xUnit – Testes Unitários Automatizados

Hélder António Brandão – ei01033@fe.up.pt
João Manuel Guerreiro – ei02047@fe.up.pt
João Miguel Monteiro Pinto – ei03119@fe.up.pt
Joel Tiago Moreira Campos – ei03066@fe.up.pt
Tiago Mota Freitas – ei031042fe.up.pt
Victor Manuel Oliveira – ei00047@fe.up.pt

Engenharia de Software

Prof. Raul Moreira Vidal
Prof. Nuno Flores

Abstracto. xUnit como ferramenta auxiliar no desenvolvimento ao software. Resultado de utilização de testes automatizados como aumento de produtividade. Redução de *bugs* através de testes frequentes e hábitos de programação. Casos específicos de utilização e implementações do conceito. xUnit no contexto de Engenharia de Software.

1 Introdução

xUnit, é o nome genérico para qualquer estrutura de testes automáticos unitários.
O teste unitário ou de unidade é um processo que consiste na verificação da menor unidade do projecto de software. Em sistemas construídos com uso de linguagens orientadas a objectos, um teste envia uma mensagem a um método, uma classe ou mesmo um objecto e verifica se tem o retorno previsto.
O teste unitário é da responsabilidade do próprio programador durante a implementação, isto é, após codificar uma classe, por exemplo, seria executado o teste de unidade. Geralmente, um teste de unidade executa um método individualmente e compara uma saída conhecida após o processamento da mesma.
O teste unitário é considerado o primeiro de uma cadeia de testes à qual um software pode ser submetido, nesta fase, não se pretende testar toda a funcionalidade de uma aplicação.

2 xUnit – Desenvolvimento do Tema

2.1 Contexto Geral de Utilização e algumas Aplicações

xUnit, o nome genérico para qualquer estrutura de testes automáticos unitários. O desenvolvimento desta área deve-se à importância dos testes efectuados na implementação de uma aplicação. Os testes de software durante o desenvolvimento permitem a redução de custos.
Este tipo de testes garante que a implementação respeita a especificação. Os testes devem ser automatizados, sendo executados sempre que necessário o que garante a qualidade de software durante todo o ciclo de vida.

2.1.1 Motivação para Utilização do xUnit

Visão tradicional: se um programador garante que um programa funciona, assume-se que o programa funciona hoje e funcionará sempre.

Os *testes* devem ser feitos:

- No início;
- Durante;
- No Fim.

Visão dinâmica: se um programa não tem um teste automatizado, assume-se que ele não funciona – cabe ao programador provar que ele funciona através de testes automatizados.

