PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET
METROPOLITAN
BEOGRAD

KI301 - KONSTRUISANJE SOFTVERA

Praksa testiranja

Lekcija 08

PRIRUČNIK ZA STUDENTE

KI301 - KONSTRUISANJE SOFTVERA

Lekcija 08

PRAKSA TESTIRANJA

Uvod

L8:PRAKSA TESTIRANJA

Sadržaj:

- Alati za podršku testiranja
- Alati za debugging
- Vežba : Zadaci i pitanja iz testiranja softvera
- Vežba - JUNIT alat
- Vežba - Konstrukciono testiranje

Poglavlje 1

Alati za podršku testiranja

ALATI ZA TESTIRANJE

Evo liste alata za testiranje.

Alati za testiranje pomažu kod testiranja softvera:

- Automatski test-okviri, npr. Junit, Nunit, CppUnit, itd.
- Alati za poredjenje različitih verzija programa tj. Diff Tools koji se koriste kod regresionog testiranja i porede izlaz iz programa sa očekivanim izlazom iz programa
- Test scaffolding ("skele za testiranje")
- Defect-tracking softver
- Error databases tj. baza podataka o greškama je vrlo koristan alat koji omogućuje analizu grešaka
- Test-data generatori , koji generišu npr. proizvoljno test-podatke

Regresivno testiranje (regression testing) tj. re-testiranje: Ako npr. imamo program tj. softver koji iztestiran, i eliminisane greške, i onda izvršimo izmenu tog programa u jednom manjem delu, onda se vrši tzv. "regresivno testiranje" gde testiramo samo da dokažemo da izmene programa nisu dovele do novih greški.

Automated tests:

- Kod regresivnog testiranja korisno je izvršiti automatizaciju testiranja
- Automatizovani testovi se mogu koristiti tokom razvoja projekta u cilju efikasnijeg testiranja, ako očekujete da se neki testovi ponavljaju
- alati koji omogućuju automatsko testiranje omogućuju i "skele", generišu ulaz, analiziraju izlaz (porede dobijeni izlaz sa očekivanim izlazom). i alati kao što su JUnit, CppUnit, NUnit, itd, omogućuju ili sve ili neke od ovih opcija.

„SOFTVERSKE SKELE“- IZGRADNJA POMOĆU TEST-OKVIRA

Postoji mnogo test-okvira (test-frameworks) koji omogućuju da se obezbede "skele" za programe, npr. JUnit za Javu, CppUnit za C++, NUnit za .NET.

Izgradnja "skela" zahteva vreme, Ali, brojni alati postoje koji olakšavaju kreiranje "skela". Ako koristite "skele" kod testiranja klasa, onda možete svaku klasu da istetsirate bez interakcije

sa ostalim klasama, što je predmet integracionog testiranja. Međutim, i kod integracionog testiranja se koriste "skele", ali tada je nekoliko softverskih jedinica npr. nekoliko klasa povezano u jednu veću celinu, gde isto treba onda koristiti "skele". "Skele" su vrlo korisne jer omogućuju da se željeni deo programa istetstira bez uticaja greški iz drugih delova programa.

Postoji mnogo test-okvira (test-frameworks) koji omogućuju da se obezbede "skele" za programe, npr.

- JUnit za Javu
- CppUnit za C++
- NUnit za .NET

Ako ne koristite ove test-okvire tj. test-alate, onda se mogu napisati nekoliko podprograma u nekoj klasi i takođe main() scaffolding-routine, da bi se istetsirala neka klasa. Npr., main() podprogram može da učitava argumente iz komandne linije (konzole), i da ih predaje podprogramima (klasnim metodama) koje se testiraju, tako da možete da testirate podprogram samostalno pre nego ga integrišete sa ostatkom programa. Kad integrišete pojedine delove programa, možete ostaviti "skele" u programu i deaktivirati ih pomoću comment-instrukcija.

VIDEO VEŽBA

Video vežba - Vrste alata za testiranje (Types of Software Testing Tools)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 2

Vežba - Upotreba JUNIT alata

JEDINIČNO TESTIRANJE (UNIT-TESTING) POMOĆU JUNIT

Najpopularniji test-okviri za Javu su JUnit i TestNG. Oni se mogu koristiti i za jedinično i za integraciono testiranje.

Pogledati link:: http://www.vogella.com/tutorials/JUnit/article.html

gde je detaljno objašnjeno i dat je primer kako se koristi alat JUnit. Može da se odnosi na JUnit 4.x i JUnit 5. Inače, ima nekoliko test-okvira, testing-frameworks, koja se mogu koristiti za Javu. Najpopularniji su: JUnit and TestNG. Ovi alati se mogu koristiti:

- za jedinične testove
- i za integracione testove.

JUnit-POKAZNA VEŽBA:

The JUnit framework: JUnit je test-okvir koji koristi anotacije da identifikuje metode koje se testiraju. JUnit je open source projekt koji se može naći na Github. Kako se definiše test u alatuJUnit **?:** JUnit-test je ustvari metoda obuhvaćena u nekoj klasi-klasi koja se koristi samo za testiranje. Ova klasa se zove:Test class. Da bismo definisali da određena metoda je test-metoda, potrebno je da se ona anotira sa @Test anotacijom. Izvršavanjem ove metode izvršava se željeni test.

