recordação)



Dados

- o grau de qualidade a assegurar
- a especificação
- o nível de abstração

O **critério de seleção de casos de teste** determina

- como criar os **casos de teste abstratos**
- e como convertê-los em **casos de teste semânticos**

Dados

- os casos de teste semânticos

O **critério de valoração** determina

- como escolher os dados a serem usados nos **casos de teste valorados**

Dados

- o critério de valoração
- os casos de teste valorados, segundo esse critério


O critério de valoração define

- os **oráculos** contidos nos casos de teste úteis

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio


4

Critério de seleção, requisitos


Laboratório de Engenharia de Software

- Uma suíte de teste deve ser
 - **confiável:**
 - acusa falhas **sempre que existam defeitos** no artefato sendo testado
 - se isso fosse **sempre** possível, então seríamos capazes de saber se encontramos todos os defeitos, infelizmente não é
 - em virtude disso uso **eficácia aproximada**
 - criam-se mutantes (versões do AST contendo defeitos inseridos propositalmente)
 - mede-se o percentual dos defeitos inseridos que foi identificado pelos testes

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
5

Critério de seleção, requisitos


Laboratório de Engenharia de Software

- Uma suíte de teste deve ser
 - **completa:**
 - testa todo o artefato segundo um padrão de completeza
 - **cobertura** do teste, exemplos
 - » cobertura de instruções
 - » cobertura de arestas
 - » cobertura de chamadas
 - » cobertura de retornos (inclusive **throws**)
 - » cobertura de caminhos
 - » cobertura de fragmentos de caminhos
 - » cobertura de *widgets*
 - » ...

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
6

Critério de seleção, requisitos



- Uma suíte de teste deve ser
 - **indiferente à escolha dos dados:**
 - dados escolhidos de modo que satisfaçam as condições de um determinado caso de teste semântico devem **acusar sempre exatamente as mesmas falhas** para um mesmo código
 - entretanto, a prática mostra que:
 - para um mesmo caso de teste semântico, existem **dados valorados que têm probabilidade maior de encontrar falhas** do que outros
 - » ou seja, na prática **a escolha faz diferença**
 - podem existir **não determinismos** que fazem com que, em diferentes execuções, um mesmo código se comporte de forma diferente para um mesmo conjunto de dados. Exemplos:
 - » uso de **variáveis não inicializadas**
 - » **multi-programação** (threads)
 - » **aprendizado**

Mar 2017

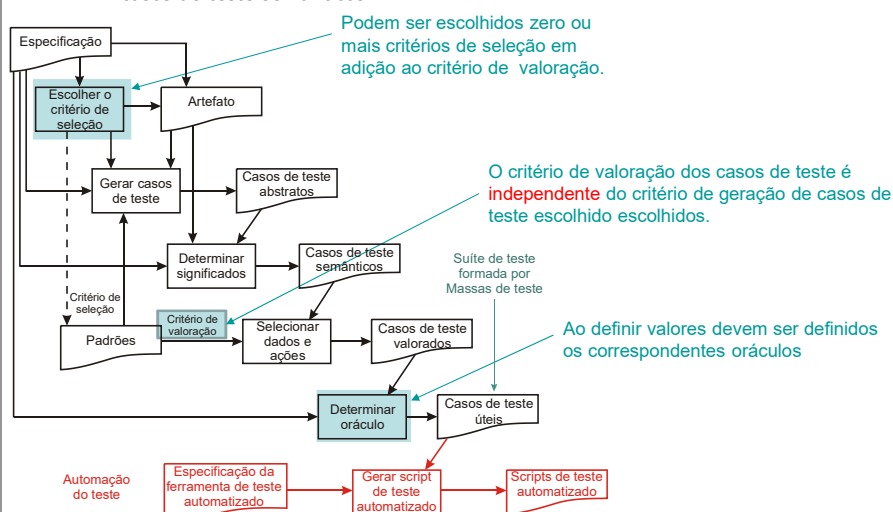
Arndt von Staa © LES/DI/PUC-Rio

7

Critério de valoração



- **Valoração dos dados:**
 - o conjunto de casos de teste valorados pode ser bem maior do que o conjunto de casos de teste semânticos



Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

8

Critério de valoração



- **Valoração dos dados:** é a escolha dos dados a serem utilizados a partir dos casos de teste semânticos
 - a escolha deve ser feita de modo a **maximizar a chance** de se encontrar uma falha
 - a observação da prática tem mostrado que a escolha de **condições de contorno** e condições “**patológicas**” aumenta esta chance
 - depois de valorar um caso de teste determina-se o **oráculo**
 - todos os oráculos devem ser **capazes de diferenciar** entre um resultado válido e uma falha
- Frequentemente o critério de valoração pode servir também como critério de seleção
 - ex. teste caixa preta de funções ou métodos simples

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

9

Qual seria a suíte de teste?



```
# include <stdio.h>
void main ( void )
{
    int    i , Num , Resto ;
    float TotalPar = 0 , TotalImpar = 0 ;
    int    NumPar = 0 , NumImpar = 0 ;
    printf( "Digite 5 números: " ) ;
    for ( i = 0 ; i < 5 ; i++ )
    {
        scanf ( "%i" , &Num ) ;
        Resto = Num % 2 ;
        if ( Resto == 0 )
        {
            TotalPar += Num ;
            NumPar ++ ;
        } else
        {
            TotalImpar += Num ;
            NumImpar ++ ;
        }
    }
    printf ( "\nMédias dos pares    = %8.1f" , TotalPar / NumPar ) ;
    printf ( "\nMédias dos impares = %8.1f\n" , TotalImpar / NumImpar ) ;
}
```

Como podemos assumir que esse código pode ser testado como se fosse “caixa preta”?

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

10

Qual seria a suíte de teste?



- como você testaria a função `findString`?
`findString(stringBase , stringProcurado,
 inxInferior , inxSuperior)`
se encontrado: retorna o índice maior ou igual a `inxInferior` do primeiro caractere de `stringProcurado` encontrado em `stringBase`
se não encontrado: retorna -1


CrITÉRIOS de valoraÇÃO



- O critério de valoração é um critério baseado em lista de controle (*checklist*)
 - critérios baseados em listas de controle baseiam-se em aquisição de conhecimento (aprendizado)
 - cada item da lista descreve uma condição a ser testada
 - crie e mantenha um “manual de testes” com os padrões de valoração
 - registre nele
 - os cenários patológicos (que frequentemente levam a falhas)
 - » condições de contorno
 - » enganos frequentes
 - » condições patológicas
 - » ...
 - quais os sintomas e como criar testes capazes de identificar falhas para cada um desses cenários

Laboratório de Engenharia de Software

Lista de controle - tempo de execução




- Exemplos de itens de lista de controle
 - verificar se ponteiro pode apontar para valor não válido (ex. NULL)
 - verificar se ocorre vazamento de recursos (ex. memória)
 - verificar esgotamento de memória
 - verificar divisão por zero
 - verificar dados ilegais para a função sendo avaliada, ex. `sqrt(-1)`
 - verificar *strings* de tamanho maior do que o limite especificado
 - verificar caracteres ilegais em campos digitados, ex. campo numérico
 - ...

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
13

Laboratório de Engenharia de Software

Lista de controle - padrão de programação



- Exemplos de itens de lista de controle
 - assegurar que ponteiro sempre aponta para valor válido (ex. !NULL)
 - assegurar que esgotamento de memória é controlado
 - assegurar que regras ACID de transações não são violadas
 - Atomicidade, Consistência, Isolamento e Durabilidade
 - assegurar que não ocorre saturação de acumulador vírgula flutuante
 - assegurar que não ocorre perda danosa de significância
 - assegurar que a composição de erros numéricos de representação e de processamento não ultrapassam a tolerância desejada
 - existe aritmética intervalar capaz de controlar isso em tempo de execução
 - assegurar que divisão por zero é impossível
 - assegurar a corretude sintática e semântica dos argumentos passados para um método
 - assegurar que *strings* de não violam o limite de tamanho
 - ...

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
14

Critérios de valoração, comparações



- Ao testar uma comparação $a \leq b$, escolha sempre os três casos:

$$a = b - \varepsilon$$

$$a = b$$

$$a = b + \varepsilon$$

- ε é o menor valor possível que torne verdadeira a relação $a + \varepsilon \geq b$ quando for verdadeira a relação $a < b$.

