Testování jednotek

Radek Mařík

CA CZ, s.r.o.

January 20, 2009

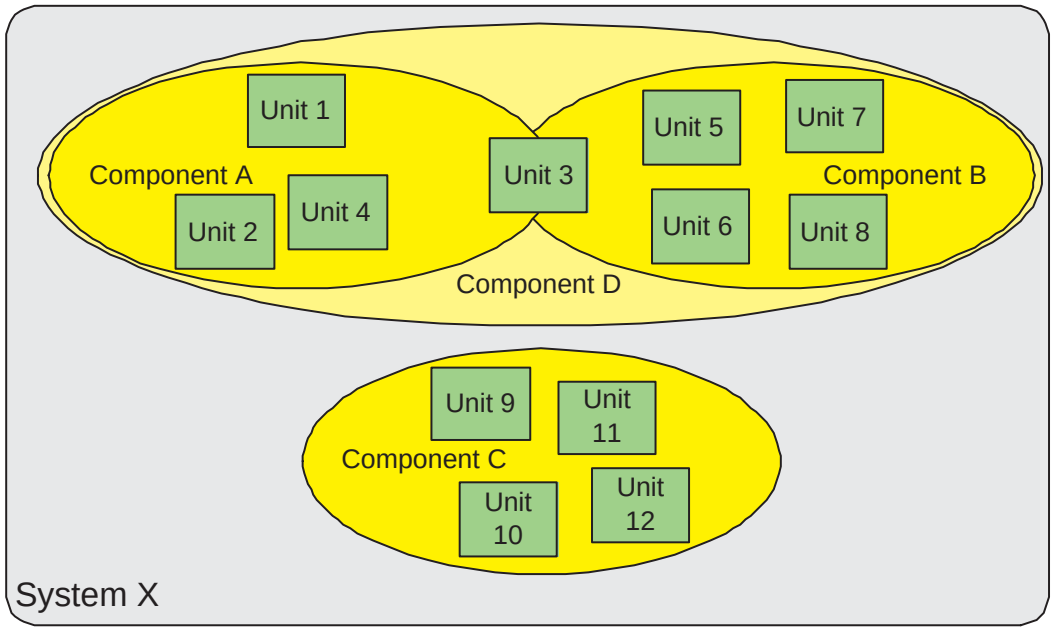
CA Labs

Transforming IT Management

Obsah

1. Základy testování jednotek
   - Principy
   - Architektura
2. Nástroje pro testování jednotek
   - Obecně
   - JUnit 3.8
   - JUnit 4.5

Terminologie



Jednotka je nejmenší testovatelný kus softwaru. Znamená to, že může být přeložen, sestaven, spuštěn a řízen testovacím přípravkem nebo řadičem.

- **procedurální programování:** program, funkce, procedura.
- **objektově-orientované programování:** třída

Komponenta, Modul je integrovaný agregát jedné a více jednotek.

Cíl

- Cílem je
  - izolování každé části programu,
  - prokázání, že jednotlivé individuální části jsou správné.
  - **otestování dané individuální metody v izolaci.**
- Každý testovací případ je nezávislý na ostatních.
- Testy zaměřující se na chování jiné než určené signaturou metody nejsou považovány za jednotkové.
- Každý test určuje striktní explicitně popsaný kontrakt, který daný kód musí splnit.
- Eliminace interakcí mezi jednotkami umožňuje jejich testování za podmínek, že potřebné další jednotky nejsou jestě implementovány.

Strategie

- Testování jednotek typicky provádí **vývojáři**.
- Testování jednotek je základním pilířem **extrémního programování** (XP).
- Automatizované, opakovatelné, a proaktivní testování.
- **Vývoj řízený testy**
  - Pokud napíšete nejprve testy, pak se hned z počátku ztotožňujete s rolí zákazníka.
  - "Zákazník má vždy pravdu" - prioritní pohled zákazníka.

Typická pravidla

1. Napiš nejprve test, který je možné přeložit (ale ne více).
   - vede k testovatelnému kódu,
   - vede k cílově-orientované implementaci kódu,
2. Nikdy nepiš test, který je bezprostředně úspěšný po jeho napsání.
   - každý test by měl ověřovat novou, ještě neimplementovanou vlastnost,
   - ověření správnosti testu reakcí na implementovanou vlastnost,
3. Začni s prázdným případem, či s něčím, co nepracuje.
4. Neboj se něčeho, co vede na triviální věci, které splní test.
   - testy, které ověřují hraniční jednoduché hraniční podmínky
5. Eliminace interakcí podporuje testovatelnost.
   - vede na udržovatelný kód, se kterým je možné pracovat i za podmínek, že okolní části kódu chybí,
6. Používej imitačních objektů.
   - poskytují jasný pohled na průběh interakcí mezi komponentami.
7. Chodící testy se neodstraňují, tj. jsou spojeny s implementovanou funkcionalitou.