Pri tome, može se koristiti assert-method (asertivna metoda) , koja postoji u JUnit-u ili nekom drugom assert-framework-u, da bi se proverio jedan očekivani rezultat nasuprot stvarnom rezultatu. Upotreba asertivne metode tj. njeno pozivanje se zove: assert ili assert statements. Poruke u assert-statements imaju određeno značenje, koje im mi zadajemo, i na ovaj način olakšavamo da identifikujemo i ispravimo problem.. Ovo je posebno korisno ako neko testira program a da ga on nije pisao već ga samo testira.

JEDINIČNO TESTIRANJE (UNIT-TESTING) SA ALATOM JUNIT - PRIMER

Dole je prikazana metoda koju testiramo.

Primer JUnit-testa :

Sledeći programski kod prikazuje jedan JUnit-test, koristeći verziju JUnit 5. Ovde se pretpostavlja da klasa MyClass postoji, i da ima metodu: multiply(int, int)

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class MyTests {

@Test
public void multiplicationOfZeroIntegersShouldReturnZero() {
MyClass tester = new MyClass(); // MyClass is tested

// assert statements
assertEquals("10 x 0 must be 0", 0, tester.multiply(10, 0));
assertEquals("0 x 10 must be 0", 0, tester.multiply(0, 10));
assertEquals("0 x 0 must be 0", 0, tester.multiply(0, 0));
}
}
```

JUNIT KONVENCIJE

Ima čitav niz konvencija za obeležavanje kod upotrebe JUnit alata.

JUnit konvencije **obeležavanja za JUnit-testove:**

- A widely-used solution for classes is to use the "Test" suffix at the end of test classes names.
- As a general rule, a test name should explain what the test does. If that is done correctly, reading the actual implementation can be avoided.
- One possible convention is to use the "should" in the test method name. For example, "ordersShouldBeCreated" or "menuShouldGetActive". This gives a hint what should happen if the test method is executed.
- Another approach is to use "Given[ExplainYourInput]When[WhatIsDone]Then[ExpectedResult]" for the display name of the test method.
- Run your test from the command line.You can also run your JUnit tests outside our IDE via standard Java code.
- Build systems like Apache Maven or Gradle in combination with a Continuous Integration Server (like Jenkins) can be used to execute tests automatically on a regular basis.

MYCLASSTEST

Ovde je ilustrovano izvršavanje testa: MyClassTest.

Sledeća klasa ilustruje kako da se izvrši MyClassTest. Ova klasa izvršava vašu test-klasu i na konzoli piše moguće defekte. Ova klasa se može izvršiti preko komandne linije (konzole)

kao bilo koji Java program, ali treba da dodate JUnit library tj. JAR file na klasnu putanju (classpath).

Napomene:

- JUnit assumes that all test methods can be executed in an arbitrary order.
- Well-written test code should not assume any order, i.e., tests should not depend on other tests.

```
package de.vogella.junit.first; import org.junit.runner.JUnitCore; import
org.junit.runner.Result; import org.junit.runner.notification.Failure;

public class MyTestRunner {
public static void main(String[] args) {
Result result = JUnitCore.runClasses(MyClassTest.class);
for (Failure failure : result.getFailures()) {
System.out.println(failure.toString());
} }}
```

ASSERT-KLASA

JUnit pruža Assert-klasu koja se koristi da se specificiraju error-messages tj. poruke o greškama..

Assert instrukcije: JUnit ima niz klasnih metoda u Assert klasi koje olakšavaju testiranje. Imena ovih metoda počinju sa rečjuassert. One omogućuju da specificirate tzv. error-message, kao i očekivani i stvarni rezultat testiranja Ove assert-metode porede stvarnu izlaznu vrednost (vraćenu tokom testiranja parčete softvera) sa očekivanom vrednošću. Ako poređenje ne uspe tj. ako je izlazna vrednost pogrešna, onda ova metoda izbacuje tzv. AssertionException.

Dole su navedene assert-metode u klasi Assert. Parametri u zagradama su opcioni, i oni su tipa String:

Tabela: Metode koje proveravaju test-rezultate
Statement- Description
fail([message])- Let the method fail. Might be used to check that a certain part of the code is not reached or to have a failing test before the test code is implemented. The message parameter is optional.
assertTrue([message,] boolean condition)-Checks that the boolean condition is true.
assertFalse([message,] boolean condition)-Checks that the boolean condition is false.
assertEquals([message,] expected, actual)-Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays

etc..

POKAZNA VEŽBA - PROGRAM ZA REZERVACIJU KARATA

Testiranje programa za rezervaciju karata upotrebom JUnit testova.

Dat je primer programa preko koga se vrši rezervacija karata. U nastavku se nalazi programski kod za četiri klase ovog programa (klasu Korisnik, klasu Rezervacija i klasu Main).

Programski kod posebne klase Korisnik:

```
package Main;

/**
 *
 * @author Nebojsa
 */
public class Korisnik {
    private String ime;
    private String prezime;
    private String grad;
    private String drzava;
    private String adresa;
    private int brojKartice;
    private int kontaktTelefon;

    public Korisnik(String ime, String prezime, String grad, String drzava, String
adresa, int brojKartice, int kontaktTelefon) {
        this.ime = ime;
        this.prezime = prezime;
        this.grad = grad;
        this.drzava = drzava;
        this.adresa = adresa;
        this.brojKartice = brojKartice;
        this.kontaktTelefon = kontaktTelefon;
    }

    public String getIme() {
        return ime;
    }

    public String getPrezime() {
        return prezime;
    }

    public String getGrad() {
        return grad;
    }

    public String getDrzava() {
        return drzava;
    }
```

```
    public String getAdresa() {
        return adresa;
    }

    public int getBrojKartice() {
        return brojKartice;
    }

    public int getKontaktTelefon() {
        return kontaktTelefon;
    }

    public void setIme(String ime) {
        this.ime = ime;
    }

    public void setPrezime(String prezime) {
        this.prezime = prezime;
    }

    public void setGrad(String grad) {
        this.grad = grad;
    }

    public void setDrzava(String drzava) {
        this.drzava = drzava;
    }

    public void setAdresa(String adresa) {
        this.adresa = adresa;
    }

    public void setBrojKartice(int brojKartice) {
        this.brojKartice = brojKartice;
    }

    public void setKontaktTelefon(int kontaktTelefon) {
        this.kontaktTelefon = kontaktTelefon;
    }

    @Override
    public String toString() {
        return "Korisnik{" + "ime=" + ime + ", prezime=" + prezime + ", grad=" +
grad + ", drzava=" + drzava + ", adresa=" + adresa + ", brojKartice=" + brojKartice
+ ", kontaktTelefon=" + kontaktTelefon + '}';
    }


}
```

Programski kod klase Rezervacija:

```
package Main;

/**
 *
 * @author Nebojsa
 */
public class Rezervacija {

    private int brojKarata;
    private String datumRezervacije ;
    private String datumIsteka;
    private String potvrdaRezervacije;

    public Rezervacija(int brojKarata, String datumRezervacije, String datumIsteka,
String potvrdaRezervacije) {
        this.brojKarata = brojKarata;
        this.datumRezervacije = datumRezervacije;
        this.datumIsteka = datumIsteka;
        this.potvrdaRezervacije = potvrdaRezervacije;
    }

    public int getBrojKarata() {
        return brojKarata;
    }

    public String getDatumRezervacije() {
        return datumRezervacije;
    }

    public String getDatumIsteka() {
        return datumIsteka;
    }

    public String getPotvrdaRezervacije() {
        return potvrdaRezervacije;
    }

    public void setBrojKarata(int brojKarata) {
        this.brojKarata = brojKarata;
    }

    public void setDatumRezervacije(String datumRezervacije) {
        this.datumRezervacije = datumRezervacije;
    }

    public void setDatumIsteka(String datumIsteka) {
        this.datumIsteka = datumIsteka;
    }

    public void setPotvrdaRezervacije(String potvrdaRezervacije) {
        this.potvrdaRezervacije = potvrdaRezervacije;
    }
```

```
    @Override
    public String toString() {
        return "Rezervacija{" + "brojKarata=" + brojKarata + ", datumRezervacije="
+ datumRezervacije + ", datumIsteka=" + datumIsteka + ", potvrdaRezervacije=" +
potvrdaRezervacije + '}';
    }


}
```

Programski kod klase SingleObject:

```
package Main;

/**
 *
 * @author Nebojsa
 */
public class SingleObject {
    private static SingleObject instance;

    private SingleObject() {
        System.out.println("Singleton(): Initializing Instance");
    }

    public static SingleObject getInstance() {
        if (instance == null) {
            synchronized (SingleObject.class) {
                if (instance == null) {
                    instance = new SingleObject();
                }
            }
        }

        return instance;
    }

    public int getKorisnikBrojKartice(int brojKartice) {
        return brojKartice;
    }

}
```

Programski kod Main klase koja se testira:

```
/**
 *
 * @author Nebojsa
 */
public class Main {

    /**
```

```
     * @param args the command line arguments
     */
    public static void main(String[] args) {
       SingleObject object = SingleObject.getInstance();
        int brojKartice = object.getKorisnikBrojKartice(12341234);

        Korisnik korisnik = new Korisnik("Nebojsa", "Gavrilovic",
"Kosjeric","Srbija","Balkanska", brojKartice , 064123456);
        System.out.println(korisnik);

        Rezervacija rezervacija = new Rezervacija(4, "22.11.2017","20.11.2017",
"true");
        System.out.println(rezervacija);
    }