- para valores **inteiros** ε é 1.
- para valores **vírgula flutuante**

- **erro absoluto:**

$$a - \varepsilon \leq b \leq a + \varepsilon$$

» depende da magnitude de a e b

- **erro relativo:**

$$1 - \varepsilon \leq b / a \leq 1 + \varepsilon$$

» independente da magnitude

» ε pode ser o número de algarismos significativos desejado

» procure usar sempre que possível erro relativo ao testar

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

15

Critérios de valoração, ordenação de strings



- Qual a ordenação: José ? JOSÉ ? JOSE ? jÓsÉ
 - para computadores cada caractere tem um valor numérico
 - a comparação simples utiliza o valor numérico
 - `strcmp` ou `memcmp`
 - `Jóse > Jose` [`ó > o`] ; `Jose < jóse` [`J < j`]
 - precisa criar, ou usar, uma função de comparação que seja insensível à caixa e à acentuação (diacríticos)
 - muitas convertem para **representação canônica** e depois comparam
 - ex. tudo minúsculo e sem acentos
 - comparação parcialmente igual → google
- Onde fica o caractere Euro: € na tabela Unicode?
 - Em ASCII é 80 hexadecimal ou 128 decimal

canônico: em conformidade com padrão, modelo, norma, ou regra

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

16

Cr terios de valora  o, ordena  o de strings



- Caso listas de *strings* sejam ordenadas deve-se testar a **sensitividade dos caracteres** nas diferentes posi  es: primeiro, segundo, meio,  ltimo
- Para **letras** temos os problemas
 - caracteres s o comparados como se fossem n meros
 - descontinuidade dos valores num ricos
 - caracteres ASCII min sculos, mai sculos e diacr ticos est o em regi es diferentes da tabela de c digo
 - v rios caracteres diferentes podem representar o mesmo caractere de compara  o
 - a == A ==   ==   ==   ==   ==   ==   == ...
 - as tabelas ASCII (8 bits) e Unicode (16 bits) s o diferentes (* bvio*) mesmo quando se considera o conjunto de caracteres latinos

Mar 2017

Arndt von Staa   LES/DI/PUC-Rio

17

Cr terios de valora  o: tabela ISO / ASCII



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8	�		,	f	"	...	†	‡	�	‰	Š	<	�		Ž	
9		\	/	"	"	•	–	—	~	™	š	>	�		ž	Ÿ
A		ı	�	�	�	�	�	�	�	�	�	�	�	�	�	�
B	�	�	�	�	�	�	�	�	�	�	�	�	�	�	�	�
C	�	�	�	�	�	�	�	�	�	�	�	�	�	�	�	�
D	�	�	�	�	�	�	�	�	�	�	�	�	�	�	�	�
E	�	�	�	�	�	�	�	�	�	�	�	�	�	�	�	�
F	�	�	�	�	�	�	�	�	�	�	�	�	�	�	�	�

parte ASCII

parte ISO

Mar 2017

Arndt von Staa   LES/DI/PUC-Rio

18

Tabela ISO / ASCII para comparação português																LES	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/	
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	
4	@	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	
5	p	q	r	s	t	u	v	w	x	y	z	[\]	^	_	
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	\0	
8	€	\0	,	f	"	...	†	‡	^	%	\0	<	\0	\0	\0	\0	
9	\0	`	'	"	"	•	—	—	~	™	\0	>	\0	\0	\0	\0	
A	\0	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯	
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿	
C	a	a	a	a	\0	\0	\0	c	\0	e	e	\0	\0	i	\0	\0	
D	\0	\0	\0	o	o	o	\0	x	\0	\0	u	\0	u	\0	\0	\0	
E	a	a	a	a	\0	\0	\0	c	\0	e	e	\0	\0	i	\0	\0	
F	\0	\0	\0	o	o	o	\0	÷	\0	\0	u	\0	u	\0	\0	\0	

'\0' corresponde ao valor numérico 0, é usado para os caracteres *letra* que não têm significado em português

