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

Tehnike testiranja

Lekcija 07

PRIRUČNIK ZA STUDENTE

KI301 - KONSTRUISANJE SOFTVERA

Lekcija 07

TEHNIKE TESTIRANJA

- Tehnike testiranja
- Poglavlje 1: Uvod u konstrukciono testiranje
- Poglavlje 2: Metode testiranja softvera
- Poglavlje 3: Test-first metoda
- Poglavlje 4: Strukturno osnovno testiranje
- Poglavlje 5: Vežbe - Metode testiranja
- Zaključak

Uvod

L7:TEHNIKE TESTIRANJA

Sadržaj:

- Uvod u konstrukciono testiranje
- Metode testiranja softvera
- Metoda test-first
- Strukturno osnovno testiranje
- Testiranje toka podataka

Poglavlje 1

Uvod u konstrukciono testiranje

RAZVOJNO TESTIRANJE

„Razvojno“ testiranje se sastoji od testiranja softverskih jedinica i testiranja softverskih komponenti.

Testianje je vrlo važno za poboljšanje kvaliteta softvera.Testiranje se obavlja od strane razvojnih softverista, ali i od strane specijalizovanih test-specijalista. Ima različitih vrsta testiranja. Testiranje softverskih jedinica, gde se ispituje neka kompletirana klasa ili podprogram ili mali program, i to izolovano od ostalog dela sistema, pri čemu je ta jedinica napisana od strane jednog rogramera ili jednog tima programera. Testiranje softverskih komponenti, gde se ispituje neki element programa, npr. klasa, paket, manji program, i to izolovano od ostalog dela sistema, ali koji je razvijen od strane nekoliko programera ili nekoliko timova programera. Integracioni testovi je kombinovano ispitivanje dve ili više klasa, paketa, komponenti, podsistema, razvijen od strane nekoliko programera ili nekoliko timova programera. Sistemsko testiranje, gde se softver testira u finalnoj konfiguraciji, uključujući integraciju sa ostalim softverom i računarskom opremom

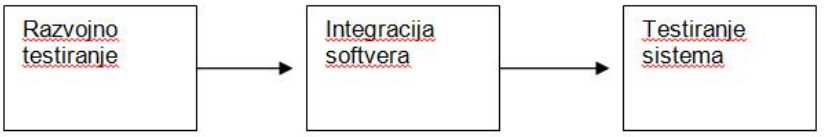
Testiranje u kompaniji koja razvija softver obuhvata testiranje softverskih jedinica, testiranje softverskih komponenti, i integracione testove. Sistemsko testiranje ide posle integracije. Ponekad i sistemsko testiranje se obavlja u kompaniji koja je razvila softver. Integracija tj. integraciono testiranje se radi u okviru testiranja, a posle „razvojnog“ testiranja. Dakle, „razvojno“ testiranje se sastoji os testiranja softvcerskih jedinica i testiranja softverskih komponenti.

Testiranje i „dedefektovanje“,debugging, nije isto, naime,

- testiranje omogućuje detekciju greški,
- a debugging, to je dijagnoza i korigovanje uzroka greške koja je već otkrivena.

TESTIRANJE SOFTVERA

Slika prikazuje testiranje softvera .



Sl. 1 -Faze testiranja softvera

SISTEMATSKI PRILAZ RAZVOJNOM TESTIRANJU-PREPORUKE ZA TESTIRANJE

Evo od čega se sastoji jedan sistematski prilaz testiranju.

Ovde se bavimo samo sa detekcijom greški, tj testiranjem. Jedan sistematski prilaz razvojnom testiranju maksimizira sposobnost nalaženja greški sa minimumom napora, i to:

- testirati svaki relevantni softverski zahtev, da bi se proverila implementacija zahteva za softver
- testirati svaki problem koji je uočen u fazi dizajna, i već tada u fazi dizajna napisati test-slučajeve
- testirati svaku liniju programa pomoću tzv baznog testiranja
- testirati tok-podataka
- koristiti listu-provere gde se navode greške iz prošlosti, tj iz prethodnih projekata kao i iz prethodnih faza dotičnog projekta

NESAVRŠENOSTI RAZVOJNOG TESTIRANJA

Razvojni softveristi obično smatraju da je pokrivanje svake linije programa adekvatno, medjutim bolji standard zahteva i testiranje svake putanje u programu.

