


Laboratório de Engenharia de Software

Aula 05

Introdução à Teste de Módulos

Alessandro Garcia
LES/DI/PUC-Rio
Agosto 2010

Avisos



Laboratório de Engenharia de Software

- Próxima semana:
 - não falem pois haverá um exercício em sala (vale parte da nota do T1)

Ago 2009


Alessandro Garcia © LES - DI/PUC-Rio

2 / 26

Laboratório de Engenharia de Software

Princípios Discutidos até aqui...

- Cada módulo deveria implementar uma única abstração
- Separação: interface da implementação de um módulo
- Maximizar a declaração e documentação de interface implícita e explícita
- Garantir corretude sintática e semântica em conexões entre funções/módulos
- Promover encapsulamento e definir TADs
- Minimizar acoplamento entre módulos
- Maximizar coesão dos elementos internos de cada módulo




Ago 2009
Alessandro Garcia © LES - DI/PUC-Rio
3 / 26

Laboratório de Engenharia de Software

Especificação

- Objetivo dessa aula
 - Definir teste de software
 - Apresentar como definir um módulo de teste manual




- Referência básica:
 - Monografia: *Arcabouço para a Automação de Testes de Programas Redigidos em C*; contido no arquivo **TesteAutomatizado.zip** acessível para *download* através do site da disciplina, aba: **Software**

4 / 30
LES/DI/PU

Laboratório de Engenharia de Software

Sumário




- Faltas, erros e falhas
- Por que testar módulos
- O que é testar programas
- Módulo controlador de teste
- Teste manual
 - controlador de teste manual

5 / 30LES/DI/PU

Laboratório de Engenharia de Software


Até aqui...



- Princípios de modularidade para programação que levam a programas de qualidade satisfatória *por construção*
 - *Módulos com interfaces bem definidas*
- É desnecessário então conduzir um teste sistemático?

6 / 30LES/DI/PU

Até aqui...




Laboratório de Engenharia de Software

- Princípios de modularidade para programação que levam a programas de qualidade satisfatória *por construção*
 - Módulos com interfaces bem definidas
- É desnecessário então conduzir um teste sistemático?
 - Não**, mesmo programas modulares por construção contém faltas
- Na verdade: **nenhuma** técnica de construção ou mesmo de testes pode assegurar que o software não tenha faltas
- Objetivo**: aplicar sistematicamente execução de testes para maximizar chances de obter software de **qualidade satisfatória**
 - encontrar **faltas remanescentes** no software

7 / 30

LES/DI/PU

Por que testar módulos?



Laboratório de Engenharia de Software


- Programas podem conter **defeitos** (ou **faltas**) que, quando exercitados, provocam **erros de funcionamento**. Quando observados estes erros passam a ser **faltas**.
 - defeito**: código errado (**falta**: a mesma coisa que defeito)
 - erro**: estado diferente do esperado ou desejado, ainda não observado
 - falha**: estado diferente do esperado ou desejado, **observado**
- Faltas podem ser introduzidos na medida que **evoluimos os módulos**
 - especificação incompleta ou **evolutiva**

8 / 30

LES/DI/PU

Laboratório de Engenharia de Software

Por que módulos podem conter faltas?




- **Humanos são falíveis** (ferramentas também ☹), logo:
 - podem se enganar ao redigir o código e inserir defeitos
 - podem se enganar ao especificar o que se deseja que o módulo faça
 - a implementação correta passa a ser uma solução correta do problema errado, ou seja a solução estará incorreta
 - podem se enganar ao especificar os requisitos de qualidade da solução
 - a implementação correta cria problemas para o usuário, ou seja a solução estará incorreta, exemplos
 - difícil de utilizar
 - tempo de resposta excessivamente demorado
 - não atende à demanda de serviço

9 / 30

LES/DI/PU

Laboratório de Engenharia de Software

Exemplo



- O uso de muitas construções da linguagem de programação (C, por exemplo) facilitam a inserção de **faltas** em programas
 - atribuição vs. comparação de valores
 - confusão: comparadores de valores
- As faltas somente se manifestarão na forma de **erros** quando certas entradas (valores) forem acionados
- **POR EXEMPLO....**

10 / 30

LES/DI/PU

Exemplo – falha introduzida no uso...



