Aplicando testes em XP com o framework JUnit

ADRIANE PEDROSO DIAS[1]
SABRINA BORBA DALCIN[1]
MARCOS CORDEIRO D'ORNELLAS[1]

UFSM - Universidade Federal de Santa Maria
PPGEP - Programa de Pós-Graduação em Engenharia de Produção
Faixa de Camobi, km 9 - 97.105-900- Santa Maria (RS)
[1] (dias.adriane,sabrinadalcin,marcosdornellas)@gmail.com

Resumo. O objetivo desse artigo é demostrar como a aplicação de testes de unidade pode auxiliar na melhora da qualidade dos softwares voltados para processamento e análise de imagens, o frameworl JUnit foi utilizado na implementação dos test cases. A metodologia XP também foi utilizada no desenvolvimento do software e dos testes.Finalmente, é apresentado alguns testes que verificam a qualidade do software analisado. Os resultados demonstram a efiencia da metodologia proposta.

Palavras-Chave: Programação Extrema, Testes de Unidade, Java.

APPLYING TESTS IN XP WITH THE JUNIT FRAMEWORK

Abstract. The goal of this article is to demonstrate how the application of unit tests can assist to improve in the softwares for image processing and analysis quality, the JUnit Framework was used in the test cases implementation. We also used the XP methodology in the software and tests development. Finally, it is presented some tests that verify the quality of the analysed software. The results demonstrates the efficiency of the proposed methodology.

Keywords: eXtreme Programming, Tests of Unit, Java.

(Received January 24, 2006 / Accepted July 10, 2006)

1 Introdução

Segundo [11], a atividade de teste é o processo de executar um programa com a intenção de encontrar um erro: um bom caso de teste é aquele que tem alta probabilidade de revelar a presença de erros e um teste bem sucedido é aquele que detecta a presença de erros ainda não descobertos. Por outro lado, o custo para correção de um erro na fase de manutenção é de sessenta a cem vezes maior que o custo para corrigi-lo durante o desenvolvimento [12].

Dessa forma, a atividade de testes é uma etapa crítica para o desenvolvimento de um software. Embora durante todo o processo de desenvolvimento de software sejam utilizados métodos, técnicas e ferramentas a fim de evitar que erros sejam introduzidos no produto, a atividade de teste continua sendo de fundamental importância para a eliminação dos erros que persistem [9].

Embora as revisões técnicas sejam mais eficientes na detecção de defeitos, os testes são importantes para complementar as revisões e aferir o nível de qualidade conseguido. Dessa forma, é importante que os testes sejam bem planejados e desenhados, para que seja possível alcançar melhor desempenho dos recursos alocados para eles.

Um dos objetivos de testar programas é maximizar a sua abrangência, isto é, permitir que o teste alcance um maior número possível de defeitos, com isso, deixando o programa com um melhor desempenho. Outro objetivo é eliminar esforços duplicados, visto que no mo-

mento que se cria um programa, já deve ser criado o seu teste, diminuindo com isso a probabilidade de problemas futuros principalmente se houver a necessidade refactoring de código.

Atualmente, observa-se que o custo do software tem ultrapassado em muito o custo do hardware [6]. Este fato torna muito importante o controle de qualidade dos programas durante a sua fase de desenvolvimento. Nesse sentido, a organização da atividade de teste deve, de alguma forma, refletir a organização da atividade do projeto. Isto implica que a arquitetura modular do software seja uma candidata natural a dirigir a verificação da aplicação. Dentro deste contexto, procurou-se realizar um estudo sobre a técnica de desenvolvimento de software eXtreme Programming (XP) com ênfase em testes de unidade de software, utilizando o framework JUnit.

Neste artigo são descritos alguns casos de testes de unidade utilizando o framework JUnit dentro de um contexto de desenvolvimento XP. Fazendo uso dessa tecnologia, serão apresentados casos de testes realizados nas classes de um software para processamento e análise de imagens.

Este artigo esta organizado da seguinte maneira: na seção 2 aborda a XP e suas práticas; na seção 3 é apresentado o framework JUnit como ferramenta de testes de unidade integrada à XP, na seção 4 é apresentado uma aplicação utilizando o framework JUnit. Por fim, na seção 5, serão apresentadas as conclusões.

