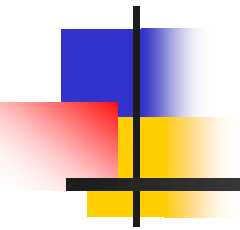


Testiranje softvera

Uvodna lekcija

Komunikacija

- Predavač: doc dr Dragan Bojić paviljon soba 23, bojic@etf.rs
- Asistent: dipl. ing Dražen Drašković soba 37, draskovic@etf.rs
- Sajt predmeta: http://si3ts.etf.rs

- Mailing lista: si3ts@lists.etf.rs, arhiva: http://lists.etf.rs

Pravila polaganja

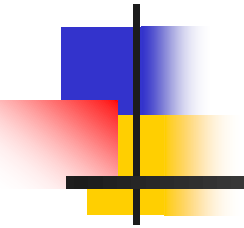
- Za polaganje ispita neophodno je imati urađene i odbranjene domaće zadatke (max 40 poena).
- U svakom ispitnom roku student može polagati ispit koji pokriva celo gradivo i boduje se sa 60 poena. Ukoliko je student polagao kolokvijume uzima se najbolja varijanta zaključivanja ocena:

1. oba kolokvijuma se računaju: Total = D1 + D2 + K1 + K2 + ispit * 1/3
2. računa se jedan od kolokvijuma: Total = D1 + D2 + Kx + ispit * 2/3
3. računaju se samo domaći i ispit: Total = D1 + D2 + ispit * 1

- Uslov za izlazak na ispit je da je (D1 + D2) >= 20. Uslov za pozitivnu ocenu je, u sve tri varijante, da broj poena u totalu koji se osvoji kroz kolokvijume i završni ispit >= 30. Granice formiranja ocena su na 51, 61, itd.

Literatura

- Materijali sa sajta predmeta (predavanja i vežbe) – dovoljni za uspešno savladavanje gradiva
- Knjige:
  - Jorgensen, Software Testing: A Craftman's Approach, 3. izdanje, 2008, ISBN: 0849374758
  - Myers, Sandler, Badgett, The Art of Software Testing, 2. izdanje, 2004, ISBN: 0471469122
  - -, Standard for Software Component Testing, British Computer Society 2001, http://www.testingstandards.co.uk/
  - Pezzand, Young, Software Testing and Analysis: Process, Principles and Techniques, 2008, ISBN:9780471455936
  - Schultz, Bryant, Langdell, Game Testing All in One, 2005, ISBN:1592003737



Uvod

Industrijski softver u odnosu na "studentske" programe

- "Studentski" softver:
  - Uglavnom se pravi u svrhu demonstracije ili iz hobija, ne rešava neki realan problem
    - Sledi da prisustvo grešaka (bagova, defekata) ne zabrinjava
  - Upotrebljava ga sam autor, tako da nije važno dokumentovanje, a bagove ispravlja sam autor ako na njih naiđe
  - Životni vek je kratak (najčešće za jednokratnu upotrebu)

Industrijski softver

- Engl. industrial strength software
- Napravljen da rešava neki poslovni problem korisnika, tj. važne aktivnosti zavise od korektnog funkcionisanja sistema => loše funkcionisanje izaziva nezadovoljstvo korisnika i finansijske, materijalne gubitke, ili čak ljudske žrtve.

Čuveni primeri softverskih otkaza

- **Aerodrom u Denveru**: 1994, zakašnjenje u otvaranju od skoro godinu dana zbog nefunkcionisanja automatizovanog sistema za transport prtljaga (inicijalna cena tr.sistema 234 M$, troškovi zakašnjenja 1 M$ dnevno)
- **Deutsche Telekom**: pogrešan proračun cene telefonskih impulsa za 1.1.96 (softverska greška: bez praznične tarife – šteta: stotine miliona DEM)
- **Prvi internet crv**:(program koji se sam umnožava i širi) 1988 zaraženo nekoliko hiljada računara, bug u virusu prouzrokovao zagušenje računara, šteta 10-100 M$. Koristio rupe u sendmailu, rsh/execu. autor R.T.Moris sin inženjera NSA uslovno i novčano osuđen, sada prof na MITu

