


Laboratório de Engenharia de Software

Teste Estrutural 3

Arndt von Staa
Departamento de Informática
PUC-Rio
Maio 2015

Especificação



Laboratório de Engenharia de Software


- Objetivo desse módulo
 - discutir teste de exceções e o uso de módulos dublê.
- Justificativa
 - Exceções são uma grande fonte de problemas e precisam ser adequadamente testados.
 - Muitos módulos (classes, objetos) dependem de outros módulos para que possam realizar os seus serviços. Diversas razões tornam mais interessante utilizar módulos dublê no lugar de módulos de produção.

Módulo dublê: módulo que, durante os testes, ocupa o lugar do de produção (*test double*, analogia: *stunt double*)

Mai 2015Arndt von Staa © LES/DI/PUC-Rio2

Laboratório de Engenharia de Software

Teste de exceções




- O que são exceções?
 - exceções são na realidade “**retornos nomeados**”
 - ao executar o **throw** a pilha é desfeita até encontrar um **catch** correspondente ao tipo (classe) do objeto **thrown**
 - retorna (**throw**) sem saber se terá alguém (**catch**) que tratará
 - retorna sem saber quem tratará
 - podem existir, na pilha de execução, vários **catch** para uma mesma exceção
 - retorna sem saber como será tratada
 - o tratador é determinado pelo tipo da exceção (classe)
 - o tratador ativado pode não corresponder ao tratamento a ser dado
 - quem escreve o tratador possivelmente não sabe em que condições específicas ocorre o **throw**
 - exceções são caras em termos de
 - recursos computacionais requeridos
 - dificuldade de assegurar a corretude da execução
 - considerando a variedade de possíveis cenários

Mai 2015
Arndt von Staa © LES/DI/PUC-Rio
3

Laboratório de Engenharia de Software

Teste de exceções




- Quando usar exceções?
 - **abordagem ortodoxa**: observar ocorrência de um erro, exemplos:
 - assertiva não vale
 - dispositivo não existe, mas deveria existir
 - faltou memória
 - anomalia na sequência de execução
 - falha **transiente**: de hardware ou de plataforma
 - componente interno, ex. base de dados, recebeu dados não válidos
 - **abordagem não ortodoxa**: um instrumento de controle da execução
 - utiliza exceções como condição de retorno

Mai 2015
Arndt von Staa © LES/DI/PUC-Rio
4

Laboratório de Engenharia de Software

Teste de exceções




- Quando não utilizar?
 - **abordagem ortodoxa**: ao invés de exceção utilizam-se **condições de retorno** sempre que a condição de execução for esperada no processamento normal, exemplos
 - arquivo a ser aberto não existe
 - fim de arquivo
 - usuário forneceu dado errado
 - **abordagem não ortodoxa**: sem restrição

Mai 2015
Arndt von Staa © LES/DI/PUC-Rio
5

Laboratório de Engenharia de Software

“Liberalização” da abordagem ortodoxa




- **Abordagem “menos” ortodoxa**:
 - usar exceção ao invés de condição de retorno se o tratador (**catch**) fica “longe” do observador (**throw**)
 - bibliotecas
 - arcabouços
 - erros de uso com tratamento **necessariamente** realizado no contexto do controle da interface do usuário
 - exceções “pulando barreiras” de arquitetura (componentes) são em geral **anomalias** (*bad smells*)
 - » frequentemente implicam problemas de manutenção

Mai 2015
Arndt von Staa © LES/DI/PUC-Rio
6

Laboratório de Engenharia de Software

Teste de exceções




- Problemas inerentes a exceções
 - exceções são propensas a defeitos
 - `throw` não tem `catch` correspondente
 - o programa é cancelado
 - erro de **casamento** entre `throw` e `catch`
 - a exceção sinalizada é capturada por um tratador incorreto
 - a exceção sinalizada é engolida (*swallowed*)
 - » `catch` faz nada
 - o tratador meramente registra no *log*, sem tentar recuperar nem dar *roll back* para um estado correto
 - infelizmente isso pode algumas vezes ser a única coisa possível
 - defeito comum é seguir em frente como se nada tivesse acontecido
 - a exceção deveria pelo menos conter um volume de dados que permita saber o que ocorreu
 - » diagnóstico na **primeira ocorrência de uma falha**

Skwire, D.; Cline, R.; Skwire, N.; *First Fault Software Problem Solving: A Guide for Engineers, Managers and Users*; Ireland: Opentask; 2009

