


Laboratório de Engenharia de Software

Critério de seleção de casos de teste

Arndt von Staa
Departamento de Informática
PUC-Rio
Março 2015

Especificação



Laboratório de Engenharia de Software

- Objetivo desse módulo
 - apresentar os conceitos relacionados com critérios de seleção de casos de teste
 - apresentar e exemplificar o uso do critério de valoração. Esse critério é da família dos critérios de condições de contorno.
- 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 2015Arndt von Staa © LES/DI/PUC-Rio2

Critério de seleção, requisitos



- Um critério de seleção de casos 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** – o percentual dos defeitos existentes que foi identificado pelos testes
 - completo:**
 - 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 *widgets*
 - » ...

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

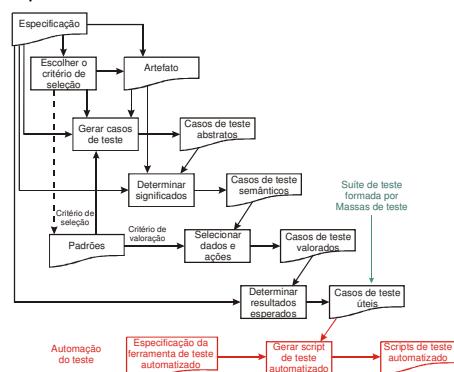
3

Critério de seleção, requisitos



- Um critério de seleção de casos de teste deve ser, cont.
 - indiferente à escolha:**
 - 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 conjuntos de dados valorados que têm **probabilidade maior de encontrar falhas** do que outros
- ou seja, 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*)



Mar 2015

Arndt von Staa © LES/DI/PUC-Rio


4

Cr terio de sele  o, requisitos


Laborat rio de Engenharia de Software

- Problema da *confiabilidade*
 - em geral n o   poss vel gerar uma massa de teste **finita e confi vel**

Mar 2015
Arndt von Staa   LES/DI/PUC-Rio
5

Cr terio de sele  o, requisitos


Laborat rio de Engenharia de Software

- Problema da *escolha*
 - diferentes escolhas de dados (valora  es) satisfazendo um mesmo caso de teste sem ntico **podem levar**   detec  o de diferentes falhas ou mesmo   n o detec  o delas
 - exemplo: o caso de teste sem ntico pode exigir que se acesse um elemento contido em uma lista de colis o de uma tabela de randomiza  o (*hash table*). O programa sob teste cont m um defeito que o faz errar ao acessar elementos da lista de colis o de  ndice **dimTab-1** (limite superior). Caso a escolha n o exercite este  ndice, o defeito n o ser  observado nem removido.
 -   uma falha de *sensitividade*

Mar 2015
Arndt von Staa   LES/DI/PUC-Rio
6

Crítério de seleção, requisitos



- Problema da *completeza* da suíte (massas) de teste
 - formas diferentes de criar os conjuntos de casos de teste podem levar a massas de teste muito diferentes
 - algum **critério de completeza** deve sempre estar identificado
 - procurem usar instrumentos de medição da cobertura da suíte de testes
 - Exemplos de critérios de completeza das suítes de teste
 - Cobertura de instruções
 - Cobertura das arestas do fluxograma
 - Cobertura de fragmentos de caminhos
 - Cobertura de caminhos inteiros
 - Cobertura de chamadas de funções
 - Cobertura dos retornos de funções, inclusive “**throw**” e chamadas a funções que jamais retornam (arrumação da casa)
 - Cobertura dos elementos da interface com o usuário
 - . . .

Mar 2015

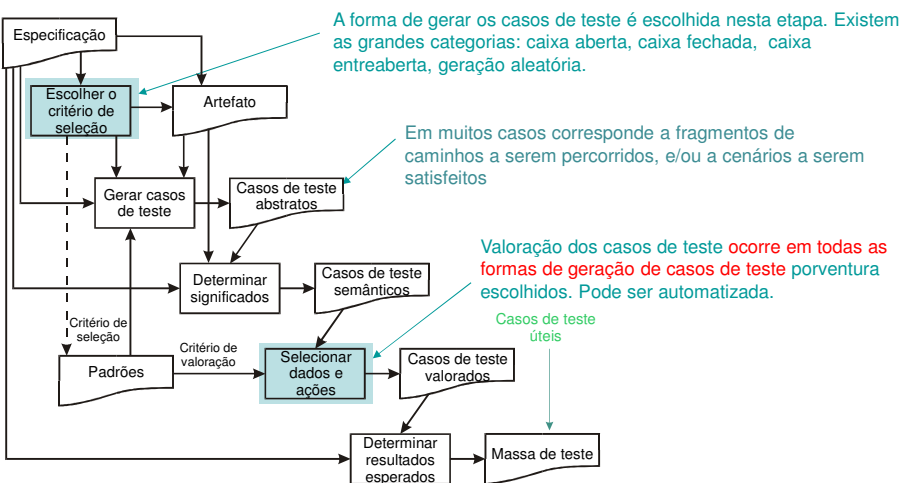
Arndt von Staa © LES/DI/PUC-Rio

7

Crítério de valoração



- **Valoração dos dados**: é a escolha dos dados a serem utilizados pelos casos de teste



