


Laboratório de Engenharia de Software

# Geração automática de casos de teste

Arndt von Staa  
Departamento de Informática  
PUC-Rio  
Maio 2014



## Especificação

Laboratório de Engenharia de Software


- Objetivo desse módulo
  - discutir uma abordagem de geração automática de suítes de teste executáveis
- Justificativa
  - a criação automática das suítes de teste
    - reduz significativamente o esforço de criação de casos de teste
    - pode alcançar uma grau de confiabilidade bastante superior ao alcançado com a criação manual de suítes de teste

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

2

**Geração automática de massas de teste**




Laboratório de Engenharia de Software

- A geração de massas de teste
  - é trabalhosa
  - requer a obediência a critérios de seleção que, muitas vezes, acabam conduzindo a um volume muito grande de casos de teste
  - não garante que sejam selecionados todos os **casos de teste relevantes**

Mai 2014Arndt von Staa © LES/DI/PUC-Rio3

**Geração automática de massas de teste**



Laboratório de Engenharia de Software

- Seria possível automatizar a geração das massas de teste?
  - reduzir o trabalho de criação de massas de teste?
  - conjugar o gerador com uma variedade de ferramentas de teste?
  - melhorar a eficácia dos testes?

Mai 2014Arndt von Staa © LES/DI/PUC-Rio4

## Geração automática de massas de teste



- Como gerar automaticamente os casos de teste?
  - geração a partir de modelos ?
  - seleção aleatória de dados pré-definidos ?
    - qual a vantagem disso com relação à geração manual de dados ?
  - geração de dados aleatórios ?
  - gerar ou procurar dados segundo regras estabelecidas?
    - procurar dados é necessário quando se trabalha com dados persistentes criados externamente ao artefato sob teste