}
```

Na datom programu za rezervisanje karata za primenjen je JUnit test. U okviru testa potrebno je utvrditi da se broj kartice ne razlikuje od očekivanog broja koji je potrebno uneti kroz program.

POKAZNA VEŽBA - PROGRAM ZA REZERVACIJU KARATA - DOBIJENI REZULTATI

Nakon izvršenih JUnit testova potrebno je uporediti dobijene rezultate sa očekivanim rezultatima testiranja.

JUnit test koji je primenjen na Main klasi:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Main;

import Main.SingleObject;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
    * @author Nebojsa
 */
public class SingleObjectTest {
```

```
    public SingleObjectTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }


    /**
     * Test of doJMBG method, of class SingleObject.
     */
    /*@Test
    public void testDoJMBG() {
        System.out.println("getBrojKartice");
        int brojKartice = 37465825;
        SingleObject object = SingleObject.getInstance();
        int expResult = 37465825;
        int result = object.getKorisnikBrojKartice(brojKartice);
        assertEquals(expResult, result);
        System.out.println("Broj kartice je ispravno unet.");


    }*/

    @Test
    public void testGetBrojKartice() {
        System.out.println("getBrojKartice");
        int brojKartice = 37465825;
        SingleObject object = SingleObject.getInstance();
        int expResult = 374658225;
        int result = object.getKorisnikBrojKartice(brojKartice);
        assertNotEquals(expResult, result);
        System.out.println("Broj kartice nije ispravno unet.");

    }

}
```

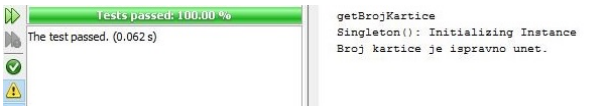
Prvi test:

```
@Test
public void testGetBrojKartice() {
    System.out.println("getBrojKartice");
    int brojKartice = 37465825;
    SingleObject object = SingleObject.getInstance();
    int expResult = 37465825;
    int result = object.getKorisnikBrojKartice(brojKartice);
    assertEquals(expResult, result);
    System.out.println("Broj kartice je ispravno unet.");

}
```

Slika 2.1 Prvi test slučaj

Prvi test slučaj treba da pokaže da se uneti broj kartice ne razlikuje od očekivanog broja koji je definisan u programu. Na slici 1 prikazan je programski kod ovog test slučaja a na slici 2 dobijeni rezultat koji prikazuje da se brojevi podudaraju.



Slika 2.2 Dobijeni rezultat prvog test slučaja

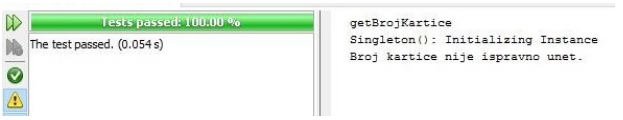
Drugi test:

```
@Test
public void testGetBrojKartice() {
    System.out.println("getBrojKartice");
    int brojKartice = 37465825;
    SingleObject object = SingleObject.getInstance();
    int expResult = 374658225;
    int result = object.getKorisnikBrojKartice(brojKartice);
    assertNotEquals(expResult, result);
    System.out.println("Broj kartice nije ispravno unet.");

}
```

Slika 2.3 Drugi test slučaj

Drugi test slučaj treba da pokaže da se uneti broj kartice ne slaže sa očekivanim brojem kartice koji je definisan u programu. Na slici 3 je prikazan programski kod drugog test slučaja a na slici 4 se nalazi dobijeni rezultat ovog testa. Brojevi se ne podudaraju i test slučaj je uspešan.



Slika 2.4 Dobijeni rezultat drugog test slučaja

ZADACI ZA INDIVIDUALAN RAD

Zadaci sa upotrebom alata JUnit.

1. Za izabranu aplikaciju, uraditi jedinično testiranje upotrebom alataJUnit, i pri tome primeniti metodu strukturnog testiranja, tj. white-box metodu ?

2.. Za izabranu aplikaciju, uraditi jedinično testiranje upotrebom alata JUnit, i pri tome primeniti data-flow metodu testiranja, dakle white-box metodu ?

3. . Za izabranu aplikaciju, uraditi jedinično testiranje upotrebom alata JUnit, i pri tome primeniti metodu funkcionalnog testiranja , tj. black-box metodu?

Primer JUnit testiranja u razvojnom okruženju NetBeans:

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Primer JUnit testiranja u razvojnom okruženju Eclipse:

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 3

Vežba - Java alati za testiranje

JAVA ALATI

Uprkos rastućoj populatrnosti IntelliJ IDEA, NetBeans, i drugih IDE, ankete ukazuju da Eclipse je još uvek najpopularniji IDE , za oko 50% Java softverista.

Pogledati pokaznu vežbu na linku:

8 tools for every Java developer's toolkit:

- https://www.loggly.com/blog/8-tools-for-every-java-developers-toolkit/

Ovih 8 alata pokrivaju Java-development, od code-building do bug-squashing. Učenje ovih Java alata može vam pomoći da poboljšate kvalitet programa i postanete vrlo efikasan Java softverista.