Nesavršenosti razvojnog testiranja se mogu sabrati na sledeći način

Razvojni softveristi pre testiraju da li softver funkcioniše nego da traže slučajeve tj izuzetke kad softver ne funkcioniše

Obično kod razvojnog testiranja se pokrije 80% testova, a vruje se da je pokrivenost 95%

Razvojni softveristi obično smatraju da je pokrivanje svake linije programa adekvatno, medjutim bolji standard zahteva i testiranje svake putanje u programu i to kako za ispravne tako i neispravne vrednosti

Nije moguće dokazati testiranjem da je program potpuno ispravan. Da bi se to uradilo trebalo bi proveriti savaki mogući ulaz, i svaku moguću kombinaciju ulaznih veličina. Medjutim, čak i za male programe to je iscrpljujuće do nemogućnosti. Npr. program koji učitava ime, adresu i telefonski broj u datoteku. I ako je svaki ovaj podatak 20 znakova dugačak, mogući broj ulaza je 10^{66}, znači nemoguće za testiranje. Medjutim, ne ide se na iscrpno testiranje, već samo se testiraju slučajevi koji testiraju različite stvari, a ne testira se ponavljanje jedne te iste stvari za sve različite vrednosti.

PROCENAT VREMENA RAZVOJA SOFTVERA ZA RAZLIČITE PROJEKTE

Kod testiranja, korisno je posmatrati klasu kao „glass-box“ a ne „black-box“ jer da bi se detaljno istetstirala klasa potrebno je posmatrati iyvorni kod i ulaze i izlaze.

Tabela: procenat vremena razvoja softvera za projekat od 100kl i 500kl (kl=1000 linija u programu)

	100k	500k
System testing	20%	20%
Integration	15%	20%
Developer testing	15%	10%
Coding+debugging	15%	10%
Detailed design	20%	15%
Architecture	15%	25%

U gornjoj tabeli dat je prikaz procenat vremena razvoja softvera za projekat od 100kl i 500kl (kl=1000 linija u programu). Kod konstruisanja softvera, tj. kod pisanja softvera, ili se piše klasa (*class*) ili podprogram tj. procedura (*routine)*, zatim se ona proveri mentalno, i posle toga testira. Nezavisno od integracije ili od sistemskog testiranja, pre ovoga potrebno je detaljno istestirati svaku softversku jedinicu (*unit testing*) pre kombinovanja sa drugim jedinicama softvera. Ako pišete nekoliko podprograma ili klasa, potrebno ih je testirati jednu po jednu. Kod testiranja, korisno je posmatrati klasu kao „*glass-box*“ a ne „*black-box*“ jer da bi se detaljno istetstirala klasa potrebno je posmatrati iyvorni kod i ulaze i izlaze. Naime, ako znate šta je unutar klase, onda možete da detaljnije iztestirate klasu. Ali, i testiranje kao „*black-box*“ isto ima svojih prednosti.

ULOGA RAZVOJNOG TESTIRANJA KOD KVALITETA SOFTVERA

Testiranje je važan deo plana za postizanje kvaliteta, a često je i jedini deo.

Testiranje je važan deo plana za postizanje kvaliteta, a često je i jedini deo. Međutim, prakse kolaborativnog razvoja (collaborative development) u različitim formama su se pokazale da omogućuju pronalaženje velikog broja greški npr. približno isto koliko i testiranje, i ove metode kolaborativnog razvoja su upola jeftinije od testiranja.