Mai 2015
Arndt von Staa © LES/DI/PUC-Rio
7

Laboratório de Engenharia de Software

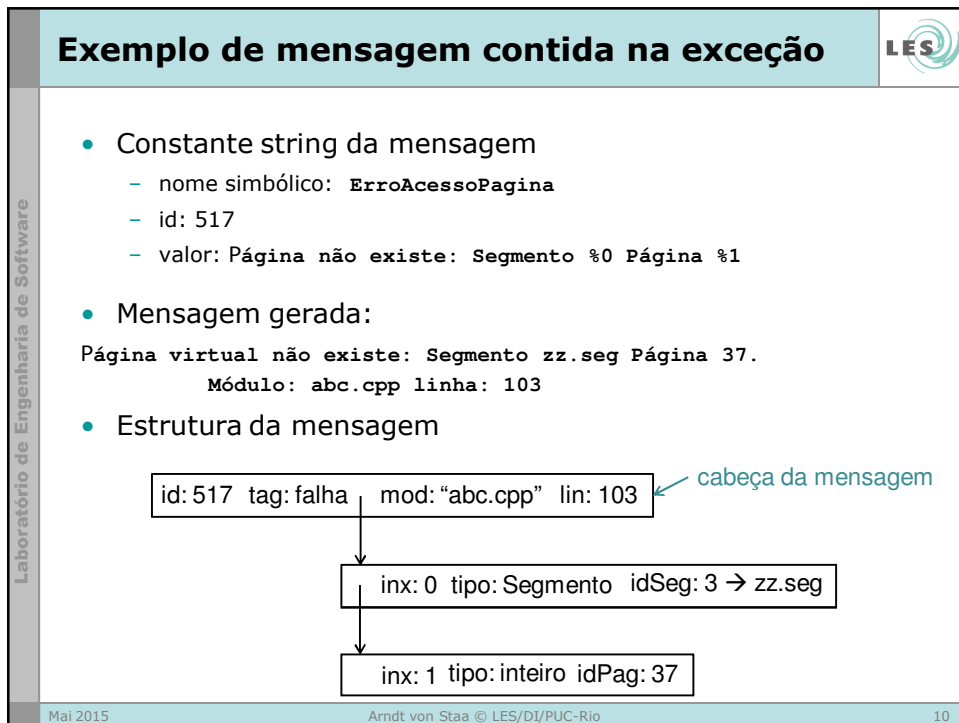
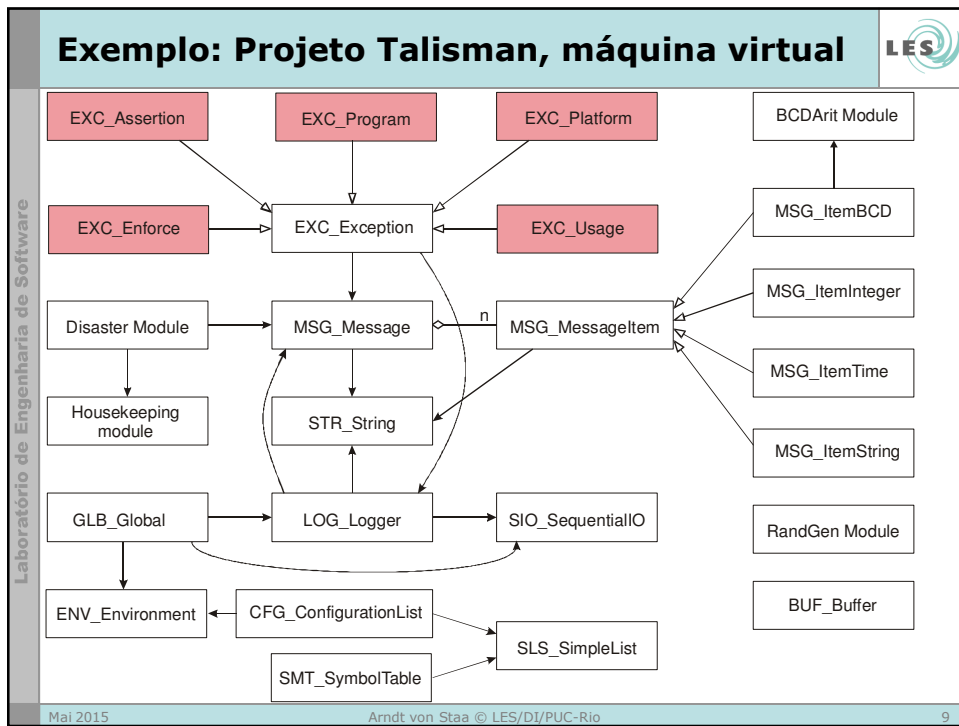
Teste de exceções



