Unidade 6: Teste de Software

E Uma Introdução às GUIs

Prof. Daniel Caetano

Objetivo: Apresentar e aplicar alguns conceitos da análise funcional.

Bibliografia: MOLINARI, 2005; MOREIRA FILHO, 2002.

INTRODUÇÃO

Até o momento já vimos conceitos suficientes para desenvolver uma aplicação bastante simples, mas completa do ponto de vista estrutural e conceitual.

Enquanto apresentamos mais da biblioteca do Java e incrementamos o software já desenvolvido, nesta aula aproveitaremos para introduzir os conceitos de teste de software.

1. O QUE É TESTE DE SOFTWARE

Teste de Software é o processo pelo qual se verifica o quão correto, completo e seguro está um software, propiciando uma forma de avaliar a qualidade do software.

O ato de testar é a realização de uma comparação crítica entre o que o software faz e o que ele deveria fazer. Assim, o teste ideal seria aquele que fosse capaz de avaliar todas as possibilidades e combinações de funções de um software. Entretanto, é matematicamente comprovado que há tipos de faltas impossíveis de serem identificados por processos automáticos e, portanto, a identificação destas ficam delegadas à inspeção humana. Como tal inspeção também pode ser falha, a grande maioria dos autores sugere que não existe software isento de faltas.

Corrobora para esta teoria o fato de que, costumeiramente, considera-se que só é possível dizer que um software funciona por um determinado número de horas sem falhas se ele já houver sido testado pelo menos por aquele número de horas. Assim, para que um software que tem objetivo de funcionar "para sempre", exigiria um período de testes também "para sempre", o que certamente é impossível.

Por esta razão, é comum diferenciar os termos "falta do software" e "falha do software". "Falta do software" é um defeito que não foi descoberto, mas cuja probabilidade de existência é alta. "Falha de software", por outro lado, é um defeito já identificado e para o qual é necessário uma solução. A etapa de teste consiste em encontrar o maior número de "faltas", quando então tornam-se "falhas", para que elas possam ser corrigidas e o software que chega às mãos do usuário seja o mais livre possível de problemas.

1.1. Por que Realizar um Teste de Software

A atividade de teste pode ser uma das mais aborrecidas de um software e, sob certo aspecto, até mesmo é uma etapa custosa. Entretanto, muito mais custosa para uma empresa pode ser a divulgação ou o uso de um software que tenha passado por uma fase de teste ineficaz. Produtos defeituosos podem arruinar o nome de uma empresa e, dependendo de quão crítica é a aplicação do software, danos irreparáveis podem ser causados, tais como a morte de seres humanos.

Estima-se que o custo de correção de um defeito é tanto maior quando mais avançado o projeto/implementação estiver. Assim, se não por outras razões, esta é uma razão para se iniciar a etapa de testes o mais cedo possível dentro de um ciclo de desenvolvimento de software.

2. TESTES AO LONGO DO DESENVOLVIMENTO

Os testes podem ser planejados e realizados desde as primeiras etapas do desenvolvimento de um software.

Já na Análise de Requisitos é importante identificar, dentre os principais requisitos e restrições, algumas das situações críticas que poderão vir a ocorrer durante o tempo de vida do software.

Na etapa de Análise de Sistema devem ser identificados com mais precisão os aspectos que são passíveis de teste, considerando o software como um todo. Juntamente com a definição dos "aspectos testáveis" devem ser identificados sob que condições estes testes devem ser aplicados.

Na fase de Projeto de Sistema devem ser criadas estratégias de teste, se serão aplicados os testes automáticos ou manuais, quais os componentes críticos. Nos estágios mais avançados do Projeto devem ser desenvolvidos pequenos aplicativos de teste de componentes, bem como teste global, formando os "test-suites", conjuntos de testes automatizados que serão usados, a cada atualização de código, se o software sendo desenvolvido está se comportando como deveria.

Tais "test-suites" são muito úteis na verificação do software após mudanças no código, para verificar se a correção de um dado problema ou a inserção de uma nova característica não introduziu nenhum novo problema. Este tipo de verificação é chamada de "teste regressivo".

Assim que os primeiros componentes do software ficarem prontos, inicia-se a etapa de testes propriamente dita, quando a grande maioria dos testes é aplicada componente a componente, certificando que seu funcionamento está de acordo com as especificações.

Com base nos testes realizados e defeitos encontrados é que é possível estabelecer um padrão de qualidade, indicando se o software está pronto para ser distribuído ao público ou não.

2.1. Testes Caixa-Branca e Caixa-Preta

Os termos teste "Caixa-Branca" e "Caixa-Preta" referem-se a quanto acesso tem o indivíduo que aplica o teste em relação ao código sendo testado.