2 XP E SUAS PRÁTICAS

Segundo [2] e [4], XP é uma disciplina de desenvolvimento de software voltada para pequenas e médias equipes, onde os requisitos são vagos e mudam freqüentemente. Desenvolvido por Kent Beck, Ward Cunningham e Ron Jeffries, a XP tem como principal tarefa a codificação, com ênfase menor nos processos formais de desenvolvimento, com uma maior disciplina de codificação e testes. Tem como princípios a comunicação, simplicidade, feedback de usuários e coragem dos desenvolvedores.

A XP valoriza a criação e automação dos testes, sendo criados antes, durante e depois da codificação. É flexível a mudanças de requisitos, valorizando o feedback com o usuário e a qualidade do código fonte final [1].

A XP enfatiza o desenvolvimento rápido do projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas. As regras, práticas e valores da XP proporcionam um agradável ambiente de desenvolvimento de software para seus seguidores, que são conduzidos por quatro valores: comunicação, simplicidade, feedback e coragem [2].

A finalidade do princípio de comunicação é manter o melhor relacionamento possível entre clientes e desenvolvedores, preferindo conversas pessoais a outros meios de comunicação. A simplicidade visa permitir a criação de código simples que não deve possuir funções desnecessárias. A prática do feedback significa que o programador terá informações constantes do código e do cliente. A informação do código é dada pelos testes constantes, que indicam os erros tanto individuais quanto do software integrado.

Em relação ao cliente, o feedback significa que ele terá freqüentemente uma parte do software totalmente funcional para avaliação, e o princípio coragem para tratar de problemas que surgem repentinamente, fazendo ajustes significativos no sistema a fim de garantir que funcione adequadamente.

O funcionamento da XP é baseado em um conjunto de regras e práticas, divididos em quatro grandes grupos: planejamento, desenvolvimento da arquitetura, codificação e testes. As práticas da XP são divididas em organizacionais (círculo externo), equipe (círculo intermediário) e individuais (círculo interno), essas organizações são mostradas na Figura 1. As práticas da XP a saber são [10], [1]:

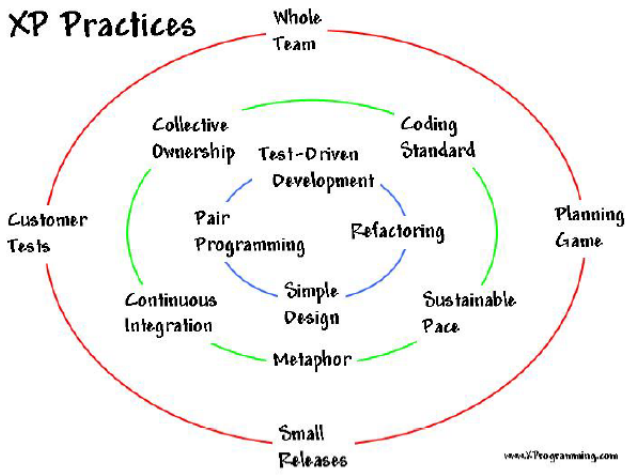

Figura 1: Práticas XP.

- Jogo do planejamento (The Planning Game). Consiste em decidir o que é necessário ser feito e o que pode ser adiado no projeto. A XP baseia-se em requisitos atuais para desenvolvimetno de software, não em requisitos futuros. Além disso, a XP procura evitar os problemas de relacionamento entre as áreas de negócios (clientes) e a área de de-

senvolvimento.