Na linku

- http://www.vogella.com/tutorials/JUnit/article.html

može se naći detaljno uputstvo za upotrebu:

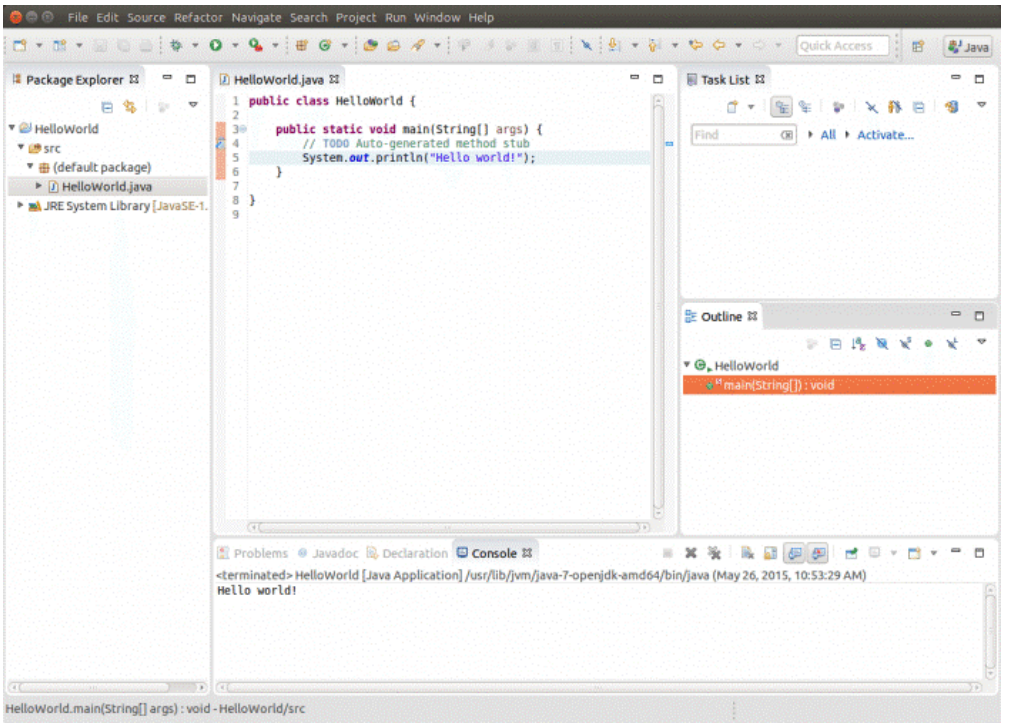
JUnit alata: Unit Testing with JUnit - Tutorial.

Eclipse podržava pisanje JUnit testova pomoću Eclipse-wizards.

ECLIPSE

Eclipse je moćan IDE, koji ima moćan interface i veoma puno plugins.

Uprkos rastućoj popularnosti IntelliJ IDEA, NetBeans, i drugih IDE-a, Eclipseje još uvek najpopularniji IDE za oko 50% Java softverista.Eclipse je moćan IDE, koji ima moćan interface i veoma puno plugins.



ALAT JUNIT

JUnit je open-source framework za pisanje i izvršavanje unit-tests.

JUnit: JUnit je open-source framework za pisanje i izvršavanje unit-tests. Neki JUnit-test sastoji se od testing-class, i testing-method, i fuunkcionalnosti koja se testira. JUnit koristi anotacije (annotations) da specificira kako su testovi strukturisani i kako se izvršuju. Npr., ako vaš program ima klasu nazvanu MathClass sa klasnim metodama za množenje i deljenje brojeva, možete da kreirate JUnit testove da proverite da li će doći do neočekivanih vrednosti . Ako zadate brojeve 2 i 5 kao ulazne parametre u metodi množenja, vi očekujete da primite 10 kao rezultat.. Ako predate broj 0 kao drugi parametar u metodi deljenja, vi očekujete ArithmeticException zbog deljenja sa nulom:

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class MathTest {
@Test
public void multiplicationTest() {
MathClass myClass = new MathClass();
assertEquals(10, myClass.multiply(2, 5));
}
@Test(expected=ArithmeticException.class)
public void divisionTest() {
MathClass myClass = new MathClass();
myClass.divide(1, 0);
}
}
```

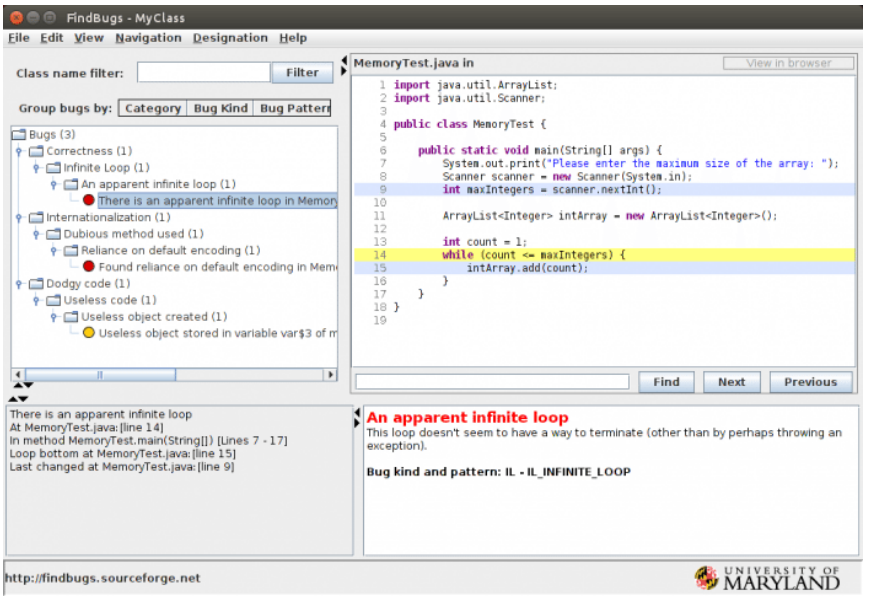
Na linku http://www.vogella.com/tutorials/JUnit/article.html može se naći detaljno uputstvo za upotrebu JUnit alata: Unit Testing with JUnit - Tutorial

ALAT FINDBUGS

Alat FindBugs je alat koji upoređuje program sa bazom podataka sa greškama (database of bugs).

FindBugs: FindBugs je alat koji upoređuje program sa bazom podataka sa greškama (database of bugs). Alat FindBugs označava instrukcije u programu gde su detektovane greške (bugs).

Verzija 3.0.1, FindBugs sadrži stotine opisa mogućih greški, bug-descriptions. Ove greške su kategorizovane u 4 nivoa, na osnovu njihove važnosti tj. težine : concern, troubling, scary, and scariest. Osim GUI, FindBugs ima i command-line-interface, i jedan Eclipse-plugin.



Slika 3.1

JAVA ALATI ZA TESTIRANJE

Koji su najkorisniji Java alati za testiranje?

Pogledati link:

- https://blog.idrsolutions.com/2015/02/8-useful-java-testing-tools-frameworks-programmers-developers-coders/

Naime, korisni Java Testing tools(& Frameworks) for Programmers, Developers and Coders.

Početak sa Java programiranjem: Ako tek započinjete Java programiranje, prvo morate da instališete JDK (Java Development Kit), koji dolazi sa Java Runtime Environment (JRE) i JVM (Java Runtime Environment). Ovo možete naći kod Oracle-a. Ovo će vam omogućiti

compile-run- test vaš Java program na vašoj mašini. Takođe vam treba jedan IDE (Integrated Developer Environment), i tu ima nekoliko opcija:

- Intellij, Eclipse and NetBeans.

JAVA ALATI ZA TESTIRANJE-NASTAVAK

Upoznajmo Java alate za testiranje.

Ovde navodimo alate za Java testing.

1. **Arquillian:** Arquillian is a highly innovative and extendible testing platform for JVM that allows developers to easily create automated integration, functional and acceptance tests for Java.

2. **JTest:** JTest also known as 'Parasoft JTest' is an automated Java software testing and static analysis software made by Parasoft. JTest includes functionality for Unit test-case generation and execution, static code analysis, data flow static analysis, and metrics analysis, regression testing, run-time error detection. There are also features that allow you to peer code review process automation and run-time error detection for e.g.:

3. **JUnit:** JUnit is a unit testing framework designed for the Java programming language. JUnit has played an important role in the development of test-driven development frameworks. It is one of a family of unit testing frameworks which is collectively known as the xUnit that originated with SUnit.

4**.TestNG**: TestNG is a testing framework designed for the Java programming language and inspired by JUnit and NUnit. TestNG was primarily designed to cover a wider range of test categories such as unit, functional, end-to-end, integration, etc. It also introduced some new functionality that make it more powerful and easier to use.

VIDEO VEŽBA

Video vežba- Java - JUnit testing in Eclipse

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

INDIVIDUALNA VEŽBA

Zadaci za individualnu vežbu .

Zadaci:

- Za izabranu aplikaciju demonstrirati alat JUnit za jedinično testiranje koristeći white-box tehniku?
- Upoznati se sa alatomJTest ?

- Za izabranu aplikaciju demonstrirati primenu JTest?

Poglavlje 4

Vežba - Konstrukciono testiranje aplikacija

TESTIRANJE APLIKACIJE KORIŠĆENJEM DEFINISANIH TESTOVA - DETERMINISTIČKI SLUČAJ

Test koji obuhvata unapred definisan cilj a to je da aplikacija na osnovu provere kredencijala korisnika u bazi podataka prikaže da su korisničko ime i lozinka pogrešni.

Deterministički slučaj testiranjaKada se test slučajevi definišu na deterministički način to znači da se unapred zna šta će se desiti I kako će program reagovati na zadate kriterijume. Mogu se jasno definsati ali i nasumično izvući zavisno od načina pristupa softverskog inženjera.Uglavnom se menja ulazni parameter a kontroliše izlazni koji je definisan pre početka testiranja.