Testiranje koje obuhvata

- unit test,
- component test,
- integration test

obično detektuje ukupno oko 50-60% postojećih greški. Dakle, software quality assurance plan treba da uključi i kolaborativno konstruisanje da bi se otkrilo što više greški, tj. greški koje se ne mogu otkriti tetstiranjem. Obično oko 50% ukupnog vremena razvoja softvera odlazi na debugging i testiranje, pri čemu je ovde u testiranje uključeno i ne-razvojno testiranje.

TESTIRANJE TOKOM KONSTRUISANJA

Ako razvijate nekoliko klasnih metoda, treba ih razvijati jednu po jednu a ne istovremeno.

Kod konstruisanja, programer prvo napiše ili podprogram (tj. klasnu metodu) ili klasu, zatim je proveri mentalno, i onda testira. Bez obzira koju vrstu integracione strategije se primenjuje, treba svaku softversku jedinicu (klasnu metodu ili klasu) detaljno istestirati pre nego se one kombinuju sa ostalim softverskim jedinicama. Ako razvijate nekoliko klasnih metoda, treba ih razvijati jednu po jednu a ne istovremeno.

Klasne metode ja mnogo lakšedebugg-ovati jednu po jednu, nego istovremeno, jer ako dodajete jednu po jednu klasnu metodu nekoj klasi, onda bilo koja nova greška je

- ili usled greške u toj klasnoj metodi
- ili usled interakcije te nove klasne metode sa ostatkom klase

Prema tome, lakše je debugg-ovati klasu ako dodajete jednu p jednu metodu nego sve istovremeno.

Kod testiranja kod konstruisanja bitno je sledeće:

- Metode kolaborativnog konstruisanja ima prednosti koje testiranje ne može da ponudi.
- Ali, često testiranje nije dobro obavljeno, jer nije istestirao sve delove programa.

Kod testiranja neke klase postoje dve vrste:

- black-box testiranje, gde se gleda šta klasa radi sa gledišta korisnika te klase, i ne posmatra se klasa iznutra tj. iza interfejsa klase
- glass-box testiranje tj. white-box testiranje, gde se gleda se analizira source-code klase i takođe ulazi i izlazi iz klase
- iblack-box i glass-box testiranje imaju svaka svoje prednosti, tako da obe metode zajedno treba koristiti a ne samo jednu metodu,
- ponekad se ignoriše glass-box testiranje i koristi samo black-box testiranje što je pogrešno

STANDARDI RELEVANTNI ZA TESTIRANJE

Evo liste relevantnih standarda.

Standardi bitni za testiranje:

- IEEE Std 1008-1987 (R1993), Standard for Software Unit Testing
- IEEE Std 829-1998 (R1993), Standard for Software Test Documentation
- IEEE Std 730-2002 (R1993), Standard for Software Quality Assurance Plans

VIDEO VEŽBA

Video vežba - Ciklusi i faze testiranja softvera -STLC (Software Testing Life Cycle) and STLC phases

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

VIDEO VEŽBA 2

Video vežba - Ciklus testiranja softvera (Software Development Lifecycle in 9 minutes!)

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

PITANJA ZA SAMOPROVERU ZNANJA

Pitanja za vežbu

- Nabrojati standarde relevantne za testiranje softvera?
- Koje su nesavršenosti razvojnog testiranja?
- Opisati sistematski prilaz testiranju softvera?
- Kako se može testirati klasa?
- Šta je to testiranje staklene kutije (glass-box testing)?

Poglavlje 2

Metode testiranja softvera

SELEKCIJA TEST SLUČAJEVA

Zaustavni kriterijum može bitiili unapred specificiran broj slučajeva u svakom subdomenu testiranja ili se posmatra broj grešaka.

