Verifikacija softvera

— Dinamička analiza softvera —

Milena Vujošević Janičić

Matematički fakultet, Univerzitet u Beogradu

Sadržaj

Pregled

Značenja poglavlja

- Testiranje i razvoj softvera — koja je uloga i gde je mesto testiranja u procesu razvoja softvera?
- Vrste testiranja — šta tačno proveravamo?
- Tehnike testiranja — gde pronaći dobre test primere?
- Načini testiranja — manuelno ili automatsko testiranje?

Dinamička verifikacija softvera

Dinamička verifikacija softvera obuhvata ...
... tehnika ispitivanja ispravnosti koda u toku njegovog izvršavanja.

Najčešći vid verifikacije softvera je ...

... **testiranje**. Testiranje se često koristi kao sinonim za verifikaciju softvera. Testiranje se često koristi i kao sinonim za validaciju i verifikaciju softvera. Testiranje se često koristi i kao sinonim za brigu o kvalitetu softvera. **Testiranje je ipak samo važan deo verifikacije softvera, važan deo validacije softvera, a V&V je važan deo brige o kvalitetu softvera.**

Ali stalno treba imati u vidu...

Edsger Wybe Dijkstra (Tjuringova nagrada 1972)
"Program testing can show the presence of bugs, never their absenc

Testiranje ne može da dokaže ispravnost softvera...
Pravilnim i sistematičnim testiranjem podižemo nivo pouzdanosti i smanjujemo verovatnoću da greške promaknu. **Testiranje se ne radi nasumično, već je važno poznavati metodologiju, procese i principe testiranja.**

1 Testiranje i razvoj softvera

1.1 Cena grešake u kontekstu vremena otkrivanja

Testiranje i razvoj softvera

Razvoj softvera

- Softver se implementira prema zahtevima korisnika sa ciljem rešavanja realnog problema ili kreiranja potrebne funkcionalnosti.
- Testiranje predstavlja važan deo životnog ciklusa razvoja softvera.
- Svako ponašanje softvera koje se ne slaže sa zahtevima uzrokovano je nekakvom ljudskom greškom u razvoju softvera i predstavlja defekt koji čini softver manje kvalitetnim.
- U zavisnosti od namene softvera, smanjen kvalitet softvera može da ima najrazličitije posledice.

Testiranje i razvoj softvera

Metodologije razvoja softvera

- U okviru razvoja softvera, cilj je da se maksimizuje profit pravljenjem prozvoda visokog kvaliteta ali u vremenskim i budžetskim granicama.
- Najzastupljenije i trenutno najpopularnije metodologije razvoja softvera promovišu paralelnu implementaciju i pisanje testova za svaku od celina koja se razvija u okviru softverskog sistema

Testiranje i razvoj softvera

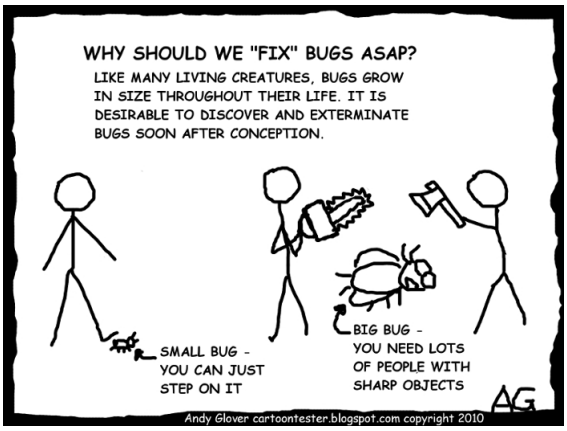
Greške

- Sa porastom složenosti projekta, raste i značaj testiranja i provera celokupnog softverskog sistema kako bi se izbegli ishodi koji mogu da unište ceo projekat.
- Kao što znamo, greške se ne mogu izbeći jer je to stvar ljudske prirode

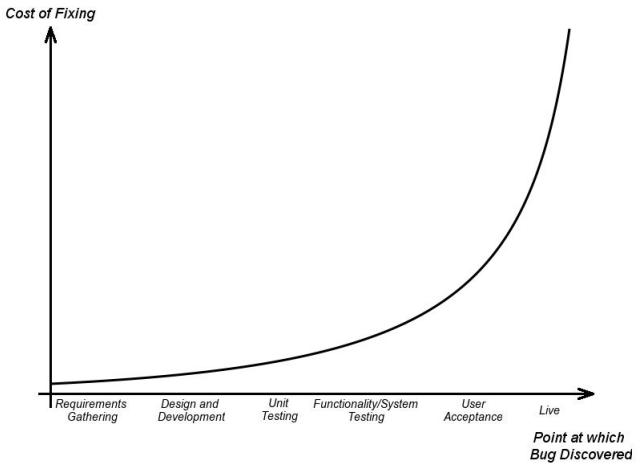
Vreme - cena

- Poželjno je sve greške detektovati što ranije u fazi razvoja softvera jer je ispravljanje grešaka jeftinije i brže u ranijim fazama razvoja softvera.

Cena grešake u kontekstu vremena otkrivanja



Cena grešake u kontekstu vremena otkrivanja



Cena grešake u kontekstu vremena otkrivanja

Faza analize zahteva
Cena greške = vreme potrebno da se utvrde i zapišu novi zahtevi

Faza kodiranja
Cena greške = dodatno vreme programera. Vreme može da varira u zavisnosti od kompleksnosti greške, ali je značajno manje nego kada se ispravlja greška koju pronađe neko drugi. Kada programer pronađe sam svoju grešku, on obično razume problem i zna kako da ga reši.

Cena grešake u kontekstu vremena otkrivanja

Faza integracije koda
Cena greške = dodatni rad programera i drugih inžinjera U ovoj fazi vreme za ispravljanje greške je obično dva puta duže jer kada se problem desi na višem nivou potrebno je da se najpre pronađe koji to tačno deo koda ili konfiguracija su bili pogrešni.

Cena grešake u kontekstu vremena otkrivanja

Sistemsko testiranje
Cena greške = dodatni rad programera i drugih inžinjera, dodatan rad program menadžera i QA tima U ovoj fazi greška zahteva da QA tester bude u stanju da reprodukuje i dokumentuje sve korake koji su potrebni da se opiše greška, QA tester treba da prijavi grešku, da joj da prioritet, da se sastane sa developerima (programerma) i da to prodiskutuje. Kada programeri isprave grešku, kôd mora ponovo da se integriše u testno okruženje, druge vrste testiranja treba da se ponovo odrade, i za tu grešku mora da se utvrdi da je stvarno popravljena. Defekat takođe mora da se isprati u okviru sistema za praćenje defekata.

Cena grešake u kontekstu vremena otkrivanja

Testiranje prihvatljivosti
Cena greške = dodatni rad programera i drugih inžinjera, dodatan rad program menadžera, QA tima i kupca/korisnika Ovo zahteva komunikaciju između testra za proveru pruhvatljivosti sa testerima sistemskog testiranja. Tester sistemskog testiranja će pokušati da reprodukuje grešku i da utvrdi da li je to greška ili sistem radi u skladu sa dizajnom. Ako reprodukovanje greške nije moguće, onda se nastavlja komunikacija sa testerom provere prihvatljivosti. Ako reprodukovanje greške jeste moguće, onda je potrebno da se dokumentuju koraci reprodukovanja greške i da se prođu sve faze kao i kod pronalaženja greške u prethodnoj fazi. Kada se greška ispravi, kôd mora ponovo da se uvede u okruženje za testiranje prihvatljivosti tako a korisnik može da nastavi testiranje.