- Pequenas Versões (Small releases). A equipe deve colocar rapidamente um sistema simples em produção, uma versão pequena, e depois entregar novas versões em poucos dias ou poucas semanas.
- Metáfora (Metaphor). Uma metáfora é uma descrição simples de como o sistema funciona. Ela fornece uma visão comum do sistema e guia o seu desenvolvimento.
- Projeto simples (Simple design). O sistema deve ser projetado o mais simples possível. Complexidade extra é removida assim que descoberta.
- Testes (Testing). Os programadores escrevem testes de unidade continuamente. Esses testes são criados antes do código e devem ser executados perfeitamente para que o desenvolvimento continue. Os clientes também escrevem testes para validar se as funções estão finalizadas.
- Refatoração (Refactoring). Os programadores reestruturam o sistema durante todo o desenvolvimento, sem modificar seu comportamento externo. Isso é feito para simplificar o sistema, adicionar flexibilidade ou melhorar o código. A refatoração deve ser feita apenas quando é necessário, ou seja, quando um desenvolvedor da dupla, ou os dois, percebe que é possível simplificar o módulo atual sem perder nenhuma funcionalidade.
- Programação pareada (Pair programming). Todo código produzido é feito em pares, duas pessoas trabalhando em conjunto na mesma máquina. O desenvolvedor que está com o controle do teclado e do mouse implementa o código, enquanto o outro observa continuamente o trabalho que está sendo feito. Uma grande vantagem da programação em dupla é a possibilidade dos desenvolvedores estarem continuamente aprendendo um com o outro.
- Propriedade coletiva (Collective ownership). Qualquer um pode alterar qualquer código em qualquer momento, o código é de propriedade coletiva.
- Integração contínua (Continuous integration). Uma nova parte do código deve ser integrada assim que estiver pronta. Conseqüentemente, o sistema é integrado e construído várias vezes ao dia.
- Semana de 40 horas (40-hour week). XP defende um ritmo de trabalho que possa ser mantido, sem prejudicar o bem estar da equipe. Trabalho além do horário normal pode ser necessário, mas fazer horas extras por períodos maiores que uma semana é sinal de que algo está errado com o projeto.
- Cliente junto aos desenvolvedores (On-site customer). Os desenvolvedores devem ter o cliente disponível todo o tempo, para que ele possa responder às dúvidas que os desenvolvedores possam ter.
- Padronização do Código (Coding standards). Os programadores escrevem o código seguindo regras comuns enfatizando a comunicação por meio do código, ou seja, padronização na arquitetura do código, para que este possa ser compartilhado entre todos os programadores.

As práticas do XP apóiam umas às outras, devem ser usadas em conjunto e todas devem ser aplicadas para se ter agilidade no processo. Aplicar as práticas de forma isolada pode não produzir a agilidade desejada [5].

3 Framework JUnit

O JUnit é um framework horizontal de código aberto, desenvolvido por Kent Beck e Erick Gamma, com suporte à criação de testes automatizados em Java. Esse framework facilita a criação de código para a automação de testes com apresentação dos resultados. Com ele, pode ser verificado se cada método de uma classe funciona da forma esperada, exibindo possíveis erros ou falhas podendo ser utilizado tanto para a execução de baterias de testes como para extensão.

Uma vez que todas as condições testadas obtiveram o resultado esperado após a execução do teste, considera-se que a classe ou método está de acordo com a especificação, incrementando a qualidade daquela unidade.

Com JUnit, o programador tem uma ferramenta muito poderosa que o ajudará a eliminar todos (ou quase todos) os bugs de seu código de maneira mais atraente. Ou seja, os programadores gostam de programar, então se criou, uma forma interessante de realizar testes onde é possível a criação de programas que realizem os testes pelo programador. É utilizando esse conceito que JUnit permite deixar a fase de teste de unidades bem mais agradável ao programador.

Para utilizar o JUnit, é necessário criar uma classe que estenda junit.framework.TestCase. A partir daí, para cada método a ser testado é necessário definir um método devem ser públicos sem retorno de argumentos, `testeXxx()` na classe de teste, esses métodos de testes devem ser `public void` e não podem receber nenhum parâ-metro. Eles criam um objeto definindo o ambiente de teste, executam esses testes utilizando o ambiente que foi criado e verificam os resultados que podem terminar, falhar ou provocar uma exceção.

JUnit facilita bastante a criação e execução de testes, mas elaborar bons testes exige mais. A XP sugere que se realizem testes em tudo [8]. Dentro deste contexto, inclui-se até mesmo os métodos `get/set`. A Figura 2 a-presenta o diagrama das principais classes da API para construir os testes.