Čuveni primeri softverskih otkaza

- **Mariner I, prva svemirska raketa za Veneru:** 1962 usled softverskog otkaza automatizovanog sistema za navođenje morala biti uništena daljinskom komandom 5 min posle lansiranja (gubitak 18 M$).
- **Ariane 5 let 501**, 1996 raspala se 40s posle lansiranja usled prekoračenja u konverziji float->int, šteta 370 M$
- **Therac 25 – računarski kontrolisani uredjaj za terapijsku radijaciju:** izmedju juna 1985. i januara 1987. 6 ljudi je predozirano (5 od njih je kasnije umrlo) kao posledica nedostajuće softverske sigurnosne brave koja bi trebala da spreči predoziranje

Industrijski softver

- Posledice potreba za kvalitetom:
  - 30% do 50% ukupnog napora (troška) je na testiranje, za stud. softver ne prelazi 5%
  - Zahtevi za planiranim razvojem po fazama, dokumentovanjem, pridržavanjem raznih standarda, ispunjavanjem različitih nefunkcionalnih zahteva (sigurnost, portabilnost, performanse)
  - Ovakav softver zahteva 10x više napora za istu funkciju od "studentskog".
  - prosečna produktivnost po osobi u celokupnom ciklusu razvoja industrijskog softvera je 300 do 1000 LOC/mes (LOC = linija izvornog koda)

Veličina industrijskog softvera

- Mali projekti <= 10 KLOC
- Srednji <= 100 KLOC
- Veliki <= 1 MLOC
- Veoma veliki – više MLOC

Industrijski softver i važnost njegovog testiranja

- Veličina industrijskog softvera:
  - Windows 2003: ~50 MLOC (miliona linija izvornog koda)
  - Linux kernel: 2.6.32 >12 MLOC
- Američki nacionalni institut za standarde (NIST) procenjuje da su 2002. god. softverski bagovi izazvali 60 milijardi $ gubitaka u američkoj ekonomiji.