- Projetar para **testabilidade**
 - antes de iniciar o desenvolvimento estabeleça um padrão de implementação para exceções
 - idealmente deveria ser válido para todos os projetos da organização
 - simplifique a captura (`catch`) A
 - evitar uma lista longa de `catches`
 - detalhe a causa
 - assegurar que se saiba exatamente a causa, conflita com A
 - determine onde foi e por que foi sinalizada a exceção
 - todas as falhas deveriam ser logadas
 - diversos erros de uso deveriam ser logados, ex. login errado
 - forneça informação de contexto do ponto da sinalização
 - ex. pilha de execução, variáveis do objeto

Araújo, T.P.; Wanderley, C.G.; Staa, A.v.; "An introspection mechanism to debug distributed systems"; 26o SBES - Simpósio Brasileiro de Engenharia de Software; Porto Alegre, RS: Sociedade Brasileira de Computação; 2012; pags 21-30

Mai 2015
Arndt von Staa © LES/DI/PUC-Rio
8



Exemplo: Estrutura de mensagem



- Cada tipo de exceção estabelece convenções de mensagens
- Cada exceção referencia uma **estrutura de mensagem**
- A estrutura de mensagem contém
 - o id da mensagem, constante *string*
 - a constante contém marcadores para a inserção de itens de mensagem
 - uma lista de 0 ou mais **itens da mensagem**
 - os itens informam valores que detalham a mensagem
 - ao gerar a mensagem da exceção, são automaticamente adicionados os itens *nome do módulo* e *número da linha de código* onde se encontra a macro **EXC_...**
- A operação **ToString()** aplicada ao objeto exceção gera o *string* da mensagem, substituindo os marcadores pelos correspondentes itens

Mai 2015

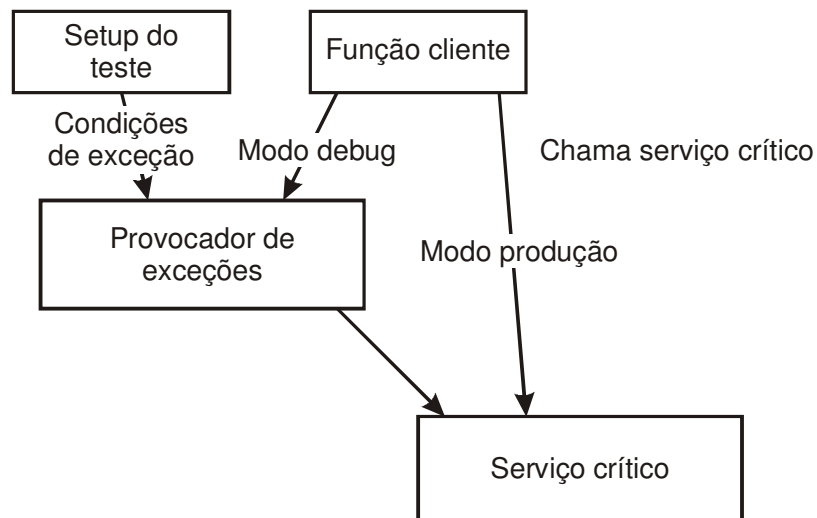
Arndt von Staa © LES/DI/PUC-Rio

11

Teste de exceções



- Como testar exceções?