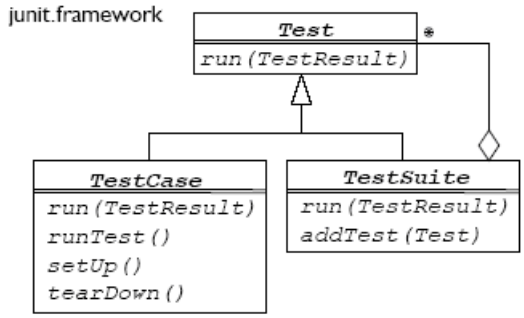

Figura 2: Principais classes da API.

Segundo [3] estas são algumas vantagens de se utilizar JUnit:

- Permite a criação rápida de código de teste enquanto possibilita um aumento na qualidade do sistema sendo desenvolvido e testado;
- Não é necessário escrever o próprio framework;
- Amplamente utilizado pelos desenvolvedores da comunidade código-aberto, possuindo um grande nú-mero de exemplos;
- JUnit é elegante e simples. Quando testar um programa se torna algo complexo e demorado, então não existe motivação para o programador fazê-lo;
- Uma vez escritos, os testes são executados rapidamente sem que, para isso, seja interrompido o processo de desenvolvimento;
- JUnit checa os resultados dos testes e fornece uma resposta imediata;
- Pode-se criar uma hierarquia de testes que permitirá testar apenas uma parte do sistema ou todo ele;
- Escrever testes com JUnit permite que o programador perca menos tempo depurando seu código;
- Todos os testes criados utilizando o JUnit são escritos em Java;
- JUnit é LIVRE.

O JUnit possui uma grande integração com outras ferramentas de desenvolvimento, como Jbuilder, Kawa, Jdeveloper, entre outros. Além disto, foram projetados as extensões do JUnit voltados para diversos segmentos como banco de dados, XML, J2EE e WEB.

4 StoneAge: Uma Aplicação

A ferramenta StoneAge foi inicialmente desenvolvida[2] dentro do Grupo de Processamento de Informação Multimídia PIGS[3] da Universidade Federal de Santa Maria, e provê funcionalidades para processamento e análise de imagens. Essa ferramenta foi implementada utilizando a linguagem de programação Java, a Application Programming Interface (API) Swing[4] e Java Advanced Image[5], em conjunto com o desenvolvimento baseado em componentes e padrões de projeto.

O grupo PIGS tem como objetivo produzir e implementar métodos de processamento e análise de imagens em microscopia quantitativa e anatomopatologia, contemplando as seguintes áreas de atuação: processamento e filtragem de imagens para melhoramento na qualidade da análise, classificação automática de imagens indexadas por conteúdo, desenvolvimento de software para processamento de imagens e imageamento médico.

Atualmente, o grupo está desenvolvendo a ferramenta StoneAge e as funcionalidades estão sendo desenvolvidas dentro do contexto da metodologia XP tendo em vista a necessidade de desenvolver um software de forma mais rápida, mas com qualidade. Para isso, uma forma de garantir a qualidade do que está sendo desenvolvido é realizar testes de unidade continuamente.

Inicialmente vem sendo realizados casos de testes de unidade, entretanto como forma de garantir 100% da confiabilidade do sistema, futuramente serão desenvolvidos outros tipos de testes como testes de sistema, testes de desempenho, testes de componentes, entre outros.

Essa é uma boa prática, herdada da "Extreme Programming"(XP), onde o teste é escrito antes de escrever o método. Desta maneira o caso de teste funciona como uma especificação da funcionalidade, executa-se primeiramente o teste e escreve-se o método evolutivamente de acordo com os resultados verificados. Com isso consegue-se restringir a funcionalidade para apenas

[2]Desenvolvida por Daniel Welfer [Welfer, 2005], Gabrielle Dias Freitas [Freitas, 2004], Grasiela Peccini [Peccini, 2004] e Marcos Cordeiro d'Ornellas.

[3]Grupo com suporte do CNPq e Capes.

[4]Java Foundation Classes (JFC/Swing): http://java.sun.com/products/jfc.

[5]Página Oficial de Java Advanced Image: http://java.sun.com/products/java-media/jai.

aquilo que foi especificado e, assim, conseguir reduzir o tempo necessário para manutenção e correções, além do retrabalho de identificação e localização dos erros.