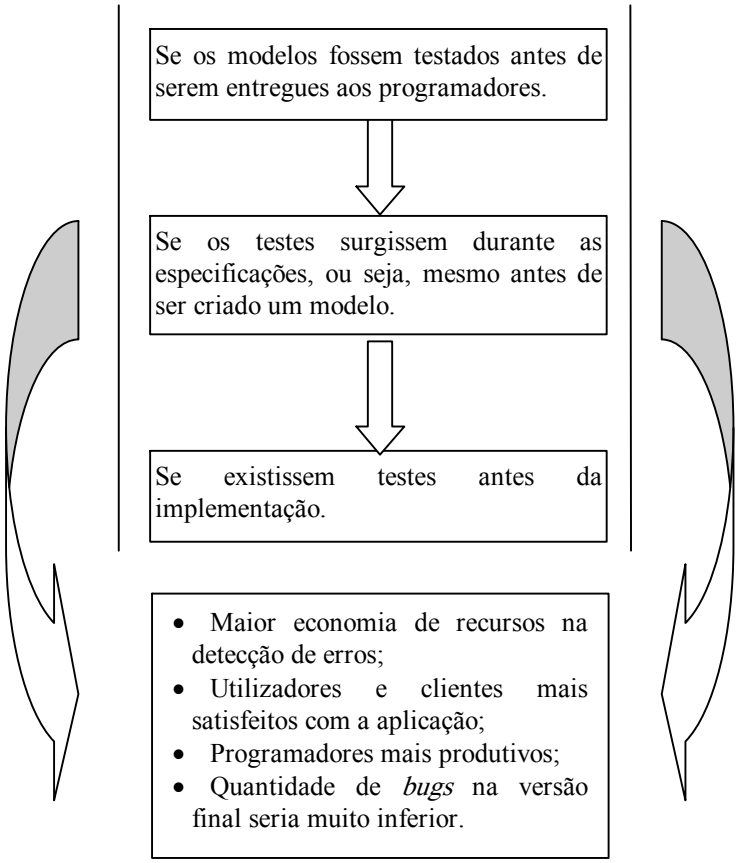

Fig. 1: Importância de testes

No desenvolvimento de software sem testes dinâmicos, surgem problemas como:

- Atrasos no cronograma;
- Menos código estável e confiável;
- Menos produtivo, logo, mais pressão existente sobre os programadores.

Na área da programação, os testes unitários consistem num processo usado na verificação do funcionamento de um módulo de código fonte.

Preferencialmente, cada unidade de teste é independente das restantes. Este tipo de teste é normalmente desenvolvido pelos programadores e não pelos utilizadores do software.
A fase de testes, representa mais de cinquenta por cento dos custos e do tempo despendido no desenvolvimento de software, desta forma, foi necessário rentabilizar este processo, daí o uso de testes automáticos unitários.

O uso de testes automáticos unitários promove:

- O design da aplicação criada;
- A produtividade dos programadores;

2.2 Raízes e Necessidades que levaram à criação do xUnit

O xUnit, ferramenta de testes unitários para software, pretendia ser um pacote de programas/ampliações e modelos de teste para ajudar quem desenvolve projectos de software a criar os seus próprios testes de erros. Por norma o xUnit costuma ser software *opensource*, o que também ajudou a sua grande difusão.

Kent Beck, foi o criador do conceito de xUnit, tendo a sua primeira implementação sido feita no início dos anos noventa, usando a linguagem de programação SmallTalk, ficou conhecido como SUnit.

A linguagem de programação SmallTalk, foi desenvolvida no início dos anos setenta no *Palo Alto Research Center* (PARC), por Alan Kay.
Trata-se de uma linguagem orientada por objectos que curiosamente nasceu de uma aposta entre funcionários dos laboratórios da *Xerox* que trabalhavam no PARC, a aposta consistia em que uma linguagem de programação baseada na ideia de troca de mensagens, não podia ser implementado numa pagina de código.

Kent Beck foi não só o mentor do projecto xUnit mas também de diversos projectos relacionados com engenharia de software como é o caso do XP, ou *Extreme Programing*, uma grande contribuição para o desenvolvimento das linguagens orientadas por objectos.

Kent Beck quando implementou a primeira versão do SUnit tinha como principal objectivo apresentar uma alternativa aos testes de software que se limitavam a simular utilizadores, e que muitas vezes tinham de ser refeitos e completamente reestruturados devido a uma pequena alteração no programa base, levando a um dispêndio de tempo muito grande por parte dos programadores a actualizarem o software de teste.
A versão original do SUnit tinha como filosofia a criação de pequenos testes (um por classe), com a finalidade de no caso de uma regressão no avanço do projecto, o erro

ou falha que deu origem a regressão, ser facilmente detectado e localizado. O facto de cada teste ser independente dos restantes, significa que qualquer alteração no projecto resulta em que apenas tenham de ser alterados os testes respectivos a cada classe envolvida nas alterações, não sendo necessário a alteração dos restantes testes, poupando assim tempo e esforço por parte de quem desenvolve.

Rapidamente a ideia de Kent Beck foi aceite e disponibiliza numa série de plataformas tais como C++: CPPUnit, Java: JUnit, JavaScripting: JSUnit, C: CUnit, PHP: PHPUnit, Python: PyUnit, entre muitas outras.

2.3 Vantagens da Utilização do xUnit na Detecção Precoce de Erros

Através da utilização de testes de código ao longo da implementação é possível reduzir a quantidade de bugs na aplicação final. Os testes do xUnit funcionam através de comparação de resultados das funções a serem testadas com valores esperados.