Mai 2014

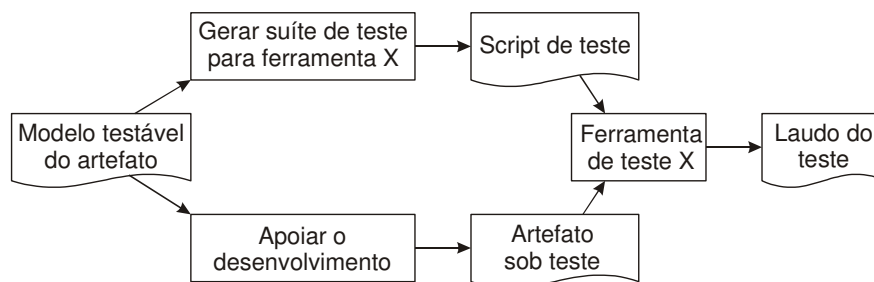
Arndt von Staa © LES/DI/PUC-Rio

5

## Geração a partir de modelos



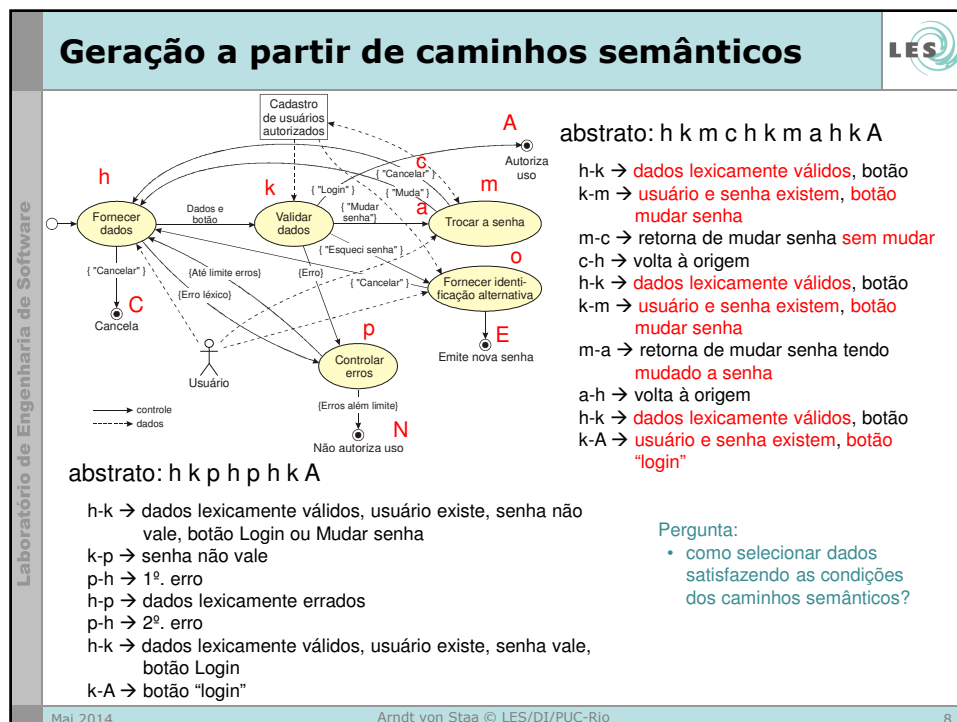
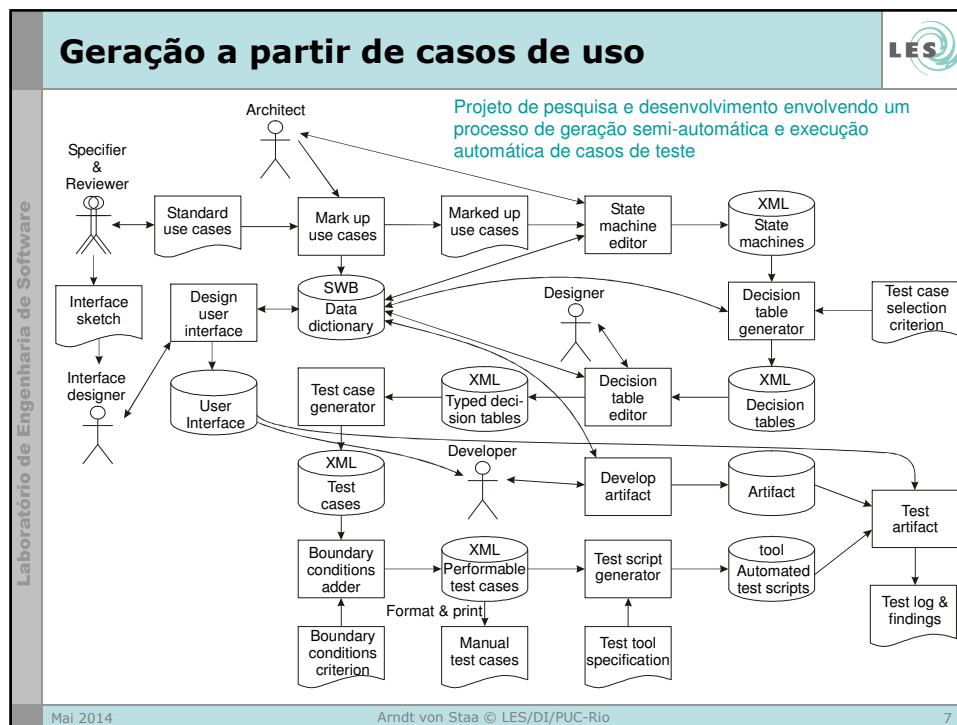
- **Modelo** é uma **especificação abstrata**
  - destina-se a guiar o desenvolvimento
  - pode servir para gerar scripts de teste
  - a automação da geração de suítes de testes leva ao **desenvolvimento dirigido por testes de aceitação**
- Esquema geral da abordagem:



Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

6



Laboratório de Engenharia de Software

## Exemplo de Junit gerado

- Ao invés de escrever o Junit pode-se gerá-lo
  - podem ser utilizadas diversas notações para isso
    - tabelas de decisão
    - máquinas de estado
    - casos de uso
    - tudo isso junto ☺
- Passa-se a poder testar características ao invés de classes ou módulos
  - evidentemente os módulos de teste para testar características poderiam também ser criados à mão

Mai 2014
Arndt von Staa © LES/DI/PUC-Rio
9

Laboratório de Engenharia de Software

## Exemplo de Junit gerado

Nome

Senha

Ingressar

Grupo Condi...	Tipo de Cam...	Identificador	Nome	R1	R2
SSV1;SSV2P	NENHUM	chave	preenche cha...	V	
SSV1;SSV2	TEXT0	chave	chave cadastr...	V	teste
SSV1;SSV3P	NENHUM	senha	preenche sen...	V	
SSV1;SSV3	TEXT0	senha	senha cadastr...	V	teste
SSV1P	CLICÁVEL	button	clica entrar	V	
			Ações		
			loginSuces...	X	
			loginUnsucce...		X
			doeNothing		