Cena grešake u kontekstu vremena otkrivanja

Program u upotrebi
Cena greške = dodatni rad programera i drugih inžinjera, dodatan rad program menadžera, QA tima i kupca/korisnika Prijava greške prati sličan postupak kao kod testiranja prihvatljivosti, ali su posledice značajno veće, a ispravljen kôd treba da se isporuči svima

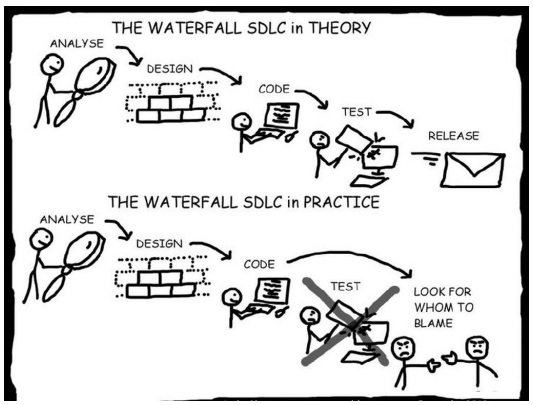
1.2 Uloga testera u razvoju softvera

Uloga testera u razvoju softvera

Projekti se vode na najrazličitije načine

- Briga o kvalitetu softvera obuhvata razna pravila i procedure, koji su različiti u zavisnosti od vrste projekata koje firma radi, ali i od zrelosti i veličine firme
- U zavisnosti od firme i projekta, QA tim može i ne mora da postoji

Koga okriviti?



Uloga testera u razvoju softvera

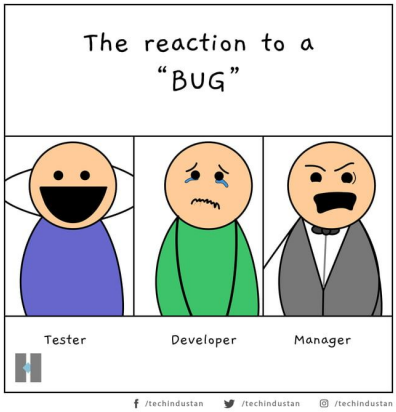
Moguće su različite raspodele obaveza i timova:

- programeri su istovremeno i testeri
- programeri i testeri rade zajedno
- programeri i testeri su potpuno razdvojeni (nekada čak i fizički na različitim lokacijama)

Česti problemi na relaciji programer-tester

- Loša komunikacija
- Međusobno nerazumevanje
- Netrpeljivost

Osnovni razlog neslaganja



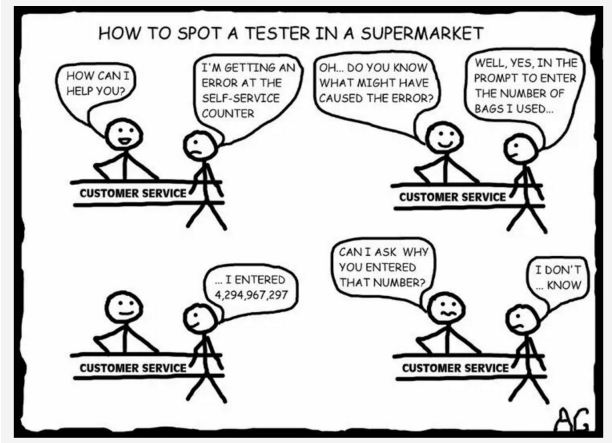
Osobine testera

Dobar tester poseduje ...

- osnovno razumevanje programiranja i procesa razvoja softvera,
- poznavanje procedura i procesa testiranja,
- poznavanje alata i skript jezika,
- poznavanje čestih grešaka i propusta, kao i nesvakidašnjih slučajeva upotrebe,
- kreativnost i potrebu za stalnim usavršavanjem
- ...

Formalna edukacija
Ne postoji škola za testiranje, niti smer za testiranje. Knjige, kursevi i sertifikati.

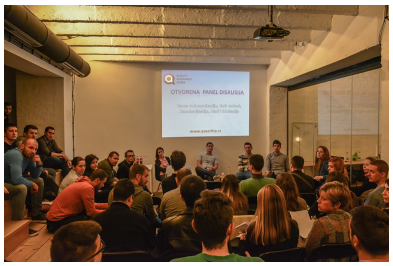
Tester u supermarketu



Važno je pratiti trendove i najnovija saznanja

Neformalna edukacija

- Meetup-ovi — QA Serbia http://www.qaserbia.rs/
- Konferencije — Belgrade Test Conference https://bg-testconference.rs/



1.3 Faze testiranja softvera

Faze testiranja softvera

Ulazni kriterijumi
Za početak procesa testiranja, postojanje koda nije neophodno. Dovoljno je imati samo jasno definisane zahteve korisnika jer testiranje počinje analizom tih zahteva. Tj, da bi započeli testiranje, potrebno je da postoji specifikacija zahteva sistema (eng. System Requirements Specification) koji treba da se izgradi kao i specifikacija zahteva softvera koji treba da se izgradi (eng. Software Requirements Specification).

Test slučaj (engl. *test case*) ...
... je dokument koji definiše ulaze u sistem i očekivane izlaze za te ulaze.

Faze testiranja softvera

Osnovne faze testiranja
Testiranje softvera se u opštem slučaju sastoji od četiri faze pri čemu svaka faza obuhvata veliki broj aktivnosti.

- Planiranje
- Analiza, dizajn i implementacija testova
- Izvršavanje
- Evaluacija testova

Planiranje

Planiranje (eng. *Test planning*)
... predstavlja pripremu za ceo proces testiranja i uključuje definisanje zadataka koje je potrebno sprovesti kao i način njihovog izvršavanja.

Planiranje definiše:

- vrste testova koje će biti sprovedene
- opseg testiranja, pristup, strategije i metode testiranja
- kriterijum završetka
- potrebne resurse i dogovara način komunikacije između članova tima

Planiranje

Plan testiranja ...
... zavisi od primenjene metodologije razvoja softvera i prilagođava se svakom konkretnom projektu.

Rezultat planiranja je skup dokumenata ...
... koji sadrže opšti pogled na sistem koji će se testirati, aktivnosti koje će biti izvršene kao i alate koji će biti korišćeni.

Analiza, dizajn i implementacija testova

Analiza, dizajn i implementacija testova (eng. *Test analysis, design and implementation*)...
... obuhvata pravljenje detaljne specifikacije načina na koje će se aktivnosti predviđene planom izvršiti.

Analiza, dizajn i implementacija testova

- Ispituje se mogućnost testiranja određenih delova koda, prikupljaju potrebni podaci i preciziraju zahtevi korisnika.
- Kreiraju se i precizna uputstva kako će se vršiti testiranje sistema
- Rezultat ove faze je skup test slučajeva i test procedura koja će biti korišćene u fazi izvršavanja testova.
- Samo izvršavanje može biti ručno i automatsko

Izvršavanje testova

Izvršavanje testova (eng. *Test execution*) ...
... je proces konkretne primene test slučajeva i test procedura formiranih na osnovu plana, analize, dizajna i implementacije. Ova faza podrazumeva i određivanje prioriteta izvršavanja testova, pripremu testova za automatizovano testiranje (ukoliko je ono deo procesa testiranja) i organizaciju testova za što efikasnije izvršavanje.

Izvršavanje testova

Izvršavanje testova ...