Assim, ao testar a função fica-se a saber com facilidade se foi correctamente implementada para responder a determinado problema. A utilização das unidades de teste reduz a dificuldade e tempo perdido em testes de software pois torna-se um processo automatizado e simples. Os testes são criados pelo programador e as cláusulas de teste são também definidas por ele. O utilizador final não tem intervenção nestes testes.

Ao criar os testes durante o desenvolvimento é mais provável não haver esquecimento de efectuar determinadas provas pois o desenvolvimento será [illegible] desenvolve software sem utilização de unidades específicas de prova, os testes podem ser feitos ao longo do desenvolvimento mas são temporários. Ou seja, faz-se um teste a determinada função e tem o valor esperado naquele momento, então, não se volta a testar, mesmo quando se muda o código, perdendo assim toda a integridade.

Actualmente existem plataformas específicas de teste para várias linguagens, algumas incluem uma interface gráfico onde se pode ver os testes e a sua conclusão, outros mostram apenas se a conclusão foi correcta ou se houveram erros.

O xUnit é uma tentativa de certificação de software, pois os testes foram efectuados sempre que necessário e certificados todos os módulos quanto ao seu funcionamento.

2.4 xUnit no Contexto de Engenharia de Software

O teste de software é muitas vezes menosprezado mas é um processo fundamental no desenvolvimento de uma aplicação de qualidade.

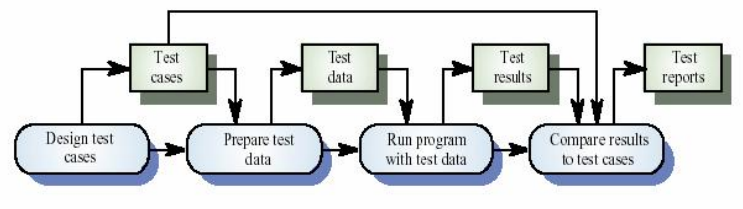

Fig. 2: Desenvolvimento de software

Sendo um processo fundamental no desenvolvimento de uma aplicação, o teste de software para ser considerado tem de apresentar características tais como: ser operável, observável, controlável, ter decomposição, simplicidade, estabilidade e compreensão, um bom teste é aquele que tem uma elevada probabilidade de revelar um erro, e um teste para ser bem sucedido é aquele que revele erros ainda não revelados.

Estes testes têm de ser sempre documentados e reproduzíveis.
A vantagem deste tipo de testes é que é feito sobre um módulo (classe). Após a escrita do código escreve-se o teste, ou, primeiro o teste e depois o código, dependendo do programador. Esta técnica permite que o teste encontre erros mais facilmente, visto que testa por classes, em vez de o fazer ao código completo.
O tempo de *debug* tem tendência a decrescer, o trabalho também, o que implica a redução do custo total de desenvolvimento do software.

A importância do xUnit na engenharia de software é elevada, pois com esta ferramenta podemos efectuar testes de melhor qualidade, comparados com os testes convencionais, isto é, podemos observar mais facilmente os erros, e verificar com melhor fiabilidade os valores das nossas variáveis e funções. Este tipo de ferramentas tem como objectivo fazer testes para que se possa confirmar a boa qualidade do projecto, de forma a obter maior confiança no final.

2.5 xUnit nas várias Linguagens: Java, C++, Http, Pascal, etc.

A abordagem deste assunto teria de começar inevitavelmente pela apresentação do JUnit, que é actualmente a ferramenta mais utilizada a este nível mas, é de bom-tom realçar que o JUnit não foi o pioneiro nesta área. O verdadeiro pioneiro a este nível foi o SUnit (Smalltalk[1]). Este sim foi o predecessor de todos os programas que actualmente se encontram para realizar tarefas de testes unitários.
Apesar disso e não descurando a sua ilegibilidade como o pai de todas as ferramentas na área do xUnit, poder-se-ia afirmar sem grande nível de contrariedade que o JUnit, se tornou o verdadeiro orgulho da família, remetendo para o anonimato toda uma

[1] Ver 2.2

outra série de ferramentas (incluindo o SUnit, o que se traduz por um muito escasso conhecimento do público, em geral de todas as outras ferramentas que não JUnit).
Isto deve-se, em parte, devido à revelância da linguagem em causa (Java), o JUnit esse sim foi o verdadeiro responsável pelo *boom*, que se deu a este nível com o aparecimento de várias outras ferramentas com funções em tudo semelhantes.