Caldeira, L.R.N.: *Geração Semi-automática de Massas de Testes Funcionais a Partir da Composição de Casos de Uso e Tabelas de Decisão*; Dissertação de Mestrado; PUC-Rio; 2010

Mai 2014
Arndt von Staa © LES/DI/PUC-Rio
10

## Exemplo de Junit gerado



```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import com.thoughtworks.selenium.DefaultSelenium;

public class TestLoginSuite
{
    public DefaultSelenium getSelenium(){
        return this.selenium;
    }

    @Test
    public void r1(){
        final DefaultSelenium selenium = getSelenium();
        waitForElement( selenium, "chave" );
        selenium.type( "chave", "teste" );
        waitForElement( selenium, "senha" );
        selenium.type( "senha", "testel23" );
        waitForElement( selenium, "button" );
        selenium.click( "button" );
        Results.loginsSuccessful( selenium );
    }

    @Test
    public void r10(){
        final DefaultSelenium selenium = getSelenium();
        Results.doesNothing( selenium );
    }
}
```

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

11

## Exemplo de Junit gerado



```
@Test
public void r4(){
    final DefaultSelenium selenium = getSelenium();
    waitForElement( selenium, "chave" );
    selenium.type( "chave", "t" );
    waitForElement( selenium, "senha" );
    selenium.type( "senha", "testel23" );
    waitForElement( selenium, "button" );
    selenium.click( "button" );
    Results.loginUnsuccessful( selenium );
}


@Test
public void r5(){
    final DefaultSelenium selenium = getSelenium();
    waitForElement( selenium, "chave" );
    selenium.type( "chave", "t" );
    waitForElement( selenium, "senha" );
    selenium.type( "senha", "teste" );
    waitForElement( selenium, "button" );
    selenium.click( "button" );
    Results.loginUnsuccessful( selenium );
}
```

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

12

## Geração automática de massas de teste




Laboratório de Engenharia de Software

- Testes com dados aleatórios
  - permitem gerar grandes volumes de dados segundo variadas funções de distribuição
    - as funções podem basear-se em gramáticas de geração
  - são úteis quando
    - o número de condições estruturais for grande
    - o número e a complexidade das regras de negócio for grande
    - se deseja que, a cada nova execução, os testes percorram um grande número de diferentes caminhos

Mai 2014
Arndt von Staa © LES/DI/PUC-Rio
13

## Geração automática de massas de teste



Laboratório de Engenharia de Software

- Como saber se o resultado do teste é coerente com os dados gerados?
  - utilizar **assertivas executáveis rigorosas**
    - controlam **todas as variáveis e estados** manipuladas pelas funções
  - utilizar **geradores de oráculos** associados à geração dos dados de teste
    - ex.: aplicar os dados a uma tabela de decisão e extrair o oráculo da tabela
  - aproveitar propriedades conhecidas dos resultados
    - ex.: usar funções inversas capazes de recompor os dados de entrada
  - . . .

Mai 2014
Arndt von Staa © LES/DI/PUC-Rio
14

## Teste com assertivas recordação



- O teste usando assertivas como oráculo depende da **qualidade das assertivas**
  - torna necessária a **introdução de redundâncias no código**
  - requer **assertivas estruturais (quase) completas**
    - devem assegurar a auto-verificação
    - porém, o custo de avaliação de uma assertiva estrutural pode ser muito elevado
  - as assertivas de entrada são **assumidas verdadeiras** ao iniciar e assegurar a corretude do fragmento de interesse
    - precisa-se verificar se isso é verdade → instrumentação
  - devem figurar nas assertivas de entrada **todos os dados, estados e recursos** usados no fragmento de interesse antes de serem redefinidos (alterados, destruídos) neste fragmento
  - devem figurar nas assertivas de saída **todos os dados, estados e recursos** criados, alterados ou destruídos no fragmento de interesse
    - é importante verificar se o efeito do fragmento de código foi atingido através de assertivas de saída sensíveis aos dados de entrada