- ... se vrši radi provere funkcionalnosti sistema.
- ... obuhvata i dodatnu aktivnost praćenja statusa problema. Ova aktivnost podrazumeva eliminaciju prijavljenih problema kao i potvrđivanje da je problem rešen.
- ... ponovno izvršavanje testova posle popravke grešaka
- Komunikacija tester - programer

Evaluacija testova

Evaluacija testova (eng. *Test evaluation*) ...
... obuhvata procenu kriterijuma završetka testiranja i izveštavanje. Svaka izmena u kodu, čak i koja podrazumeva popravljanje grešaka, može da dovede do novih grešaka. Iz tog razloga se, za različite oblasti testiranja, definiše kriterijum završetka testiranja u odnosu na rezultate izvršavanja test skritpova, procenta nerešenih bagova ili preostalog vremena za testiranje. Proces evaluacija uključuje i pregled rezultata dobijenih analizom izlaza test slučajeva.

Kreiranje izveštaja (eng. Test Summary Report) ...
... je opis šta je testirano. Na osnovu ovog izveštaje se utvrđuje da li je implementirani sistem spreman za korišćenje u skladu sa korisničkim zahtevima.

Evaluacija testova

Izlazni kriterijumi (eng. Exit Criteria) ...
... određuju da li je testiranje kompletirano i da li je aplikacija spremna za korišćenje u skladu sa korisničkim zahtevima. Uključuju *Test Summary Report*, izračunavanje raznih metrika i izveštaj o defektima (eng. Defect Analysis Report).

Aktivnosti zatvaranja testiranja
Testiranje se zatvara kada je softver isporučen korisniku, mada može se desiti i u nekim drugim situacijama, na primer, kada je projekat otkazan ili je neki cilj postignut. Tokom ove faze, test skriptovi i dokumentacija se arhiviraju, dok se primenjeni proces testiranja analizira i diskutuje o tome šta je bilo dobro, a šta ne.

2 Vrste testiranja

Vrste testiranja — šta se proverava

Različite podele testiranja softvera

- Funkcionalno testiranje (funkcionalnost aplikacije) — obuhvata testiranje na različitim nivoima, testove prihvatljivosti, regresione testove, istraživačke testove...

- Nefunkcionalno testiranje (neophodni tehnički kvaliteti aplikacije)

Podela po nivoima testiranja
Mogu se testirati pojedinačni moduli, grupe modula (vezanih namenom, upotrebom, ponašanjem ili strukturom) ili ceo sistem. U skladu sa pomenutom podelom, prema nivou testiranja, razlikujemo testove jedinice koda, komponentne, integracione i sistemske testove.

2.1 Testiranje jedinica koda

Testiranje jedinica koda

Testiranje jedinica koda (eng. *Unit testing*)

- Proverava se funkcionisanje delova sistema koji se nezavisno mogu testirati.
- U zavisnosti od konteksta i programske paradigme, to mogu biti podprogrami, klase, manje ili veće celine formirane od tesno povezanih jedinica.
- Ovom vrstom testiranja prolazi se svaki i najmanji deo sistema.
- Jedinični testovi definisani su standardom *IEEE Standard for Software Unit Testing*.
- Dobra podrška u alatima za automatsko izvršavanje i proveravanje ovih testova (sastavni deo razvojnog okruženja)

Testiranje jedinica koda

Testiranje jedinica koda

- Cilj jediničnih testova je utvrđivanje da li izolovani delovi koda imaju predviđenu funkcionalnost.
- Ukoliko kôd komunicira sa mrežom, bazom podataka ili fajl sistemom, to se u okviru testova apstrahuje u nekakve fiksirane vrednosti. Ukoliko kôd komunicira sa drugim klasama, modulima ili komponentama sistema, i to se apstrahuje. Dozvoljena je samo direktna komunikacija sa memorijom.
- Testiranje metodama bele kutije
- Testove piše programer
- Ukoliko postoje greške unutar jedinice koda, one bi trebalo da budu otkrivene u ovoj fazi.

2.2 Komponentno i integraciono testiranje

Komponentno testiranje

Komponentno testiranje (eng. *Component testing*)

- ... proverava komponente sastavljene od više jedinica koda.
- Jedinice koda koje su proverene da ispravno rade u izolaciji, sada se testiraju na nivou komponente i proverava se komunikacija između jedinica koda.
- Neformalno, komponenta je skup povezanih jedinica koda koje imaju zajednički interfejs prema ostalim komponentama.
- Komponente se proveravaju odmah po njihovom kreiranju pri čemu se testiranje može vršiti izolovano od ostatka sistema, u zavisnosti od izabranog modela razvoja.

Komponentno testiranje

Komponentno testiranje

- Komponentno testiranje je veoma slično sa testiranjem jedinica koda, samo što su komponente malo veće
- Sama jedinica koda je u prethodnoj fazi testiranja izolovana u potpunosti od spoljašnjeg sistema, dok je u okviru komponentnog testiranja sada napuštena izolacija na nivou same komponente, ali i dalje ostaje izolacija u okviru povezanosti sa drugim komponentama i spoljašnjim sistemom.
- S obzirom da se u komponentnom testiranju integrišu osnovne jedinice koda, ovo je vrsta integracionog testiranja, ali na nižem nivou.
- Komponentno testiranje može raditi programer a može i tester, u zavisnosti od vrste projekta

Integraciono testiranje

Integraciono testiranje (eng. *Integration testing*)

- ... proverava saradnju između komponenti koji predstavljaju jednu celinu sistema.
- Ispituje se da li su veze između komponenti dobro definisane i realizovane, tj. da li komponente komuniciraju na način opisan u specifikaciji projekta.
- Tokom integracionog testiranja mogu se naći propusti u komunikaciji između komponenti
- Integracionim testovima proverava se da različite komponente sistema rade ispravno zajedno. https://www.youtube.com/watch?v=0GypdsJulKE
- Testiranje metodama crne kutije
- Testiranje koje rade testeri

2.3 Sistemsko testiranje

Sistemsko testiranje

Sistemsko testiranje (eng. *System testing*) ...

- ... obuhvata proveravanje sistema kao celine.
- Ispituje se da li je ponašanje sistema u skladu sa specifikacijom zadatom od strane klijenta.
- Ovde se zahteva i potpun pristup bazi i hardverskim delovima sistema.
- Sistemsko testiranje može da uključuje i **funkcionalne** i **nefunkcionalne** aspekte sistema
- U sistemsko testiranje nekada se ubrajaju **istraživačko testiranje** i **testiranje prihvatljivosti**, a nekada se ove dve vrste testiranja izdvajaju nezavisno.

Istraživačko testiranje

Istraživačko testiranje (eng. *Exploratory testing*)

- Tokom istraživačkog testiranja testeri pronalaze i proveravaju druge eventualne pravce korišćenja softverskog sistema.
- Ovo podrazumeva istaknutu kreativnost testera. Ova vrsta testiranja obuhvata aktivnosti prepoznavanja, kreiranja i izvršavanja novih test slučajeva (onih koji nisu bili predviđeni test planom).
- Istraživačko testiranje uglavnom ima smisla kada je aplikacija u svom finalnom obliku, kada tester može videti i druge alternativne pravce korišćenja sistema koji ranije nisu mogli biti predmet testiranja.
- Ukoliko se ova faza testiranja preskoči, postoji opasnost da neke funkcionalnosti sistema ne budu pokrivene testovima.

Testovi prihvatljivosti

Testovi prihvatljivosti (eng. *Acceptance testing*)

- ... treba da omoguće klijentima i korisnicima da se sami uvere da je napravljeni softver u skladu sa njihovim potrebama i očekivanjima.
- Validacija softvera
- Ovu vrstu testiranja izvode i procenjuju korisnici, a razvojni tim im pruža pomoć oko tehničkih pitanja, ukoliko za tim ima potrebe.
- Klijent može da proceni sistem na tri načina: **referentnim** testiranjem, **pilot** testiranjem i **paralelnim** testiranjem.