Esses testes, no âmbito de processamento e análise de imagens, devem ser realizados com o objetivo de aumentar a confiabilidade e garantir a qualidade do software desenvolvido, além de reduzir a ocorrência de erros e riscos associados ao desenvolvimento, tendo em vista que grande parte dos algoritmos implementados são extremamente complexos.

A ferramenta StoneAge possui duas classes principais que são a `Main` e a `LittleWindow`. A classe `Main` (classe principal do sistema) representa a classe mãe onde é efetuado a abertura dos arquivos gráficos, as informações de ajuda do sistema e outras. É a classe principal por gerenciar a chamada aos pacotes os quais os componentes estão inseridos. Toda vez que um arquivo gráfico é aberto essa classe invoca um objeto da classe `LittleWindow` que tem por finalidade exibir essa imagem na tela e proporcionar todos as operações até então desenvolvidas [13].

A classe `LittleWindow` (unidade principal da interface com o usuário) tem por finalidade exibir a imagem na tela e proporcionar todos as operações até então desenvolvidas. Assim, esta classe comporta-se como um processo filho capaz de apresentar diferentes comportamentos [7].

O presente artigo explora casos de testes realizados nas classes `Main` e `ShowImage` da ferramenta StoneAge utilizando framework JUnit. O JUnit facilita a criação automática de testes, com apresentação dos resultados. Pode-se verificar se cada método de uma classe funciona de forma esperada, exibindo uma possível falha.

No entanto, os testes escritos caracterizam-se como testes de sistema, pois cada teste inicializa as funções básicas da ferramenta de forma integrada e atua como um usuário da infra-estrutura executando diferentes casos de uso (chamando os métodos da interface) com entradas (parâmetros) corretas ou não.

Dentre as funcionalidades existentes na classe `Main`, realizou-se testes de unidade na função `Open`, a qual, é responsável por abrir um arquivo gráfico. Posteriormente realizou-se testes na classe `ShowImage` a qual é responsável por exibir uma janela interna, tipo `LittleWindow`, que mostra um arquivo gráfico aberto.

Na função `Open` da classe `Main` testou-se a chamada de abertura de um objeto gráfico (imagem) escolhido e, também as extensões desse arquivo aceitas pela ferramenta StoneAge. Na Figura 3 pode-se visualizar a função `Open` da classe `Main`, a qual realizou-se os testes.

```
public void open()
{
    PlanarImage src; // tipo do arquivo
    if ( getOpenNumber() == 0 )
    {
       jfilechooser = new
            JFileChooser("C:/WINDOWS/Desktop");
    }
    else
    {
    String ultimoCaminho = (String)
        ultimoPath.elementAt(getOpenNumber() -1);
       jfilechooser = new
         JFileChooser( ultimoCaminho );
    }
    jfilechooser.setFileSelectionMode
         ( JFileChooser.FILES_ONLY );

    ExampleFileFilter filterOpen = new
          ExampleFileFilter();
    filterOpen.addExtension("JPG");
    filterOpen.addExtension("JPEG");
    filterOpen.addExtension("BMP");
    filterOpen.addExtension("PNG");
    filterOpen.addExtension("GIF");
    filterOpen.addExtension("TIFF");
    filterOpen.addExtension("PNM");

    jfilechooser.setFileFilter(filterOpen);

    result = jfilechooser.showOpenDialog(this);

    //se o usuário cancelou a operação
    if (result == JFileChooser.APPROVE_OPTION)
    {
    this.filename = jfilechooser.getSelectedFile();
    String name =
          jfilechooser.getName(this.filename);
    setImageName( name );

    File imagePath = jfilechooser.getSelectedFile();
    String path = imagePath.getPath();
    setImagePath( path );
    setImageExtension
          ( filterOpen.getExtension( filename ) );
    src = JAI.create("fileload", getImagePath() );
    setSrc( src );

    new ShowImage( src, getImagePath(),
          desktop );
    setOpenNumber(getOpenNumber() + 1);
    ultimoPath.add( path );
    }
    else
     jfilechooser.cancelSelection();
  }
```

Figura 3: Função Open.