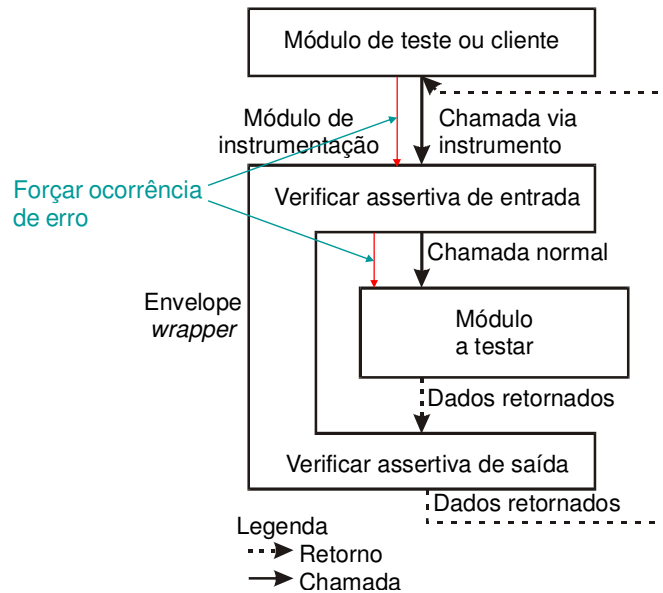


Mai 2015

Arndt von Staa © LES/DI/PUC-Rio

12

Teste de exceções: instrumentação



Teste de exceções: arcabouço teste C++



```

46 == Add numbers of different size, generates exception "loss of data"
47 =ConvertIntToBCD      0 4 -3785328
48 =ConvertIntToBCD      1 5 108303031
49 =Add                  0 1
50 =ExceptionProgram     0 363 'c'
...
106 == Convert 10 siz 1 to BCD, generates exception overflow
107 =ConvertIntToBCD     0 1 10
108 =ExceptionProgram    0 361 'c'

-----
46 == Add numbers of different size, generates exception "loss of data"
>>> 1 Line 49 Tester caught a program exception >>
      ERROR: Loss of data while assigning a BCD number.
      Exception thrown in Line: 536 File: ..\sources\bcdarit.cpp
<<< 0 Line 50 Expected exception has been ignored.

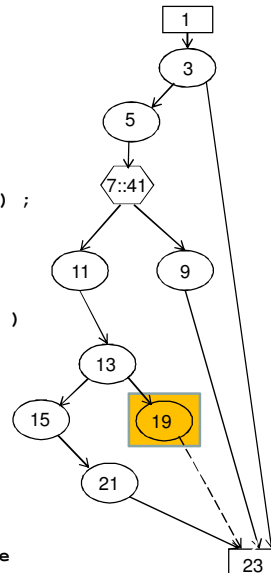
106 == Convert 10 siz 1 to BCD, generates exception overflow
>>> 1 Line 107 Tester caught a program exception >>
      ERROR: Overflow in BCD conversion
      Exception thrown in Line: 161 File: ..\sources\bcdarit.cpp
<<< 0 Line 108 Expected exception has been ignored.
    
```

Grafo de estrutura considerando try-catch



```

1. void VMC_PageFrame :: WritePageFrame (
2. {
3.   if ( changeLevel < VMC_NOT_CHANGED )
4.   {
5.     try
6.     {
7.       SEG_SegmentRoot::GetRoot() -> WritePage(
8.         idSegment , idPage , pageValue ) ;
9.       changeLevel = VMC_NOT_CHANGED ;
10.    }
11.    catch ( EXC_CException * pExc )
12.    {
13.      if ( changeLevel == VMC_IGNOREABLE_CHANGE )
14.      {
15.        changeLevel = VMC_NOT_CHANGED ;
16.        delete pExc ;
17.      } else
18.      {
19.        throw pExc ;
20.      } /* if */
21.    } /* end catch */
22.  } // end if
23. } // End: VMF !Write page value contained in frame
    
```



Grafo de estrutura considerando try-catch

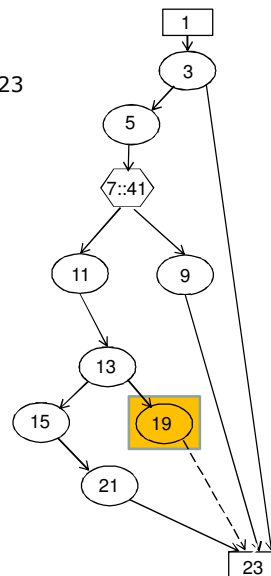


Expressão

1: 3 (5 [41] (11 13 (15 21 | 19 23e) | 9) | ϕ) 23

Caminhos

- 1: 3 23
 - página não está marcada como alterada
- 1: 3 5 [41] 9 23
 - alterada, gravada sem problema
- 1: 3 5 [41] 11 13 15 21 23
 - alterada, exceção ao gravar, alteração descartável
- 1: 3 5 [41] 11 13 19 23e
 - alterada, exceção ao gravar, exceção a propagar



Grafo de chamada considerando try-catch



- Problemas
 - nem sempre se conhecem todas as exceções que poderão ser geradas
 - a lista de `catch` poderá estar incompleta
 - nem sempre se sabe se passará uma exceção ou não
 - não existe `try-catch` no código
 - nem sempre as exceções são capturáveis
 - o padrão 1998 de C++ não explicita exceções de sistema, e.g. falta de memória
 - neste caso o `catch` tenderá a ser um `catch all`
 - o `catch` de uma exceção pode estar (frequentemente está) em um módulo desconhecido ao módulo que contém o `throw`
- Solução proposta
 - programação defensiva, e
 - criar e obedecer a padrões ao programar exceções


Teste de exceções, critérios de cobertura



- Assegurar que
 - cada `throw` foi executado pelo menos uma vez no conjunto de todos os casos de teste
 - cada `catch` foi executado pelo menos uma vez no conjunto de todos os casos de teste
- Problema
 - isso pode ser “fácil” ao testar módulos
 - mas é difícil ao testar construtos complexos
 - o ponto em que se encontra o `throw` pode distar muito do ponto em que se encontra o `correspondente catch` (caminho longo no grafo de chamadas)
 - pode ser difícil forçar a ocorrência da causa
 - muitas vezes o caminho é não realizável
 - pode ser difícil forçar causas detectadas pelo interpretador (Java, C#) ou pelo run-time (C++) ou por bibliotecas