Testiranje softvera je izvršavanje tj. rad („egzekucija“) softvera sa test podacima. Jedan od osnovnih problema kod testiranja softvera je selekcija test-slučajeva (test-case selection). Postoje sledeće metode za selekciju test-slučajeva:

Funkcionalna, koja se bazira na specifikaciji softverskih zahteva (software requirements specification)

Strukturna, koja se oslanja na strukturu izvornog koda programa

Data-flow, tj. testiranje toka podataka u programu

Random tj. proizvoljno trestiranje

Sa druge strane, zaustavni kriterijum (stopping criterion) može biti

ili unapred specificiran broj slučajeva u svakom subdomenu testiranja (gde se domen svih test-slučajeva deli na subdomene)

ili se posmatra broj grešaka u softveru koji se pojavljuje kod testiranja, pa ako se ovaj broj smanji do neke male vrednosti koja je zadata.

FUNKCIONALNO TESTIRANJE

Obično može da se generiše jedan test-slučaj za svaki različit tip izlaza iz programa.

Kod funkcionalnog testiranja se koristi softverska specifikacija (*software requirements specification*) da se definišu subdomeni koji treba da se testiraju. Obično može da se generiše jedan test-slučaj za svaki različit tip izlaza iz programa. Npr. svaka *error-message* treba da se generiše pomoću testova. Onda, svi specijalni slučajevi treba da se testiraju. Dalje, uobičajene greške kod upotrebe softvera treba da se testiraju. Ovi test- slučajevi omogućuju razvojnom softveristi kod razvojnog testiranja (konstrukcionog testiranja) da proveri da li se razvijeni softver ponaša kao što se očekuje.

Primer :Klasičan primer je program koji kao ulaz koristi tri broja NumberA, NumberB, NumberC, i koji treba da analizira ove brojeve polazeći od pretpostavke da ova tri broja su tri strane trougla, i da kao rezultat tj. kao izlaz da tip trougla

Kao primer funkcionalnog testiranja, domen testiranja se može podeliti na subdomene, gde svaki subdomen odgovara različitom tipu trougla:

raznostrani trougao, jednakokraki trougao, jednakostrani trougao

Takodje, mogu se identifikovati dve situacije sa greškama:

subdomen sa lošim ulaznim podacima (*bed inputs*)

subdomen gde podaci ne odgovaraju trouglu (npr. jedna strana trougla mora biti manja od zbira druge dve strane)

Takodje, treba kod specificiranja test-slučajeva razmotriti razne kombinacije ulaznih podataka, i takodje za svaki test-slučaj naznačiti koji je izlaz iz programa.

PRIMER TEST-SLUČAJEVA

Evo primera test-slučajeva za program za trougao.

Evo test-slučajeva:

Subdomeni su: raznostrani trougao, jednakokraki trougao, jednakostrani trougao, nije trougao, loši ulazi

Raznostrani trougao:

4,5,6; 5,4,6; 4,6,5

Jednakokraki trougao:

6,6,9; 6,9,6; 9,6,6; 9,9,6; 9,6,9; 6,9,9

Jednakostrani trougao:

6,6,6

Nije trougao:

8,2,5; 5,8,2; 1,5,9

Loši ulazi

0,0,0

-4,5,6

4,-6,-5

STRUKTURNO TESTIRANJE

Kod testiranja svake instrukcije, traži se da se svaka instrukcija u programu izvrši bar jedanput od strane nekog test-slučaja u toku testiranja test-slučajevima.

Strukturno testiranje se bazira na strukturi izvornog koda programa (*source code*). Postoje tri vrste strukturnog testiranja:

Testiranje svake instrukcije u program (Every-Statement Coverage)

Testiranje svake grane u programu (Every-BranchTestiranje)

Testiranje svake putanje (Every-Path Testing)

Kod testiranja svake instrukcije, traži se da se svaka instrukcija u programu izvrši bar jedanput od strane nekog test-slučaja u toku testiranja test-slučajevima. Tako da se test slučajevi biraju jedan po jedan dok se sve instrukcije u programu ne izvrše.