A função `Open` instancia um objeto do tipo `JFileChooser`, que é a janela interna que permite localizar e abrir um arquivo gráfico (imagem). Após, foi criado um construtor do tipo `ExemploFileFilter` o qual é uma classe da Sun que é utilizada para filtrar a extensão dos arquivos que serão abertos. As extensões permitidas para abertura de um arquivo pela ferramenta StoneAge são JPG, JPEG, BMP, PNG, GIF, TIFF

e PNM. Por fim, é passada a classe `ShowImage`.

Os testes de unidade realizados nos métodos da função `Open`, Figura 3, tiveram como objetivo verificar se o arquivo está sendo aberto corretamente e suas possíveis extensões. Neste caso, para desenvolver os testes com o JUnit, foi necessário criar uma classe que estenda `junit.framework.TestCase`, para em seguida definir um método do tipo `public void testImagemAberta()` na classe de teste. A Figura 4, mostra os casos de teste de unidade realizados para o método da classe `Main` descrito na Figura 3.

```
public void testImagemAberta() {
      main.open();
      try{
           System.out.println( "Extensão " +
               main.getImageExtension() );
           } catch( Exception e ){
           }
       extension = main.getImageExtension();

       if ( (extension.equals("jpg")) ||
           (extension.equals("JPG")) ||
           (extension.equals("jpeg")) ||
           (extension.equals("JPEG") ) )
          assertEquals( "2.0", "jpg", extension );
       else
       if ( (extension.equals("bmp")) ||
           (extension.equals("BMP")) )
          assertEquals( "3.0", "bmp", extension );
       else
       if ( (extension.equals("tiff")) ||
           (extension.equals("TIFF")) )
          assertEquals( "4.0", "tiff", extension );
       else
       if ( (extension.equals("png")) ||
           (extension.equals("PNG")) )
           assertEquals( "5.0", "png", extension );
       else
       if ( (extension.equals("pnm")) ||
           (extension.equals("PNM")) )
            assertEquals( "6.0", "pnm", extension );
       else
       if ( (extension.equals("gif")) ||
            (extension.equals("GIF")) )
            assertEquals( "7.0", "gif", extension );
       else
       if ( (extension != "jpg") || (extension != "JPG")   ||
          (extension != "jpeg") || (extension != "JPEG") ||
          (extension != "bmp") || (extension != "BMP") ||
          (extension != "tiff")  || (extension != "TIFF") ||
          (extension != "png")  || (extension != "PNG") ||
          (extension != "gif")   || (extension != "GIF") )
            assertFalse(extension, true);
   }
}
```

Figura 4: Casos de testes realizados para a função Open.

No caso de teste `testImagemAberta` iniciou-se o teste chamando a função `Open`, o qual abre uma caixa de diálogo para escolha do arquivo que será aberto. Posteriormente a escolha do arquivo, é feito o teste para ver se a extensão do arquivo escolhido pelo usuário é permitida pela ferramenta StoneAge.

Para verificar se a extensão do arquivo aberto é igual à extensão permitida utilizou-se o método `assertEquals()`, o qual tem como objetivo comparar dois valores, (o valor esperado e o valor recebido).

Na classe `ShowImage` realizou-se testes de unidade para verificar se os métodos e variáveis responsáveis por abrir uma janela interna estão recebendo os valores atribuídos. A Figura 5, mostra a classe `ShowImage`, o qual se realizou os testes.

```
public ShowImage( PlanarImage src, String imagePath,
    JDesktopPane desktop ){
    setShowImage( src );
    sourcesVector.addSources( src );
    int svs = sourcesVector.getVSources().size();

     try{

       LittleWindow littleWindow = new LittleWindow
            ( imagePath, dst, desktop );
       desktop.add(littleWindow);
       littleWindow.setLocation( 10 * svs , 10 * svs );
       littleWindow.setSelected( true );
       }catch (Exception ex){

       JOptionPane.showMessageDialog( null,
            ex.getMessage(),
            "Try Again!",
            JOptionPane.INFORMATION_MESSAGE
            );
       }
  }
  public void setShowImage( PlanarImage dst ){
    this.dst = dst;
  }
  public PlanarImage getShowImage( ){
    return dst;
  }
}
```

Figura 5: Classe ShowImage.