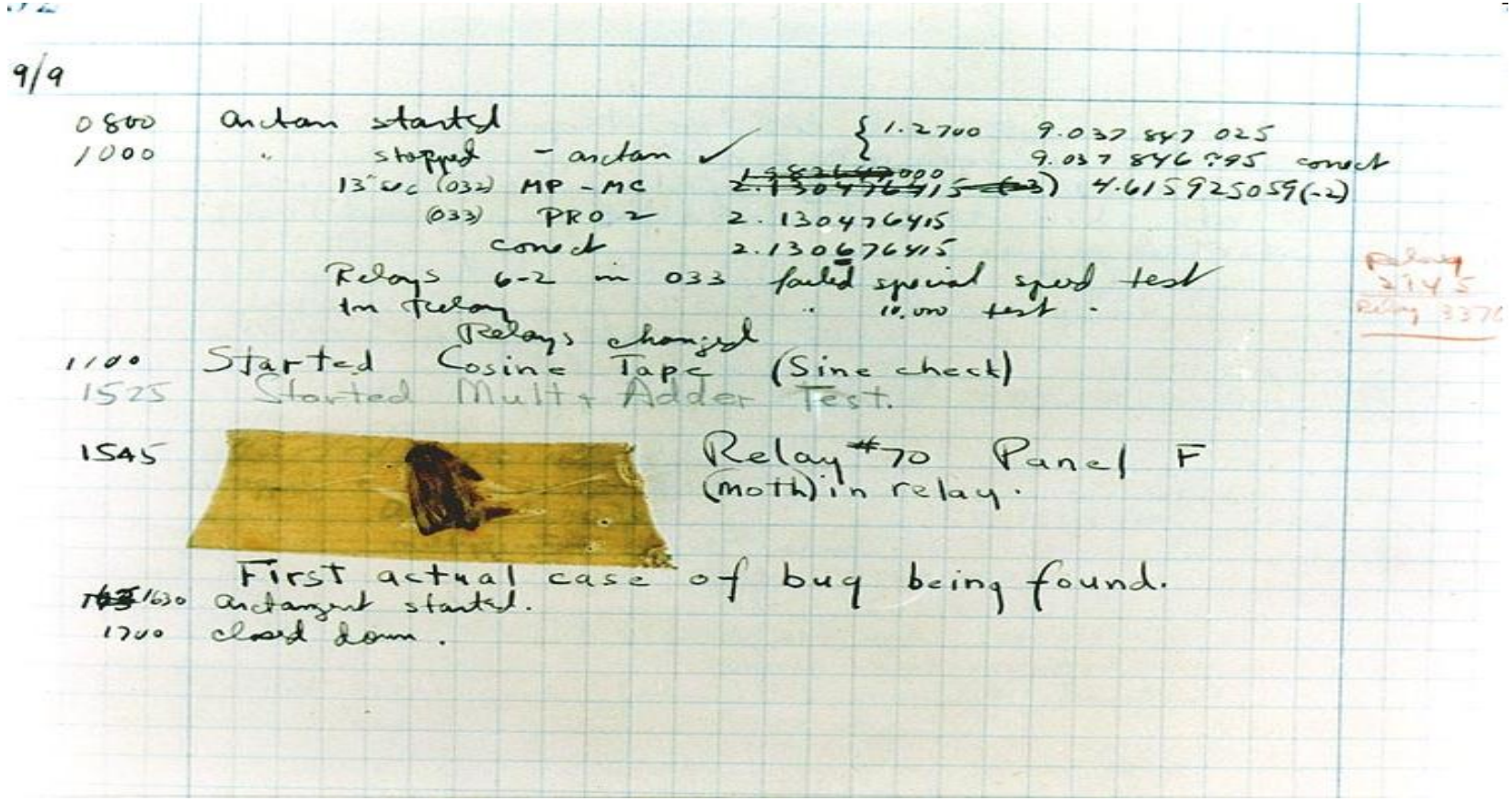
Razvoj softvera u odnosu na druge tehničke discipline

- Jedno ispitivanje u amer. ministarstvu odbrane pokazalo je da se čak 70% otkaza opreme može pripisati softveru (u sistemima punih električnih, mehaničkih i hidrauličnih komponenata)
- Druge tehničke discipline su znatno zrelije, softver je često slaba tačka.
- Otkazi fizičkih sistema javljaju se usled fizičkih i električnih promena uzrokovanih starenjem
- Softver ne stari, greške se nalaze u njemu od početka, a mogu se manifestovati u vidu otkaza i posle dužeg ispravnog rada.

Terminologija u vezi sa greškama i testiranjem

- Izvor: Glossary of Software Engineering Terminology. ANSI/IEEE Std. 729–1983
- **Greška (Error)**
  - Napravi je čovek, na primer, prilikom specifikacije zahteva ili kodiranja programa
- **Mana, defekat (Fault)**
  - posledica greške (na primer, programu nešto nedostaje, ili ima funkciju ali neispravno - "bug")
- **Otkaz (Failure)**
  - Nemogućnost sistema da obavi zahtevanu funkciju najčešće se javlja aktiviranjem (izvršavanjem) defektnog koda.

Prva kompjuterska "bubica"

