UNIVERSIDADE FEDERAL DO PARANÁ - UFPR
Bacharelado em Ciência da Computação

SOFT
DISCIPLINA: Engenharia de Software **DATA: _____/_____/_____**
AULA NÚMERO: 16 **PROFESSOR: Andrey**

APRESENTAÇÃO

O objetivo desta aula é apresentar e discutir conceitos relacionados a Testes de Software.

DESENVOLVIMENTO

Teste Caixa Preta ou Teste Comportamental

O teste caixa-preta tem como foco os requisitos funcionais do software. Esta técnica de teste exercita o software em função das entradas e das saídas correspondentes para cada módulo ou subsistema.

Classes de Equivalência

É uma técnica usada para reduzir a quantidade de casos de testes a um montante gerenciável, mas mantendo um elevado grau de cobertura. Uma Classe de Equivalência representa um conjunto de valores válidos ou inválidos para condições de entrada.

O conjunto de entradas possíveis é dividido em partições;
Os elementos de um subconjunto são equivalentes em relação a uma característica;
Os subconjuntos são disjuntos.

A técnica admite que se as entradas estão divididas em partições de equivalência, o comportamento do sistema será o mesmo para qualquer entrada escolhida em uma mesma partição.

Dessa forma não é preciso testar todas as entradas possíveis, basta testar uma opção de cada partição.

Valores de Fronteira

É uma técnica em testes de software utilizada para testar valores nas "fronteiras" das faixas de entrada, por exemplo...

Considere as seguintes regras organizacionais, sobre contratação baseada em idade, em um programa de uma empresa de recursos humanos:

0-16 Não Contrata
16-18 Contrata meio-expediente
18-55 Contrata tempo integral
55-99 Não Contrata

Uma regra correta seria:

Se (idadeCandidato >= 0 && idadeCandidato <= 15) estadoContratação = "Não";
Se (idadeCandidato >= 16 && idadeCandidato <= 17) estadoContratação = "Meio";
Se (idadeCandidato >= 18 && idadeCandidato <= 54) estadoContratação = "Integral";
Se (idadeCandidato >= 55 && idadeCandidato <= 99) estadoContratação = "Não";

Conjuntos de valores interessantes para serem testados são {-1,0,1},{15,16,17},{17,18,19},{54,55,56} e {98,99,100}.
Dependendo das pré-condições, valores como {-42, 1001, FRED, %$#@} também devem ser testados

Passos:
Identifique as classes de equivalência.

Identifique as fronteiras de cada classe.
Crie casos de teste para cada valor de fronteira, escolhendo um ponto abaixo, um acima e outro ponto na fronteira.

Tabela de Decisão

Tabelas de decisão é uma excelente técnica para capturar alguns tipos de requisitos do sistema e para documentar modelagens internas do sistema. Elas são utilizadas para registrar regras de negócio complexas. Também servem como guia para criar casos de teste.

Tabelas de decisão representam regras de negócio complexas baseadas em um conjunto de condições:

Pairwise

Cada um tem uma grande quantidade de combinações a serem testadas.

Cada um tem uma grande quantidade de combinações que podem ser muito arriscadas de não serem testadas.

Como selecionar um subconjunto razoavelmente pequeno e eficiente, que ache uma grande variedade de erros, mais, do que você esperaria de tal subconjunto?

A resposta é não tentar testar todas as combinações de valores e variáveis e sim testar todos os pares de variáveis.

Exemplos:

Para um sistema com 4 parâmetros diferentes cada um podendo assumir 3 valores, resulta em 3^4 = 81 combinações. Com Pairwise reduz-se para 9 testes.

Para um sistema com 13 parâmetros diferentes cada um podendo assumir 3 valores, resulta em 3^{13} = 1,594,323 combinações. Com Pairwise reduz-se para 15 testes.

Para um sistema com 20 parâmetros diferentes cada um podendo assumir 10 valores, resulta em 10^{20} (100.000.000.000.000.000.000) combinações. Com Pairwise reduz-se para 180 testes.