Esse método é responsável por exibir uma janela interna, do tipo `LittleWindow`, onde passa três parâmetros, o primeiro é uma `planarImage`, o segundo é uma `string` que mostra o caminho do arquivo e o terceiro é o `JDesktopPane` a ser instânciado. A Figura 6, mostra os casos de testes de unidade realizados para a classe `ShowImage`.

```
  public void testImageReturn() {
       main.open();

  showImage.setShowImage(main.getSrc());
       try{
          main.getSrc();
          }
            catch( Exception e )
            {
            }
       assertNotNull(showImage.dst);

  assertNotNull(showImage.getShowImage());
       }
  }
```

Figura 6: Casos de testes realizados para a classe ShowImage.

O teste `testImageReturn` pertencente a classe de teste `TestShowImage`, teve como objetivo verificar se o método e a variável da classe `ShowImage` estão recebendo e mostrando o arquivo aberto. Para fazer esse teste utilizou-se o método `AssertNotNull()`, cujo objetivo é verificar se a referência ao objeto passado não é nula.

Posteriormente criou-se uma "classe de teste mãe", chamada `TestaTudo`, a qual representa uma composição de testes, cuja finalidade é acionar várias classes de teste em um único lugar, sendo usado pelo `TestRunner` para saber quais métodos devem ser executados como teste. A Figura 7, mostra um exemplo de unidade de teste "mãe".

```
import junit.framework.Test;
import junit.framework.TestSuite;

public class TestaTudo {

   public static void main(String[] args) {
   junit.textui.TestRunner.run(suite());
   }

     public static Test suite() {

         TestSuite suite= new TestSuite();
         suite.addTest(TestMain.suite());
         suite.addTest(TestShowImage.suite());
         return suite;
      }
}
```

Figura 7: Exemplo de unidade de teste "mãe".

Neste caso, o teste `TestaTudo` pertencente a classe de teste `TestaTudo` tem como objetivo possibilitar construir uma hierarquia de testes onde uma `Suite` chama a outra e assim por diante. Na Figura 8 é apresentado o resultado com sucesso da classe `TestaTudo`.

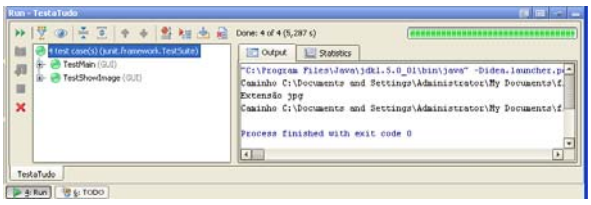

Figura 8: Caso de teste com sucesso.

Na Figura 8 foi apresentado o caso de teste `TestaTudo`, o qual, se verificou através do modo gráfico que os testes `testImagemAberta()` da classe `Main`, e `testImageReturn()` da classe `ShowImage`, obtiveram sucesso, garantindo a qualidade das funções básicas da ferramenta.

Mas vale lembrar que em caso de insucesso a ferramenta mostra uma barra vermelha. Na Figura 9 é apresentado um caso de teste mal sucedido.

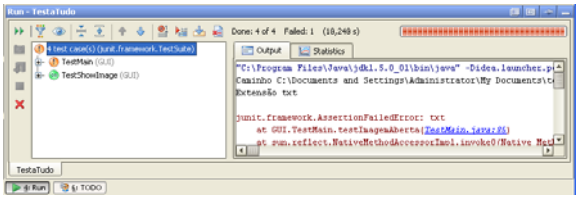

Figura 9: Caso de teste mal sucedido.

No caso de teste apresentado na Figura 9 constatou-se que a classe de teste `TestMain` teve um erro na função `testImagemAberta`, tendo em vista que a mesma recebeu um arquivo com formato "txt", sendo que essa extensão não é suportada pela ferramenta StoneAge a qual, só aceita extensões do tipo gráfico.

5 Conclusão

A XP é uma metodologia voltada para o aumento de qualidade e produtividade no desenvolvimento de software. Para implementar a XP é preciso fazer a equipe se unir em torno de algumas práticas simples, obter feedback suficiente e ajustar as práticas para a sua situação particular. Ela pode ser implementadas aos poucos, porém a maior parte das práticas são essenciais. Nem todos os projetos são bons candidatos para o uso da XP.

