Testiranje koda - JUnit

Bojan Tomić

Testiranje programa

- Dinamička provera ponašanja programa izvođenjem konačnog broja testova i upoređivanjem sa očekivanim ponašanjem programa
- Programska mana (“software fault”)
  - Neki nedostatak u okviru programa
  - Ostaje sakrivena dok se ne desi neka situacija
- Programska greška (“software failure”)
  - Manifestovana programska mana

Testiranje programa

- Zadatak testiranja
  - Otkrivanje što više programskih mana izvršavanjem i proverom što većeg dela programa da bi se mane manifestovale kao greške
- Cilj testiranja
  - Stvaranje određenog nivoa sigurnosti da program radi ono što je opisano zahtevima

Klasifikacija testova

- Prema vrsti zahteva
- Testovi za proveru funkcionalnih karakteristika
  - unit (pojedinačni) - testiranje jedne klase
  - integration - testiranje grupe povezanih klasa
  - system - testiranje celog programa
  - acceptance - testiranje celog programa od strane krajnjeg korisnika
- Testovi za proveru nefunkcionalnih karakteristika
  - brzina rada, bezbednost, opterećenje, preopterećenje...

Klasifikacija testova

- Prema kriterijumu dostupnosti koda
- Princip “crne kutije” (“black-box”)
  - Kod se tretira kao crna kutija
  - Porede se ulazi sa očekivanim izlazima, a zanemaruje se interno funkcionisanje koda
  - Testiraju se samo javno dostupne klase, metode i atributi
  - Prednost - kada se promeni interni kod neke metode, svi testovi i dalje funkcionišu
  - Mana - zbog “nepoznavanja” unutrašnje strukture koda mogu se propustiti neki testovi

Klasifikacija testova

- Princip “bele kutije” (“white-box”, “glass-box”)
  - Kod se tretira kao da je javno dostupan - postoji uvid u unutrašnje funkcionisanje
  - Testiraju se svi delovi koda: javni, privatni, zaštićeni i sa podrazumevanim pristupom
  - Pokrivenost testovima - obezbediti da svaka linija koda bude proverena nekim testom
  - Prednost - pokrivenost testovima obezbeđuje određenu sigurnost
  - Mana - veliki broj testova je neupotrebljiv čim se kod promeni

Klasifikacija testova

- Princip “sive kutije” (“grey-box”)
  - Kombinacija prethodna dva pristupa
  - Testiraju se samo javno dostupne klase, metode i atributi
  - Testovi se pišu sa uvidom u unutrašnje funkcionisanje koda
  - Zadržava prednosti i otklanja mane prethodnih pristupa

Klasifikacija testova

- Regresivno testiranje (“regression testing”)
  - Izvršava se kada se kod promeni
  - Ima za cilj da utvrdi da promene u kodu nisu narušile funkcionalnost već napisanih delova koda
  - Sastoji se od ponovnog izvršavanja svih “starih” testova - onih koji su napisani pre promene koda a još uvek su aktuelni