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

15

## Geração automática de massas de teste



- Como saber que a massa de teste gerada é boa ?
  - usar **mutantes** para avaliar a eficácia do teste assegurada pelo método de geração
  - **medir a cobertura** para avaliar a completeza assegurada pelo método de geração
    - todos os comandos
    - todas as arestas
    - todos (fragmentos de) caminhos

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

16



## Geração automática de massas de teste



- Pode-se gerar **sem regras definidas**?
- Como gerar **com** regras definidas ?
  - geração segundo uma gramática
  - geração *ad hoc*
    - regras intuitivas criadas sob medida para o artefato a ser testado
  - geração a partir de grafos, ou máquinas de estados
    - geram-se os caminhos
    - geram-se dados em acordo com as regras desses caminhos
  - geração usando técnicas de busca em conjuntos complexos
    - muitas vezes baseadas em princípios de inteligência artificial
    - *search based software engineering* (SBSE) aplicado a testes
      - McMinn, P.; "Search-based software test data generation: a survey"; *Software Testing, Verification and Reliability* 14(2); 2004; pp. 105-156
  - ...

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

17

## Gramática "livre de contexto" especificação 1 / 2



```
<gramática>      ::= 0 - [ ( <produção> | <declara_função> ) ] ;
<produção>       ::= <não_terminal> ' ::= '
                  ( <frase> | <expressão lógica> ) ';' ;
<não_terminal>   ::= '<' $nome '>' ;
<frase>          ::= ( <sequência>
                      | <alternativa>
                      | <repetição>
                      | <declara_token> ) ;
<sequência>      ::= 1 - [ <elemento> ] ;
<alternativa>    ::= '(' <frase> 0 - [ '|' <frase> ] ')' ;
<repetição>      ::= <cardinalidade> '[' <frase> ']' ;
<cardinalidade>  ::= ( $num | $num '-' | $num '-' $num ) ;
<elemento>       ::= ( <terminal> | <não_terminal> ) ;
<terminal>      ::= ( $lexema ) ;
```

exemplos de elementos léxicos  
\$xxx – é um conjunto de  
elementos léxicos

lexema – é qualquer coisa retornada pelo analisador léxico, exemplos: '|' ' ::= ' \$num \$nome até \$lexema


Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

18

Laboratório de Engenharia de Software

## Gramática "livre de contexto" especificação 2 / 2



```

<declara_função> ::= <nome_função> '(' <parâmetros> ')'
                    ' ::= ' $expressão_lógica ;
<parâmetros>      ::= <parâmetro> 0 - [ ',' <parâmetro> ] ;
<parâmetro>       ::= $nome ;
<chama_função>    ::= <nome_função> '(' <parâmetros> ')' ;
<nome_função>     ::= $nome ;
<declara_token>   ::= <tipo> <nome_token> ',' $expressão_lógica ;
<tipo>            ::= $idTipo ;
<nome_token>      ::= $nome ;

```

*\$expressão\_lógica* – corresponde à "gramática" de expressões lógicas descrita na aula de assertivas

- indica a regra que o *token* deve satisfazer


Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

19

Laboratório de Engenharia de Software

## Exemplo: gramática para datas (oráculo)



```

<data> ::= <ano> '/' <mês> '/' <dia> ;
<ano>  ::= int a, 1900 <= a && a <= 3000 ;
<mês>  ::= int m, 1 <= m && m <= 12 ;
bissexto( a ) ::= a div 400 || ( a div 4 && ! ( a div 100 ) ) ;
<dia>  ::= int d, ( 1 <= d ) && (
    m ∈ { 1, 3, 5, 7, 8, 10, 12 }    => d <= 31 ,
    m ∈ { 4, 6, 9, 11 }             => d <= 30 ,
    m = 2 && bissexto( a )           => d <= 29 ,
    m = 2 && ! bissexto( a )         => d <= 28 ) ;

```

- Segundo a regra acima um ano corresponde a  $365 + 97 / 400$  dias = 365,2425
- Um ano Tropical medido é aproximadamente 365,24219 dias. Considerando o calendário Gregoriano, o cálculo corresponde a um erro de um dia a aproximadamente cada 3300 anos

