Testavimas. JUnit

Rankinio testavimo privalumai

Išmokstamumas – testuojant rankiniu būdų reikia mažiau techninių žinių, nei rašant automatizuotus testus, todėl mokymosi laikotarpis trumpesnis.

Aplikacijos padengimas testais – daugelyje atvejų automatizuoti testai nepadengia viso funkcionalumo, kurį gali padengti rankinis testavimas.

Arčiau tikrų vartotojų – joks įrankis negali pakeisti žmogaus išmonės bei patirties. Rankiniu būdu testuojant galima pagauti daugiau specifiniu defektų, vykdant kitokius žingsnius, ar kitokia tvarka.

Greičiau atnaujinami testavimo atvejai – nereikia programuoti naujų testavimo atvejų, užtenka juos trumpai aprašyti.

Rankinio testavimo trūkumai

Atima daug laiko – kadangi viskas atliekama rankomis, įvykdyti kiekvieną testavimo atvejį kainuoja daug laiko.

Nuobodus - tuos pačius testavimo atvejus reikia kartoti keletą kartų, todėl tai gali atsibosti.

Reikalauja daug žmogiškųjų išteklių – kuo daugiau reikia ištestuoti rankiniu būdu, tuo daugiau testuotojų reikia.

Mažiau patikima – rankiniame testavime gali pasitaikyti žmogiškųjų klaidų

Automatinio testavimo privalumai

Greitas vykdymas – testai vykdomi greičiau, negu juos atliekant rankiniu būdu.

Mažesnės išlaidos – kadangi naudojami automatiniai įrankiai, mažiau testuotojų reikia, jie gali užsiimti sudėtingesnėmis užduotimis.

Pernaudojamumas – tie patys testai gali būti leidžiami ant skirtingų prietaisų/aplinkų.

Patikimumas – kiekvieną kartą atliekami tie patys žingsniai – išvengiama žmogiškųjų klaidų.

Greiti rezultatai – atsiradus naujai versijai, automatinis testavimas padeda greičiau sužinoti, ar versija tinkama – pateikia atlikto testavimo išvadas

Automatinio testavimo trūkumai

Reikalauja daug techninių žinių – automatinių testų rašymui reikia techninių žinių apie programavimą. Tam reikia apmokyti esamus darbuotojus, arba įdarbinti naujų specialistų.

Didelės pradinės investicijos – darbuotojų apmokymai bei nauji įrankiai – tai papildomos investicijos.

Neįmanoma ištestuoti visko – testų, tikrinančių vartotojo sąsajos draugiškumą (angl. user friendliness) ištestuoti neįmanoma.

Automatinių testų tipai

Moduliniai testai (angl. unit tests) – jie tikrina kodo vienetus atskirai vieną nuo kito. Kiekvienas kodo vienetas (funkcija, klasė, metodas) turi savo vieną ar kelis testus, kuriuos dažniausiai rašo programuotojas.

API testai (angl. Application Programming Interface) – jie tikrina verslo logiką, ar funkcionalumas, skaitomumas, vykdymas (angl. performance) atitinka reikalavimus

GUI testai (angl. Graphical User Interface) – tai vartotojo sąsajos testavimas. Automatizuojama pelės bei klaviatūros paspaudimai, rašymas į laukelius. Tai padeda greitai rasti defektus daugeliu atvejų, pavyzdžiui atliekant regresinį testavimą, ar pildant ilgas formas, kas užima daug laiko

Junit testų naudojimas

Junit testai – klasės paveldinčios TestCase klasę, ir joje turi būti surašyti metodai skirti testavimui. Minimali testavimo klasė:

```
import junit.framework.TestCase;

public class TestUser extends TestCase {

} // Testuos User klasę
```

Klasės pavadinimas privalo būti testuojamos klasės pavadinimas su prefixu arba sufixu Test. Pavyzdžiui: TestMyClass arba MyClassTest

Testų rašmas

Kiekvieną testą turi sudaryti tokia tvarka:

- Sukuriami kintamieji ir testui skirta aplinka
- Įvykdomos funkcijos kurios bus testuojamos
- Tikrinama ar po funkcijų rezultatai yra tokie kokių tikimasi

```
public void testEmptyList() {
   Bowl emptyBowl = new Bowl();
   assertEquals(“Size of an empty list should be zero.”,  0, emptyList.size());
   assertTrue(“An empty bowl should report empty.”,  emptyBowl.isEmpty());
}
```

Assert metodai

Kiekvienas assert metodas turi kelis pagrindinius parametrus:

Pranešimas, laukiama reikšmė, gauta reikšmė

Dirbant su slankaus kablelio skaičiais nurodoma papildoma reikšmė – paklaida.

Kiekvienas assert metodas taip pat turi alternatyvų metodą be pranešimo.

Assert metodai

assertTrue(String *message*, Boolean *test*)

assertFalse(String *message*, Boolean *test*)

assertNull(String *message*, Object *object*)

assertNotNull(String *message*, Object *object*)

assertEquals(String *message*, Object *expected*, Object *actual*) (uses equals method)

assertSame(String *message*, Object *expected*, Object *actual*) (uses == operator)

assertNotSame(String *message*, Object *expected*, Object *actual*)

Testavimo klasė

Testavimo klasė taip pat gali turėti papildomus metodus:

konstruktorių

protected void setUp() – paleidžiamas prieš kiekvieną testą (gali būti naudojamas kintamųjų inicializavimui) Test fixture creates and initializes instance variables, etc.

protected void tearDown() - paleidžiamas po kiekvieno testo (failų ar kitų resursų uždarymui)