Výhody I

- Umožňuje změny implementace
  - kontroluje stabilitu funkcionality při refaktorování kódu,
  - jednotkové testy odráží zamýšlené použití kódu,
  - dobrý návrh pokrývá všechny použitelné cesty.
- Zesiluje separaci rozhraní od implementace
  - Zaměření pouze na jednotku vyžaduje minimalizaci interakce jednotky s okolím.
  - Případná rozhraní jsou explicitně definována, aby bylo možné prostředí jednotky vytvořit pomocí náhrad.
  - Nalezení a eliminování nadbytečných závislostí mezi jednotkami.

Výhody II

- Zjednodušuje integraci
  - při návrhu a implementaci zdola-nahoru zjednodušuje integrační testy.
- Poskytuje "živou" dokumentaci
  - vývojář může vyčíst funkcionality z testovacího kódu,
  - umožňuje porozumění API jednotky,
  - identifikuje jak
    - positivní chování, tj. cílenou funkcionalitu,
    - tak negativní chování, např. při nevhodných parametrech či zpracování výjimek
  - na rozdíl od běžné dokumentace se vyvíjí současně s implementací kódu.
- Chování identifikované testy podporuje lepší komunikaci s ostatními programátory
  - chování a jeho protokoly jsou explicitně zachyceny,
  - při žádostech o změny selhávající testy ihned identifikují problémy,
  - eliminuje případy použití s vedlejšími skrytými účinky,

Nevýhody, Omezení

- Nechytí všechny chyby programu.
- Testuje pouze funkcionalitu jednotek. Nenalezne chyby
  - integrační,
  - výkonostní,
  - systémové.
- Jako všechny ostatní formy testování
  - může pouze ukázat **přítomnost chyb**,
  - ale nemůže prokázat jejich **absenci**.
- K identifikaci příčin chyb je nutné podpořit systémem řízení změn.

Testovací prostředí

- angl. unit testing framework,
- je software zajišťující testování jednotek,
  - spouštění celých sad testů, případně vybraných částí,
  - spuštění vybraného testu.
- vytváří proměnlivé podmínky testů,
- monitoruje chování jednotek a jejich výstupů,
  - průběžné sledování běhu testů,
  - generovaní reportů
- umožňuje analýzu výsledků,
- skládá se z exekučního modulu a sady testovacích scriptů

Koncepty

Příslušenství (angl. fixture) - sada objektů, které jsou testovány.

Testovací případ (angl. test case)

- třída, která definuje příšlušenství pro řadu testů,
- definuje proměnnou pro každou položku příslušenství,
- zaručuje vytvoření a likvidaci příslušenství.

Nastavení (angl. setup) - metoda použitá pro inicializaci proměnných před každým testem či sadou testů.

Úklid (angl. tearDown) - metoda pro uvolnění zdrojů alokovaných nastavením.

Testovací sada (angl. test suite) - soubor testovacích případů.

Řešení chybějících jednotek

Náhrada (angl. test double) - obecný pojem použitý pro jakýkoliv objekt, jehož účelem je doručit funkcionalitu reálného objektů pro účely testování.

Prázdný object (angl. dummy object) - objekt se předává, typicky jako parametr, ale není de facto použit.

Padělek (angl. fake object) - plná funkcionalita řádné implementaci, typicky nevhodná pro nasazení v reálné aplikaci,

- nahrazení reálné databáze jednoduchou databází v paměti.

Imitátor (angl. mock object) - objekty plní případně kontrolují vybranou specifikaci jejich volání,

- typicky specifikují sekvenci volání, tj. verifikují chování

Zástupce (angl. stub) - je schopen dodat odpovědi v omezených případech cílených jednotlivými testy, tj. verifikuje stav.

- mohou zaznamenávat i průběh volání.