- de cláusulas **if**....

(versão simplificada de uma função de mapeamento)

```
int map( int i ) {  
    if( i > 0 )  
        if( i > 10 )  
            return 10;  
    else  
        return -1;  
    return 0;  
}
```

A intenção do programador é a seguinte:

i > 10	retornar 10
i in 1..10	retornar 0
i <= 0	retornar -1



- Qual é o resultado de chamar a função *map* com o valor de argumento **11**?

LES/DI/PUC-Rio

11 / 30

Exemplo



```
int map( int i ) {  
    if( i > 0 )  
        if( i > 10 )  
            return 10;  
    else  
        return -1;  
    return 0;  
}
```

A intenção do programador é a seguinte:

i > 10	retornar 10
i in 1..10	retornar 0
i <= 0	retornar -1



- Qual é o resultado de chamar a função *map* com o valor de argumento **5**?

LES/DI/PUC-Rio

12 / 30

Exemplo



`public int map(int i) {`
 `if(i > 0)`
 `if(i > 10)`
 `return 10;`
 `else`
 `return -1;`
 `return 0;`
`}`

associação desejada (green box around the outer if)

associação real (red box around the inner if)

falta ou defeito (red arrow pointing to the inner if)

Qual é a falta introduzida pelo programador?

- Qual é o resultado de chamar a função *map* com o valor de argumento -3?

A intenção do programador é a seguinte:

$i > 10$	retornar 10
$i \text{ in } 1..10$	retornar 0
$i \leq 0$	retornar -1

O comportamento real é:

$i > 10$	retornar 10
$i \text{ in } 1..10$	retornar -1
$i \leq 0$	retornar 0

O compilador, na prática, associa a cláusula *else* com o if interno

LES/DI/PUC-Rio

13 / 30

Exemplo



`public int map(int i) {`
 `if(i > 0)`
 `if(i > 10)`
 `return 10;`
 `else`
 `return -1;`
 `return 0;`
`}`

associação desejada (green box around the outer if)

associação real (red box around the inner if)

falta ou defeito (red arrow pointing to the inner if)

Quais são as possíveis situações de erro?

Erros: momento em que tais retornos são gerados

A **intenção** do programador é a seguinte:

$i > 10$	retornar 10
$i \text{ in } 1..10$	retornar 0
$i \leq 0$	retornar -1

O comportamento real é:

$i > 10$	retornar 10
$i \text{ in } 1..10$	retornar -1
$i \leq 0$	retornar 0

O compilador, na prática, associa a cláusula *else* com o if interno

LES/DI/PUC-Rio

14 / 30

Exemplo



public int map(int i) {
 if(i > 0)
 {
 if(i > 10)
 return 10;
 else
 return -1;
 return 0;
 }
}

associação desejada (green box around the first if)

associação real (red box around the nested if-else)

falta ou defeito (red arrow pointing to the nested if-else)

Erros: momento em que tais retornos são gerados

FALHA: pode ser observada pelo usuário ou administrador...

A **intenção** do programador é a seguinte:

i > 10	retornar 10
i in 1..10	retornar 0
i <= 0	retornar -1

O comportamento real é:

i > 10	retornar 10
i in 1..10	retornar -1
i <= 0	retornar 0

O compilador, na prática, associa a cláusula *else* com o if interno

LES/DI/PUC-Rio

15 / 30

O que é testar um módulo?