Evo primera testiranja svake instrukcije:

Posmatramo ponovo primer programa koji učitava i onda analizira tri brojne vrednosti i kao rezultat daje informaciju a li je trougao tipa raznostran ili jednakokrak ili jednakostran. Dole je data tabela koja prikazuje za razne test-slučajeve da li je izvršena neka instrukcija (linija) u programu. Program ima 10 instrukcija i posmatra se nekoliko test-slučajeva, i za svaki test-slučaj se analizira koje instrukcije su izvršene, I možemo videti da su sve instrukcije izvršene, neke više puta a neke samo jednom.

PRIMER PSEUDOKODA

U donjoj tabeli je dat pseudokod programa.

Tabela: pseudokod instrukcije i test-slučajevi

Instrukcija

1. READ: NumA, NumB,NumC

2. WRITE: NumA, NumB,NumC

3. IF(NumA==NumB OR NumB==NumC OR NumA=NumC)

4. WRITE: two equal sides

5. IF(NumA==NumB AND NumB==NumC)

6. WRITE: three equal sides

7. IF(NumA>=NumB+NumC OR NumB>=NumC+NumA OR NumC>=NumA+NumB)

8. WRITE: IMPPOSIBLE

9. IF(NumA<=0 OR NumB<=0 OR NumC<=0)

10. WRITE: BAD DATA

TEST-SLUČAJEVI ZA INSTRUKCIJE PSEUDOKODA

Ovde se navode test-slučajevi za instrukcije u pseudokodu.

Instrukcija_____Test-slučaj_____4,5,6_____4,4,6_____6,6,6_____0,1,2

1_____yes_____yes_____yes_____yes

2 _____ yes _____ yes ____ yes _____ yes

3 _____ yes _____ yes ____ yes _____ yes

4 _____ yes _____ yes ____ yes

5 _____ yes _____ yes ____ yes _____ yes

6 _____yes

7 _____ yes _____ yes ____ yes _____ yes

8 _____ yes

9 _____ yes _____ yes _____ yes _____yes

10

PRIMER TESTIRANJA SVAKE GRANE U PROGRAMU

Pod granama se podrazumevaju grane u dijagramu toka programa (control flow graph).

Kod testiranja svake grane u programu, potrebno je da se iizvrši svaka grana u programu bar jedan put kod testiranja test slučajevima. Tako da se test slučajevi biraju jedan po jedan dok se sve grane u programu ne izvrše. Pri tome pod granama se podrazumevaju grane u dijagramu toka programa (*control flow graph*). U ovom dijagramu se if-instrukcije prikazuju tako da se vidi grananje u programu.

Evo primera testiranja svake grane u programu:Prethodno je potrebno nacrtati dijagram toka programa tj. dijagram kntrolne strukture programa. Evo tog dijagrama za prethodni program:

I evo tabele test-slučajeva: Test-slučajevi

Grana 4,5,6 4,4,6 6,6,6 0,1,2

1,2,3-4 ___yes ___ yes ___ yes

1,2,3-5 ___yes

4-5 ___ ___ ___ __ yes ___yes ___yes

5-6 _____ _____ _____ ___yes

5-7 ___ ___yes ___ yes _____yes

6-7 ___ ___ ___ ___ _____yes

7-8 _____ _____ _____ _____ ___ yes

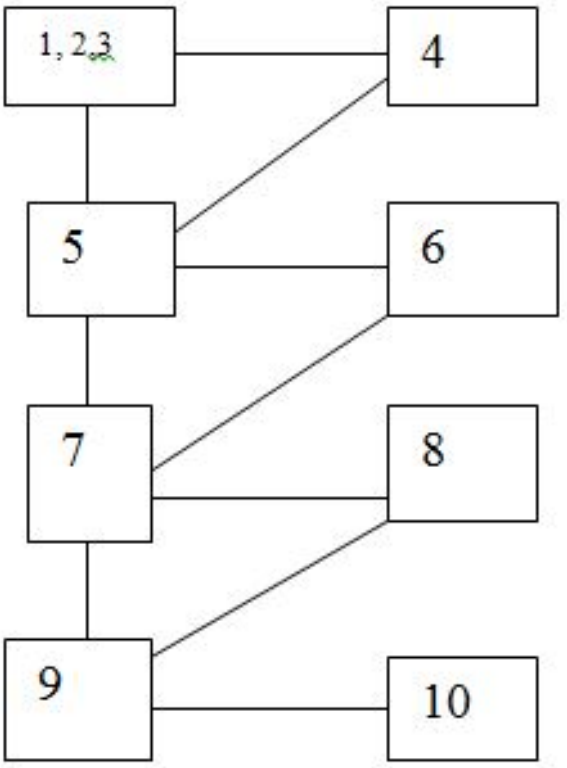
7-9 yes yes yes

8-9 _____ _____ _____ _____ ___ yes

9-10 _____ _____ _____ _____ ___yes

PRIMER DIJAGRAMA TOKA PROGRAMA

Dat je dijagram toka programa koji se testira.



Slika 2.1 Dijagram toka programa

TESTIRANJE PUTANJI , I RANDOM TESTIRANJE

U praksi se ne ide na testiranje svih putanja već jednog redukovanog broja putanja pošto neke putanje nisu izvodljive u programu u praksi.

Kod testiranja svake putanje, potrebno je testirati svaku putanju na dijagramu toka programa. A putanja je jedinstven niz instrukcija koje se izvršavaju kod rada programa za neki test-slučaj. Npr. na prethodnom primeru programa imamo putanje: 1,2,3,5,7,9 ili 1,2,3,5,7,9,10, itd. Medjutim, u praksi se ne ide na testiranje svih putanja već jednog redukovanog broja putanja pošto neke putanje nisu izvodljive u programu u praksi.

SLUČAJNO TJ. PROIZVOLJNO TESTIRANJE (TESTIRANJE NASUMICE)

Random testiranje (slučajno testiranje) se obavlja slučajnim tj. nasumičnim izborom test-slučajeva. Ovaj prilaz ima prednosti da je brz I da otklanja predrasude test-inženjera. Npr. kod prethodnog primera programa koji učitava i analizira tri broja koja treba da predstavljaju strane trougla, može se koristiti random-number-generator (program koji proizvoljno generiše brojeve), i svaka tri uzastopna broja koje generiše ovaj generator brojeva možemo uzeti kao test-slučaj. Na ovaj način možemo brzo generisati i testirati veliki broj test-slučajeva.

VIDEO VEŽBA

Video vežba - Tipovi testiranja softvera (Types of Software Testing)

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

VIDEO VEŽBA 2

Video vežba - Tehnika testiranja crne kutije (Black box testing techniques)

ZADACI ZA VEŽBU

Zadaci

- Uraditi jedan primer funkcionalnog testiranja za izabrani podprogram?
- Uraditi jedan primer strukturnog testiranja za izabrani podprogram?

Poglavlje 3

Test-first metoda

METODA PROGRAMIRANJA PRVO-TESTIRAJ (METODA TEST-FIRST PROGRAMMING)

Bolje je napisati test-slučajeve pre kodiranja nego posle kodiranja. Ima više razloga za to.

Ovde analiziramo metodu "prvo-testiraj", tj. test-first programming, dakle metodu testiranja gde se pišu testovi na samom početku procesa programiranja, dakle unapred a ne unazad. Bolje je napisati test-slučajeve pre kodiranja nego posle kodiranja. Ima više razloga za to, npr:

- Otkrivaju se greške ranije, i lakše se ispravljaju ove greške
- Ako su zahtevi za softver loše definisani, kod pisanja test-slučajeva to se otkriva
- Pisanje test-slučajeva se povoljno odražava na dizajn i zahteve softvera, i onda se pravi bolji izvorni kod
- Isto je vreme potrebno za pisanje test-slučajeva, pre i posle kodiranja
- test-first programming je vrlo korisna praksa kod testiranja

SWEBOK: Test-first programming (tj. Test-Driven Development—TDD) je popularna razvojna metoda kod oje se test-slučajevi pišu pre pisanja izvornog koda programa. Metoda test-first programming omogućuje

- da se detektuju defekti ranije i lakše koriguju u poređenju sa tradicionalnim načinom, tj. "posle-testiraj".
- Takođe, ova metoda tera programere da razmišljaju o softverskim zahtevima i dizajnu softvera pre kodiranja, i na taj način ranije se otkrivaju problemi u zahtevima i dizajnu.