Quando o aplicador dos testes tem acesso a todo o código sendo testado, permitindo que ele realize alterações de forma a facilitar seu teste, além de poder testar o código de formas não usuais, este tipo de teste é denominado teste "Caixa-Branca".

Por outro lado, quando o aplicador dos testes tem acesso ao componente ou software como um todo, sem poder saber o que ocorre internamente no componente, apenas utilizando-o segundo as especificações e pontos de entrada fornecidos pelo fabricante, este é então chamado de teste "Caixa-Preta".

Os testes do tipo "Caixa-Branca" são normalmente realizados nos primeiros estágios de teste (alpha testing), embora algumas vezes continuem sendo usados posteriormente. Entretanto, os testes do tipo "Caixa-Branca" são, em geral, realizados apenas pelos próprios desenvolvedores.

Os testes do tipo "Caixa-Preta" são realizados por todo o tempo de desenvolvimento, mas se intensificam à medida em que os componentes são finalizados e as primeiras versões completas do sistema são produzidas. Muitas vezes o teste "Caixa-Preta" é realizado por um grupo seleto de usuários que nada têm a ver com a equipe de desenvolvimento (beta testing).

Recentemente tem-se falado nos testes "Caixa-Cinza", em situações em que o aplicador dos testes não tem acesso ao código, também estando preso à documentação do fabricante com relação à interface do componente ou sistema. Entretanto, no que se denomina "Caixa-Cinza" é possível acessar o estado interno do componente sendo testado, após cada operação (e não apenas avaliar as saídas com base nas entradas, como seria numa "Caixa-Preta" comum).

2.2. Fases de Teste

A primeira etapa dos testes, normalmente quando o software sequer está desenvolvido e muitos de seus componentes ainda estão pela metade, é chamada de "fase alpha". Nesta etapa, que se estende dos primeiros dias até o momento em que o software é liberado para o

público (seja ele restrito ou não), os aplicadores de testes são apenas aqueles que fazem parte da equipe de desenvolvimento, direta ou indiretamente, usando técnicas de "Caixa-Branca".

Mais para o fim desta primeira etapa, uma equipe de testes já deve existir junto à equipe de desenvolvimento, que passa a usar técnicas de "Caixa-Cinza" e "Caixa-Preta" para a realização de testes mais completos.

A segunda etapa dos testes, quando ele é liberado para um pequeno público externo à equipe de desenvolvimento, é chamada "fase beta". Nesta fase usuários comuns estarão utilizando o software em situações de dia-a-dia, usando técnicas de "Caixa-Preta", avaliando se encontram novas falhas no sistema e, caso as encontrem, devem informá-las à equipe de desenvolvimento.

Quando a equipe de desenvolvimento recebe um relatório de defeito destes (bug-report), a primeira atitude é tentar reproduzi-lo e, uma vez reproduzido, tentar identificar a origem e corrigi-lo.

As fases alpha e beta servem para garantir que quando um software chegue à terceira etapa (chamado "nível GA", de "General Availability", ou "Disponibilidade Geral"), que é quando o software é liberado ao grande público (muitas vezes na forma de produto comercial), ele esteja praticamente isento de defeitos.

Alguns autores chamam etapa das versões oficiais de "fase gamma", mas este termo tem sido usado de forma pejorativa, sugerindo que pela quantidade de defeitos que os software comerciais atuais possuem, essa seria não uma fase de produto final, mas sim uma terceira fase de testes.

2.3. Níveis de Teste

Há basicamente três níveis de teste: o nível funcional, o nível de componente e o nível de sistema, além do "teste regressivo" que sempre deve ser realizado.

O nível funcional (ou unitário) é aquele que testa funções ou métodos de um objeto, identificando se, para entradas comuns e incomuns o comportamento da saída é adequado. Este nível visa verificar se, num nível mais básico, o software desenvolvido se comporta como o esperado.

O nível de componente é o nível em que as bibliotecas e os objetos são testados como um todo, verificando se os comportamentos de suas diversas funções e métodos são coerentes entre si e produzem resultados satisfatórios. Este nível visa verificar se, num nível de atividades, o software desenvolvido se comporta como o esperado.

O nível de sistema é o nível mais alto, em que a integração entre diversas bibliotecas e objetos é testado, verificando se o comportamento de cada componente frente aos outros são coerentes e iguais aos esperados, se não há problemas na interface entre estes componentes.

Este nível visa verificar se o sistema está cumprindo as funções propostas no Documento de Especificação de Requisitos.