Laboratório de Engenharia de Software

Teste de exceções, padrão de programação




- Recursos locais e exceções
 - ao terminar um método através de uma exceção todos os recursos ancorados localmente devem ser liberados
 - infelizmente é frequente não serem liberados
 - vazamento de recursos
- Solução genérica
 - crie um **objeto de controle** do(s) recurso(s)
 - o construtor (ou um preenchedor – *build*) aloca o(s) recurso(s)
 - o destrutor desaloca os recurso(s)
 - agora basta assegurar que o objeto seja criado e destruído
 - idealmente dentro de um método que inicia a manipulação do(s) recurso(s)

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

Laboratório de Engenharia de Software

Teste de exceções, padrão de programação



Solução Java:

- use a construção `try{ ... } finally{ ... }`
 - o objeto de controle dos recursos é alocado antes do bloco `try`
 - o bloco `try` contém o código, ou chama métodos que potencialmente podem gerar exceções
 - frequentemente o `throw` está em algum método chamado a partir do corpo do `try`
 - o objeto de controle dos recursos é liberados no bloco `finally`
 - o bloco `finally` sempre será executado, mesmo se não ocorrer uma exceção
 - o uso de blocos `catch` é opcional

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

Teste de exceções, padrão de programação



- Solução C++:

- sempre que possível use **objetos locais** ao invés de referências para objetos dinâmicos

```
AlgumaClasse algumObjeto ;
```

- se necessário crie um **struct** encapsulado no método e que contém as referências, o construtor e o respectivo destrutor. Aloque uma instância (objeto) local deste **struct**. Esboço:

```
struct tpEnvelope
{
    Recurso * pRecurso ;
    tpEnvelope( ) { pRecurso = NULL ; }
    ~tpEnvelope( ){ delete pRecurso ; }
} envelope ;
. . .
envelope.pRecurso = new Recurso( ) ;
. . .
```

Bjarne Stroustrup's C++ Style and Technique FAQ http://www.research.att.com/~bs/bs_faq2.html

Teste de exceções, padrão de programação



- *Smart pointers* C++

- `std::auto_ptr<tipo> p1`
- isso não faz parte do padrão C++

- Padrão C++ 2011

- `std::unique_ptr<tipo> p2`
 - não admite cópia, atribuição `pA = pB` apaga (NULL) o ponteiro `pB`
- `std::shared_ptr<tipo> p3`
 - admite cópia, `delete` ocorrerá somente quando a última referência for eliminada
 - problema: referências circulares
 - solução: `std::weak_ptr<tipo> p4`
 - cópia de `shared_ptr` para `weak_ptr` não altera contador,
 - destruição de `weak_ptr` também não
- para destruir um *smart pointer* usa-se a função `p.reset ()`

- `std::unique_ptr<int> p1 = new int(5);`
`std::unique_ptr<int> p2 = p1; //erro de compilação`
`std::unique_ptr<int> p3 = std::move(p1);`
 //Torna p3 dono do espaço, p1 torna-se nulo
`p3.reset(); //elimina o espaço de dados`
`p1.reset(); //faz nada`
- `std::shared_ptr<int> p1 = new int(5);`
 //Contador de referência ao espaço == 1
`std::shared_ptr<int> p2 = p1;`
 //Contador de referência == 2
`p1.reset(); //Reduz contador, não exclui: contador >0`
`p2.reset(); //Reduz contador, torna-se ==0 então exclui`

Planejamento da integração

Módulos duplê

Módulos duplê



- **Módulo duplê** é um módulo que, durante os testes, toma o lugar de um módulo de produção
 - *test double*, analogia: *stunt double*
 - comum ao desenvolver sistemas de forma incremental
- Tipos de módulos duplê
 - **controle** (*driver*) – controla a realização do teste
 - não é bem um duplê...
 - módulo de teste específico, arcabouço de apoio ao teste
 - **fictício** (*dummy*) – criado para não dar erro de compilação
 - não deve ser utilizado no processamento do teste
 - **enchimento** (*stub*) – geram respostas atreladas à suíte de teste
 - **simulação** (*mock object*) – implementam uma simulação da especificação
 - ex. base de dados residente em memória