Testovi prihvatljivosti

Referentno testiranje
Kod referentnog testiranja, klijent generiše test slučajeve koji predstavljaju uobičajne uslove u kojima sistem treba da radi. Ove testove izvode korisnici kako bi procenili da li je softver implementiran u skladu sa očekivanjima.

Pilot testiranje
Pilot testiranje podrazumeva instalaciju sistema na privremenoj lokaciji i njegovu upotrebu. U ovom slučaju, testiranje se vrši simulacijom svakodnevnog rada na sistemu.

Testovi prihvatljivosti

Paralelno testiranje
Paralelno testiranje se koristi tokom razvoja, kada jedna verzija softvera zamenjuje drugu ili kada novi sistem treba da zameni stari. Ideja je paralelno funkcionisanje oba sistema (starog i novog) čime se korisnici postepeno privikavaju i prelaze na korišćenje novog sistema.

Nefunkcionalno testiranje

Nefunkcionalni zahtevi sistema
Brzina, efikasnost, otpornost na otkaze, uklapanje u okruženje u kojem će se sistem koristiti.

Testiranje performansi
Tokom testiranja performansi, izvršavaju se testovi **konfiguracije**, **kapaciteta** i **kompatibilnosti**.

Testovima konfiguracije ...
... ispituje se ponašanje sistema u različitim hardverskim i softverskim okruženjima. Različite konfiguracije namenjene su različitim korisnicima sistema. Ovim testovima proveravaju se sve konfiguracije sistema.

Nefunkcionalno testiranje

Testovima kapaciteta ...
... proverava se ponašanje sistema pri obradama velikih količina podataka. Proverava se i ponašanje sistema u slučaju kada skupovi podataka postignu svoje maksimalne kapacitete.

Testovima kompatibilnosti ...
... proverava se način ostvarivanja komunikacije sistema sa drugim spoljnim sistemima.

Regresiono testiranje ...

Regresiono testiranje ...
... se radi nakon izmena u razvoju sistema, da bi se utvrdilo da nije došlo do lošeg rada nekih funkcija koje nisu bile obuhvaćene izmenama. Regresioni testovi garantuju da su performanse novog sistema barem jednake performansama starog. Regresioni testovi se sprovode i za testiranje funkcionalnih osobina softvera (tj ne mora da bude vezano samo za performanse).

Testovi bezbednosti

Testovi bezbednosti

- Pri testiranju važna karakteristika je bezbednost sistema.
- Testovima bezbednosti proverava se da li su određene funkcionalnosti dostupne isključivo onim korisnicima kojima su namenjene.
- Proveravaju se i dostupnost, integritet i poverljivost svih skupova podataka.

Instalaciono testiranje

Instalaciono testiranje

- Ova vrsta testiranja izvodi se instaliranjem softvera na klijentskoj mašini.
- Prilikom instaliranja, sistem se konfiguriše u skladu sa okruženjem.
- Ukoliko je potrebno, sistem se povezuje sa spoljnim uređajima i sa njima uspostavlja komunikaciju.
- Instalacioni testovi se izvršavaju u saradnji sa korisnicima.
- Ispituje se da li uslovi na klijentskoj mašini i okruženju negativno utiču na neke funkcionalne ili nefunkcionalne osobine sistema.
- Kada rezultati testiranja zadovoljavaju potrebe klijenta, testiranje se prekida i sistem se formalno isporučuje.

3 Tehnike testiranja

Strategije testiranja

Plan testiranja ...
... nastaje u fazi planiranja testiranja i to je dokument koji sadrži informacije o fokusu i obimu testiranja, definiše kriterijum pokrivenosti, rasporede testiranja, osobine koje se testiraju i procenjuje potrebne resurse.

Prilikom planiranja testiranja, potrebno je definisati pristup testiranju koji daje odgovor šta želimo da testiramo i kako želimo to da ostvarimo.

Strategije testiranja

Strategija testiranja ...
... je vodič koji se prati za postizanje ciljeva testiranja i izvršavanje testova koji se pominju u planu testiranja. U okviru strategije testiranja, definišu se ciljevi, okruženja, pristupi, automatizacija i **tehnike**, nepredviđene situacije i analiza rizika.

Definisanje strategije testiranja je najvažniji dokument vezan za testiranje.

Odnos
Ako je plan testiranja destinacija, onda je strategija testiranja uputstvo, mapa, kako da se na tu destinaciju stigne.

Kako pronaći dobre test primere?

Strategije testiranja
Metod određivanja reprezentativnog skupa podataka nad kojima će se vršiti testiranje:

- Visok potencijal otkrivanja grešaka
- Relativno mala veličina
- Relativno velika brzina izvršavanja
- Visok stepen poverenja u pouzdanost softvera

Pokrivenost koda (engl. *coverage*)
Značenje pojma pokrivenosti zavisi od konteksta u kojem se javlja. Uopšteno, pokrivenost je broj nekih elemenata programa (engl. *items*) ispitanih testovima u odnosu na ukupan broj tih elemenata.

Gde pronaći dobre test primere?

Specifikacija programa
Testiranje crne kutije (engl. *black box testing*) — generisanje test primera bez razmatranja interne strukture koda već isključivo na osnovu specifikacije. Ovakav način testiranja se fokusira na ponašanje sistema, posmatrano iz korisničkog ugla. Zadatak testera je da sistemu pruži ulaze, a zatim da proveri izlaze u odnosu na datu specifikaciju.

Drugi nazivi su i ...
... funkcionalno testiranje (engl. *functional testing*), testiranje ponašanja (engl. *behavioural testing*), testiranje vođeno podacima (engl. *data driven testing*).

Gde pronaći dobre test primere?

Kôd programa
Testiranje bele kutije (engl. *white box testing*) — generisanje test primera na osnovu interne strukture koda, npr jedinični testovi. Kriterijum pokrivenosti koda: broj izvršenih putanja, broj izvršenih instrukcija, broj izvršenih grana...

Drugi nazivi su i ...
... strukturno testiranje (engl. *structural testing*), testiranje vođeno logikom (engl. *logic driven testing*).

Gde pronaći dobre test primere?

Specifikacija i kôd programa
Testiranje sive kutije (engl. *gray box testing*)— mešovita strategija Predstavlja sredinu između modela crne i bele kutije. Kod tehnika ovog modela postoji neki uvid u unutrašnju strukturu sistema, ali ne u toj meri kao kod modela bele kutije. Koristiti se kod komponentnog i integracionog testairanja Tehnika koju koriste i programeri i testeri

Odnos
Testiranje crne kutije je testiranje iz ugla korisnika dok je testiranje bele kutije testiranje iz ugla programera.

3.1 Testiranje crne kutije

Testiranje crne kutije

Fokus na ulazu i izlazu

Testiranje crne kutije

Isprobavanja svih mogućih ulaza (engl. *exhaustive input testing*)
Za netrivijalne programe nije moguće koristiti ovu tehniku.

$a \cdot x^2 + b \cdot x + c = 0$
Rešenje

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Ako je svaka promenljiva tipa int32, broj različitih test primera za potpuno testiranje je

$$2^{32} \cdot 2^{32} \cdot 2^{32} = 2^{96}$$

Testiranje crne kutije

Metod crne kutije

- Cilj tehnika ovog modela da pronađu prihvatljiv broj test slučajeva (tj. kombinacija ulaza) koji će otkriti što više grešaka.
- Takav cilj se ostvaruje postavljanjem nekih pretpostavki o ispitivanom softveru.
- Prednost ovog pristupa jeste mogućnost potpunog razdvajanja programera i testera.

Metod klasa ekvivalencije

Testiranje pomoću klasa ekvivalencije (engl. *equivalence class testing*)...

