USP - Universidade de São Paulo

IME - Instituto de Matemática e Estatística

MAC0332

Engenharia de Software

Testes

Marco Aurélio Gerosa

gerosa@ime.usp.br

Conceitos Básicos

- O que é qualidade de software?
- Como garantir a qualidade de software?
- Validação x verificação
  - estamos construindo o produto certo?
  - estamos construindo certo o produto?
- Prevenção de defeitos é melhor do que remoção de defeitos (Mays, 1990)
- Um desenvolvedor não é indicado para testar seu próprio código (Weinberg, 1971)
- Aproximadamente 80% dos feitos são provenientes de 20% dos módulos (Pareto, 1897)(Endres, 1975)

Testes

- Engano => Imperfeição no código => Resulta em um falha => Manifesta um erro [IEEE 610.12, 1990].
  - Defeito é uma palavra genérica para imperfeição, falha ou erro.
- Verificação no final de cada fase (pode ser tarde demais)
- Validação no final do projeto (mais tarde ainda)
- Atividades contínuas de testes devem ser feitas durante o projeto
  - Teste de unidade
  - Teste de integração
  - Teste de aceitação
- Grupo de qualidade
- Grupo de teste

Testes de software

- Prejuízos de aproximadamente $59.5 bilhões na economia dos EUA (Fonte: NIST/2002)
- Tipos: caixa branca, preta ou cinza
- Fases: de unidade, de integração e de sistema
- Tipos: aceitação, performance, stress etc.
- Testes automatizados

Falhas

- Entre 60 a 70% das falhas são nos requisitos, análise ou projeto (Boehm, 1970).
- Exemplo: Jet Propulsion Laboratory inspections (1992)
  - 1.9 falhas por página de especificação
  - 0.9 por página de projeto
  - 0.3 por página de código
- Corrigir uma falha em fases mais avançadas do desenvolvimento demanda:
  - Alterar código e documentação
  - Executar teste de regressão
  - Reinstalar o produto nos clientes

Testes de unidade

- Não é “testes unitários”
- Testes automatizados
  - Possibilita rodar uma bateria grande de testes sem depender da intervenção ou interpretação humana dos resultados
  - Rápido de executar – podem ser executados a cada alteração do sistema
  - Os testes podem cobrir todo código produzido
  - Oferece documentação sobre a funcionalidade do sistema
  - Mais segurança na manutenção

- JUnit - http://www.junit.org
  - Kent Beck, Erich Gamma
  - Open Source

Exemplo de teste

```
import br.usp.ime.mac0332.Calculadora;

public class TesteCalculadora {

    private Calculadora calculadora;

    @Before
    public void setUp() throws Exception {
        calculadora = new Calculadora();
    }

    @After
    public void tearDown() throws Exception {
        calculadora = null;
    }

    @Test
    public void somaPositivos() {
        assertEquals(42, calculadora.soma(11, 31));
    }

    @Test
    public void somaNegativos() {
        assertEquals(-42, calculadora.soma(-11, -31));
    }

    @Test
    public void divide() {
        assertEquals(42, calculadora.divide(84, 2));
    }

    @Test(expected = ArithmeticException.class)
    public void dividePorZero() {
        calculadora.divide(84, 0);
    }
}
```

Asserts

- assertEquals
- assertTrue / assertFalse
- assertSame / assertNotSame
- assertNull / assertNotNull
- assertArrayEquals

JUnit 3 x Junit 4

JUnit 3

```
import junit.framework.*;

public class MultiplicationTest extends TestCase {
   public void testMultiplication() {
     assertEquals("Multiplication", 6, 3 * 2);
   }
}
```