Test pogrešan username i password Svrha ovog testa je da proveri ispravnost logovanja u aplikaciju, kao i mogućnost prijavljivanja pogrešnih podatakakojekorisnik uneseu ponuđena polja.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Test obuhvata R2 zahtev korisnika iz tabele 1.Detaljan opis testa,njegovepostavke i pregled prikazan je u tabeli.

Naslov	Pogrešan username i password	Rev 1	Autor	Nebojša Gavrilović	Datum	3.10.2014
Cilj	Cilj je provera ispravnosti prijavljivanja greške za pogrešan login		Reference			
Test uslovi		Vreme neophodno za izradu test slučaja	10 min	Neophodno vreme za izvršenje test slučaja		5min

	Opis postavke za testiranje
•	Podešavanja aplikacije su takva da jemoguće pristupiti samo uz username i password koji postoji u bazi
•	Pokrenuta je aplikacija
•	Korisnik koristi tastaturu za unos podataka

	Definicija testa			Izvršenje testa	
	Uslovi	Ulazni podaci	Očekivani rezultati	Aktuelni rezultati	Broj problema
•	Korisnik je izvršio autentifikaciju, uneo pogrešne podatkei kliknuo na dugme login	Unešeni su usernamei password	Od sistema se očekuje da ukoliko podacine postojeu bazi korisniku prikažegrešku za pogrešanunos podataka.		

	Opispostuslova
•	Korisnik vidi da je u pitanju pogrešan unos podataka
•	Aplikacija očekuje od korisnika da ponovo unese prave podatke

Slika 4.1 Deterministički način testiranja

TESTIRANJE APLIKACIJE KORIŠĆENJEM DEFINISANIH TESTOVA - AD HOC TESTIRANJE

Karakteriše ga to što inženjer koji je kreirao aplikaciju zna koji format dokumenata može biti unešen i na osnovu toga kreira test u kome se unosi pogrešan fajla.

AD Hoc testiranjePod AD Hoc testiranjem se najčešće podrazumeva znanje softverskog inženjera u radu i testiranju sličnih aplikacija na osnovu kojih definiše metod testiranja aplikacije. Često se dešava da se isti problemi javljaju na sličnim aplikacijama pa to ponekad može biti dobar i koristan plan. Ovo testiranje može pokazati na koje još načine se može izvršiti provera aplikacije i testiranje funkcionalnosti iste.

Test pogrešan format fajla Svrha ovog testa je da proveri ispravnost ubacivanja fajlova u aplikaciju.Ukoliko korisnik pokuša da ubaci format fajla koji nije podržan, aplikacija će prikazati definisanu grešku. Primer koda koji je testiran prikazan je na slici.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Detaljan opis testa,njegove postavke i pregled prikazan je u tabeli.

Naslov	Pogrešan format fajla	Rev 1	Autor	Nebojša Gavrilović	Datum	3.10.2014
Cilj	Cilj je provera ispravnosti prijavljiva greške za pogrešan format fajla		Reference			
Test uslovi	Vreme neophodno za izradu test slučaja		10min	Neophodno vreme za izvršenje test slučaja		7min

	Opispostavkezatestiranje
•	Podešavanja aplikacije s utakva da jemoguće ubaciti samo dokumentu PDF ili Word formatu
•	Pokrenuta je aplikacija
•	Korisnik koristi fajlove sa svog računara za unos u aplikaciju

	Definicija testa			Izvršenje testa	
	Uslovi	Ulazni podaci	Očekivani rezultati	Aktuelni rezultati	Broj problema
•	Korisnikjeizabrao pogrešan format fajla koji se ubacujeikliknuo ubaci	Signal da je potrebno ubaciti novi fajl	Odsistema se očekuje da ukolikoje pogrešan format fajla prikaže korisnikudajeformat pogrešan.		

	Opis postuslova
•	Korisnik vidi daje u pitanju pogrešan format fajla
•	Aplikacija očekuje od korisnika da ponovo unese podržan format

Slika 4.2 AD Hoc testiranje

ZADACI ZA INDIVIDUALAN RAD

Na osnovu primera koji su dati u okviru vežbi, potrebno je uraditi sledeće zadatke.

Zadatak 1:

U programskom jeziku Java razviti softver koji za uneti datum rođenja izdaje horoskopski znak.

Zadatak 2:

Za napisani program uraditi strukturno testiranje.

Zadatak 3:

U cilju merenja pokrivenosti koda koristiti alat Cobertura.

Zadatak 4:

Pogledati link: http://cobertura.github.io/cobertura/

Na ovom linku dat je detaljan opis alata Cobertura. Cobertura je besplatan Java-alat za proračun procenta programa pristuplenom pomoću testova. Ovaj alat se koristi da se otkrije koji delovi Java-progama nisu pokriveni testovima.

Poglavlje 5

Vežba : Zadaci i pitanja iz testiranja softvera

VIDEO VEŽBA

Video vežba - Primeri testiranja tehnikom bele kutije (White Box Testing Techniques with Examples)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

VIDEO VEŽBA 2

Video vežba - Tehnika testiranja bele kutije koristeći dijagrame (WHITEBOX TESTING AND ITS TECHNIQUES WITH DIAGRAMS)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