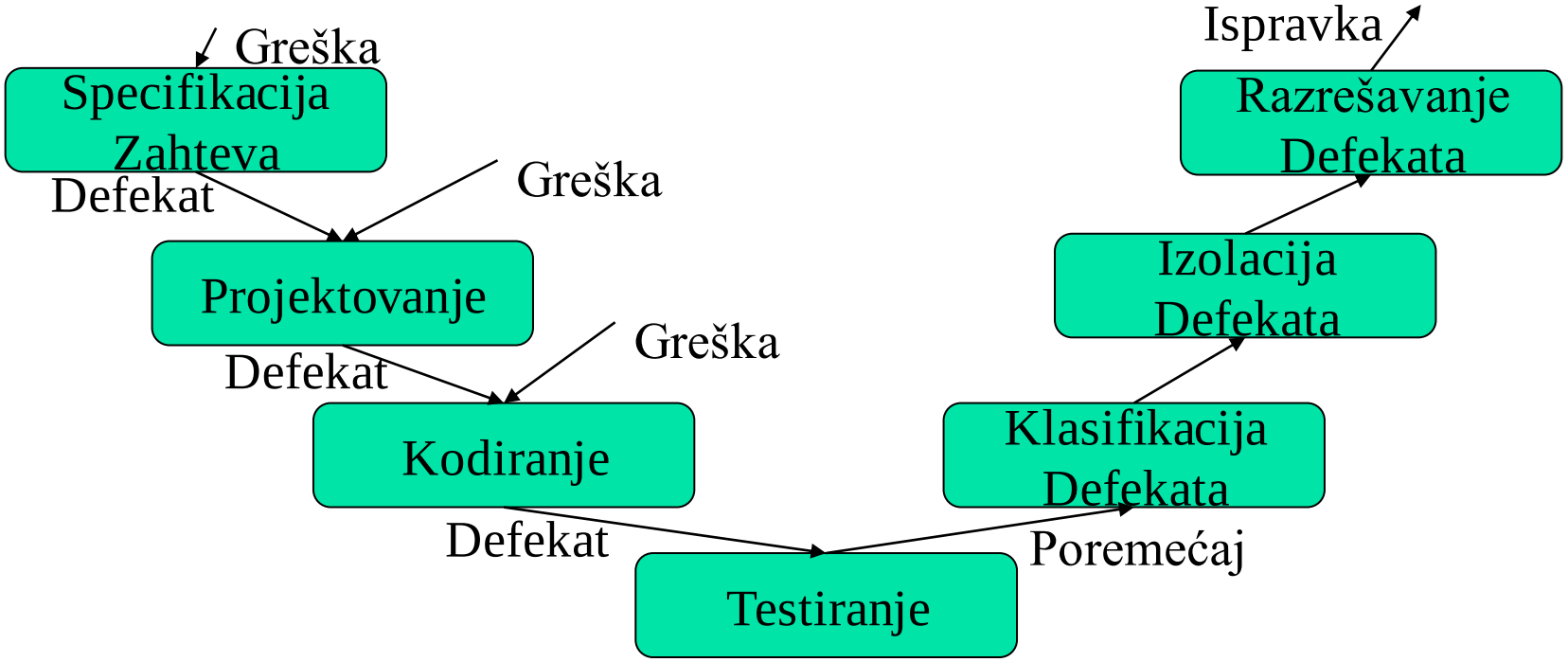


Moljac pronađen u releju računara testiranog na Harvard univerzitetu septembra 1945 i prikačen u dnevnik

Terminologija

- **Poremećaj (Incident)**
  - simptom koji korisniku obznanjuje otkaz
- **Testiranje**
  - postupak izvršavanja softvera sa test primerima. Dva su različita cilja testiranja: da se nađu otkazi, ili da se demonstrira ispravno izvršavanje
  - Testiranje može otkriti otkaze, ali potom treba eliminisati defekte – debagovati program, što je posebna aktivnost
- **Test primer (Test case)**
  - poseduje identitet, i ispituje određeno ponašanje programa. TP poseduje skup ulaznih veličina i skup očekivanih rezultata.

Životni ciklus testiranja



- Može se uočiti da u fazi razvoja postoje tri mesta gde može nastati greška, što rezultuje defektima koji se prenose u ostale razvojne faze.
- Ukratko, u prve tri faze bugovi se ubacuju, u fazi testiranja otkrivaju, a u poslednje tri faze izbacuju.
- Faza razrešavanja defekata pruža takođe priliku za nastajanje grešaka i novih defekata.