Mar 2015

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
 - idealmente a escolha deve ser feita de modo a **maximizar a chance** de se encontrar um problema
 - a observação da prática tem mostrado que a escolha de **condições de contorno** aumenta esta chance
 - podem-se adicionar condições que são frequentes causadoras de problemas
 - lista de controle (*checklist*), exemplos
 - overflow
 - saturação de acumulador
 - perda de significância
 - divisão por zero
 - dados ilegais para a função sendo avaliada, ex. `sqrt(-1)`
 - caracteres ilegais para o campo sendo preenchido, ex. campo numérico
 - ...

Mar 2015

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 ) ;
}
```

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

10

Laboratório de Engenharia de Software

Quais são os erros? São sempre detectáveis?

```

char MeuString[ ] = "abcde" ;
strcpy( MeuString , "123456" ) ;

char MeuString[ ] = "abcde" ;
strncpy( MeuString , "123456" , sizeof( MeuString ) ) ;

strncpy( MeuString , "123456" , strlen( MeuString ) ) ; //qual o problema?

void Copia( int dimStrDest , char * StrDest , char * StrOrg ) ;
. . .
char MeuString[ ] = "abcde" ;
Copia( strlen( MeuString ) , MeuString , "123456" ) ;
. . .
void Copia( int dimStrDest , char * StrDest , char * StrOrg )
{
    assert( dimStrDest >= strlen( StrOrg ) ) ;
    strcpy( StrDest , StrOrg ) ;
}

```

LES

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
11

Laboratório de Engenharia de Software

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:**


$$1 - \varepsilon \leq b / a \leq 1 + \varepsilon$$
 - » independe da magnitude
 - » ε pode ser o número de algarismos significativos desejado
 - » procure usar sempre que possível erro relativo ao testar

LES

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
12

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, -1 + 1 e 1 + -1 e similares
 - tenta observar erros ridículos, ex. dívida de R\$-0,00
 - teste sempre 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 2015
Arndt von Staa © LES/DI/PUC-Rio
13

Laboratório de Engenharia de Software

Critérios de valoração, valores numéricos



Procure sempre provocar as condições pouco comuns:

- **overflow**
 - sintoma
 - em vírgula flutuante e em inteiro: resultado é maior do que a capacidade de representação
 - 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

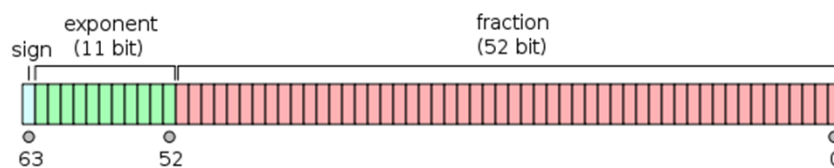
O contrário de um dado válido é um dado não válido, mas não um dado inválido

Mar 2015
Arndt von Staa © LES/DI/PUC-Rio
14

Vírgula flutuante: IEEE 754 double



- 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 2015

Arndt von Staa © LES/DI/PUC-Rio

15

Vírgula flutuante decimal, para ilustrar



- $10^{**} \text{exp} + \text{fração}$
 - as frações devem estar **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 2015

Arndt von Staa © LES/DI/PUC-Rio

16

Critérios de valoração, valores numéricos



- **underflow** em vírgula flutuante
 - sintoma
 - em vírgula flutuante: número absoluto 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
- **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
- **divisão por zero**

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

17

Exemplo de saturação



- | | |
|--|--------------------------------|
| • adições sucessivas | • adições sucessivas |
| • decimal, 2 dígitos na fração | • decimal, 3 dígitos na fração |
| 5023 5023 5011 5011 50230 50230 51712 51710 | |
| 5074 5097 5013 5024 50740 50970 51290 52100 | |
| 5011 5110 5023 5047 50110 51108 50990 52109 | |
| 5171 5181 5074 5112 51710 51818 50740 52116 | |
| 5013 5182 5099 5121 50130 51831 50230 52118 | |
| 5129 5211 5129 5150 51290 52112 50130 52119 | |
| 5099 5211 5171 5212 50990 52121 50110 52120 | |
- ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
- parcelas não ordenadas parcelas ordenadas soma flutuante


formato vírgula flutuante do exemplo: *eef* valor real: $10^{**}(ee-50)^{*}ff$

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

18

CrITÉRIOS de valoraÇão, ordenaÇão de strings




Laboratório de Engenharia de Software

- Caso listas de *strings* sejam ordenadas deve-se testar a **sensitividade dos caracteres** nas diferentes posições: primeiro, segundo, meio, último
- Para **letras** temos os problemas
 - caracteres são comparados como se fossem números
 - descontinuidade dos valores numéricos
 - caracteres ASCII minúsculos, maiúsculos e diacríticos estão em regiões diferentes da tabela de código
 - vários caracteres diferentes podem representar o mesmo caractere de comparação
 - a == A == á == à == â == ã == ä == ...
 - as tabelas ASCII (8 bits) e Unicode (16 bits) são diferentes (*óbvio*) mesmo quando se considera o conjunto de caracteres latinos

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

CrITÉRIOS de valoraÇão, ordenaÇão de strings



Laboratório de Engenharia de Software

- 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
 - 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

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

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

Critérios de valoração: 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	<	œ		ž	Ÿ
9		`	'	"	"	•	—	—	~	™	š	>	œ		ž	Ÿ
A		ı	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	÷	ø	ù	ú	û	ü	ý	þ	ÿ	

parte ASCII

parte ISO

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

21

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

21

Tabela ISO / ASCII para comparação português

	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	ı	¢	£	¤	¥	¦	§	¨	©	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

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

22

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

22

Algoritmo de comparação