- ... je tehnika koja se koristi da smanji broj test slučajeva na prihvatljiv nivo, pritom održavajući razumnu pokrivenost testovima. Ovde se pokrivenost odnosi na procenat svih mogućih ulaza koji će biti ispitan testovima.
- Ovu jednostavnu tehniku koriste intuitivno skoro svi testeri, iako oni možda nisu svesni da je to formalna metoda oblikovanja testova.
- Klasa ekvivalencije predstavlja skup podataka koji se tretiraju jednako od strane modula ili koji treba da proizvedu isti rezultat.

Metod klasa ekvivalencije

Klase ekvivalencije

- Iz tačke gledišta testiranja, sve vrednosti podataka u okviru jedne klase su *ekvivalentne* svim ostalim vrednostima u okviru te klase. Konkretno, očekujemo da:
  - Ako jedan test slučaj u jednoj klasi ekvivalencije detektuje grešku, *svi* ostali test slučajevi u okviru iste klase ekvivalencije će verovatno detektovati istu grešku.
  - Ako jedan test slučaj u jednoj klasi ekvivalencije ne detektuje grešku, *nijedan* drugi test slučaj u okviru iste klase ekvivalencije verovatno neće detektovati grešku.
- Koraci za korišćenje testiranja pomoću klasa ekvivalencije su jednostavni. Prvo, identifikujemo klase ekvivalencije. Zatim, pravimo test slučaj za svaku klasu ekvivalencije.

Primer

Tabela 1: Pravila organizacije pri zapošljavanju

Godine	**Pravilo**
0-16	Ne zaposliti
16-18	Može se zaposliti samo sa pola radnog vremena
18-55	Može se zaposliti sa punim radnim vremenom
55-99	Ne zaposliti

Broj validnih test slučajeva može se smanjiti sa 100 (testiranje za svaku godinu starosti) na 4 (testiranje jedne godine starosti za svaku klasu ekvivalencije, npr 10, 17, 30, 70). Nevalidni test slučajevi -5, 105.

Metod klasa ekvivalencije

Input or Output Event	Valid Equivalence Classes	Invalid Equivalence Classes
Enter a number	1~99	> 99 0 Negative numbers An expression that yields an invalid number, such as 5 – 5, which yields 0 Letters and other non-numeric characters

Često je lakše identifikovati ispravne klase ekvivalencije od neispravnih.

Metod graničnih vrednosti

Testiranje graničnih vrednosti (engl. *boundary value testing*)

- Testiranje pomoću klasa ekvivalencije je najosnovnija tehnika oblikovanja testova i ona nas vodi do ideje o testiranju graničnih vrednosti.
- Testiranje graničnih vrednosti se fokusira na granice zato što se tu krije mnogo grešaka.
- Greška koju programeri često čine je pogrešno kodiranje testova nejednakosti. Primer toga je pisanje $>$ znaka umesto $\geq$ znaka.

Metod graničnih vrednosti

Testiranje graničnih vrednosti
Koraci za korišćenje testiranja graničnih vrednosti su jednostavni.

- Identifikujemo klase ekvivalencije.
- Identifikujemo granice svake klase ekvivalencije.
- Pravimo test slučaj za svaku graničnu vrednost birajući jednu tačku na granici, jednu tačku ispod granice i jednu tačku iznad granice. *Ispod* i *iznad* su relativni termini i zavise od jedinica vrednosti podataka.
- Tačke ispod i iznad granice mogu biti u drugim klasama ekvivalencije i treba voditi računa da se testovi ne dupliraju.
- Više dimenzija u klasama ekvivalencije, realni brojevi

Primer

Tabela 2: Pravila organizacije pri zapošljavanju

Godine	**Pravilo**
0-16	Ne zaposliti
16-18	Može se zaposliti samo sa pola radnog vremena
18-55	Može se zaposliti sa punim radnim vremenom
55-99	Ne zaposliti

Primećuje se problem na granicama svake klase. Starost *16* je uključena u dve različite klase ekvivalencije (kao što su i 18 i 55). Prvo pravilo kaže da ne zapošljavamo osobe sa 16 godina. Drugo pravilo kaže da se osobe sa 16 godina mogu zaposliti sa pola radnog vremena. Ovo je greška u specifikaciji sistema.

Primer

Validni test slučajevi u ovom primeru su naredne vrednosti na granici ili blizu granice: {-1, 0, 1}; {15, 16, 17}; {18, 19}; {54, 55, 56}; {98, 99, 100}.

Tabela 3: Ispravljena pravila organizacije pri zapošljavanju

Godine	Pravilo
0-15	Ne zaposliti
16-17	Može se zaposliti samo sa pola radnog vremenom
18-54	Može se zaposliti sa punim radnim vremenom
55-99	Ne zaposliti

Tabele odlučivanja

Tabele odlučivanja (engl. *decision table*)

- Izrada tabela odlučivanja je tehnika za prikaz složenih poslovnih pravila u lako čitljivom obliku pomoću koje se mogu napraviti i test slučajevi.
- Prvu grupu redova tabele čine uslovi nad ulazom, a drugu moguće akcije.
- Kolone tabele predstavljaju pravila koja jedinstvenoj kombinaciji uslova dodeljuju odgovarajuće akcije.

Tabele odlučivanja

Tabele odlučivanja

- Uslovi pravila mogu biti binarni ili sa više od dve vrednosti. Iz prvih se može direktno izvesti tačno jedan test slučaj, dok iz drugih može više njih.
- Izbor različitih test slučajeva iz jednog pravila može se vršiti u kombinaciji sa drugim tehnikama, kao što su klase ekvivalencije ili granične vrednosti
- Kada imamo više pravila kod kojih akcija ne zavisi od vrednosti nekog uslova možemo ih spojiti u jedno pravilo (engl. *table collapsing*). Takav uslov u novom pravilu označavamo sa '-' i nazivamo nebitnim (engl. *don't care*).

Primer

Banka

- Klijent zahteva isplatu gotovine na bankomatu neke banke. Sistem treba da odluči da li će da odobri isplatu.
- Odlučivanje vrši pomoću podataka o sredstvima na računu i o dozvoljenom minusu.
- Način odlučivanja je prikazan tabelom odlučivanja

	Pravilo 1	Pravilo 2	Pravilo 3
Uslovi			
Dovoljno sredstava na računu	Da	Ne	Ne
Dozvoljen minus	-	Da	Ne
Akcije			
Isplata odobrena	Da	Da	Ne

Primer

Tabela odlučivanja za zahtev isplate gotovine

- Prvo pravilo je dobijeno spajanjem dva pravila. Kada ima dovoljno sredstava na računu, nezavisno od toga da li je minus dozvoljen ili ne, isplata se odobrava.
- Iz drugog pravila može se izvesti test slučaj tako što se za ulaz uzme da korisnik nema dovoljno sredstava na računu i da mu je dozvoljen minus. Zatim se izlaz iz programa poredi sa očekivanom akcijom, a to je da je isplata odobrena.

Dijagrami stanja

Dijagram stanja (engl. *state-transition diagram*) ...
... kompaktno opisuje kompleksne zahteve sistema i njegov način interakcije sa spoljašnjim svetom. Primenjuje se kod sistema čije akcije zavise od akcija izvršenih u prošlosti i koji reaguju na spoljašnje događaje.

Dijagrami stanja

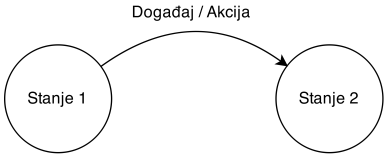
Osnovnu strukturu dijagrama čine:

Stanje Čuva znanje o prošlim događajima i definiše reakciju na buduće.