VIDEO VEŽBA

Video vežba - Šta je to TDD ? (What is Test Driven Development (TDD)? With an example)

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

ZADACI ZA VEŽBU

Zadaci

- Uraditi jedan primer primene metode test-first programming na primeru podprograma?
- Na izabranoj aplikaciji primeniti metodu test-first programming?

Poglavlje 4

Strukturno osnovno testiranje

STRUKTURNO OSNOVNO TESTIRANJE?

Ako je to logički iskaz, tj logiča instrukcija if ili while, onda je potrebno varirati testiranje tako da je kompletno istestiran taj iskaz

Structured basis testing: Ideja ovog prilaza je da svaku instrukciju u programu treba istestirati bar jednom. Ako je to logički iskaz, tj logiča instrukcija ***if*** ili while, onda je potrebno varirati testiranje tako da je kompletno istestiran taj iskaz. U nekim prilazima testiranju, testiraju se sve putanje u programu. To je slično sa prilazom strukturnog baznog testiranja, ali je potrebno dodati ideju o minimalnom skupu test-slučajeva.

Minimalan broj putanja tj test-slučajeva za strukturno bazno testiranje se može odrediti jednostavno na sledeći način:

startovati sa 1 za pravolinijsku putanju kroz podprogram

dodati 1 za svaki od sledećih ključnih reči, if, while, repeat, for, and, i or

dodati 1 za svaki slučaj u iskazucase**,** a ako nema slučaj **default** onda dodati još 1

PRIMER OSNOVNOG STRUKTURNOG TESTIRANJA

Svaka ključna reč u programu, if, while, repeat, for, and , i or, može biti ili istinita ili neistinita, i treba proveriti sve alternacije.

Evo jednog **primera** proračuna broja putanja kroz jedan program u Javi:

```
x=y;

x=x+5;

if (x<20) {

x=x+50;

}

x=x-50;
```

Kod ovog primera, 1 je za sam program, plus 1 za instrukciju if, što je ukupno 2. Znači, treba imati dva test-slučaja, jedan test-slučaj za x<20, jedan test-slučaj za x>20. U ovom primeru treba 2 test-slučaja. Medjutim, ne bilo koja 2 testa, ako se test-slučajevi ne izaberu pažljivo,

onda treba više testova od minimalnog broja. Svaka ključna reč u programu,if, while, repeat, for, and, i or,može biti ili istinita ili neistinita, i treba proveriti sve alternacije.

VIDEO VEŽBA

Video vežba -Tehnika testiranja "bele kutije" (What is White Box Testing)

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

ZADACI ZA VEŽBU

Zadaci

- Ilustrovati kroz primere metodu strukturnog osnovnog TESTIRANJA?

Poglavlje 5

Vežbe - Metode testiranja

ODREDITI BROJ PUTANJA U SLEDEĆEM PROGRAMU U JAVI:

Ovde posmatramo sledeći primer: Odrediti broj putanja u sledećem programu u Javi:

Primer: Odrediti broj putanja u sledećem programu u Javi:

```
total =0;
for(id=0; id<numE; id++) {if(m_emp[id].govRet<MAX_GOV){
govRet = ComGovRet(m_emp[id].);}
comRet = 0;
if(m_emp[id].wantsRet&&
Eligib(m_emp[ id ] ) ) {comRet = GetRet(m_emp[ id ]); }
gPay = ComGPay(m_emp[ id ]); perRet = 0;
if(EligPerRet(m_emp[ id ]) ){
perRet = PerRetCon(m_emp[ id ], comRet, gPay);}
With = ComWith(m_emp[ id ]);
nPay = gPay – with – comRet – govRet -perRet;
PayEmp(m_emp[ id ], nPay);
totalW = totalW + with;
totalGovRet = totalGovRet+govRet;
totalRet = totalRet+ comRet; }
PayRec(totalWith, totalGovRet, totRet);
```

REŠENJE ZA PRIMER PROGRAMA U JAVI

Daje se rešenje za test-slučajeve.