2.4. Testes de Cobertura

Dado que a identificação de um defeito em um trecho de código é diretamente proporcional ao número de vezes que tal trecho de código é executado, há regiões do código que são executadas muitas vezes e facilmente podem ter suas faltas reveladas. Por outro lado, há trechos de código que são raramente executados (como regiões de tratamento de erros, por exemplo) e, portanto, dificilmente terão suas faltas reveladas.

Por esta razão, é preciso ter um cuidado adicional com regiões do código que são pouco executadas. Mas para tomar este cuidado adicional, é preciso, antes de mais nada, identificar tais regiões.

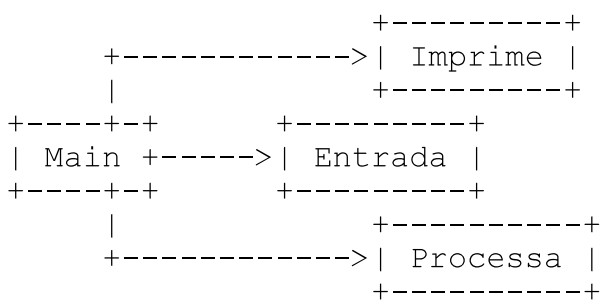
Existem software específicos que podem ser linkados em conjunto com seu software, de forma a realizar um "profiling" e identificar as regiões do código que foram alvo dos menores tempos de processamento (e, portanto, foram executadas menos vezes). Estas regiões são as mais propensas a possuírem erros que não serão identificados.

Uma vez que tais regiões tenham sido identificadas, podem ser adotadas duas estratégias (separadamente ou conjuntamente): inspeção rigorosa do texto e planejamento de testes específicos que provoquem a execução daqueles trechos de código, de forma a tentar encontrar falhas nos mesmos.

3. IMPLEMENTANDO EM CÓDIGO

Esta aula tem dois objetivos: produzir testes simples para os nossos métodos e utilizar a classe JOptionPane para converter a aplicação de avaliação de alunos da aula passada em uma aplicação com janelas.

A estrutura era essa:



O código final da aula passada é apresentado abaixo:

Main.java

```
package Notas;                  // Use essa linha APENAS no NetBeans
import java.util.*;

public class Main {

   public static void imprime(String texto) {
      System.out.println(texto);
      }

   public static double entrada(String texto) {
      Scanner teclado = new Scanner(System.in);
      System.out.print(texto);
      return (teclado.nextDouble());
      }

   public static boolean processa(double av1, double av2, double av3, double freq) {
      double media;
      if (av2 < 4.0 || freq < 75.0) return false;
      if (av1 < 4.0 && av3 < 4.0) return false;
      if (av1 >= av3) media = (av1 + av2) / 2;
      else media = (av3 + av2) / 2;
      if (media >= 6.0) return true;
      return false;
      }

   public static void main(String[] args) {
      double av1, av2, av3, freq;
      av1 = entrada("Digite a nota AV1: ");
      av2 = entrada("Digite a nota AV2: ");
      av3 = entrada("Digite a nota AV3: ");
      freq = entrada("Digite a frequência: ");

      if (processa(av1,av2,av3,freq)) imprime("Aluno Aprovado!");
      else imprime("Aluno reprovado!");
      }
   }
```

Antes de mais nada, modifiquemos o programa para que fique gráfico. Para isso, iremos substituir o método imprime por este:

```
   public static void imprime(String texto) {
      JOptionPane.showMessageDialog(null,texto,"Mensagem!", JOptionPane.PLAIN_MESSAGE);
      }
```

Lembrando de inserir, no início do programa, a diretiva:

```
import javax.swing.*;
```

E o método entrada por este:

```
   public static double entrada(String texto) {
      String valor = JOptionPane.showInputDialog(texto);
      return (Double.parseDouble(valor));
      }
```

O código resultante está abaixo:

Main.java

```
package Notas;                  // Use essa linha APENAS no NetBeans
import javax.swing.*;

public class Main {

   public static void imprime(String texto) {
      JOptionPane.showMessageDialog(null,texto,"Mensagem!", JOptionPane.PLAIN_MESSAGE);
      }

   public static double entrada(String texto) {
      String valor = JOptionPane.showInputDialog(texto);
      return (Double.parseDouble(valor));
      }

   public static boolean processa(double av1, double av2, double av3, double freq) {
      double media;
      if (av2 < 4.0 || freq < 75.0) return false;
      if (av1 < 4.0 && av3 < 4.0) return false;
      if (av1 >= av3) media = (av1 + av2) / 2;
      else media = (av3 + av2) / 2;
      if (media >= 6.0) return true;
      return false;
      }

   public static void main(String[] args) {
      double av1, av2, av3, freq;
      av1 = entrada("Digite a nota AV1: ");
      av2 = entrada("Digite a nota AV2: ");
      av3 = entrada("Digite a nota AV3: ");
      freq = entrada("Digite a frequência: ");

      if (processa(av1,av2,av3,freq)) imprime("Aluno Aprovado!");
      else imprime("Aluno reprovado!");
      }
   }
```