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

19

Algoritmo de comparação

The logo for LES (Laboratório de Engenharia de Software) is located in the top right corner. It consists of the letters "LES" in a bold, sans-serif font, followed by a circular graphic containing three curved lines that suggest a globe or a stylized 'e'.

```
int i ;
int sizPrefix = min( length_1 , length_2 ) ;
for ( i = 0 ; i < sizPrefix ; i ++ )
{
    if ( characterConversionTable[ string_1[ i ] ] <
        characterConversionTable[ string_2[ i ] ] )
    {
        return LESS ;
    } else if ( characterConversionTable[ string_1[ i ] ] >
        characterConversionTable[ string_2[ i ] ] )
    {
        return GREATER ;
    } /* if */
} /* while */
/* o mais curto é igual à parte inicial do mais longo */
if ( length_1 < length_2 ) return LESS ;
if ( length_1 > length_2 ) return GREATER ;
return EQUAL ;
```

Laboratório de Engenharia de Software

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

27


Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

20

Laboratório de Engenharia de Software

Critérios de valoração, enumerações




- Cada elemento de uma **enumeração** deve ser testado
 - e se o conjunto de enumeração for muito extenso?
 - caracteres são enumerações
 - dependendo do caso pode-se simplificar
 - testando intervalos 'a' <= ch <= 'z'
 - mas pode-se utilizar intervalos para testar os caracteres ISO/ASCII válidos em português?
 - e se o computador usar a codificação EBCDIC (IBM: Extended Binary Coded Decimal Interchange Code)
 - ou 'a' <= ch <= 'i'
 - ou 'j' <= ch <= 'r'
 - ou 's' <= ch <= 'z'

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

Laboratório de Engenharia de Software


Critérios de valoração, tamanho



- Ao testar **valores de tamanho variável**
 - por exemplo: arquivos, vetores, *strings*
 - gere casos de teste para
 - tamanho zero
 - tamanho mínimo-1 , caso exista limite inferior
 - tamanho mínimo
 - tamanho médio
 - tamanho máximo
 - tamanho máximo+1 , caso exista limite superior
 - no caso de strings teste ainda: tamanho máximo+número grande
 - teste de sensibilidade a agressões
 - vulnerabilidade decorrente da falta de controle de extravasão de *buffer*

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

Critérios de valoração, pertinência




Laboratório de Engenharia de Software

- Ao **acessar valores pertencentes a um conjunto** dinamicamente criado, ex. lista, arquivo sequencial
 - considere sempre
 - conjunto vazio
 - conjunto contendo exatamente um elemento
 - e conjunto contendo três ou mais elementos
 - para um conjunto com 3 ou mais elementos acesse
 - o primeiro elemento
 - um elemento mais ou menos no meio do conjunto
 - o último elemento

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

Critérios de valoração, pertinência



Laboratório de Engenharia de Software

- Ao **procurar elementos pertencentes a conjuntos ordenados**
 - o elemento inexistente imediatamente anterior ao primeiro
 - em um *conjunto ordenado* deve ser possível gerar um valor anterior ao primeiro elemento
 - o primeiro elemento
 - o elemento inexistente entre o primeiro e o segundo
 - em conjunto ordenado deve ser possível gerar
 - um elemento mais ou menos no meio do conjunto
 - um elemento inexistente mais ou menos no meio do conjunto
 - em conjunto ordenado deve ser possível gerar
 - o elemento inexistente entre o penúltimo e o último
 - em conjunto ordenado deve ser possível gerar
 - o último elemento
 - o elemento inexistente imediatamente após ao último
 - em conjunto ordenado deve ser possível gerar

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

Critérios de valoração, nomes



- Ao testar **nomes de arquivos**
 - nome (*string*) nulo
 - nomes com caracteres ilegais, ex.: ? * / \ < > " ' | . , ;
 - sempre para os casos existe e não existe
 - nome sem extensão
 - nome com extensão igual ao *default*
 - nome com extensão diferente do *default*
 - nome com duas extensões (ex. xpto.x.y)
 - nome com diretório absoluto
 - nome com diretório relativo
 - nome com dispositivo diferente do corrente, ex. disco

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

25

Exemplo: quais seriam os casos de teste?