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

20

## Geração de datas aleatórias (parcial)



`<ano> ::= int a, 1900 <= a && a <= 3000 ;`

- não vale: `a == 1899`; `a == 3001`
- vale: `a == 1900` ; `a == 1901` ; `a == 2999` ; `a == 3000`; e escolhas randômicas entre 1902 e 2998

`bissexto( a ) ::= a div 400 || ( a div 4 && ! ( a div 100 ) ) ;`

- são bissextos a div 400: 2000, 2400, 2800
- não são bissextos a div 100: 1900, 2100, 2200, ...
- são bissextos a div 4: 1904, 1908, 1912, ...
- não são bissextos: 1901, 1902, 1903, 1905, ...
  - precisa-se criar um gerador capaz de interpretar a regra e gerar dados que satisfaçam e outros que não satisfaçam as regras
  - gerar um número aleatório e aplicar a regra não é suficiente, pois não assegura os casos especiais enumerados acima

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

21

## Geração de dados segundo uma gramática



Exemplo: gerar nomes de um programa Java, C#, C ou C++

`<Nome> ::= <InicioNome> <RestoNome> ;`

`<InicioNome> ::= ( <Letras> | '_' ) ;`

`<RestoNome> ::= 0 - 31 [ <CharCont> ] ;`

`<CharCont> ::= ( <Letras> | <Digitos> | '_' ) ;`

`<Letras> ::= ( <Maiusculas> | <Minusculas> ) ;`

`<Digitos> ::= $Numerais ;`

`<Maiusculas> ::= $UpperCase_ASCII ;`

`<Minusculas> ::= $LowerCase_ASCII ;`

- Como gerar?
  - quais são as distribuições das alternativas?

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

22

## Geração de dados segundo uma gramática



```

<Nome>      ::= <InicioNome> <RestoNome> ;
<InicioNome> ::= ( 92% <Letras>
                  | ' ' ) ;
<RestoNome>  ::= 0 - 31 { 2% 0 ; 30% 1 - 5 ; 33% 6 - 12 ;
                        24% 13 - 22 ; 10% 23 - 30 } [ <CharCont> ] ;
<CharCont>   ::= ( 75% <Letras>
                  | 20% <Digitos>
                  | ' ' ) ;
<Letras>     ::= ( 10% <Maiusculas>
                  | <Minusculas> ) ;
<Digitos>    ::= $Sel( "0123456789" ) ;
<Maiusculas> ::= $Sel( "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ) ;
<Minusculas> ::= $Sel( "abcdefghijklmnopqrstuvwxyz" ) ;
    
```

alguma função de distribuição

A última opção de uma lista possui frequência necessária para se chegar a 100%

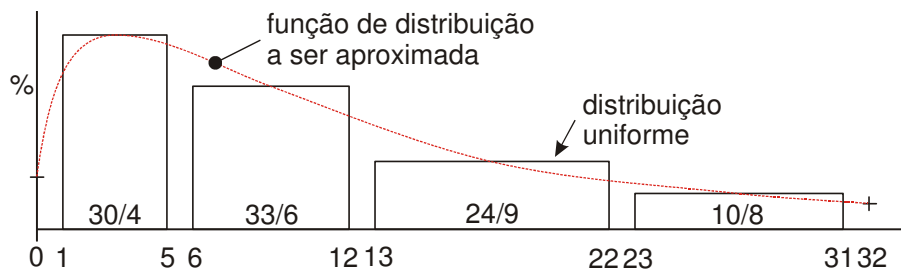
\$Sel usa distribuição uniforme

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

23

## Função de distribuição aproximada



Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

24

## Geração de dados segundo uma gramática



```
static char LETRAS_MIN[ ] = "abcdefghijklmnopqrstuvwxyz" ;
static char LETRAS_MAI[ ] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ;
static char DIGITOS[ ] = "0123456789" ;
static int  dimTamanhos = 5 ;
static struct
{
    int frequenciaTamanho ;
    int tamMin ;
    int tamMax ;
} vtTamanho[ ] =
    { { 2 , 0 , 0 } ,
      { 32 , 1 , 5 } ,
      { 65 , 6 , 12 } ,
      { 89 , 13 , 22 } ,
      { 99 , 23 , 31 } } ;
static int  vtSelecaoTamanho[ ] = { 2, 32, 65, 89, 99 } ;
static int  vtDistribuicaoChar[ ] = { 75, 95 } ;
static int  frequenciaSublinhado = 8 ;
static int  inxCharNome ;
static char Nome[ DIM_NOME ] ;
```