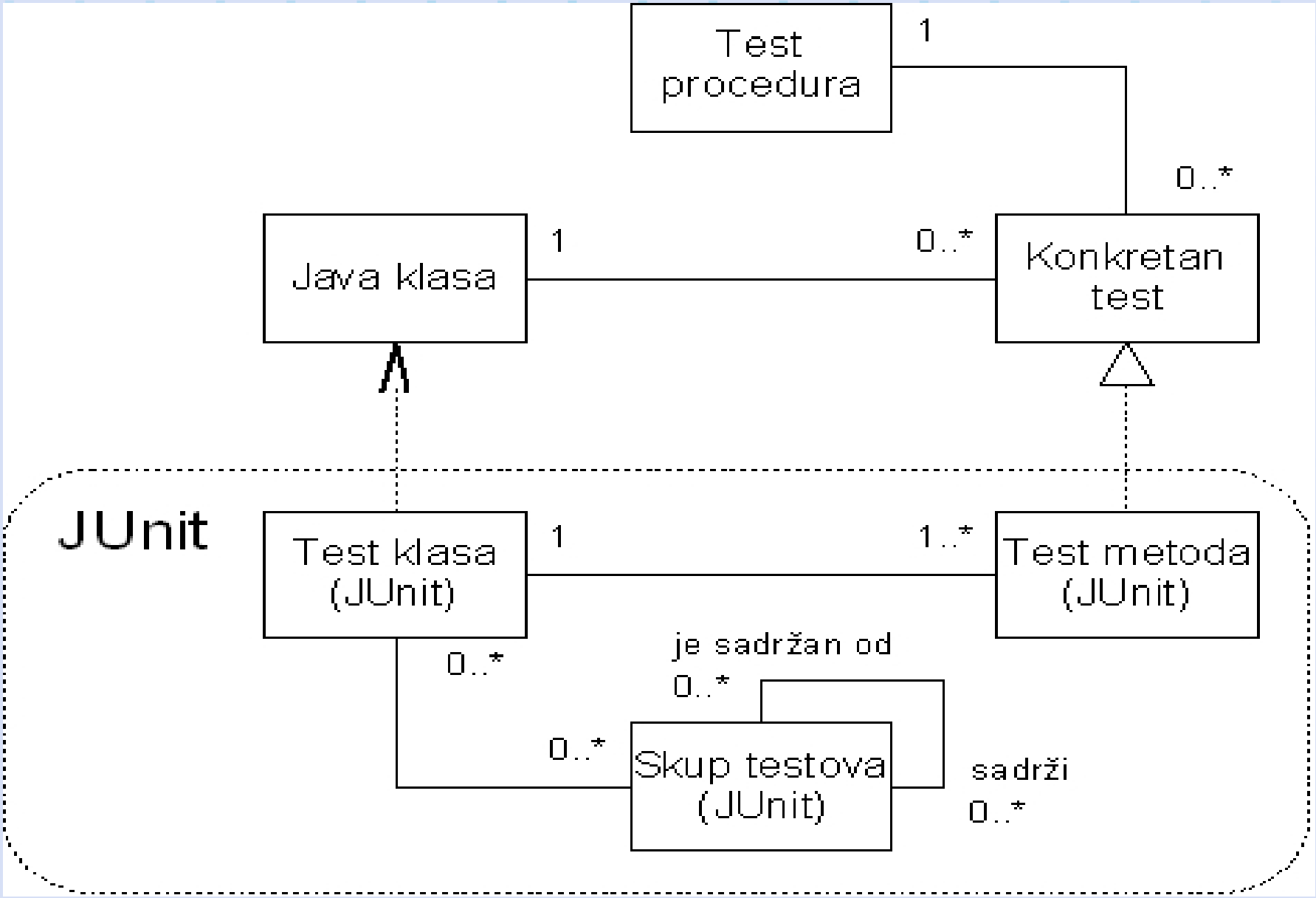
JUnit

- Besplatan alat za pojedinačno (“unit”) i integrativno (“integration”) testiranje Java koda
- Autori: Kent Beck i Erich Gamma
- Sajt: www.junit.org (aktuelna verzija JUnit 4.x)
- “de-facto” standard među alatima za testiranje
- Verzije za druge programske jezike:
  - C# (NUnit), Python (PyUnit), Fortran (fUnit), C++ (CPPUnit)
- Alternativa: TestNG

JUnit - karakteristike

- Jednostavan za korišćenje, testovi se pišu lako i uz minimum napora (testovi su obične Java metode)
- Baza testova se čuva, lako pokreće i po potrebi nadopunjuje (omogućava regresivno testiranje)
- Automatska provera rezultata testiranja i prijava grešaka
- Hijerarhijsko organizovanje testova prema klasama, paketima isl.
- JUnit je već integrisan u Eclipse i NetBeans

JUnit - arhitektura



JUnit - arhitektura

- Za svaku klasu iz programa se pravi po jedna **test klasa (“TestCase”)**
  - npr. KonvertorValute - KonvertorValuteTest
- Svaki konkretan test se implementira preko jedne **test metode**
  - npr. za metodu izracunajDin - testIzracunajDin
- Svaka test klasa može da sadrži više test metoda
- Test klase se mogu organizovati u **skupove testova (“TestSuite”)**

JUnit - sintaksa

- Primer klase koju bi trebalo testirati

```
public class KonvertorTemperature {

public static double konvertujCUF (double tc){
    if (tc < -273.15)
      throw new RuntimeException("Temperatura"
       +" ne moze biti ispod apsolutne nule");

    return tc*9/5+32;
  }
}
```