Modelo

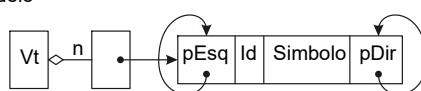


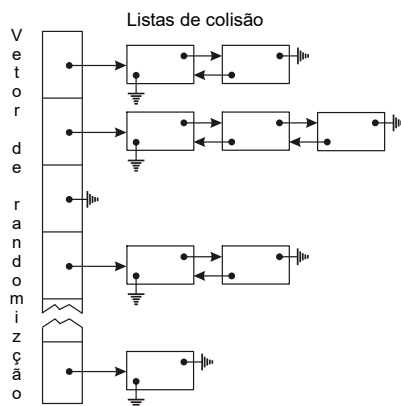
Tabela hash (randomização)

- $n > 1$
- Para todos os *Simbolos*

```
{
    0 <= ObterInxHash(
        Simbolo ) < n
}
```
- Para todos os *Simbolos* da lista *inx* : $0 \leq inx < n$

```
{
    inx == ObterInxHash(
        Simbolo )
}
```
- cada lista é duplamente encadeada e ordenada segundo *Simbolo*

Exemplo




Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

26

Laboratório de Engenharia de Software

Exemplo



- $n == 1 \rightarrow$ ilegal
- $n == 5$

Por que não $n == 2$?


 - criar símbolos crescentes A, B, C, D, E, F, G tal que $\text{ObterHash}(\text{símbolo}) == 0$
 - procurar e não encontrar D
 - inserir D e, depois, procurar e encontrar D
 - procurar e não encontrar C
 - procurar e não encontrar E
 - inserir B e, depois, procurar e encontrar B
 - procurar e não encontrar A
 - procurar e não encontrar C
 - inserir F e, depois, procurar e encontrar F
 - procurar e não encontrar E
 - procurar e não encontrar G
 - procurar e encontrar B
 - procurar e encontrar D
 - procurar e encontrar F

Que instrumentação precisa ser adicionada para poder realizar o teste?

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

Laboratório de Engenharia de Software

Exemplo



- $n == 5$

- criar símbolos A, B, C, D, E, F, G tal que $\text{ObterHash}(\text{símbolo}) == 2$
 - procurar e não encontrar D
 - inserir D e, depois, procurar e encontrar D
 - procurar e não encontrar C
 - procurar e não encontrar E
 - inserir B e, depois, procurar e encontrar B
 - procurar e não encontrar A
 - procurar e não encontrar C
 - inserir F e, depois, procurar e encontrar F
 - procurar e não encontrar E
 - procurar e não encontrar G
 - procurar e encontrar B
 - procurar e encontrar D
 - procurar e encontrar F

Precisa realmente de todos eles?

Lista e hash não são independentes?

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

Exemplo




- $n == 5$
 - criar símbolos A, B, C tal que $\text{ObterHash}(\text{simbolo}) == 4$
 - procurar e não encontrar B
 - inserir e, depois, procurar e encontrar B
 - procurar e não encontrar A
 - procurar e não encontrar C
 - procurar e encontrar B
 - criar símbolos A tal que $\text{ObterHash}(\text{simbolo}) == 1$
 - procurar e não encontrar A
 - criar símbolos B tal que $\text{ObterHash}(\text{simbolo}) == 3$
 - procurar e não encontrar B



APÊNDICE

Laboratório de Engenharia de Software

Critérios de valoração, valores numéricos




- Ao testar tabelas, métodos ou funções que recebam valores numéricos representando alguma grandeza (ex. tempo, dinheiro, distância, dimensão de vetor, etc.)
 - teste sempre para 0
 - tenta observar defeitos de divisão por zero
 - teste sempre para -1
 - tenta observar falta de controle do uso ilegal de valores negativos (ex. `sqrt(-1)`)
 - se a máquina em uso codifica inteiros usando magnitude com sinal, teste com 0, "-a + b" e "a + -b" e similares
 - tenta observar erros ridículos, ex. dívida de R\$-0,00
 - teste também para valores razoáveis considerando a aplicação
 - permite verificar se o resultado corresponde a o que o usuário espera
 - o que o usuário espera pode ser diferente de o que a especificação (oráculo) determina!