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

25

## Geração de dados segundo uma gramática



```
void Gerar( void )
{
    memset( Nome , 0 , DIM_NOME ) ;
    inxCharNome = 0 ;
    GerarNome( ) ;
} /* end Gerar */

<Nome>      ::= <InicioNome> <RestoNome> ;

void GerarNome( void )
{
    GerarInicioNome( ) ;
    GerarRestoNome( ) ;
} /* end GerarNome */
```

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

26

## Geração de dados segundo uma gramática



```
<InicioNome> ::= ( 92% <Letras>
                    | '_' ) ;
void GerarInicioNome( void )
{
    int Selecao ;

    Selecao = ALT_GerarDistUniforme( 0 , 100 ) ;
    if ( Selecao <= frequenciaLetras )
    {
        GerarLetras( ) ;
    } else
    {
        InserirChar( '_' ) ;
    } /* if */
} /* End GerarInicioNome */
```

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

27

## Geração de dados segundo uma gramática



```
<RestoNome> ::= 0 - 31 { 5% 0 ; 30% 1 - 5 ; 35% 6 - 12 ;
                    17% 13 - 22 ; 10% 23 - 30 } [ <CharCont> ] ;
void GerarRestoNome( void )
{
    int Selecao ;
    int tamGera ;
    int i ;
    Selecao = ALT_GerarFrequencia( dimTamanhos , vtSelecaoTamanho , 100 ) ;
    if ( Selecao == 0 )
    {
        return ;
    } else if ( Selecao < dimTamanhos )
    {
        tamGera = ALT_GerarDistUniforme(
            vtTamanho[ Selecao ].tamMin , vtTamanho[ Selecao ].tamMax ) ;
    } else
    {
        tamGera = 32 ;
    } /* if */

    for ( i = 0 ; i < tamGera ; i++ )
    {
        GerarCharCont( ) ;
    }
} // GerarRestoNome
```

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

28

## Geração de dados segundo uma gramática



```
<CharCont> ::= ( 75% <Letras>
                | 20% <Digitos>
                | '_' ) ;
```

```
void GerarCharCont( void )
```

```
{
```

```
    int Selecao ;
```

```
    Selecao = ALT_GerarFrequencia( 2,
                                    vtDistribuicaoChar , 100 ) ;
```

```
    switch ( Selecao )
```

```
    {
```

```
        case 0 : GerarLetras( ) ;
```

```
                break ;
```

```
        case 1 : GerarChar( DIGITOS ) ;
```

```
                break ;
```

```
        default : InserirChar( '_' ) ;
```

```
                break ;
```

```
    } // switch
```

```
} // GerarCharCont
```

o exemplo não está tratando de  
forma específica as letras  
maiúsculas e minúsculas

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

29

## Geração de dados segundo uma gramática



```
void GerarLetras( )
```

```
{
```

```
    int Selecao ;
```

```
    int caixa = ALT_GerarDistUniforme( 0 , 100 ) ;
```

```
    if ( caixa <= 10 )
```

```
    { Selecao = ALT_GerarDistUniforme(
        0 , strlen( LETRAS_MAI ) - 1 ) ;
```

```
        InserirChar( LETRAS_MAI[ Selecao ] ) ;
```

```
    } else
```

```
    { Selecao = ALT_GerarDistUniforme(
        0 , strlen( LETRAS_MIN ) - 1 ) ;
```

```
        InserirChar( LETRAS_MIN[ Selecao ] ) ;
```

```
    } /* end if */
```

```
} // Gerar Letras
```

```
void InserirChar( char Ch )
```

```
{
```

```
    assert( inxCharNome < DIM_NOME ) ;
```

```
    Nome[ inxCharNome ] = Ch ;
```

```
    inxCharNome++ ;
```

```
} /* End Inserir char */
```

Má prática:  
• todas as constantes  
deveriam ser  
declaradas com um  
nome simbólico