JUnit - sintaksa

- Primer odgovarajuće test klase:

```
import static org.junit.Assert.*;
import org.junit.Test;

public class KonvertorTemperatureTest {

//TEST METODE

}
```

JUnit - sintaksa

- Konvencija nalaže sledeće:
  - Naziv test klase bi trebalo da bude “naziv klase koja se testira” + “Test”
  - U test klasi bi trebalo da se nalaze samo oni testovi koji se odnose na jednu konkretnu klasu
  - Test klasa bi trebalo da bude na istom mestu u hijerarhiji paketa kao i osnovna klasa - formiranje paralelne hijerarhije osnovnih i test klasa

JUnit - sintaksa

- Primer test metode

```
@Test
public void testKonvertujCUFSveOk() {

 double expectedValue = 41.0;
 double value =
       KonvertorTemperature.konvertujCUF(5);

 assertEquals(expectedValue,value,0.0001);
}
```

JUnit - sintaksa

- Prethodna test metoda proverava situaciju kada se konverzija odvija bez problema
- Naziv test metode po konvenciji počinje sa reči “test” (ali nije obavezno)
- Anotacija **@Test** pre svake test metode
- JUnit koristi dve osnovne metode za proveru očekivanog i stvarnog rezultata
  - assertEquals
  - assertTrue
- One obaveštavaju da li je test prošao (“passed”) ili pao (“failed”)

JUnit - sintaksa

- assertEquals (Object expected, Object result)
  - Provera jednakosti očekivanog i stvarnog rezultata
  - Prvi ulazni parametar je očekivani rezultat a drugi stvarni rezultat
  - Postoji i treći parametar ako se porede dva realna broja - maksimalno odstupanje
  - Test je prošao ako su ove dve vrednosti jednake a, u suprotnom je test pao
  - Prima dva objekta bilo koje klase i poredi ih pozivanjem njihove “equals” metode
  - Mogu joj se kao ulazni parametri proslediti i prosti tipovi podataka (automatsko obmotavanje)

JUnit - sintaksa

- assertTrue (boolean condition)
  - Metoda koja služi za proveru nekog logičkog uslova
  - Može se koristiti umesto assertEquals kada je to zgodnije
  - Test je prošao kada se izraz u zagradi evaluira na “true” a pao je u suprotnom

JUnit - sintaksa

- Metode assertEquals i assertTrue imaju i svoje preklopljene podvarijante, a postoje i druge “assert” metode:
  - assertFalse
  - assertArrayEquals (poredi dva niza)
  - assertNotSame
  - assertNotNull
  - assertNull
  - assertSame
  - assertThat

JUnit - sintaksa

- Primer test metode

```
@Test(expected = java.lang.RuntimeException.class)
public void testKonvertujCUFIspodNule() {

    KonvertorTemperature.konvertujCUF(-273.16);

}
```

JUnit - sintaksa

- Prethodna test metoda testira situaciju kada bi testirana metoda trebalo da baci izuzetak
- Testiranje se vrši tako što se izaziva odgovarajuća situacija i hvata se izuzetak
- Parametar **expected** u okviru anotacije **@Test**
- Provera se sastoji u tome da se uporedi tip bačenog izuzetka sa očekivanom klasom izuzetka

JUnit - sintaksa

- Dve metode za (opcionu) pripremu test situacije
  - setUp (anotacija @Before ispred metode)
  - tearDown (anotacija @After ispred metode)
- Koriste se da naprave pripremu za izvršavanje testa tj. da “počiste” sve efekte posle njegovog izvršavanja
- Ako se napišu, **“setUp” se izvršava pre izvršenja svake test metode, a “tearDown” posle izvršenja svake test metode**

JUnit - sintaksa

- Ove metode se koriste da npr. naprave nove instance svih potrebnih objekata pre testa da bi se izbegli “bočni efekti”
- Još dve metode koje se izvršavaju pre početka prvog testa i posle završetka poslednjeg testa:
  - setUpBeforeClass (@BeforeClass)
  - tearDownAfterClass (@AfterClass)