- **Teste é** uma das técnicas **dinâmicas** de controle de qualidade
- Um teste é um **experimento controlado** em que se confronta o comportamento **observado** com o comportamento **esperado**
 - **teste de correitude:** existem diferenças entre o especificado e o esperado?
- Como fazer isso?
 - uma **forma de teste** é submeter o módulo a dados escolhidos e comparar o **resultado obtido** com o **resultado esperado**, calculado a partir da especificação e dos dados fornecidos
 - Ex.: comparação em que o valor obtido não está dentro dos limites de tolerância aceitáveis
 - por exemplo, $0 \geq \text{Nota} \leq 10$

16 / 30

LES/DI/PU

Como testar um módulo?

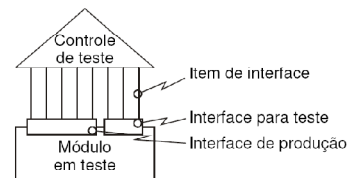


- Para testar um módulo tornam-se necessários
 - um **módulo controlador do teste** desenvolvido para testar o módulo sob teste
 - o módulo controlador **exercita** o **módulo sob teste** através de sua **interface**
 - uma **massa de teste**, isto é um conjunto de **casos de teste**

Como testar um módulo?

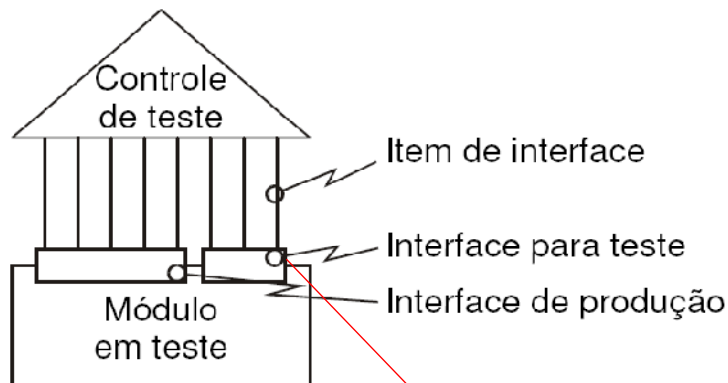


- Uso de um **módulo controlador do teste** desenvolvido que auxilia a execução de casos de teste para testar um módulo sob teste



- **Três formas:**
 - **Teste manual:** controlador exibe um menu e comparação é feita pelo próprio testador à olho nu
 - **Teste de comparação automatizado:**
 - Casos de teste são implementados em C
 - Casos de teste são *redigidos em scripts* em uma linguagem com uma sintaxe própria

O que é um módulo controlador do teste?



deve ser facilmente removível;
instrumentação: discutido em aulas futuras

19 / 30

LES/DI/PU

O que é uma interface de um módulo?



- Considerando a linguagem C, é um **módulo de definição** (módulo cabeçalho, *header file*, arquivo .h)
- Exemplo, módulo árvore – opera sobre uma única árvore:

item da interface

```
ARV_tpCondRet ARV_InserirEsquerda( char Valor ) ;  
ARV_tpCondRet ARV_InserirDireita( char Valor ) ;  
ARV_tpCondRet ARV_IrEsquerda( void ) ;  
ARV_tpCondRet ARV_IrDireita( void ) ;  
ARV_tpCondRet ARV_ObterValorCorrente( char * pValor ) ;
```

20 / 30

LES/DI/PU

O que seria um controlador?



- Exemplo de organização para teste manual
- O módulo controlador de teste exibe o menu de escolha do comando de teste:

```
1 Inserir nó à esquerda
2 Inserir nó à direita
3 Ir para nó filho à esquerda
4 Ir para nó filho à direita
5 Obter valor do nó corrente
6 Exibir a árvore em formato parentetizado
99 Terminar
Escolha a opção:
```

21 / 30

LES/DI/PU

O que seria um controlador?



- Escolhida a opção entra-se em um **switch** e realiza-se a operação. Exemplo para uma das funções:

```
switch ( idAcao )
{
    ...
    case AcaoInserirEsq:
        printf( "    Inserir aa esquerda" ) ;
        printf( "\n    Forneça o valor do nó a inserir" ) ;
        scanf( " %10s" , Valor ) ;
        CondRet = ARV_InserirEsquerda( Valor ) ;
        printf( "\nCondicao de retorno: %d\n" , CondRet ) ;
        break ;
    ...
} /* switch */
```

receber valor do nó

item de interface sendo testado

imprime-se resultado que será comparado visualmente pelo testador com o resultado esperado

22 / 30

LES/DI/PU

Laboratório de Engenharia de Software


Qual seria um cenário de realização de testes?