Mai 2015

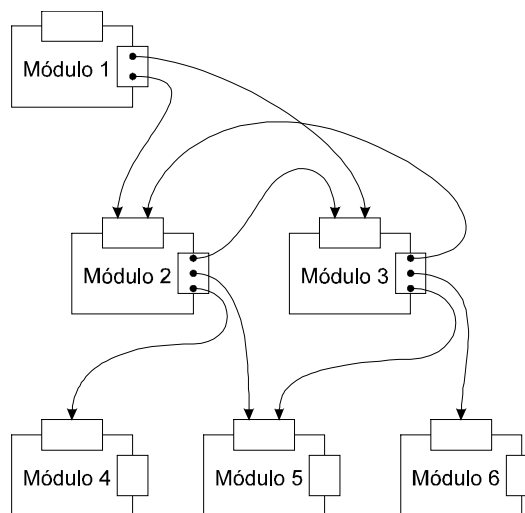
Arndt von Staa © LES/DI/PUC-Rio

25

Planejamento da integração



- Desenvolvimento “tudo de uma vez” (*big bang*)



Mai 2015

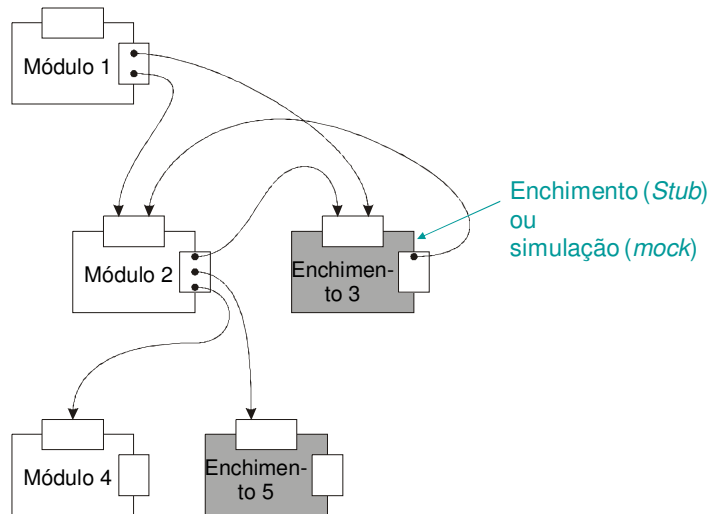
Arndt von Staa © LES/DI/PUC-Rio

26

Planejamento da integração



- Desenvolvimento incremental **descendente** (*top down*)



Mai 2015

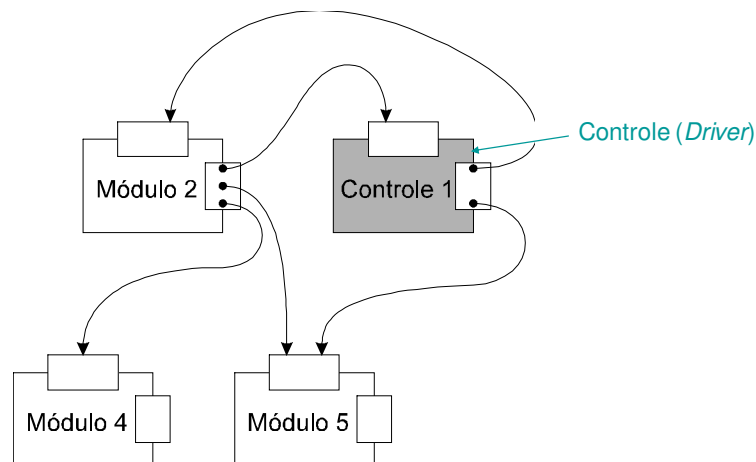
Arndt von Staa © LES/DI/PUC-Rio

27

Planejamento da integração



- Desenvolvimento incremental **ascendente** (*bottom up*)



Mai 2015

Arndt von Staa © LES/DI/PUC-Rio

28

Planejamento da integração



- **Overhead** é o esforço (custo) **necessário**, porém adicional ao custo "líquido" do desenvolvimento
 - embora não participe do uso produtivo boa parte do código e de artefatos overhead devem **permanecer vivos**
 - **subsistema de manutenção**
 - overhead **não é retrabalho inútil**
 - o custo líquido corresponde ao custo de desenvolvimento somente dos artefatos necessários para poder utilizar o software, assumindo desenvolvimento correto por construção
 - clientes e usuários em geral não sabem a dimensão do overhead
 - desenvolvedores sabem?
- Procure uma sequência de desenvolvimento e integração que **torne pequeno o overhead**


Planejamento da integração



- Exemplos de overhead provocado pelos testes
 - criação de suítes de teste e dos scripts de reconstrução dos construtos a serem testados
 - incorporação de instrumentação no código
 - ex. assertivas executáveis, verificadores estruturais
 - desenvolvimento de ferramentas de apoio aos testes
 - ex. geradores de suítes de testes aleatórios
 - desenvolvimento de armaduras de apoio aos testes
 - ex. o contexto necessário para poder testar o módulo sob teste
 - desenvolvimento de dublês convencionais
 - ex. controle, fictício, enchimento
 - desenvolvimento de simuladores (*mocks*)
 - embora um simulador seja um dublê, ele realiza muito mais tarefas do que um dublê convencional, tende a ser complexo