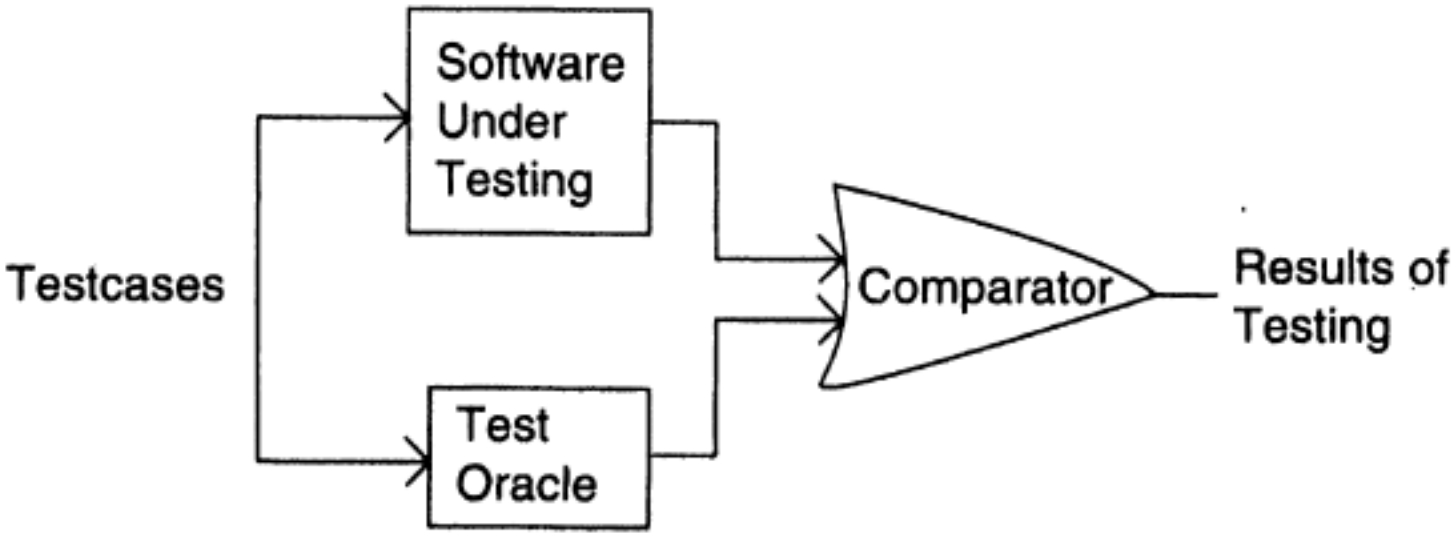
Definicija testiranja softvera

- proces izvršavanja programa sa ciljem nalaženja grešaka (G. Myers)
- U širem smislu, ispitivanje softverskog proizvoda ili servisa sa ciljem objektivnog utvrđivanja kvaliteta.
- Deo šire oblasti osiguranja kvaliteta (SQA), koja se bavi kontrolom procesa izrade da se obezbedi kvalitetan finalni proizvod u predvidivom vremenskom roku.

Test primeri i serije testova

- **Test primer** je trojka [I, S, O] gde
  - I predstavlja ulazne podatke
  - S je stanje sistema u kome će podaci biti uneti
  - O je **očekivani** izlaz
- **Serija testova** je skup svih test primera
- Test primeri se ne biraju na slučajan način. Umesto toga treba ih **projektovati**.

Očekivani rezultati testa



- Predviđanje, predikcija rezultata testa (test oracle)
  - Prediktor testa je mehanizam, nezavisan od samog programa, koji se može upotrebiti za proveru ispravnosti rada programa za test primere.
  - Konceptualno, test primeri se zadaju programu i prediktoru i njihovi izlazi potom međusobno upoređuju

Prediktor testa

- Može biti čovek ili automat
- Ljudi koriste specifikaciju programa da odluče šta je ispravno ponašanje programa. Međutim, specifikacije programa su često sa greškama, nepotpune ili dvosmislene, a i ljudi mogu napraviti previd
- Automatizovni prediktori koriste formalne specifikacije i tačni su onoliko koliko je specifikacija tačna. Formalne specifikacije često ne postoje za program (nije ih lako napraviti)