Prelaz Promena iz jednog stanja u drugo.

Događaj Nešto izvan sistema što preko interfejsa izaziva prelaz.

Akcija Operacija sistema izazvana prelazom.



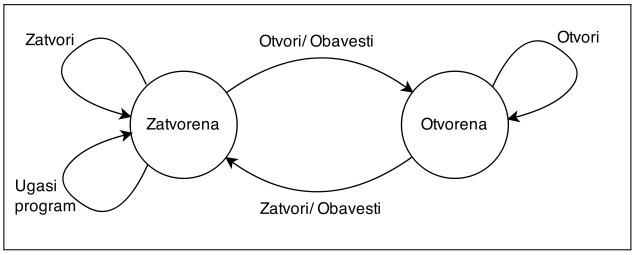
Dijagrami stanja

Dijagrami stanja

- U svakom trenutku sistem se nalazi u nekom od konačno mnogo stanja i čeka na neki događaj.
- Kombinacija stanja i događaja određuje stanje u koje sistem prelazi.
- Pri prelasku sistem može da izvrši još neku akciju, obično pravljenje nekih izlaza.
- Ovakav sistem se može modelovati konačnim automatom (engl. *finite state machine*).
- Dijagram stanja je jedan od načina prikaza takvog modela.

Primer

Posmatramo softverski sistem za maloprodaju koji ima ugrađenu opciju za otvaranje i zatvaranje (fioke) kase.



Dijagrami stanja

Generisanje test slučajeva

- Test slučajeve možemo da pravimo obilaskom, jer dijagram stanja predstavlja vrstu usmerenog grafa.
- Pri pravljenju skupova test slučajeva možemo zahtevati različite nivoe pokrivenosti, pri čemu se pravi kompromis između pokrivenosti i količine testova.
- Primer dobrog komprimisa je skup testova koji omogućava da se svaki prelaz ispita bar jednom.
- Pored toga, možemo zahtevati da se svako stanje ili svaka putanja kroz dijagram obiđu bar jednom.

Primer

Test slučajevi

- Otvori, zatvori, ugasi program
- Jedan primer test slučaja koji aktivira svaki prelaz datog dijagrama bar jednom dat je sledećim nizom naredbi: otvori, otvori, zatvori, zatvori, ugasi program.
- Pri izvršavanju navedenih naredbi proverava se da li sistem reaguje u skladu sa zadatim zahtevima.

Tabele stanja

Tabele stanja

- Konačni automat koji modeluje sistem se može prikazati i tabelama stanja (engl. *state transition tables*).
- Osnovna prednost tabela stanja jeste njihov sistematični pristup, prikazuju sve moguće kombinacije stanja i događaja. Takvim pristupom mogu da se uoče situacije u kojima ponašanje sistema nije definisano, što može da spreči pojavu grešaka.
- Kod tabela stanja, iz svakog reda se može direktno izvesti jedan test slučaj.

Primer

Tabela stanja koja odgovara prikazanom dijagramu stanja

Trenutno stanje	**Događaj**	**Akcija**	**Naredno stanje**
Zatvorena	otvori	obavesti	Otvorena
Zatvorena	zatvori	-	Zatvorena
Zatvorena	ugasi program	-	Zatvorena
Otvorena	otvori	-	Otvorena
Otvorena	zatvori	obavesti	Zatvorena
Otvorena	ugasi program	-	Nedefinisano

Primer

Test slučajevi

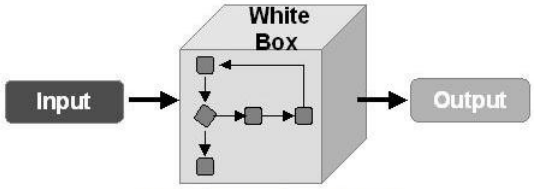
- Problematičan događaj nastaje kada korisnik sistema želi da ugasi program. Opasnost od ostavljanja otvorene kase nije navedena u dijagramu stanja, ali jeste u tabeli stanja.
- Moguća rešenja su da sistem upozori korisnika i spreči zatvaranje programa ili da automatski zatvori kasu.

Pogađanje grešaka

Pogađanje grešaka (engl. *error guessing*) ...
Ova tehnika se oslanja na iskustvo i procenu testera. To je umetnost pogađanja gde bi greška mogla da bude skrivena. Za ovu tehniku ne postoje specifični alati niti uputstva.

3.2 Testiranje bele kutije

Testiranje bele kutije



Testiranje metodama bele kutije

Testiranje bele kutije (eng. white box testing)

- Ovakav način testiranja podrazumeva znanje o unutrašnjoj strukturi softvera.
- Testove najčešće piše programer, ali može i tester
- Programer/tester kreira test slučajeve na osnovu izučavanja implementacije
- Ova vrsta testiranja je skupa i sprovodi se obično za sisteme kod kojih su greške skupe

Testiranje metodama bele kutije

Testiranje bele kutije

- Ovom vrstom testiranja ispituju se različite putanje kroz program
- Koristi se najčešće za pisanje jediničnih testova, ali može i za integraciono i sistemsko testiranje
- Mogu se testirati putanje kroz jedinicu koda, putanje između različitih jedinica koda za vreme integracije, i putanje između podsistema za vreme sistemskog testiranja.

Osnovni koraci

Osnovni koraci

1. Razumevanje koda
2. Pisanje testova i njihovo izvršavanje

Zbog prvog koraka, najčešće ovu vrstu testiranja sprovode sami programeri

Testiranje bele kutije

Tehnike testiranja bele kutije

- Analogno ispitivanju svih kombinacija ulaza kod tehnika zasnovanih na modelu crne kutije, može se zahtevati ispitivanje svih putanja kroz program.
- Međutim, takav pristup je nepraktičan, a često i nemoguć zbog prevelikog broja mogućih putanja kroz program
- Zbog toga tehnike nastoje da omoguće kreiranje praktično prihvatljivog broja test slučajeva, ali i da obezbede visok nivo pokrivenosti.
- Pre početka testiranja, odgovarajući nivo pokrivenosti treba biti izabran.

Pokrivenost

Pokrivenost

Pokrivenost putanja (Path Coverage) Mera prolaska kroz moguće putanje (potpuna pokrivenost: sve moguće putanje programa su izvršene bar jednom)

Pokrivenost naredbi (Statement Coverage) Mera izvršavanja naredbi programa (potpuna pokrivenost: svaka naredba programa je izvršena bar jednom)

Pokrivenost grana/odluka (Branch/Decision Coverage) Mera prolaska kroz grane programa (potpuna pokrivenost: svaka odluka u programu je doneta bar jednom)

```
Path Coverage = (Number of paths exercised / Total Number of paths in the program) x 100 %
Statement Coverage = (Number of Statements Exercised / Total Number of Statements) x 100 %
Decision Coverage = (Number of decisions outcomes tested / Total Number of decision Outcomes) x 100 %
```

Pokrivenost

Pokrivenost

Pokrivenost uslova (Condition Coverage) Mera ispitivanja uslova programa (potpuna pokrivenost: svaki uslov u svakoj odluci je uzeo sve moguće vrednosti bar jednom)

Pokrivenost višestrukih uslova (Multiple Condition Coverage) Mera ispitivanja višestrukih uslova programa (potpuna pokrivenost: svaka moguća kombinacija uslova u svakoj odluci je ispitana bar jednom)

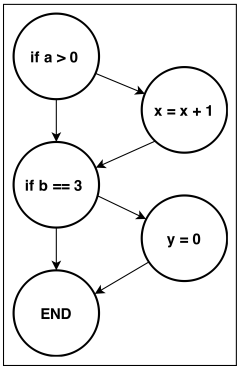
Pokrivenost funkcija (Function Coverage) Mera poziva svih funkcija programa

... ...

