Tarkvara testimine

Lühiülevaade

Mõisted

Testimine on osa tarkvaraarenduse protsessist, mille käigus hinnatakse tarkvara kvaliteeti. Lihtsamalt juhul on **testimise eesmärgiks** programmist (loogika)vigade leidmine. Vigu võib otsida nii programmi käivitades kui ka koodi läbi vaadates.

Testimise eesmärk ei ole vigade puudumise tõestamine.

Staatiline testimine – programmi ei käivitata, vigu otsitakse lähtetekstis.

Dünaamiline testimine – vigu otsitakse töötavast programmist.

Mõisted

Edukas on test siis kui leiti viga.

Edsger W. Dijkstra: *"Programmi testimise abil saab näidata, et programmis on vigu, kuid ei saa tõestada vigade puudumist."*

„Program testing can be used to show the presence of bugs, but never to show their absence!” *(Dijkstra (1970) "Notes On Structured Programming" (EWD249), Section 3 ("On The Reliability of Mechanisms"), p. 7.)*

Staatiline testimine

Koodi läbivaatus on kollektiivne üritus ja selleks on kontrollküsimustikud.

Küsimustiku eesmärgiks on probleemide süstematiseerimine.

Erinevate programmeerimiskeelte puhul on sobilikud veidi erinevad küsimustikud.

Järgnev küsimustik tugineb konspektile: http://tepandi.ee/tks-loeng.pdf

Küsimustik.Andmete kirjeldamine

Kas muutujad on ilmutatult kirjeldatud? Osades keeltes kontrollivad seda kompilaatorid.

Kas muutujad on algväärtustatud õigesti?

Kas muutujad on sarnaste või segadusse ajavate nimetustega, näiteks L1 ja Ll või Bing0 ja BingO.

Küsimustik.Muutujate kasutamine

Kas muutuja poole pöördumisel on muutujal väärtus olemas? Mõned kompilaatorid kontrollivad ja annavad hoiatuse.

Kas indeks võib minna väljapoole lubatud piire (stringis, massiivis, listis jm)?

Kas indeksile omistatakse täisarv?

Küsimustik.Arvutusvead

Kas arvutatakse lubamatute andmetüüpidega? Mõnes keeles aitab kompilaator tuvastada.

Kas juhtub üle- või alatäitumist (tulemus kaob väiksuse tõttu, väärtus ületab andmetüübi piiri)?

Kas võib toimuda jagamist nulliga?

Kas tehete järjekord on õige (sulud!)?

Küsimustik.Võrdluste vead

Kas võrreldakse erinevat tüüpi muutujaid: teksti ja arvu?

Kas märgid (näiteks *'suurem' / 'suurem-võrdne')* on õigesti kasutatud.

Kas võrdlustehete järjekord on antud õigesti?

Kas võrreldakse ujukomamuutujat mingi kindla arvuga ja võrdumistehtega?

Küsimustik.Juhtimise vead

Kas programmi struktuuri määravad taanded (või muud märgised) on õiged ning tsüklite ja tingimuslausete piirid õigesti määratud?

Kas while-tsükkel lõpetab töö (tingimus saab muutuda vääraks *(false)*)?

Mis juhtub, kui tsükli tingimus on kohe *false*?

Dünaamiline testimine

Programmi käivitamine vigade leidmise eesmärgil:

- Koostatakse testandmete komplektid: sisendid ja oodatavad väljundid
- Programm käivitatakse sisendandmetega. Kui väljund on selline nagu oodati, on programm selle testi läbinud.
- Kui väljund on teistsugune, on ilmselt programmis viga. Viga tuleb üles otsida, parandada ja programmi uuesti testida (ja mitte ainult nende andmetega, mis eelnevalt vea andsid).

On palju dünaamilise testimise meetodeid, mis sobivad erinevates tarkvaraarenduse faasides. Vaatame paari lihtsamat võtet, millega oma väikeseid programme testida.

Funktsionaalne testimine

Programmi vaadatakse kui musta kasti – me ei tea, mis seal sees on.

Testide koostamisel on aluseks spetsifikatsioon (ülesande kirjeldus), mitte kood.

Kõigi võimalike andmetega ei suudeta kunagi programmi testida, seega tuleb valida testid targalt.

Olulisemad meetodid testide koostamiseks:

- Ekvivalentsiklassid
- Piirjuhud

Ekvivalentsiklassid

Idee – sarnaste andmete puhul peaks programm ühte moodi käituma: kui ühe tekib viga, tekib ka teisega.

Kuidas leida sarnased andmed? Andmed jaotatakse **ekvivalentsiklassidesse**. See võib olla:

- Järjestatud tõkestatud vahemik
- Järjestatud tõkestamata vahemik
- Sarnaselt käituvate andmete hulk
- Erinevalt käituvate andmete hulk
- Valik / tingimus (nt ei/ja)

Ekvivalentsiklassid

Erinevat tüüpi klassidest andmete valimine:

- järjestatud tõkestatud vahemik: võta testandmed vahemiku seest, enne vahemikku ja pärast vahemikku
- järjestatud tõkestamata vahemik: võta testandmed vahemiku seest ja väljast
- ühte moodi käituvate elementidega hulk: võta testandmed hulga seest ja väljast
- erinevalt käituvate elementidega hulk: testi kõiki elemente
- tingimus: proovi mõlemaid variante (tingimus täidetud/täitmata)

Piirolukorrad

Piirolukorrad seonduvad ekvivalentsiklassidega. Piiridel esineb kõige rohkem vigu eraldi testid.