JUNIT

O que é JUnit?
Um framework que facilita o desenvolvimento e execução de testes de unidade em código Java

- Uma API para construir os testes: junit.framework.*
- Aplicações para executar testes: TestRunner

Há várias formas de usar o JUnit. Depende da metodologia de testes que está sendo usada

- Código existente: precisa-se escrever testes para classes que já foram implementadas
- Desenvolvimento guiado por testes (TDD): código novo só é escrito se houver um teste sem funcionar

JUnit para testar código existente

Exemplo de um roteiro típico
1. Crie uma classe que estenda junit.framework.TestCase para cada classe a ser testada

```
import junit.framework.*;
class SuaClasseTest extends TestCase {...}
```

2. Para cada método xxx(args) a ser testado defina um método public void testXxx()* no test case

SuaClasse:
```
   public boolean equals(Object o) { ... }
```
SuaClasseTest:
```
public void testEquals() {...}
```

Usará reflection para descobrir métodos que começam com "test"

3. Crie um método estático suite()* no test case
```
    public static Test suite() {
          return new TestSuite(SuaClasseTest.class);
    }
```

Cada método testXXX() do seu TestCase é um teste
Escreva qualquer código que sirva para verificar o correto funcionamento da unidade de código testada
Use asserções do JUnit para causar a falha quando resultados não estiverem corretos

Principais asserções
assertEquals(objetoEsperado, objetoRecebido)
assertTrue(valorBooleano)
assertNotNull(objeto)
assertSame(objetoUm, objetoDois)
fail()

Exemplo:
Uma classe que realiza conversões entre temperaturas Celsius e Farenheit.

```
import java.math.BigDecimal;
public class Conversoes {
       public BigDecimal fahrToCels(BigDecimal fahr) {
              double fahrenheit = fahr.doubleValue();
              double celsius = (5.0/9.0) * (fahrenheit - 32);
              return new BigDecimal(celsius);
       }
       public BigDecimal celsToFahr(BigDecimal cels) {
              double celsius = cels.doubleValue();
              double fahrenheit = (9.0/5.0) * celsius + 32;
              return new BigDecimal(fahrenheit);
       }
}
```

A classe de testes pode ficar assim:

```
import junit.framework.*;
import java.math.BigDecimal;
public class ConversoesTest extends TestCase {
    Conversoes conv = new Conversoes();
    public void testFahrToCels() {
        assertEquals(new BigDecimal(100),
                     conv.fahrToCels(new BigDecimal(212)));
        assertEquals(new BigDecimal(-40),
                     conv.fahrToCels(new BigDecimal(-40)));
    }
    public void testCelsToFahr() {
        assertEquals(new BigDecimal(212),
                     conv.celsToFahr(new BigDecimal(100)));
        assertEquals(new BigDecimal(-40),
```

```
        conv.celsToFahr(new BigDecimal(-40)));
    }
}
```

ATIVIDADE

1. Construa uma classe em java que possua os seguintes métodos e construa uma classe de etstes usando JUNIT para testá-la:
   Um método que receba um número e retorna a lista dos seus divisores
   Um método que receba dois números e retorna o máximo divisor comum pelo método de Euclides.

BIBLIOGRAFIA BÁSICA

PRESSMAN, R. S.. Engenharia de Software. 6a. ed. McGraw-Hill. 2006

CARVALHO, A. M. B. R.; CHIOSSI, T. C. S.. Introdução à engenharia de software. Editora da Unicamp. 2001.

OLIVEIRA, M. Técnicas de Testes Caixa-Preta. Apresentação no II Encontro Brasileiro de Teste de Software, outubro 2007.

ROCHA, ELDER DA.. Desenvolvimento guiado por testes com JUNIT. Apresentação no Congresso Brasil Software Week – FenaSoft. São Paulo: 2003. Disponível em <http:\\www.argonavis.com.br/palestras/fenasoft2003/**junit**/**JUnit**_Fenasoft.pdf>, acessado em 25/11/2009.