TESTIRANJE-ZADACI I PITANJA?

Ovde se daju pitanja i zadaci iz testiranja.

- Šta obuhvata razvojno testiranje ?
- Objasniti razliku izmedju testiranja i *debugging*-a ?
- Napisati pseudokod programa koji proverava da li je trougao pravougaoni. Napisati funkcionalne testove za ovaj program. ?
- Za prethodni program konstruisati strukturne testove. ?
- Napisati jedan program u Javi sa jednom for-petljom I dva if-grananja, I za ovaj program definisati test slučajeve koristeći strukturno minimalno testiranje. ?
- Za program iz prethodnog zadatka definisati test slučajeve pomoću *data-flow* testiranja. ?

DOMAĆI ZADATAK 8

Domaći zadatak 8 se dobija direktno od asistenta nakon poslatog četvrtog domaćeg zadatka ili na poseban zahtev studenta.

Domaći zadatak 8 je individualan za svakog studenta, i dobija se po definisanim instrukcijama.

Domaći zadatak se imenuje:

KI301-DZ8-ImePrezime-brIndeksa gde su vrednosti Ime, Prezime i br.Indeksa vaši podaci.

Domaći zadatak je potrebno poslati na adresu asistenta: **nebojsa.gavrilovic@metropolitan.ac.rs** sa naslovom (subject mail-a) **KI301-DZ8**.

Posebno je potrebno voditi računa o pravilnom imenovanju mail-a prilikom slanja domaćih zadataka.

Napomena:

Domaći zadaci treba da budu realizovani u zadatku navedenom razvojnom okruženju i da predstavljaju jedinstveno rešenje svakog studenta. Prepisivanje i preuzimanje programskog koda sa interneta strogo je zabranjeno.

Poglavlje 6

Zaključak

ZAKLJUČAK

Softverske skele“ se grade da bi omogućili ispitivanje dela nekog programa tj. softvera koji se pravi iz više delova. Npr. softverska skela može biti specijalno napravljena klasa koja se može kristiti od strane druge klase koja se testira. Takva klasa se zove npr. „mock object“ ili „stub object“. Slično, može se isti prilaz koristiti kod testiranja podprograma, i tada se koriste „stub routines“. Ove pomoćne klase ili podprogrami se mogu napraviti manje ili više realistični da zamenjuju ostatak programa. Npr. oni mogu da omogućuju:
testiranje podatka koji im se zada
Dijagnostičke poruke, npr. „eho“ ulaznih parametara
Prikazuje return-values za neki interaktivni ulaz
Pojednostavljena verzija nekog objekta ili nekog podprograma
Druga vrsta softverske skele je veštački podprogram („fake routine“) koji poziva podprogram koji se testira (tzv. „test driver“), gde npr. može
da se traže ulazni podaci da se unesu interaktivno
da se izvrši čitav niz predefinisanih testova
itd
Konačno, dummy-file je takođe jedna vrsta scaffolding-a, gde imamo jednu mini verziju realnog fajla koji ima iste tipove komponenti kao što realni tj. veliki fajl ima, dakle mini imitacija velikog fajla je taj dummy-file.

ZAKLJUČAK 2

Test-data generatori tj. random-data genberatori generišu proizvoljno tzv. test-cases tj. test-slučajeve. Npr. da proizvede 100 proizvoljnih test-slučajeva, koji omogućuju da se nađu greške u programu, npr. 1 greška za 100 test-slučajeva. Evo nekih komentara u vezi random-data generatora:
treba dobro dizajnirati random-data generator da simulira korisnike
ovi generatori test-slučajeva mogu da generišu neobične kombinacije podataka koje inače ne biste predvideli
ovi generatori mogu da generišu lako ogroman broj test-slučajeva
ove generatore treba profinjavati na osnovu iskustva, naime na osnovu rezultata testova može se videti da li je potrebno redizajnirati random-data generator
Baza podataka greški, tj. Error Database, je moćan alat za testiranje. To je baza podataka koja memoriše greške koje su bile prethodno pronađene. Ove baze podataka greški omogućuju:
provera greški koje se ponavljaju
praćenje korekcija greški
itd

LITERATURA

1. Code Complete: A practical handbook of software construction, by S. McConnell, Microsoft Press, ISBN 0-7356-1967-0, https://www.amazon.com/Code-Complete-Practical-Handbook-Construction/dp/0735619670
2. SWEBOK-V3, https://www.computer.org/web/swebok/v3
3. Theory and Problems of Software Engineering - Schaum's Outline Series, by David Gustafson, ISBN 0-07-137794-8, http://www.mhebooklibrary.com/doi/abs/10.1036/0071377948

Linkovi:

JUnit tool, http://www.vogella.com/tutorials/JUnit/article.html

Intro to Testing, http://www.cs.toronto.edu/~sme/CSC302/

Testing Strategies, http://www.cs.toronto.edu/~sme/CSC302/