Repare que a máquina virtual java "trava" após completá-lo. Para forçar o programa a sair, é necessário usar o método **System.exit**:

Main.java

```
package Notas;                  // Use essa linha APENAS no NetBeans
import javax.swing.*;

public class Main {

   public static void imprime(String texto) {
      JOptionPane.showMessageDialog(null,texto,"Mensagem!", JOptionPane.PLAIN_MESSAGE);
      }

   public static double entrada(String texto) {
      String valor = JOptionPane.showInputDialog(texto);
      return (Double.parseDouble(valor));
      }

   public static boolean processa(double av1, double av2, double av3, double freq) {
      double media;
      if (av2 < 4.0 || freq < 75.0) return false;
      if (av1 < 4.0 && av3 < 4.0) return false;
      if (av1 >= av3) media = (av1 + av2) / 2;
      else media = (av3 + av2) / 2;
```

```
       if (media >= 6.0) return true;
       return false;
       }

    public static void main(String[] args) {
       double av1, av2, av3, freq;
       av1 = entrada("Digite a nota AV1: ");
       av2 = entrada("Digite a nota AV2: ");
       av3 = entrada("Digite a nota AV3: ");
       freq = entrada("Digite a frequência: ");

       if (processa(av1,av2,av3,freq)) imprime("Aluno Aprovado!");
       else imprime("Aluno reprovado!");

       System.exit(0);
       }
    }
```

Agora vamos realizar alguns testes básicos no método de processamento: tudo zero, notas negativas, frequencia negativa, além dos casos já vistos na aula anteior para verificar se está tudo ok.

Main.java

```
package Notas;                  // Use essa linha APENAS no NetBeans
import javax.swing.*;

public class Main {

    public static void imprime(String texto) {
       JOptionPane.showMessageDialog(null,texto,"Mensagem!", JOptionPane.PLAIN_MESSAGE);
       }

    public static double entrada(String texto) {
       String valor = JOptionPane.showInputDialog(texto);
       return (Double.parseDouble(valor));
       }

    public static boolean processa(double av1, double av2, double av3, double freq) {
       double media;
       if (av2 < 4.0 || freq < 75.0) return false;
       if (av1 < 4.0 && av3 < 4.0) return false;
       if (av1 >= av3) media = (av1 + av2) / 2;
       else media = (av3 + av2) / 2;
       if (media >= 6.0) return true;
       return false;
       }

    public static void main(String[] args) {
       double av1, av2, av3, freq;

       if (processa(0,0,0,0)==false) imprime("Ok!");
       else imprime("Erro!");
       if (processa(-1,-1,-1,75)==false) imprime("Ok!");
       else imprime("Erro!");
       if (processa(10,10,10,-10)==false) imprime("Ok!");
       else imprime("Erro!");
       if (processa(3,4,8,75)==true) imprime("Ok!");
       else imprime("Erro!");
       if (processa(3,3,9,75)==false) imprime("Ok!");
       else imprime("Erro!");
       if (processa(3,4,8,70)==false) imprime("Ok!");
       else imprime("Erro!");
       if (processa(3,10,3,80)==false) imprime("Ok!");
       else imprime("Erro!");
       if (processa(10,3,10,100)==false) imprime("Ok!");
```

```
        else imprime("Erro!");


        av1 = entrada("Digite a nota AV1: ");
        av2 = entrada("Digite a nota AV2: ");
        av3 = entrada("Digite a nota AV3: ");
        freq = entrada("Digite a frequência: ");

        if (processa(av1,av2,av3,freq)) imprime("Aluno Aprovado!");
        else imprime("Aluno reprovado!");

        System.exit(0);
        }
    }
```

4. EXERCÍCIOS

A) Modifique o programa, criando um método imprimeErro para imprimir o resultado dos testes no prompt, usando o método System.out.println.

B) Tente adaptar o programa da SOMA realizado no item 4 da unidade 3 para usar as janelas, como este.

5. BIBLIOGRAFIA

MOLINARI, L. **Testes de Software: Produzindo Sistemas Melhores e Mais Confiáveis**. Editora Érica, 2005.

MOREIRA FILHO, T. **Projeto e Engenharia de Software: Teste de Software**. Alta Books, 2002.