Potreba za projektovanjem test primera

- Skoro svaki netrivijalni sistem ima ekstremno veliki domen ulaznih podataka => isrcpno testiranje za svaku zamislivu kombinaciju ulaza je nemoguće ili nepraktično
- Slučajno izabrani test primer može biti bez značaja ako izlaže grešku koja je detektovana nekim drugim test primerom

Projektovanje test primera

- Broj test primera ne određuje koliko je testiranje delotvorno
- Da bismo uočili grešku u sledećem kodu

  if(x>y) max = x; else max = x;
- {(x=3, y=2); (x=2, y=3)} zadovoljava
- {(x=3, y=2); (x=4, y=3); (x=5, y = 1)} ne zadovoljava
- Svaki test primer trebalo bi da razotkrije različitu grešku

Kriterijumi selekcije testova

- Kako da znamo koju ulaznu vrednost da uzmemo za testiranje, a koju ne?
- Mora se definisati kriterijum selekcije:
  - Za dati program P i njegovu specifikaciju S, kriterijum selekcije testova definiše uslove koje mora da zadovolji serija testova T.
  - Na primer, ako je kriterijum da se svaki iskaz programa mora izvršiti najmanje jednom, tada T zadovoljava ako za svaki iskaz u P postoji bar jedan test primer u T koji izazva izvršavanje tog iskaza.

Osobine kriterijuma selekcije

- Osnovne osobine su pouzdanost i validnost
- Kriterijum C je pouzdan ako svaka serija testova (nije važno koju izaberemo) koja zadovoljava C detektuje iste greške.
- Kriterijum C je validan ako, za svaku grešku u programu, postoji neka serija testova koja zadovoljava C koja će otkriti grešku.
- Ako je kriterijum C pouzdan i validan, a serija testova koja zadovoljava C prođe uspešno, onda program nema grešaka.

Problemi sa kriterijumima selekcije

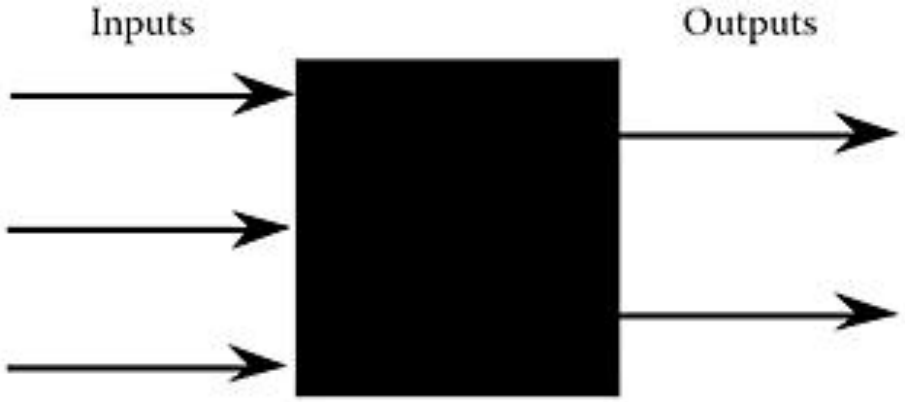
- Na žalost, nema načina da se za proizvoljan program identifikuje validan kriterijum.
- Kriterijum koji je validan i pouzdan, a da ga može zadovoljiti konačna serija testova isto najčešće nije moguće naći.
- Zato se najčešće u realnosti definišu kriterijumi koji nisu ni pouzdani ni validni, kao što je "90% iskaza bi trebalo izvršiti bar jednom".
- Za većinu kriterijuma nije moguće automatizovati generisanje testova zadovoljavaju kriterijum.
- *[Dijkstra] Testiranje programa može se koristiti da pokaže prisustvo grešaka u programu, nikada njihovo odsustvo.*
- Pošto se dakle sve greške u programu ne mogu garantovano otkriti testiranjem i uz navedena ograničenja kriterijuma, najčešće se u testiranju primenjuje neka kombinacija kriterijuma.
- Sve navedeno ima posledice na odluku o završetku testiranja o čemu će posebno biti reči kasnije