JUnit - preporuke za pisanje testova

- Testove pisati pre (“test-first”) ili uporedo sa implementacijom koda a nikako posle
- Napraviti paralelnu hijerarhiju test klasa u odnosu na klase koje se testiraju
- Pokrivenost testovima bi trebalo da bude kompletna - testirati sve alternative (scenarije) u izvršavanju
- Testovi bi trebalo da budu međusobno nezavisni (bez “bočnih efekata”)

JUnit - preporuke za pisanje testova

- Posle svake izmene koda izvršiti regresivno testiranje
- Svaki put kada se u toku izvršavanje gotovog programa pojavi greška (“bug”) **NE ISPRAVLJATI ODMAH GREŠKU U KODU**
  - Identifikovati (brojem) i opisati grešku
  - Napraviti test koji simulira grešku
  - Tek onda ispraviti grešku
  - Na ovaj način se formira baza grešaka - “bug database” i baza testova za poznate greške
  - Testovi za poznate greške onemogućavaju njihovo ponovno pojavljivanje

Postupak za funkcionalno testiranje klasa (opšti)

- Problemi:
  - Odakle bi trebalo početi sa testiranjem?
  - Šta bi sve trebalo da se proveri testovima?
  - Koliko testova bi trebalo napisati?
  - Kojim redosledom bi trebalo pisati testove?
  - Koje klase (ili metode) se moraju testirati, a koje ne?
- Problem je **nesigurnost** u vezi sa tim šta sve zapravo treba testirati

Postupak za funkcionalno testiranje klasa (opšti)

- Testirati funkcionalna ograničenja nad svim (Vlajić S., Tomić B.):
  - ***Atributima (vrednosna ograničenja)*** – utiču na moguće vrednosti atributa sistema.
  - ***Asocijacijama (strukturna ograničenja)*** – odnose se na asocijacije (veze) između elemenata sistema, njihov referencijalni integritet i kardinalnosti.
  - ***Metodama (ograničenja ponašanja)*** – odnose se na metode sistema i utiču na njegovo ponašanje.

Postupak za funkcionalno testiranje klasa (opšti)

- Ograničenja nad atributima

Ograničenja tipa atributa – definišu tip vrednosti koje atribut neke klase može da ima.

Test: pokušati sa unosom neke vrednosti koja nije odgovarajućeg tipa (za taj atribut) i očekivati da sistem javi grešku. Ako se greška ne pojavi, test je pao.

Test: pokušati sa unosom neke vrednosti koja JESTE odgovarajućeg tipa (za taj atribut) i očekivati da sistem zapamti novu vrednost. Ako se pojavi greška ili se ne zapamti nova vrednost, test je pao.

Ograničenja dozvoljenog skupa vrednosti atributa – raspon vrednosti za neki konkretan atribut.

Test: pokušati sa unosom neke vrednosti koja nije dozvoljena i očekivati da sistem javi grešku. Ako se greška ne pojavi, test je pao.

Test: pokušati sa unosom neke vrednosti koja JESTE dozvoljena i očekivati da sistem zapamti novu vrednost. Ako se pojavi greška ili se ne zapamti nova vrednost, test je pao.

Postupak za funkcionalno testiranje klasa (opšti)

- Ograničenja nad atributima (nastavak):

Ograničenja na međuzavisnosti vrednosti atributa jedne klase – dve vrste:

Ograničenja na vrednosti istog atributa u odnosu na druga pojavljivanja iste klase – ovo se prvenstveno odnosi na identifikatore tj. primarne ključeve ("PRIMARY KEY") i jedinstvene ("UNIQUE") vrednosti.

Test: pokušati sa unosom dve instance sa istim ključem (identifikatorom) i očekivati da sistem javi grešku. Ako se greška ne pojavi, test je pao.

Test: pokušati sa unosom dve instance sa različitim ključem i očekivati da sistem zapamti obe. Ako se pojavi greška ili se ne zapamte obe, test je pao.

Ograničenja na vrednosti različitih atributa u okviru istog pojavljivanja klase, npr. atributi "prodajni_kurs", "srednji_kurs" i "kupovni_kurs" klase "Valuta" (srednji kurs je uvek između ova dva).

Test: pokušati sa unosom vrednosti više atributa koji narušavaju ovo ograničenje i očekivati grešku. Ako se greška ne pojavi, test je pao.