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

30

## Geração de dados no arcabouço C++



- O comando de geração deve disparar o reconhecimento de uma sub-linguagem
  - diversos parâmetros configuram a geração
- Deve integrar a geração e o apoio à diagnose
  - um comando do gerador permite
    - informar até onde devem ser geradas e executadas de forma automática as operações a realizar
    - executam-se os comandos e ao terminar salva-se o estado
    - após é gerado o texto de n comandos entre os quais deve estar aquele que gerou a falha observada
    - para replicar: utiliza-se o estado após gerar normal e aplicam-se os comandos adicionais
    - pode-se agora usar um *trace* ou *debugger* para determinar a causa do problema

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

31

## Versão usada no teste de btree



```
== Generate and test random data
=GenerateTest          Btree01  UseSeg  BtreeHead01
+GenerationControl    10000    1000    1  quantos  numChaves  semente
+VerificationFreq      0        500
+BreakpointHandling    0        7
+ActionDistribution    35        25    40  %ins  %subst  %del
+SizeDistribution       10        2      %doItem  tamLimite
+SizeDistribution       40        15
+SizeDistribution       30        25
+SizeDistribution       20        34
+ParameterListEnd
```

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

32



## Teste usando dados aleatórios, processo

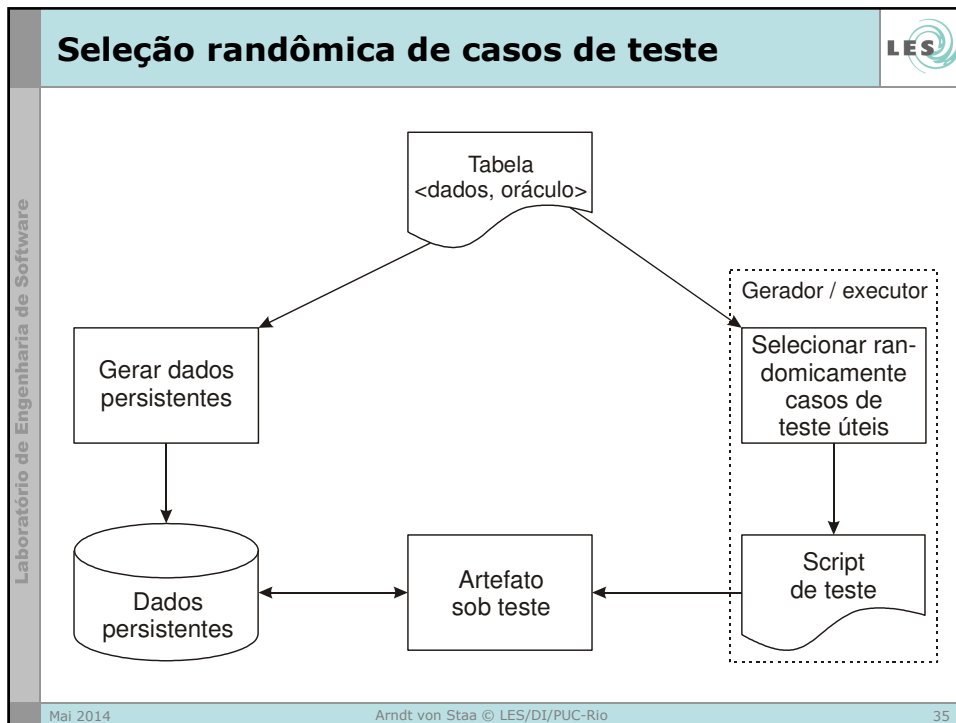


- Desenvolva um módulo de teste específico capaz de gerar **dados** e de selecionar **comandos aleatórios** (chamadas de funções) em acordo com a especificação do módulo
  - deve ser capaz de gerar dados que executem completamente o módulo a ser testado
- **Instrumente** o módulo a testar com contadores para medir a cobertura e outras propriedades
- Utilize como oráculo um **verificador estrutural** e/ou **assertivas pontuais executáveis**
- Ao terminar o teste destrua integralmente as estruturas alocadas para verificar a ocorrência de vazamento de memória ou de outros recursos

## Teste com dados previamente gerados



- Em muitas ocasiões não se pode gerar dados aleatórios irrestritos, exemplo:
  - para testar o componente Login é necessário que o cadastro de usuários esteja disponível
  - entretanto, o módulo "gerente do login" poderia criar um cadastro com dados aleatórios
  - se, ao fazer isso, for criada uma tabela auxiliar, esta tabela poderia ser utilizada pelo testador do componente Login para gerar os dados
  - podem-se criar distribuições que provoquem erros de fornecimento de dados a partir de seleções randômicas dentro da tabela