Odgovor:

1 za program, 1 za for, 1 za if, 1 za if, 1 za &&, 1 za if, ukupno 6.

Test - slučajevi bi bili:

svi bulovi uslovi su istiniti

prvi for-uslov je neistinit

prvi if-uslov je neistinit

drugi if-uslov je neistinit i prvi deo od AND je neistinit

drugi if-uslov je neistinit I drugi deo od od AND je neistinit

treći if'uslov je neistinit

VIDEO VEŽBA

Video vežba - Primer testiranja toka podataka (Data Flow Testing Part 2 Example)

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

ZADACI ZA SAMOSTALAN RAD

Zadaci za individualnu vežbu.

1. Za izabranu aplikaciju primeniti metodu strukturnog testiranja.

2. Za izabranu aplikaciju primeniti metodu testiranja toka podataka.

DOMAĆI ZADATAK 7

Domaći zadatak 7 se dobija direktno od asistenta nakon poslatog četvrtog domaćeg zadatka ili na poseban zahtev studenta.

Domaći zadatak 7 je individualan za svakog studenta, i dobija se po definisanim instrukcijama.
Domaći zadatak se imenuje:
KI301-DZ7-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-DZ7.
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

Testianje je vrlo važno za poboljšanje kvaliteta softvera.Testiranje se obavlja od strane razvojnih softverista, ali i od strane specijalizovanih test-specijalista. Ima različitih vrsta testiranja. Testiranje softverskih jedinica, gde se ispituje neka kompletirana klasa ili podprogram ili mali program, i to izolovano od ostalog dela sistema, pri čemu je ta jedinica napisana od strane jednog rogramera ili jednog tima programera. Testiranje softverskih komponenti, gde se ispituje neki element programa, npr. klasa, paket, manji program, i to izolovano od ostalog dela sistema, ali koji je razvijen od strane nekoliko programera ili nekoliko timova programera. Integracioni testovi je kombinovano ispitivanje dve ili više klasa, paketa, komponenti, podsistema, razvijen od strane nekoliko programera ili nekoliko timova programera. Sistemsko testiranje, gde se softver testira u finalnoj konfiguraciji, uključujući integraciju sa ostalim softverom i računarskom opremom
Testiranje u kompaniji koja razvija softver obuhvata testiranje softverskih jedinica, testiranje softverskih komponenti, i integracione testove. Sistemsko testiranje ide posle integracije. Ponekad i sistemsko testiranje se obavlja u kompaniji koja je razvila softver. Integracija tj. integraciono testiranje se radi u okviru testiranja, a posle „razvojnog“ testiranja. Dakle, „razvojno“ testiranje se sastoji os testiranja softvcerskih jedinica i testiranja softverskih komponenti.
Ovde se bavimo samo sa detekcijom greški, tj testiranjem. Jedan sistematski prilaz razvojnom testiranju maksimizira sposobnost nalaženja greški sa minimumom napora, i to:
testirati svaki relevantni softverski zahtev, da bi se proverila implementacija zahteva za softver
testirati svaki problem koji je uočen u fazi dizajna, i već tada u fazi dizajna napisati test-slučajeve
testirati svaku liniju programa pomoću tzv baznog testiranja
testirati tok-podataka
koristiti listu-provere gde se navode greške iz prošlosti, tj iz prethodnih projekata kao i iz prethodnih faza dotičnog projekta

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