- Numa forma **indisciplinada** o testador:
 1. ativa o programa
 2. menu de teste de itens de interface é disponibilizado
 3. seleciona uma opção (digitando 99 termina a execução)
 4. fornece os dados solicitados
 5. **compara visualmente** o resultado obtido com o resultado que ele **imagina** deveria ser retornado
 6. **procura corrigir** o programa caso o resultado esteja errado
 7. repete a partir de **1**

23 / 30
LES/DI/PU

Laboratório de Engenharia de Software

Como detectar discrepâncias?



- comparação resultando em diferenças entre o esperado
 - por exemplo, obtido em processamento de inteiros
- comparação em que o valor obtido não está dentro dos limites de tolerância aceitáveis
 - por exemplo, $0 \geq \text{Nota} \leq 10$
- estrutura de dados que não satisfaz as suas assertivas estruturais ou regras de negócio inválidas
 - por exemplo: nós de uma árvore convencional com *dois pais*, aluno pode ser cadastrado em um número maior de disciplinas que o permitido
- um arquivo, ou base de dados, cujo conteúdo e estrutura não corresponde ao esperado
 - Por exemplo, colunas adicionais em tabelas de dados persistentes

24 / 30
LES/DI/PU

Laboratório de Engenharia de Software

Vantagens e desvantagens do teste manual

- Vantagens
 - É relativamente simples de programar.
 - É mais fácil verificar a corretude caso esta se baseie em valores aproximados
 - evidentemente, isto requer que o testador saiba determinar quais as aproximações aceitáveis
- Desvantagens
 - O usuário **não sabe quando testou tudo** que deveria ser testado
 - teste incompleto pode deixar defeitos remanescentes
 - O usuário **imagina** o resultado esperado
 - O usuário realiza testes que **repetem condições já testadas**
 - teste redundante adiciona custo sem contribuir para descobrir novos defeitos
 - O usuário corrige imediatamente ao encontrar uma falha
 - após eliminar o defeito que provocou a falha, o **módulo precisa ser retestado**
 - correção imediata aumenta o número de retestes

LES

25 / 30
LES/DI/PU

Laboratório de Engenharia de Software

Exemplo

- Exiba o exemplo árvore com teste manual
 - diretório: \autotest\manual
- Não esquecer de antes de executar o programa
 - Executar o gmake antes: **gmake /CExemploManual.COMP**
 - Compilar todos os arquivos: **CompilaTudo.bat**
- Executar ExemploManual.exe

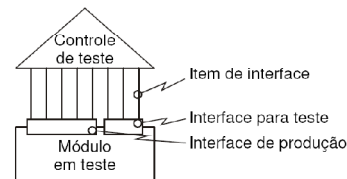
LES

26 / 30
LES/DI/PU

Como testar um módulo?



- Uso de um **módulo controlador do teste** desenvolvido que auxilia a execução de casos de teste para testar um módulo sob teste



- **Três formas:**
 - **Teste manual:** controlador exibe um menu e comparação é feita pelo próprio testador à olho nu
 - **Teste de comparação automatizado:**
 - Casos de teste são implementados em C
 - Casos de teste são *redigidos em scripts* em uma linguagem com uma sintaxe própria

27 / 30

LES/DI/PU

Como reduzir o custo do reteste?