Laboratório de Engenharia de Software

Planejamento da integração




- Exemplos de *overhead* adicionado por um dublê
 - desenvolvimento do dublê
 - garantia da qualidade do dublê
 - manutenção do dublê (**co-evolução**)
 - suíte de teste e scripts de reconstrução usando o dublê
 - desenvolvimento da suíte para teste do módulo isolado
 - reformulação da suíte de teste e dos scripts de reconstrução para operar com módulos de produção
 - depois de dispor do componente completo, o uso de dublês pode tornar-se um estorvo

Mai 2015
Arndt von Staa © LES/DI/PUC-Rio
31

Laboratório de Engenharia de Software

Planejamento da integração



- Prefira seguir uma abordagem **descendente** (*top-down*) ao desenvolver um conjunto de módulos, esta abordagem
 - torna visível logo no início a funcionalidade do programa, mesmo que incompleta
 - facilita a observação do progresso à medida que funcionalidades (*features*, características) forem adicionadas
 - requer dublês (enchimento e/ou simuladores)
- Use uma abordagem **ascendente** (*bottom-up*) somente quando
 - existir alguma precedência de desenvolvimento que não permita o uso de dublês, exemplo
 - serviços de persistência podem tornar necessário um dublê quase idêntico ao módulo de produção

Mai 2015
Arndt von Staa © LES/DI/PUC-Rio
32

Módulos duplê: exemplo de enchimento



- Módulos de enchimento (*stub*) simples

```
int E( int ParametroX )    método duplê fictício
{
    throw new EXC_NotImplemented( ParametroX ) ;
}

int F( int ParametroX )
{
    EXC_ASSERT( ParametroX == 3 ) ;
    return 10 ;
}

int G( int ParametroX )
{
    EXC_ASSERT( ParametroX == 73 ) ;
    return ParametroX / 2 ;
}
```

exceção a ser tratada pelo arcabouço de teste

o projeto da suíte de teste deve assegurar que o **assert** não falhe

alguma função simples

Mai 2015

Arndt von Staa © LES/DI/PUC-Rio

33

Módulos duplê: exemplo de enchimento



- Módulos de enchimento um pouco mais complexos

```
int H( int ParametroX )
{
    tpElem * pElem = ProcurarTabela( TabelaH , ParametroX ) ;
    if ( pElem == NULL )
    {
        throw new EXC_Exception( ErroParametro ) ;
    }
    return pElem->valRetorno
}

int L( int parametroX )
{
    int numLidos = RDR_ReadLine( "ii" , &controle , &valor ) ;
    if ( ( controle != parametroX )
        || ( numLidos != 2 ) )
    {
        throw new EXC_Exception( ErroSincroniaScript ) ;
    }
    return valor ;
}
```

exceção a ser tratada pelo arcabouço de teste

funções do arcabouço de teste

a suíte de teste deve assegurar sincronia entre o caso de teste e os valores lidos

Mai 2015

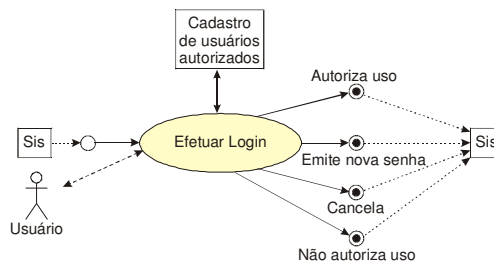
Arndt von Staa © LES/DI/PUC-Rio

34

Módulos de simulação



- Simulam o comportamento de módulos ou componentes necessários para poder testar um módulo ou componente
- Ex. para testar o componente Login precisamos de uma **armadura de teste**:
 - um **simulador do sistema** Sis
 - um **simulador de banco de dados**
 - inicializado para conter o cadastro de usuários autorizados
 - um **simulador de interação de usuário** para que o teste possa ser automatizado
 - uma infraestrutura para **reconstruir** o construto e **reexecutar** os testes automatizados