O JUnit foi criado por Kent Beck e Erich Gamma, é uma ferramenta desenvolvida especificamente com o intuito de proporcionar ao seu utilizador um método sólido consistente e fiável (mediante limites previamente definidos) acerca da capacidade das classes e métodos resolverem os problemas a que se propõem resolver.
Esta ferramenta conseguiu o que nenhuma outra havia feito previamente: teve sucesso. E juntamente com o sucesso, obteve experiência que foi posteriormente aplicada aos outros ramos que entretanto emergiram na área do teste unitário.

> Citando Elliotte Rusty Harold:
>
> "JUnit ...,... is almost indisputably the single most important third-party Java library ever developed. As Martin Fowler has said, "Never in the field of software development was so much owed by so many to so few lines of code." JUnit kick-started and then fueled the testing explosion. Thanks to JUnit, Java code tends to be far more robust, reliable, and bug free than code has ever been before. JUnit ...bring's the benefits of unit testing to a wide range of languages. nUnit (.NET), pyUnit (Python), CppUnit (C++), dUnit (Delphi), and others have test-infected programmers on a multitude of platforms and languages."

JUnit é uma ferramenta já bastante desenvolvida:

O que faz:
- Fornece um *template* para escrita de testes com instalação e execução;
- Permite a organização de testes hierarquizados;
- Permite a execução de testes de forma automática e fácil;
- Separa o relato dos testes da execução permitindo a utilização de diferentes várias aplicações de teste no mesmo ambiente.

O que ainda não faz:
- Gerar testes automaticamente para uma unidade a testar;
- Fornecer cobertura estatística;
- Indicar quando foi escrito um teste de fraca robustez, que não prova a eficácia da unidade.

Outras implementações:
- CppUnit

A primeira implementação do CppUnit foi realizada por Michael Feathers.
a própria página de desenvolvimento deste software que note-se, é livre, o nome do programa vem creditado da seguinte forma: CppUnit - *C++ port of JUnit*.
Muito embora o CppUnit inicialmente tenha sido o que o seu nome ainda hoje descreve, tem actualmente implementado algumas funcionalidades extras, e já se encontra em desenvolvimento a versão 2.0 que se espera ser mais do que do que uma mera transposição do JUnit.

- NUnit

NUnit é uma *framework* também *opensource* para aplicações Microsoft™ .NET. Por sua vez, esta mesma plataforma possui uma extensão *NUnit.Forms* que é uma expansão ao próprio NUnit também ela em *Opensource*, que se encarrega especificamente de permitir a manipulação de testes aos elementos de interface de utilizador nos formulários Windows™.
Foi também ela importada do JUnit, mas foi completamente remodelada estando actualmente escrita em C# tendo sido reestruturada para ter proveito de todas as funcionalidades da linguagem .NET.
Apesar de ser *opensource* e serem aceites contribuições de todos os que queiram participar, os principais responsáveis são Michael C. Two, Charlie Poole, Jamie Cansdale e Gary Feldman.

2.6 Casos Específicos de Utilização: JUnit

O JUnit é uma implementação bastante simples do conceito xUnit. A sua popularidade deveu-se uma integração semitransparente com o trabalho do programador.
O JUnit facilita a detecção de erros no código Java.

A sua utilização baseia-se numa *framework* composta por classes base (*TestCase, TestSuite, Test-Runner*), que serão esclarecidas mais à frente. Os testes codificados pelo programador derivam de TestCase que implementa a Classe *Test*.