Postupak za funkcionalno testiranje klasa (opšti)

- Ograničenja nad atributima (nastavak):

Ograničenja na međuzavisnosti vrednosti atributa više klasa – Na primer, polje “ukupan_iznos” klase “Račun” je suma polja “iznos_stavke” klase “Stavka_računa”.

Test: pokušati sa unosom vrednosti atributa koji narušavaju ovo ograničenje i očekivati grešku. Ako se greška ne pojavi, test je pao.

Test: pokušati sa unosom vrednosti atributa koji NE NARUŠAVAJU ograničenje i očekivati da sistem zapamti sve nove vrednosti. Ako se pojavi greška ili se ne zapamte nove vrednosti, test je pao.

Postupak za funkcionalno testiranje klasa (opšti)

- Ograničenja nad asocijacijama

Ograničenja kardinalnosti veza – svaka asocijacija između dve klase ima svoju donju i gornju kardinalnost.

Test: pokušati sa kreiranjem instanci i stvaranjem asocijacija koje narušavaju minimalnu i maksimalnu kardinalnost i očekivati grešku. Ako se greška ne pojavi, test je pao.

Test: pokušati sa kreiranjem instanci i stvaranjem asocijacija koje NE NARUŠAVAJU ograničenja kardinalnosti i očekivati da sistem zapamti sve. Ako se pojavi greška ili se ne zapamte nove vrednosti, test je pao.

Ograničenja referencijalnog integriteta – ova ograničenja se odnose na situaciju kada se jedno od dva pojavljivanja klase koja su u vezi obriše ili izmeni. U kontekstu relacionih baza podataka, ovo su “ON UPDATE” i “ON DELETE” ograničenja.

Test: pokušati sa brisanjem ili ažuriranjem instanci i očekivati da sistem obriše ili ažurira sve referentne instance. Ako sistem ne ažurira referentne instance ili se javi greška, test je pao.

Postupak za funkcionalno testiranje klasa (opšti)

- Ograničenja nad metodama

Ograničenja tipa vrednosti parametara metoda – ova ograničenja se odnose na tipove parametara metoda.

Test: pokušati sa unosom vrednosti parametra koja nije odgovarajućeg tipa i očekivati da sistem javi grešku. Ako se greška ne pojavi, test je pao.

Test: pokušati sa unosom vrednosti parametra koja JESTE odgovarajućeg tipa i očekivati da sistem izvrši metodu. Ako se pojavi greška, test je pao.

Ograničenja skupa vrednosti parametara metoda – ograničenje skupa vrednosti koje se mogu uneti kao parametri metoda. Na primer, nije dozvoljeno da neki parametar ima “null” vrednost.

Test: pokušati sa unosom vrednosti parametra koja nije dozvoljena i očekivati da sistem javi grešku. Ako se greška ne pojavi, test je pao.

Test: pokušati sa unosom vrednosti parametra koja JESTE dozvoljena i očekivati da sistem izvrši metodu. Ako se pojavi greška, test je pao.

Postupak za funkcionalno testiranje klasa (opšti)

- Ograničenja nad metodama (nastavak):

Ograničenja ponašanja (pod uslovom da su prva dva tipa ograničenja zadovoljena) – ova ograničenja se odnose na ponašanje metode u situaciji da su uneti parametri u okviru prethodnih ograničenja. Ovde se testiraju **svi osnovni i alternativni scenariji** jedne metode. Primer osnovnog scenarija - unos novog filma u bazu podataka. Primer alternativnog scenarija - pokušaj unosa već postojećeg filma u bazu podataka i javljanje greške.

Test (za svaki osnovni scenario): pokušati sa unosom vrednosti parametra koji su dozvoljeni i očekivati da se metoda izvrši kako je definisano. Ako se pojavi greška, ili ako metoda nije izvršila sve promene koje je bilo potrebno, test je pao.

Test: (za svaki alternativni scenario): pokušati sa unosom vrednosti parametra koji su dozvoljeni, ali podesiti stanje tako da se izvršava alternativni scenario i očekivati da se metoda izvrši kako je definisano alternativnim scenariom. Ako se ne izvrši alternativni scenario (to je najčešće da se javi greška), test je pao.