## Teste usando dados aleatórios: comentários

- Vantagens
  - não se precisa gerar manualmente a massa de teste
    - é gerado um número grande de dados
    - dados exploram uma grande variedade de combinações
  - pode ser repetido com diferentes parâmetros exercitando o módulo de diferentes maneiras
    - como se está utilizando um gerador de números pseudo-aleatórios o teste é repetível se os números aleatórios forem gerados a partir de uma mesma semente
    - mudando a semente mudam-se os caminhos percorridos
  - é rápido
    - a rapidez é proporcional ao volume de dados gerados
  - custo baixo
    - não requer muito trabalho humano

Mai 2014      Arndt von Staa © LES/DI/PUC-Rio      36

**Teste usando dados aleatórios: comentários**




Laboratório de Engenharia de Software

- Desvantagens
  - requer algum trabalho de condicionamento do módulo sob teste
    - instrumentação
    - assertivas estruturais ou outra forma de oráculo
  - requer a programação das funções de geração
  - requer uma inspeção e um teste cuidadoso das funções de geração da massa de teste
    - o módulo de teste específico pode tornar-se complexo

Mai 2014
Arndt von Staa © LES/DI/PUC-Rio
37

**Teste usando dados aleatórios: comentários**




Laboratório de Engenharia de Software

- A forma de teste discutida foi utilizada em
  - componente *btree*
  - componente editor de texto
  - uma vez removidos os defeitos, nenhum defeito novo foi encontrado em mais de 15 anos de uso

Mai 2014
Arndt von Staa © LES/DI/PUC-Rio
38

Laboratório de Engenharia de Software

## Geração automática de suítes de teste



- Alguns comentários
  - muitos "geradores" publicados na literatura geram somente casos de teste abstratos e, algumas vezes, os casos de teste semânticos
    - como exemplos vejam os "geradores" de caminhos a partir de máquinas de estado ou de diagramas de fluxo
  - por enquanto os geradores são projetados para determinadas classes de problemas
  - uma grande parte destes geradores são resultados de projetos de pesquisa ainda não convertidos em produtos
  - vale a pena procurar literatura tratando de soluções específicas

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

39

Laboratório de Engenharia de Software

## Referências bibliográficas



- Caldeira, L.R.N.; *Geração de Massas de Teste para Aplicações WEB a Partir da Composição de Casos de Uso e Tabelas de Decisão*; Dissertação de Mestrado, DI/PUC-Rio, 2010
- Edwards, S.H.; "A Framework for Practical, Automated Black-Box Testing of Component-Based Software"; *Software Testing, Verification and Reliability* 11(2); New York: John Wiley & Sons; 2001; pages 97-111
- Heumann, J.; "Generating Test Cases from Use Cases"; *The Rational Edge e-zine* www.ibm.com/developerworks/rational/rationaledge/ International Business Machines; 2001 ; Buscado em: 22/jan/2009; URL: www.ibm.com/developerworks/rational/library/content/RationalEdge/jun01/GeneratingTestCasesFromUseCasesJune01.pdf
- Lachtermacher, L.; *O uso de tabelas de decisão para a automação da geração e da execução de casos de teste*; Dissertação de Mestrado, DI/PUC-Rio, 2010
- Pacheco, C.; Lahiri, S.K.; Ernst, M.D.; Ball, T.; "Feedback-directed Random Test Generation"; *Proceedings of the 29th International Conference on Software Engineering - ICSE'07*, Minneapolis, 2007; Los Alamitos, CA: IEEE Computer Society; 2007; pags 75-84
- Wanderley, C.G.; *Ferramenta de Auxílio à Automação de Testes de Interfaces Gráficas Desenvolvidas com C++*; Relatório de Iniciação Científica, PIBIC/PUC-Rio; 2011

Mai 2014

Arndt von Staa © LES/DI/PUC-Rio

40

Laboratório de Engenharia de Software

LES

FIM

Mai 2014Arndt von Staa © LES/DI/PUC-Rio41