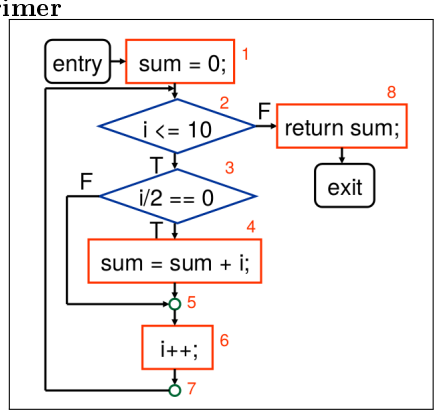
Primer

```
if (a > 0) { x = x + 1; }
if (b = = 3) { y = 0; }
```

Test a=6, b=3 pokriva sve naredbe, ali ne i sve putanje



Primer



Putanja 1 2 3 4 5 6 7 2 8 pokriva sve naredbe Putanja 1 2 3 5 6 7 2 3 4 5 6 7 2 8 pokriva sve naredbe i svaka odluka je doneta na različite načine bar jednom

Hijerarhija pokrivenosti

```
Pokrivenost putanja
      \/
Pokrivenost višestrukih uslova
      \/
Pokrivenost uslova
      \/
Pokrivenost odluka
      \/
Pokrivenost naredbi
```

Pokrivenost koda

Kako izračunati nivo pokrivenosti koda?

- Postoje različiti alati za izračunavanje nivoa pokrivenosti testovima https://stackify.com/code-coverage-tools/
- Alati mogu da budu sastavni deo alata za razvoj softvera ili se mogu pokretati nezavisno

- gcov, Cobertura, CodeCover, Coverage.py, Emma, Gretel, Hansel,JaCoCO, JCov ...
- Visual Studio Testing Tools https://docs.microsoft.com/en-us/visualstudio/test/improve-code-quality (opcija Analyze Code Coverage)
- EclEmma (JaCoCO za Eclipse http://www.eclemma.org/, ranije Emma za Eclipse)
- ...

Kako izabrati putanje?

Neka pravila

- Bolje je imati više manjih putanja kroz više test primera nego jednu komplikovanu putanju
- Najbolje je kada su putanje male varijacije drugih putanji
- Petlja može da ima beskonačno mnogo putanja:
  - Preskoči petlju
  - Prođi jednom kroz petlju
  - Prođi dva dva puta kroz petlju
  - Ako postoji maksimalan broj prolaza n, prođi n-1 i n puta kroz petlju

Automatizacija
Da li se i šta u postupku izbora putanji može automatizovati?

Kako izabrati putanje?

Testiranje baznih putanja (engl. *basis path testing*)
je sačinjeno od sledećih koraka:

1. Izvođenje grafa toka upravljanja iz softverskog modula
2. Izračunavanje ciklomatične kompleksnosti grafa (**C**)
3. Odabir skupa **C** baznih putanja
4. Pravljenje test slučaja za svaku baznu putanju
5. Izvršavanje ovih testova

Testiranje baznih putanja

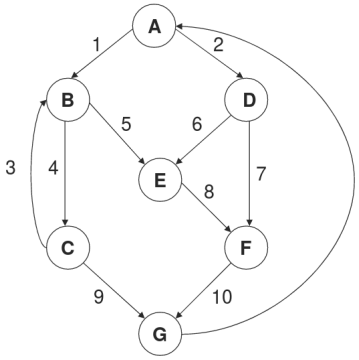
Ciklomatična kompleksnost (engl. *cyclomatic complexity*)
... je metrika koja se koristi da se izračuna kompleksnost softvera. To je kvantitativna mera broja linearno nezavisnih putanja kroz kôd programa. Izračunava se pomoću jednačine:

$$C = grane - čvorovi + 2 * broj\_povezanih\_komponenti$$

Kreiranjem i izvršavanjem svih baznih test slučajeva, garantuje se i pokrivenost grana i pokrivenost naredbi zato što skup baznih putanja pokriva sve grane i čvorove grafa kontrole toka.

Primer

C = 10 - 7 + 2 = 5, algoritam: u svakom čvoru odluke promeniti odluku, počevši od najdonjeg



Path 1: A, B, C, G.
Path 2: A, B, C, B, C, G.
Path 3: A, B, E, F, G.
Path 4: A, D, E, F, G.
Path 5: A, D, F, G.

Osnovni problem: pretpostavka o mogućnosti izbora ovih putanja i kako odabrati putanje koje su dostižne.

Testiranje grafa toka podatka

Testiranje grafa toka podatka (engl. *data flow graph*)

- ... koristi graf kontrole toka da istraži ispravnost upotrebe podataka
- Anomalije upotrebe podataka uključuju:
  - Promenljive koje se koriste a nisu inicijalizovane
  - Inicijalizovane promenljive koje se ne koriste
  - Promenljive koje su više puta definisane pre upotrebe
  - Dealokacija promenljive pre prve upotrebe
  - ...

3.3 Druge tehnike testiranja

Druge tehnike testiranja

Tehnike sive kutije, metamorfno testiranje

- Ukoliko nemamo pristup kompletnom kodu, onada se mogu koristiti tehnike sive kutije, koje kombinuju pristupe bele i crne kutije, u odnosu koji odgovara datoj situaciji
- Kako primeniti tehnike testiranja ukoliko ne znamo koji je odgovarajući izlaz za neki konkretan ulaz?

Druge tehnike testiranja

Kako generisati odgovarajuće parove ulaz-izlaz?

- Jedno rešenje može da bude da se koristi više implementacija algoritma čiji se izlazi nad istim test primerima porede: ako su rezultati različiti, onda bar jedan od algoritama ima grešku

- Ova tehnika se često ne može primeniti pošto često ne postoji više implementacija istog algoritma ili su te implementacije previše skupe i zahtevaju puno vremena
- Takođe, ako se različite implementacije kreiraju od strane istih ljudi, moguće je da oni prave iste greške

Metamorfno testiranje

Problem proročišta — kako odrediti da li je izlaz iz testiranog programa ispravan?

- U nekim situacijama proročište nije dostupno ili ga je previše teško iskoristiti.
- Problem proročišta je izražen u oblastima kao što su računarska grafika, konstrukcija kompilatora, mašinsko učenje
- Metamorfno testiranje je metod koji testira programe bez korišćenja proročišta. Umesto toga koriste se osobine algoritma koji se testira kako bi se generisali dodatni test primeri i automatski verifikovali njihovi izlazi.

Metamorfno testiranje

Metamorfne relacije

- Većina aplikacija ima neka svojstva takva da za određene promene ulaza mogu da se predvide neke karakteristike novog izlaza, uz poznavanje prvobitnog izlaza.
- Na primer, "ukoliko se ulaz poveća za n onda će izlaz da se poveća za n^2
- Na primer, "ukoliko je ulaz po modulu k isti, onda je i odgovarajući izlaz isti"
- "Izvorni" i "naknadni" test primeri mogu generisati na osnovu datih metamorfnih relacija.
- Opisana tehnika je bazirana na jednostavnom konceptu, nije teška za implementaciju, može se automatizovati i nezavisna je od određenog programskog jezika.
- Ključni korak pri ovoj tehnici je identifikacija metamorfnih relacija koje daju neku vezu između više ulaza i njihovih izlaza za dati program.

Metamorfno testiranje

Kako izabrati odgovarajuće metamorfne relacije?

- Postoje principi koji se mogu poštovati, iz perspektive testiranja metodama bele kutije ali takođe i iz testiranja metodama crne kutije.