JUnit 4

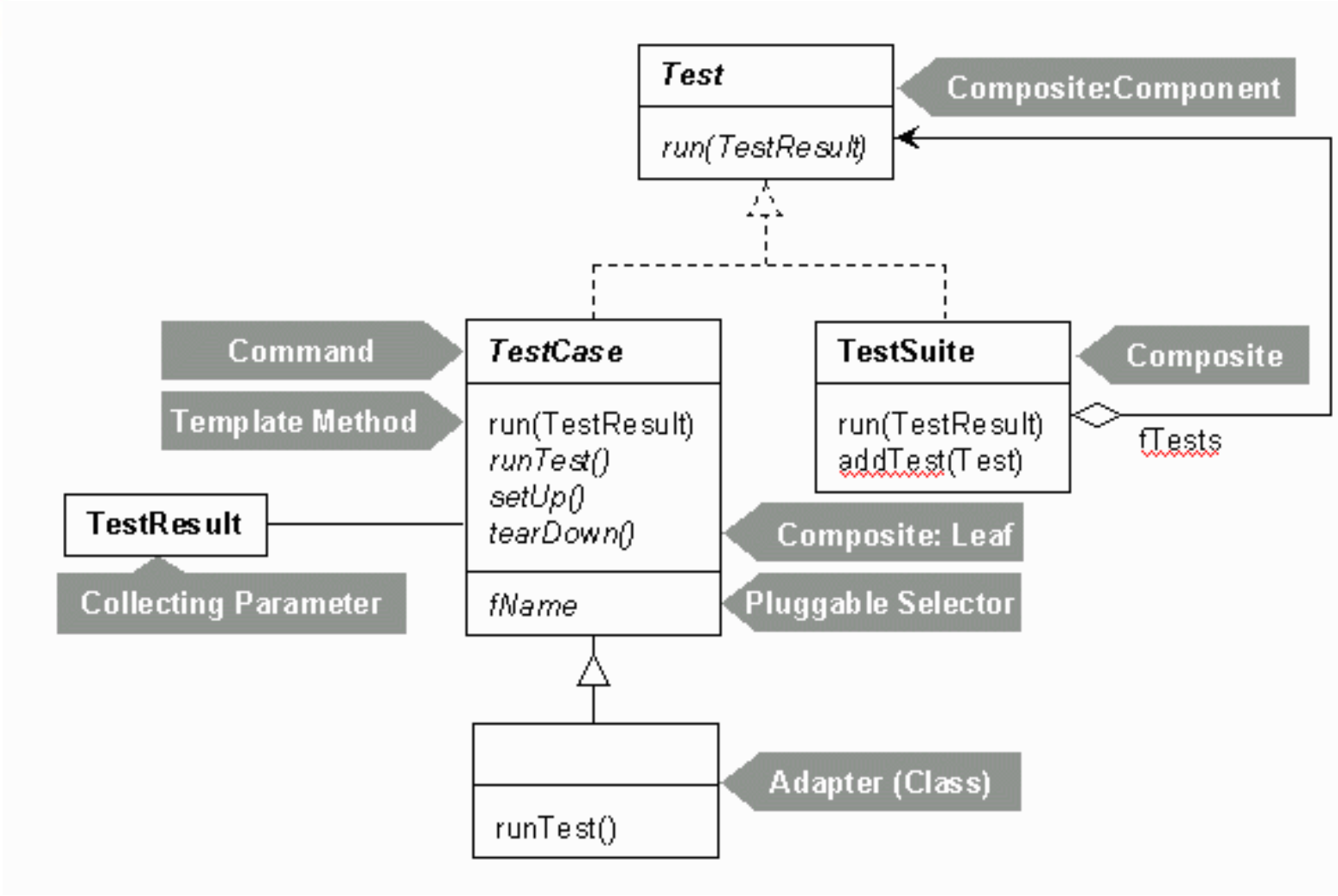
```
import org.junit.*;

public class MultiplicationTest {
   @Test
   public void multiplication() {
     Assert.assertEquals("Multiplication", 6, 3 * 2);
   }
}
```

JUnit 3 x JUnit 4

JUnit 3	JUnit 4
public void testX()	@Test
public void setUp()	@Before
public void tearDown()	@After
assertEquals()	Assert.assertEquals()

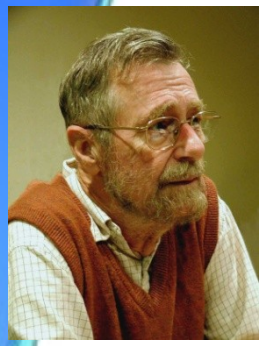
Arquitetura JUnit



Princípios para escrever testes

- Código dos testes deve ser simples
- Testes podem conter erros
- Não devem exigir intervenção humana
- Devem documentar o sistema

Citações



"Program testing can be used to show the presence of bugs, but never to show their absence."