Přehled prostředí - vybrané příklady

- Smalltalk
  - Kent Beck publikoval myšlenku prostředí pro testování jednotek v roce 1998. Tato myšlenka a navržený protokol byl pak převzat a implementován řadou dalších (10^2) programovacích prostředí.
  - SUnit
- Java
  - TestNG
  - JUnit
- Python
  - PyUnit
- C++
  - CppUnit
  - CxxUnit
- .NET
  - NUnit
  - Visual Studio Team Edition
- Delphi
  - DUnit

JUnit

- http://www.junit.org
- jednotkové testování pro jazyk Java
- podpora vývojovými prostředky
  - Ant,
  - Maven,
  - NetBeans
  - Eclipse.
- JUnit 3.8
  - lokalizace testů založena na
    - dědičnosti tříd,
    - reflexi,
    - konvenci jmen.
- JUnit 4.x, nyní 4.5
  - založen na vlastnostech Java 5
    - anotace
    - statický import

Architektura JUnit 3.8

- *Příkazová šablona* pro definici testů
  - *TestCase* je příkazový objekt, který obsahuje implementace testovacích metod
  - *testXXX()* definuje testovací metodu (začíná "test"),
  - *assert()* metodu a řadu jejích variant lze použít pro porovnání očekávaných a aktuálních výsledků.
  - *setUp()* a *tearDown()* metody inicializují a ruší společné objektu příslušenství pro každou testovací metodu zvlášť.
- *Kompoziční šablona* pro vytvoření hierarchie testů
  - *TestSuite* definuje hierarchii testů,
  - vytváření testů obvykle automatizováno užitím reflexe a konvencí jmen,
  - postupy se velmi liší,
- *Běh testů*
  - **Textové rozhraní:** java junit.textui.TestRunner junit.samples.AllTests
  - **Grafické rozhraní:** java junit.swingui.TestRunner junit.samples.AllTests

JUnit 3.8 příklad: testovací metody

```
import junit.framework.TestCase;
public class AdditionTest extends TestCase {
  private int x = 1;
  private int y = 1;

  @Override protected void setUp() {
    y = 2;}

  public void testAddition() {
    int z = x + y;
    assertEquals(3, z);}

  protected void tearDown() {
    System.gc();}
  ......
}
```

JUnit 3.8 příklad: testovací sada

```
import junit.framework.*;
public class AdditionTest extends TestCase {

  .......

  public static Test suite(){
    return new TestSuite(AdditionTest.class);
  }

  public static void main(java.lang.String[] argList){
    junit.textui.TestRunner.run(suite());
  }
}
```

Architektura JUnit 4.5

- *Příkazová šablona* pro definici testů
  - testovací třída se neodvozuje z TestCase,
  - *assert()* metodu a její varianty lze použít podobně jako v JUnit 3.8.
  - *@Test* dekorace definuje testovací metodu,
  - *@Before* dekorace označuje metody inicializují společné objektu příslušenství pro každou testovací metodu zvlášť.
  - *@After* dekorace označuje metody rušící společné objektu příslušenství po každé testovací metodě.
  - *@BeforeClass* dekorace označuje veřejné statické metody běžící před třídou.
  - *@AfterClass* dekorace označuje veřejné statické metody běžící po třídě.
  - *@Ignore* dekorace označuje ignorovaný test.

Architektura JUnit 4.5

- *Kompoziční šablona* pro vytvoření hierarchie testů
  - *@RunWith(Suite.class)* dekorace specifikuje třídu testovací sady
  - *@SuiteClasses(TestA.class, TestB.class)* dekorace specifikuje třídy testů patřící do příslušné sady
- *Běh testů*
  - přímá podpora v IDE nebo **ant**
  - lze použít podpory vytvořené v JUnit 3.8 pomocí adaptérů

JUnit 4.5 příklad: testovací metody

```
import org.junit.*;
import static org.junit.Assert.*;
public class AdditionTest {
  private int x = 1;
  private int y = 1;

  @Before public void setUp() {
    y = 2;}

  @Test public void testAddition() {
    int z = x + y;
    assertEquals(3, z);}

  @After public void tearDown() {
    System.gc();}
  ......
```

JUnit 4.5 příklad: testovací sada

```
import junit.framework.JUnit4TestAdapter; //from 3.8
public class AdditionTest {

  .......

  public static junit.framework.Test suite() {
    return new JUnit4TestAdapter(AdditionTest.class);
  }

  public static void main(java.lang.String[] argList){
    junit.textui.TestRunner.run(suite());
  }
}
```

Literatura I