Princip različitosti u izvršnim putevim

- Dobro je izabrati metamorfne relacije koje rezultuju najvećim razlikama u izvršnim putevima između izvornih i naknadnih test primera: ako imamo veliku razliku izmedu izvršnih puteva, imamo i veliki prostor u kojem može da se izrazi propust u softveru.
- Međutim, često nije očigledno koje metamorfne relacije će rezultovati većoj razlici između izvršnih puteva, tako da je u tim slučajevima neophodno pokrenuti program i analizirati pokrivenost koda za tako generisane test primere.

Metamorfno testiranje

Relacija ekvivalentnosti

- Relacija ekvivalentnosti kao relacija između relevantnih izlaza se pokazala kao bolja od ostalih relacija pošto je ekvivalentnost uži uslov od ostalih neekvivalentnih uslova
- Metamorfne relacije sa relacijom ekvivalentnosti se lakše mogu prekršiti od ostalih relacija, čime bi trebalo da se detektuje veći broj grešaka

Domensko znanje

- Potrebno je dobro domensko poznavanje problema kako bi primena ovog vida testiranja bila efikasna.

Metamorfno testiranje — primeri

Računanje standardne devijacije niza brojeva

- Permutacija reda elemenata ne utiče na konačni rezultat
- Množenje svake vrednosti sa -1, ne utiče na konačni rezultat pošto devijacija od srednje vrednosti u oba slučaja ostaje ista.
- Ako se svaki broj pomnoži sa nekom konstantom, standardna devijacija tog novog niza brojeva bi trebalo da je srazmerna standardnoj devijaciji originalnog niza.
- ...

Metamorfno testiranje — primeri

Konstrukcija kompilatora

- Teško je verifikovati ekvivalenciju između izvornog koda i objektnog koda.
- Za dati izvorni kôd, mogu se dodati određene linije koje mu ne menjaju semantiku (dodavanje nule izrazu ne menja vrednost izraz, uslov `if (true) ...` takođe ne menja semantiku...).
- Za tako generisane programe, trebalo bi da bude generisan izvršni kôd koji se ponaša na isti način

Metamorfno testiranje — primeri

Mašinsko učenje

- Intuitivne metamorfne relacije koje se mogu koristiti za algoritme klasifikacije:
  - Konzistentnost sa afinim transformacijama
  - Permutacija labela klase
  - Permutacija atributa
  - Dodatak neinformativnih atributa
  - ...

4 Načini testiranja

Automatizacija u testiranju

Podela automatizacije

- Način generisanja test primera
- Način izvršavanja test primera

Generisanje test primera ...
... se može automatizovati samo za neke vrste testiranja (na primer za metode bele kutije, za neke nefunkcinalne testove... više o tome kasnije u nastavku kursa). Za većinu funkcionalnog testiranja neophodno je manuelno generisati test primere.

Automatizacija u testiranju

Izvršavanje test primera

- Validacija softvera testiranjem najčešće zahteva ručno izvršavanje testova.
- U mnogim slučajevima je moguće automatizovati izvršavanje testiranja.
- Tehnike automatizacije procesa testiranja — sastavni deo alata za razvoj softvera, alati za kontinuiranu integraciju softvera, alata za testiranje specifičnih vrsta softvera...

Automatizacija u testiranju

Izvršavanje test primera

- Automatsko izvršavanje test primera u okviru alata za razvoj sofvtera
- Najčešće vezano za testiranje jedinica koda
- xUnit framework (JUnit, CppUnit ...)
- Obično povezan i sa automatskim računanjem pokrivenosti koda

Alati za kontinuiranu integraciju softvera

Konitunirana integracija ...
... je neophodna za razvoj softvera gde članovi time unose izmene na dnevnom nivou. Svaku izmenu je potrebno objediniti sa tekućim stanjem koda, potvrditi automatskom izgradnjom koda i testiranjem kako bi se detektovala potencijalne greške u najkraćem vremenskom roku. Prilikom svakog objedinjavanja, sistem je integrisan (sve promene do tog trenutka su objedinjene u projekat), izgrađen (kôd je kompajliran u paket ili u izvršni fajl), testiran (pokreću se automatski testovi), arhiviran (verzionisan i sačuvan) i primenjen (učitan na sistem gde se može izvršavati). Ovakvim pristupom se smanjuje cena projekta, vreme projekta i rizik pri objavljivanju novih verzija.

Alati za kontinuiranu integraciju softvera

Kontinuirana integracija softvera

- Jenkins https://jenkins.io/
- Buildbot
- Travis CI
- GitLab CI
- Circle CI
- TeamCity by Jetbrains
- Bamboo by Atlassian
- ....

Testiranje veb aplikacija

Selenium

- Selenium je softver otvorenog koda
- Selenium je skoro standard za automatizovano testiranje veb aplikacija
- Selenium podržava testiranje veb aplikacija u gotovo svim dostupnim pretraživačima
- Test skriptovi mogu biti pisani u različitim programskim jezicima kao što su C#, Java, Ruby, Python i Perl i pokretani na Windows, Macintosh ili Linux platformama.
- Na strani kursa postoje literatura za razne vrste alata za testiranje, uključujući Selenium

Ali stalno treba imati u vidu...

Edsger Wybe Dijkstra (Tjuringova nagrada 1972)
"Program testing can show the presence of bugs, never their absenc

Testiranje ne može da dokaže ispravnost softvera...
Pravilnim i sistematičnim testiranjem podižemo nivo pouzdanosti i smanjujemo verovatnoću da greške promaknu. **Testiranje se ne radi nasumično, već je važno poznavati metodologiju, procese i principe testiranja.**

Literatura

Literatura
A Practitioner's Guide to Software Test Design, Lee Copeland

Literatura na srpskom

- **Testiranje** — Iz materijala za izradu master rada *„Automatsko generisanje test primera uz pomoć statičke analize i rešavača Z3"* student Ana Đorđević, mentor Milena Vujošević Janičić, Matematički fakultet
- **Pregled osnovnih tehnika testiranja** — Seminarski rad u okviru kursa Metodologija stručnog i naučnog rada, Lazar Mrkela, Ivan Milić Matematički fakultet

Literatura na srpskom — dodatni materijali

- Seminarski rad, Rastko Đorđević: Metamorfno testiranje.
- Seminarski rad, Lazar Mrkela, Ivan Milić: Pregled osnovnih tehnika testiranja
- Master rad, Ana Đorđević: Automatsko generisanje test primera uz pomoć statičke analize i rešavača Z3
- Master rad, Ana Mitrović: Primena jezila Scala u paralelizaciji rasplinutog testiranja
- Seminarski rad, Petar Mićić: Automatsko generisanje test primera korišćenjem genetskog algoritma
- Seminarski rad, Lazar Mladenović: JUnit
- Seminarski rad, Dalma Beara: Selenium – alat za testiranje veb aplikacija
- Seminarski rad, Stefan Pantić: CppUnit - Biblioteka za testiranje C++ koda
- Seminarski rad, David Gavrilović: Komponentno i integraciono testiranje u programskom jeziku C++
- Seminarski rad, Dijana Zulfikarić: Biblioteka PyTest
- Seminarski rad, Nikola Dimić: WebdriverIO - savremen alat za testiranje
- Seminarski rad, Luka Milošević: Fuzz testiranje
- Seminarski rad, Filip Jovašević: Testiranje metodama klase ekvivalencije i graničnih vrednosti
- Seminarski rad, Stefan Stanišić: Razvojni okvir Jasmine
- Seminarski rad, Nikola Kovačević: Jedinično testiranje pomoću xUnit alata (u C#.NET aplikacijama)