Goddard Space Flight Center; *FSW Unit Test Standard*; Flight Software Branch; Code 582; 2006; Buscado em: 06/abril/2009; URL: <http://software.gsfc.nasa.gov/AssetsApproved/PA2.4.2.2.1.doc>

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
31

Laboratório de Engenharia de Software

Critérios de valoração, valores numéricos



Procure sempre criar casos de teste que provoquem:


- overflow
 - sintoma
 - em inteiros e em vírgula flutuante: o resultado da soma é maior do que a capacidade de representação, teste para os dois dados positivos e para os dois negativos, mistura não pode dar *overflow*
 - em inteiro: o resultado é menor do que pelo menos uma das parcelas
 - » $a + b < a$ ou $a + b < b$, sendo $a > 0$ e $b > 0$
 - exemplos
 - somas ou multiplicações envolvendo números grandes, ou muitos números
 - em vírgula flutuante: divisão de número grande (expoente positivo grande) por número muito pequeno (expoente negativo grande)
 - próximo slide tem a codificação de números vírgula flutuante

O contrário de um dado válido é um dado não válido, mas não um dado inválido

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
32

Laboratório de Engenharia de Software

Critérios de valoração, valores numéricos




Procure sempre criar casos de teste que provoquem:

- **divisão por zero**
 - sintoma erro acusado pelo hardware
 - em vírgula flutuante teste para “quase zero”, ex 10^{-n} onde n é próximo do limite da codificação de vírgula flutuante usada pela máquina
 - sintoma overflow
- **underflow** em vírgula flutuante
 - sintoma
 - em vírgula flutuante: número absoluto é diferente de zero, porém menor do que o menor representável na codificação usada pela máquina
 - exemplo
 - em vírgula flutuante: divisão de número muito pequeno por número muito grande

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
33

Laboratório de Engenharia de Software

Critérios de valoração, valores numéricos



Procure sempre criar casos de teste que provoquem:

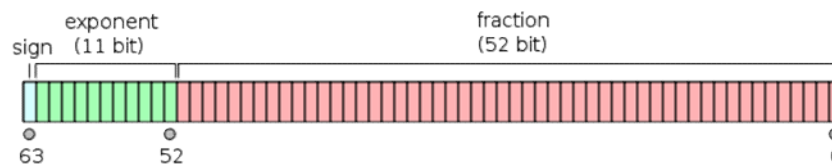
- **saturação de acumulador**
 - sintoma
 - em vírgula flutuante: somas sucessivas podem não alterar o valor do resultado
 - exemplos
 - em vírgula flutuante: ocorre quando se tenta somar um número pequeno a um número grande → o pequeno é tratado como zero
 - soma de um número muito grande de pequenas parcelas

Mar 2017
Arndt von Staa © LES/DI/PUC-Rio
34

Vírgula flutuante: IEEE 754 double



- Algumas vezes é importante conhecer o hardware, ex.
- Formato de números vírgula flutuante *double* (64 bits)
 - Deslocamento expoente: $0x3FF$ (metade de 11 bits)
 - Expoente mínimo = $-0x3FF = -1023_{10}$
 - Expoente máximo = $0x7FF - 0x3FF = 1024_{10}$
 - a fração é sempre normalizada: o expoente deve ser ajustado de modo que o primeiro bit da fração seja não nulo
 - consequentemente ele não precisa ser representado na codificação
 - o número de bits da fração é então 53 bits
 - Valor-bin = $\text{exp-bin}(\text{exp} - 0x3FF) * 1.\text{fracao}$



[Wikipedia] – Obs. existem diversas codificações de valores especiais.

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

35

Vírgula flutuante decimal, para ilustrar



- $10^{** \text{exp} + \text{fração}}$
 - as frações são **normalizadas**: $0.1 \leq \text{fração} < 1.0$
 - codexp é codificado $50 + \text{exp}$ o que permite gerar expoentes de $-50 \leq \text{exp} \leq 49$ $\text{exp} = \text{codexp} - 50$
- exemplos de valores:
 - 1. = 5110 ($10^{*0,1}$)
 - 0.1 = 5010
 - 100 = 5310
 - 0.001 = 4810

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

36

LES

- soma
flutuante

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

37

LES

Mar 2017

Arndt von Staa © LES/DI/PUC-Rio

38