mais adiante serão apresentados mais detalhes

Mai 2015

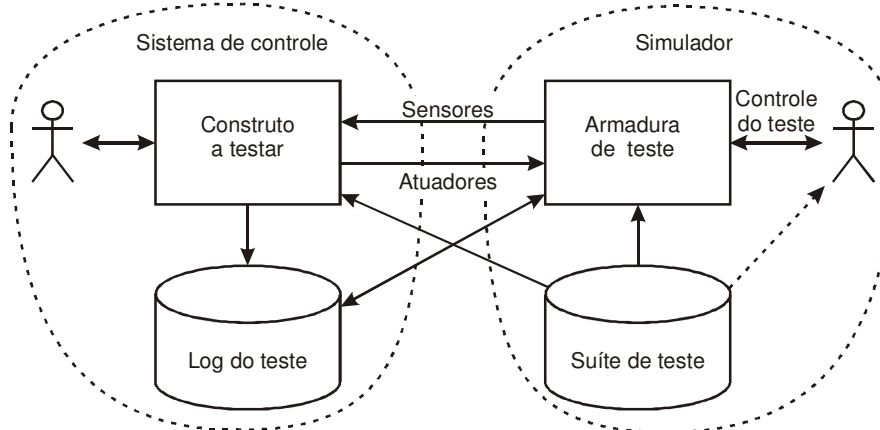
Arndt von Staa © LES/DI/PUC-Rio

35

Armadura de teste



- Contexto montado para poder testar na ausência ou impossibilidade de uso de um contexto de produção



Mai 2015

Arndt von Staa © LES/DI/PUC-Rio

36

Módulos dublê: módulos de simulação (*mock*)



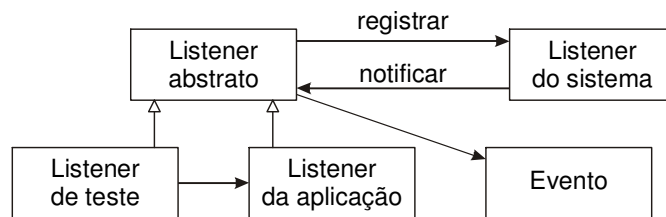
- Exemplos de situações que pedem por simuladores
 - o teste do módulo necessita uma armadura de apoio ao teste
 - o módulo de imitação permite sequenciar os eventos
 - o módulo real possui comportamento não determinístico
 - o módulo de imitação permite sequenciar os eventos
 - o módulo real demorará para estar disponível
 - desenvolvimento em paralelo com várias equipes
 - desenvolvimento de software embarcado
 - o módulo real possui comportamento difícil de provocar
 - ex. erros de comunicação
 - o módulo real induz esperas
 - o módulo real contém interface com o usuário
 - para automatizar é necessário simular o “uso”
 - o módulo sob teste interage com serviços complementares
 - ex. uso de uma função de *call back*

Mai 2015

Arndt von Staa © LES/DI/PUC-Rio

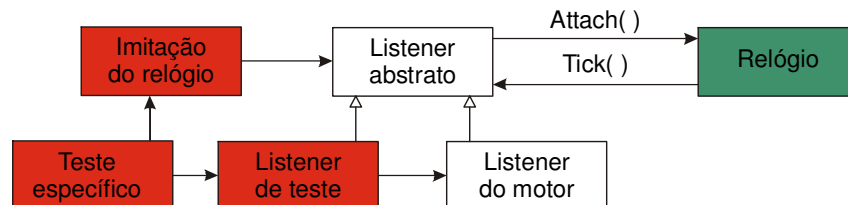
37

Exemplo de módulos de imitação



Padrão de projeto *Listener de teste* – adaptação do padrão *Observer*

Adaptação para monitorar ou simular eventos do relógio




Mai 2015

Arndt von Staa © LES/DI/PUC-Rio

38

Laboratório de Engenharia de Software

Módulos dublê: módulos de simulação



```

public interface Ambiente
{
    public long getTime( ) ;
    ...
}
--- Implementação de produção
public class Contexto implements Ambiente
{
    public long getTime( )
    {
        return System.currentTimeMillis( ) ;
    }
    ...
}
--- Implementação de simulação
public class MockContexto implements Ambiente
{
    private long currentTime ;
    public long getTime( ) { return currentTime ; }
    public void setTime( long aTime ) { currentTime = aTime ; }
    ...
}


```

Que tal corrigir para que o relógio continue atuando?

Mai 2015
Arndt von Staa © LES/DI/PUC-Rio
39

Laboratório de Engenharia de Software

Módulos dublê: módulos de imitação



- Existem diversas bibliotecas para a criação de MockObjects:

Framework	URL
Mockito	http://mockito.org/
EasyMock	http://easymock.org/
Mockachino	http://code.google.com/p/mockachino/
PowerMock	http://code.google.com/p/powermock/
jMock	http://www.jmock.org/
JMockit	http://code.google.com/p/jmockit/
Unitils	http://unitils.sourceforge.net/
	...

Site: [Comparing Java Mock Frameworks – Part 1: The Contenders](#)

Mai 2015
Arndt von Staa © LES/DI/PUC-Rio
40

- Mai 2015

Mai 2015