Com a XP os testes devem ser simples e eficientes dando ênfase ao design simples para resolução do problema, permitindo sempre que necessário escrever novos testes. A realização de testes de software permite a criação de códigos bem mais confiáveis e estáveis em um tempo de desenvolvimento bem menor, pois os usuários estão cada vez mais informados e principalmente críticos, quanto à qualidade dos sistemas adquiridos.

Inicialmente, o teste de unidade focaliza cada unidade para garantir que os aspectos de implementação de cada uma estejam corretos. Apesar desses testes não garantirem a total corretude do software analisado, ajudam nitidamente a diminuir os erros que o software possui durante todo o seu ciclo de desenvolvimento, permitindo assim que se agilize o processo de desenvolvimento do software garantindo sua qualidade.

No âmbito de processamento e análise de imagens, apesar de ainda não haver referências específicas para essa área, notou-se que os testes são de vital importância para garantir a qualidade da ferramenta desenvolvida, além de, garantir a confiabilidade para os usuários que utilizarão das funções que a ferramenta StoneAge disponibiliza.

Esse artigo apresentou a aplicação de testes de unidade na ferramenta StoneAge utilizando o framework JUnit, cujo propósito foi garantir, através dos testes, a qualidade das funções básicas dessa ferramenta, e atuando como um usuário da infra-estrutura, executando diferentes casos de uso (chamando os métodos da interface) com entradas (parâmetros) corretas ou não.

Os resultados apresentados no decorrer desse trabalho mostraram a importância de se utilizar testes no processo de desenvolvimento de um software em conjunto com uma metodologia ágil de desenvolvimento. Além disso, pôde-se verificar ao longo dos testes realizados, que as funcionalidades básicas desenvolvidas demonstram estar corretas, realizando suas operações de forma esperada.

Para finalizar, cabe destacar que o framework JUnit mostrou-se um ótimo recurso para testes de unidade, principalmente por prover a separação do código de teste do código do produto, facilitando sua integração com a maioria das ferramentas de desenvolvimento Java, bem como extensões para vários segmentos. Sendo também de fácil utilização, permitindo visualizar os testes em formato texto, GUIs AWT e Swing, e ainda a extensão do framework.

Referências

[1] Astels, D., Miller, G., and Novak, M. eXtreme Programming: Guia Prático. 2002.

[2] Beck, K. *eXtreme Programming Explained*. Addison Wesley Publishing Company, New York, 2000.

[3] Clark, M. JUnit Primer. *Web Journal*, 2005.

[4] Cunningham, W. Extreme Programming. *Web Journal*, 2005.

[5] da Costa Filho, E. G., Penteado, R., Silva, J. C. A., and Braga, R. T. V. Padrões e Métodos Ágeis: agilidade no processo de desenvolvimento de software. In *5ª Latin American Conference on Pattern Languages of Programming - SugarLoafPloP*, pages 145–157, Campos do Jordão, 2005.

[6] Dias, A. P., Ferreira, L., and Pillat, F. Ferramentas para Automação de Testes. In *Revista do Centro de Ciências da Economia e Informática, Bagé RS*, volume 8, pages 50–57, 2004.

[7] Freitas, G. D. *IGraPI: Interface Gráfica da Ferramenta para Processamento de Imagens StoneAge*. Monografia, Universidade Federal de Santa Maria, Santa Maria, RS, 2004.

[8] Jeffries, R., Anderson, A., and Hendrickson, C. *eXtreme Programming Installed*. Addison Wesley Publishing Company, New York, 2001.

[9] Maldonado, J. C. *Critérios Potenciais usos: Uma Construção ao Teste Estrutural de Software*. PhD thesis, Campinas, SP, 1991.

[10] Mentor, O. Extreme Programming. *Web Journal*, 2002.

[11] Myers, G. J. *The Art of Software Testing*. Addison Wesley Publishing Company, New York, 1999.

[12] Pressman, R. *Engenharia de Software*. Makron Books, New York, 1995.

[13] Welfer, D. *Padrões de Projeto no Desenvolvimento de Sistemas de Processamento de Imagens*. Dissertação de mestrado, Universidade Federal de Santa Maria, Santa Maria, RS, 2005.