Edsger W. Dijkstra

“Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead.” Martin Fowler.

“Any program feature without an automated test simply does not exist.”

Kent Beck.

Citações

- “If a program feature lacks an automated test, we assume it doesn’t work. This seems much safer than the prevailing assumption, that if a developer assures us a program feature works, then it works now and forever.” JUnit Cook’s Tour
- “Telling me that your program works now doesn’t help me, because it doesn’t assure me that your program will work one minute from now after I integrate, and it doesn’t assure me that your program will still work in five years, when you are long gone.” Junit Cook’s Tour
- “Inspecionar para prevenir defeitos é bom; Inspecionar para encontrar defeitos é desperdício” - Shigeo Shingo

Mock Objects

- Simulam objetos reais
- Úteis para isolar o teste de um objeto do outro ou quando temos objetos que são difíceis de criar, reproduzir, lerdos, que ainda não existem etc.
- Você configura a expectativa do comportamento do objeto
- Frameworks
  - JMock
  - EasyMock
  - Mockito

Mockito

- stub – antes da execução
- verify – após a execução

- Depois de criado, o Mock Object irá “lembrar” todas as interações com ele.

Exemplo

```
class FiltraPessoasMaioresDeIdade {
        private RepositorioDePessoas pessoas;
        public FiltraPessoasPorIdade(RepositorioDePessoas pessoas) {
                this.pessoas = pessoas;
        }

        public List<Pessoa> filtra() {
                List<Pessoa> filtradas =new ArrayList<Pessoa>();
                for(Pessoa p : pessoas.pegaTodas()) {
                        if (p.getIdade() >= 18) filtradas.add(p);
                }
                return filtradas;
        }}

class FiltraPessoasMaioresDeIdadeTest {

@Test
public void deveFiltrarApenasPessoasMaioresDe18Anos() {
        RepositorioDePessoas pessoas = mock(RepositorioDePessoas.class);
        Pessoa maior = new Pessoa("José", 21);
        when(pessoas.pegaTodas()).thenReturn(Arrays.asList(maior, new Pessoa("criança", 10)));
        FiltraPessoasMaioresDeIdade filtro = new FiltraPessoasMaioresDeIdade(pessoas);
        List<Pessoa> pessoas = filtro.filtra();
        assertEquals(1, pessoas.size());
        assertSame(maior, pessoas.get(0));
}}
```

Exemplos

```
import static org.mockito.Mockito.*;

List mockedList = mock(List.class);
mockedList.clear();
verify(mockedList).clear();
```

```
List mock = mock(List.class);
when(mock.get(0)).thenReturn("one");
when(mock.get(1)).thenReturn("two");
when(mock.get(2)).thenThrow(new RuntimeException());
(...)
```

Argument Matchers

- when(mockedList.get(anyInt()).thenReturn(“olá”);
- verify(mockedList).get(anyInt());
- argThat(isValid())

Outras funções

- InOrder – verifica ordem de chamada de métodos
- doThrow – verifica o lançamento de exceção
- times(n) – número de chamadas
- atLeastOnce(), atMost(n), atLeast(n) etc.
- Exemplo
  - mockedList.add("once");
  - mockedList.add("twice"); mockedList.add("twice");
  - // testa
  - verify(mockedList).add("once");
  - verify(mockedList, times(1)).add("once");
  - verify(mockedList, times(2)).add("twice");
  - verify(mockedList, never()).add("never happened");
  - verify(mockedList, atLeastOnce()).add(“twice");

- http://docs.mockito.googlecode.com/hg/latest/org/mockito/Mockito.html