<embed/it>

Testování

PV168
7. 3. 2017 & 14. 3. 2017
Petr Adámek

Testování

- Co je to testování a proč testovat
- Základní pravidla
- Druhy testů
  - Podle způsobu provedení
  - Podle cíle
- Jednotkové testování
- Principy
- Mock objekty

Testování aplikací

- Ověřuje soulad implementace se specifikací a s očekáváním zákazníka.
- Je důležitou součástí procesu řízení kvality vývoje software
- Na rozdíl od formální verifikace neumožní odhalit všechny potenciální chyby

Zakládní pravidla

- Testy by měly být reprodukovatelné.
- Testy by měly být deterministické, tj. měly by mít na začátku vždy stejné vstupní podmínky.
- Testy by měly být nezávislé, tj. nebýt ovlivněny ostatními testy.
- Testy by měly být levně opakovatelné.

Druhy testování podle metody

- **Ruční testování:**
  - nízké vstupní náklady, drahé opakování;
  - obtížné zajištění reprodukovatelnosti, determinismu a nezávislosti
- **Automatizované testování:**
  - vysoké vstupní náklady, levné opakování;
  - snadné zajištění reprodukovatelnosti, determinismu a nezávislosti.

Druhy testování podle cíle

- Jednotkové testování
- Integrační testování
- Funkční testování
- Akceptační testování
- Testování výkonu a škálovatelnosti
- Testování uživatelské přívětivosti
- Testování bezpečnosti

Jednotkové testování

U jednotkového testování se snažíme otestovat jednotlivé komponenty vyvíjeného systému na té nejnižší úrovni.

Jednotlivé testované komponenty by měly být izolovány od svého okolí, aby se zamezilo vlivu tohoto okolí na testovanou komponentu.

Interakce s okolím je simulována pomocí falešných objektů, které simulují chování okolí v konkrétním testovacím scénáři (viz Mock Objekty).

Čím je lépe provedená dekompozice, tím je snadnější jednotkové testování.

Nástroje

- **Testovací rámce**
  - JUnit
  - TestNG
- **Knihovny pro ověřování platnosti invariantů**
  - Hamcrest
  - AssertJ

Příklad

```
public class CalculatorTest {

    private Calculator c;

    @Before
    public void setUp() {
        c = new Calculator();
    }

    @Test
    public void testDivide() {
        assertEquals( 9, c.divide( 99, 10));
        assertEquals(10, c.divide(100, 10));
    }

    @Test(expected = IllegalArgumentException.class)
    public void testDivideByZero() {
        c.divide(100, 0);
    }
}
```

Postup

- **Základní pravidla**
  - Výstupem testu je ANO/NE (boolean)
  - Nejdříve testy, potom kód (viz XP a TDD).
  - Při opravě chyby nejdříve testy, potom oprava (ochrana proti regresím)
  - Triviální get/set metody se netestují
  - Testujeme všechny nestandardní situace a hraniční hodnoty
  - Single responsibility: jeden test = jedna testovaná věc
  - Chybové hlášky a komentáře nejsou vždy potřeba
  - Testy se spouští po každé změně

Příklad – testování výjimek (JUnit)

```
public class CalculatorTest {

    private Calculator c = new Calculator();

    @Test(expected = IllegalArgumentException.class)
    public void testDivideByZeroA() {
        c.divide(100, 0);
    }

    @Test
    public void testDivideByZeroB() {
        try {
            c.divide(100, 0);
            fail();
        } catch (IllegalArgumentException ex) {};
    }

    @Rule
    public ExpectedException expectedException = ExpectedException.none();

    @Test
    public void testDivideByZeroC() {
        ExpectedException.expect(IllegalArgumentException.class);
        c.divide(100, 0);
    }
}
```

Testování databázových aplikací

- **Jak na testování databázových aplikací?**
  - Použití ORM
  - Abstraktní DAO objekt
  - In-memory databáze
  - DBUnit

Příklad

- **BodyManagerImplTest.java** (https://goo.gl/4FYzCj)

Interakce s okolím

- **Komponenty by se měly testovat izolovaně**
  Je ale nutné nějak simulovat interakci s okolím:
- **K tomu slouží tzv. Mock objekty.**
- **Tyto objekty musí být typově kompatibilní se simulovanou komponentou.**
  - Dědění
  - Implementace rozhraní (vhodnější)
- **Mock objekty můžeme vytvářet ručně (pracné) nebo pomocí nástrojů**
  - Mockito, EasyMock, jMock

Příklad – ruční tvorba Mock objektů

```
public class CurrencyConvertorTest {

    @Test
    public void testConvert() {

        ExchangeRateTable exchangeRateTable = new ExchangeRateTable() {
            public BigDecimal getExchangeRate(Currency currency) {
                return BigDecimal.valueOf(28.2);
            }
        };

        CurrencyConvertor convertor = new CurrencyConvertor(exchangeRateTable);
        Currency czk = Currency.getInstance("CZK");
        BigDecimal actualResult = convertor.convert(czk, BigDecimal.valueOf(10));
        BigDecimal expectedResult = BigDecimal.valueOf(282.0);

        assertThat(expectedResult)
                .isEqualTo(actualResult);
    }
}
```

Příklad – Mockito

```
public class CurrencyConvertorTest {

    @Test
    public void testConvertMockito() {

        Currency czk = Currency.getInstance("CZK");

        ExchangeRateTable exchangeRateTable = mock(ExchangeRateTable.class);

        when(exchangeRateTable.getExchangeRate(czk))
                .thenReturn(BigDecimal.valueOf(28.2));

        CurrencyConvertor convertor = new CurrencyConvertor(exchangeRateTable);
        BigDecimal actualResult = convertor.convert(czk, BigDecimal.valueOf(10));
        BigDecimal expectedResult = BigDecimal.valueOf(282.0);

        assertThat(expectedResult)
                .isEqualTo(actualResult);
    }
}
```

Závěr