Kui piir on reaalarv, teha testid sellel piiril, sellest veidi suuremal ja väiksemal väärtusel.

Kui ülemist (alumist) piiri pole antud, testida väga suure positiivse (negatiivse) arvuga.

Kui piir on täisarv N, testida väärtusi N, N-1, N+1

Kui sisendandmeteks on järjestamata hulgad, siis piirolukordi ei teki.

Programmi teksti järgi

Testide koostamisel on aluseks programmi kood/struktuur. Erinevad tehnikad:

- **Lause adekvaatsuse kriteerium** – iga lause peab vähemalt üks kord olema töötanud.
- **Haru adekvaatsuse kriteerium** – iga haru peab olema vähemalt üks kord töötanud (ka tühjad harud).
- **Andmetepõhine testimine** – testid tehakse andmestruktuuride baasil.
- **Tsüklite testimine** – tugineb tsükli korduste arvule, testitakse min ja maks arvu ümber.

Juhuslikud sisendid

Tavaliselt nii alguses tehakse – kuidas oma programme testid? :)

Sobib näiteks koormustestimiseks.

Andmed peab genereerima sarnaselt programmi kasutusprofiilile, mitte suvaliselt.

Juhuslikkus peab olema matemaatiline, mitte programmeerija oma.

Ainsa meetodina kindlasti ei sobi.

Kokkuvõte

Ühtegi meetodit ei kasutata eraldi. Igal meetodil on oma plussid ja miinused.

Ükski meetod ei anna täielikku kindlust ja garantiid.

On hea, kui keegi teine testid koostab ja testib, sest programmeerija ise ei tavaliselt pisiavalt motiveeritud oma koodi põhjalikult testima. Ja teiselt teades omaenda loogikat koostab ta pigem testid, mis vigu ei anna.

Silumisest

Allikas: A. Downy Think Python

Silumine on (loogika)vigade tuvastamine ja parandamine.

Silumine on nagu detektiivitöö. Sherlock Holmes: *“When you have eliminated the impossible, whatever remains, however improbable, must be the truth.”*

Parem on teha vigu õppimise ajal ja meelega kui pärast ja kogemata – katseta, arvuti ei lähe katki.

Veendu, et kood, mida Sa vaatad on sama kood, mida Sa käivitad (programm mitmes kohas või salvestamata).

Silumisest

Veateade ütleb, millal viga avastati, kuid see ei pruugi olla vea tekkekoht – tekkida võis juba varem.

Aita end väljatrükkidega. Mõistlikul viisil ja mõistlikus kohas olevad print-laused aitavad programmi käitumise loogikast aru saada.

Ära tee juhuslikke muudatusi!

Vigade klassifikatsioon

Süntaksivead (ingl *syntax error*) – tähendab programmeerimiskeele reeglite rikkumist. Programm sellisel juhul ei käivitu. Algaja programmeerija teeb neid vigu palju ja leidmine on vaevaline.

Täitmisaegsed vead (ingl *runtime error*) – ilmnevad programmi täitmise ajal. Pythonis nimetatakse ka **erind** (ingl *exeption*)

Semantika e **loogikavead** (ingl *semantic error*) - programm töötab, kuid ei tee õiget asja. Samas veateateid ei tule ja nende vigade märkamine ja kõrvaldamine on keeruline.

Täitmisaegsed vead

Tüüpilised näited: nulliga jagamine või muu lubamatu tehe, ebasobiv sisend, puuduv fail, ...

Pythonis on erindid, mis katkestavad samuti programmi täitmise:

- NameError – kirjeldamata muutuja
- TypeError – ebasobiv andmetüüp
- IndexError – jadas liiga suur indeks

Osa Pythoni erindeid leitakse mõnes teises programmeerimis keeles juba kompileerimise käigus ja koodi ei käivitata: nt NameError või TypeError. Selline tuvastamine on võimalik, kui muutujad on eelnevalt deklareeritud.

Semantikavead

Loogikaviga: aitab avastada testimine.

Vea lokaliseerimiseks tuleb aru saada seosest oma programmi teksti ja arvuti käitumise vahel – trüki ja katseta programmi tükk haaval!!

Ära kirjuta liiga pikki avaldisi.

Jälgi tehete järjekorda, lisa ()!!

Mõtle, võimaluse korral distantseeru, eriti kui:

sa saad tigedaks,

leiad, et arvuti vihkab Sind,

teed programmis juhuslikke muudatusi.

Pythoniga seonduvalt:

Mõtteid raamatust „How to Think Like a (Python) Programmer“

Sea eesmärk*: leia süntaksi vead enne käivitamist.*

Loe programm läbi ning veendu, et:

- ühelgi muutujal ei ole nimeks võtmesõna
- juhtlause lõpus (if, while, for, def) on koolon.
- treppimine on järjekindel. Soovitav on kasutada ainult tühikuid või ainult tab-e. Taanded ühe suurusega.
- kõigil stringidel on ja lõpusümbol.
- kõigil sulgudel on paarilised.
- võrdlemine toimub == ja mitte =-ga.