```
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 */
if ( Length_1 < Length_2 ) return LESS ;
if ( Length_1 > Length_2 ) return GREATER ;
return EQUAL ;
```

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

23

Crítérios de valoração, enumerações




- Cada elemento de uma **enumeração** deve ser testado
 - 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)
 - problema e se o conjunto de enumeração for muito extenso?

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

24

Critérios de valoração, tamanho




Laboratório de Engenharia de Software

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

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
 - acesse
 - o primeiro elemento
 - um elemento mais ou menos no meio do conjunto
 - o último elemento

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

Critérios de valoração, pertinência



- 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


Critérios de valoração, pertinência



- 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

Laboratório de Engenharia de Software

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

Laboratório de Engenharia de Software

Critérios de valoração



- O critério de valoração é um exemplo de critério **baseado em lista de controle** (*checklist*)
 - critérios baseados em listas de controle são critérios que se baseiam no aprendizado (aquisição de conhecimento)
 - o conhecimento deve ser registrado
 - para não se perder com o tempo
 - para reduzir o tempo de treinamento quando ingressar um novo desenvolvedor na equipe
 - crie e mantenha um “**manual de testes**” com os padrões de valoração
 - registre nele os problemas encontrados com alguma frequência e quando e como devem ser realizados os testes para identificar a ocorrência desses problemas
 - cenário de teste

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

Exemplo: quais seriam os casos de teste?



Modelo

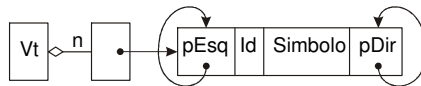


Tabela hash (randomização)

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

```

{
    0 <= ObterInxHash(
        Simbolo ) < n
}

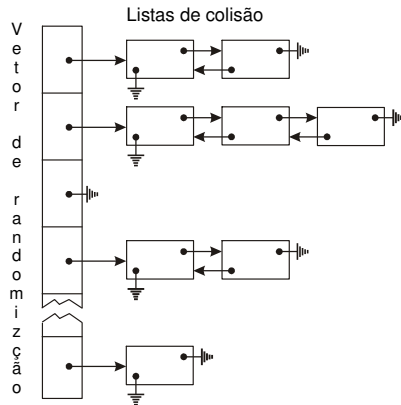
```
- Para todos os *Simbolos* da lista *inx* : $0 \leq inx < n$

```

{
    inx = ObterInxHash(
        Simbolo )
}

```
- cada lista é duplamente encadeada e ordenada segundo *Simbolo*

Exemplo



Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

31

Exemplo



- $n == 1 \rightarrow$ ilegal
- $n == 5$
 - criar símbolos A, B, C, D, E, F, G tal que $ObterHash(\text{simbolo}) == 0$
 - 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 2015

Arndt von Staa © LES/DI/PUC-Rio

32

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 2015

Arndt von Staa © LES/DI/PUC-Rio

33

Exemplo



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

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

34

Laboratório de Engenharia de Software

LES

FIM

Mar 2015

Arndt von Staa © LES/DI/PUC-Rio

35