- **Automatizar a execução** dos testes: implementar um programa (controlador) que...
 - estabelece e executa a sequência de comandos de teste em uma espécie de programa
 - ... Quantas vezes for necessário...
 - ... Mesmo quando módulo é modificado ou incrementado mais tarde
 - emitir um relatório da execução do teste (laudo)
- Automação dos testes é **mais efetivo** na procura de **classes específicas de faltas**
 - devido ao **reuso dos casos de teste**

28 / 30

LES/DI/PU

Exemplo simplório



ativação do item de interface sendo testado

instância de caso de teste

compara obtido com o esperado

```
. . .
ContaCaso ++ ;
if ( CriarArvore( ) != ARV_CondRetOK )
{
    printf( "\nErro ao criar árvore" ) ;
    ContaFalhas ++ ;
}
ContaCaso ++ ;
if ( InserirEsquerda('a' ) != ARV_CondRetOK )
{
    printf( "\nErro ao inserir nó raiz da árvore" ) ;
    ContaFalhas ++ ;
}
ContaCaso ++ ;
if ( IrPai( ) != ARV_CondRetEhRaiz )
{
    printf( "\nErro ao ir para pai de nó raiz" ) ;
    ContaFalhas ++ ;
}
. . .
```

29 / 30

LES/DI/PU

Por que é simplório?



- O código é muito **repetitivo**
 - viola a regra de evitar duplicações de código
- O **módulo controlador** do teste pode tornar-se muito **extenso**
 - dificulta verificar se o teste é um bom teste
- As mensagens impressas **não explicitam o porquê** da falha
 - quais foram os valores que causaram a mensagem?
- **Não identifica** com precisão o **objetivo** de cada caso de teste
 - o leitor do código precisa inferir o objetivo a partir da leitura
- O código **não produz um laudo** do teste

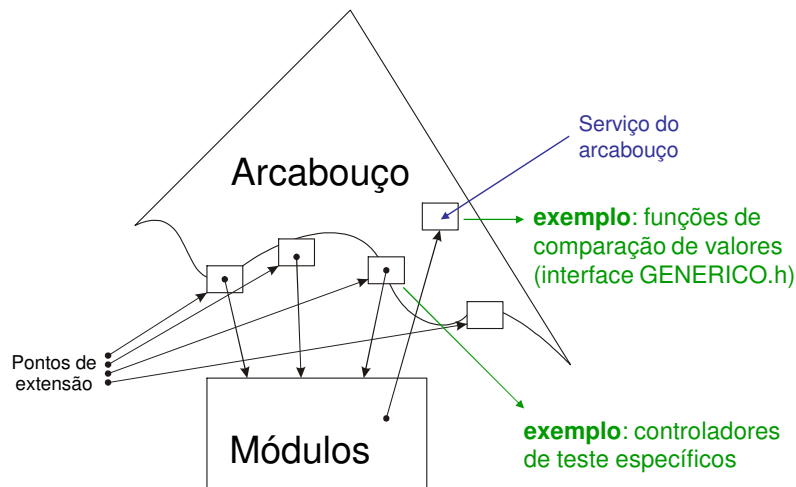
30 / 30

LES/DI/PU

Como reduzir o volume de repetição?



- Criar um **arcabouço** (framework) com as **funções de comparação**.



31 / 30

LES/DI/PU

Trabalho



- Referência básica:
 - Monografia: *Arcabouço para a Automação de Testes de Programas Redigidos em C*; contido no arquivo **TesteAutomatizado.zip** acessível para *download* através do site da disciplina, aba: **Software**
- Comecem a instalar o arcabouço
 - Seção 2 da monografia dá todos os passos de instalação
 - Ler o arquivo "...readme" para verificar como instalar
 - O arcabouço precisa ser compatível com a versão da plataforma de desenvolvimento utilizada
 - portanto, antes de utilizá-la, ela precisa ser recompilada
 - "Recomenda-se fortemente..." = leia-se "DEVE LER"
 - Comece a utilizar os exemplos e arcabouço de testes...
- Enunciado já está no sítio da disciplina

Ago 2009

Alessandro Garcia © LES - DI/PUC-Rio

32 / 26



Aula 05

Introdução à Teste de Módulos

Alessandro Garcia
LES/DI/PUC-Rio
Agosto 2011