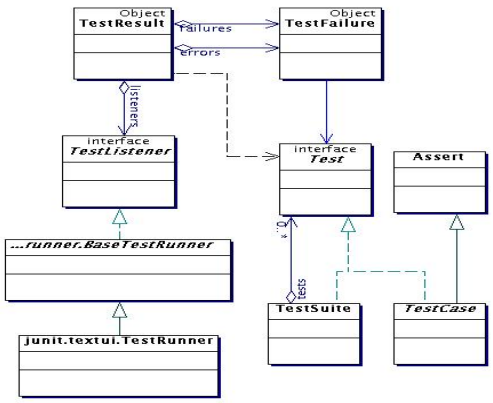

Fig. 3: Diagrama de Classes

JUnit é constituida essencialmente pelas seguintes classes:

- Uma *Test-Case* é uma classe típica de teste: cada uma das restantes classes representadas depende desta. A *Test-Case* é uma subclasse da classe *Test*.
- Uma *Test-Suite* representa um conjunto de testes. Ela deriva igualmente da classe *Test* e tem umas instâncias desta (Design Pattern "Composite"), que permite ter indiferentemente uma *Test-Case* ou outras *Test-Suite*.
- Uma *Test-Runner* permite de executar uma lista de *Test*.

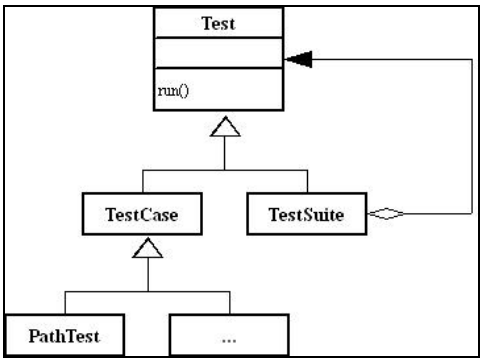

Fig. 4: Elementos da *framework*

Para utilizar este *framework*, basta escrever as classes de teste derivada da *Test-Case*, pôr uma instância de cada teste dentro de uma hierarquia de *Test-Suite* e por fim, dar todo o conteúdo ao *Test-Runner* para que ele executar.

Um exemplo prático da utilização do JUnit é:

Código da classe em teste:

```
public class Testa{
  int n;
  public static void Testa(int n){
     this.n = n;
  }

  public int getN(){
     return n;
  }
}
```

Código das classes de teste:

```
public class TesteTesta extends TestCase  {
   public TesteTesta(String testname){
       super(testname);
   }

   public void testegetN(){
       Testa testa= new Testa(5);
       assertTrue(testa.getN()==5);
   }
   //outros testes aqui
}

public class TodosTestes {
    public static Test suite() {
       TestSuite suiteTesta = new TestSuite("testa");
       suiteTesta.addTest(TesteTesta.class);
       return suiteTesta;
    }
```

Este exemplo mostra um teste ao método de criação de uma classe.
É bastante simples expandir os testes efectuados pois é apenas necessário acrescentar à classe *TesteTesta* os outros testes a efectuar.

3 Conclusão

Seria muito positivo se os programadores menos experientes aprendessem cedo, que testar é um processo fácil e de máxima importância e não um *extra* que é realizado no fim de escrever código.

O ensino cuidadoso da verificação não automatizada de código, inicia o processo de aprendizagem de como simular e verificar a execução de um algoritmo.
A verificação manual, é um processo testado e provado que resulta em bastantes erros não detectados durante o desenvolvimento de *software*.

xUnit , pelo contrário, é uma ferramenta de testes nova e tecnologicamente inovadora.

Se os programadores menos experientes aprenderem a utilizar xUnit para o desenvolvimento de testes, poderão descobrir o *gozo* em realizar testes e também tornarem-se melhores programadores.

4 Referências Bibliográficas

- http://nunit.com/testweb/
- http://www-128.ibm.com/developerworks/opensource/library/os-junit/?ca=dgr-lnxw07JUnite

- http://www-128.ibm.com/developerworks/java/library/j-junit4.html?ca=dgr-lnxw01JUnit4
- http://www.linux.ie/articles/tutorials/junit.php
- http://cppunit.sourceforge.net/doc/1.8.0/
- http://cppunit.sourceforge.net/cgi-bin/moin.cgi
- http://sourceforge.net/projects/cppunit
- http://cppunit.sourceforge.net/cppunit2/