Klasifikacije testiranja

- Uočavamo dve vrste klasifikacije
  - Prema nivou granularnosti na
    - Jedinično
    - Integraciono
    - Sistemsko
  - Druga klasifikacija (uglavnom za jedinični nivo) je prema pristupu testiranja:
    - Funkcionalno
    - Strukturno

Funkcionalno testiranje

- Program se posmatra kao funkcija koja mapira vrednosti iz ulaznog domena u izlazni. Implemenacija nije poznata, program predstavlja "crnu kutiju".



Funkcionalno testiranje

- Jedina informacija koja se koristi za određivanje test primera je specifikacija programa.
- Prednosti funkcionalnog pristupa:
  - Test primeri su nezavisni od konkretne implementacije, tako da su upotrebljivi i pri promeni implementacije
  - Razvoj testova može teći u paraleli sa razvojem implementacije, čime se smanjuje potrebno vreme
- Mane ovog pristupa:
  - Često postoji velika redundantnost testova u seriji
  - Mogućnost da se deo implementacije ne pokrije testovima (npr. ne može se otkriti da je u program zaražen virusom, pošto ga nema u specifikaciji)

Strukturno testiranje

- Strukturno testiranje fokusira se na implementaciju programa. Naziva se još pristup bele kutije (providne kutije).
- Cilj nije da se izvrše sve moguće funkcije programa, već da se izvrše/aktiviraju različite programske i strukture podataka u programu.
- Različiti kriterijumi strukturnog testiranja znatno preciznije od kriterijuma funkc. testiranja definišu koje programske strukture treba pokriti (npr. iskaze, odluke, putanje)
- Ovim testiranjem nije moguće otkriti da li neki element specifikacije nije implementiran u programu, pošto se specifikacija ne uzima u obzir za selekciju testova

Nivoi testiranja

- Jedinično testiranje
  - Verifikuje ponašanje delova softvera koji se mogu izdvojiti od ostatka i odvojeno testirati. To mogu biti individualni potprogrami ili klase, ili veće celine sastavljene od čvrsto spregnutih komponenata (npr. izvorni modul u jednom fajlu)

Nivoi testiranja

- Integraciono testiranje
  - Verifikuje interakciju između softverskih modula (koji su prethodno jedinično testirane), da li se pravilno kombinuju u podsisteme. Naglasak je na proveri intefrejsa modula. Može se shvatiti kao provera dizajna.
  - Proučićemo razne strategije postepene (inkrementalne) integracije koja se preferira u odnosu na sastavljanje svih komponenata odjednom, tj. “Big bang” integraciju.

Nivoi testiranja

- Sistemsko testiranje
  - Proverava ponašanje sistema kao celine u odnosu na specifikaciju. Pošto je većina funkcionalnih zahteva proverena na nižim nivoima testiranja, ovde je naglasak na nefunkcionalnim zahtevima kao što su sigurnost, brzina, pouzdanost, eksterni interfejsi prema drugim aplikacijama i operativnom okruženju.

Program

- Uvod u testiranje softvera.
- Manuelno i automatizovano testiranje.
- Jedinično testiranje.
  - Tehnike bele kutije.
    - Tehnike pokrivanja koda zasnovane na toku kontrole.
    - Tehnike toka podataka (sve c-upotrebe, sve-p-upotrebe, sve DU putanje, sve bazicne putanje, McCabeov baseline metod).
    - LCSAJ testiranje
  - Tehnike crne kutije.
    - Deljenje na klase ekvivalencije.
    - Uzročno posledični grafovi.
    - Analiza graničnih vrednosti.
    - Testiranje zasnovano na modelu stanja
    - Testiranje sintakse
    - Kombinatorno testiranje zasnovano na ortogonalnim nizovima
    - Testiranje zasnovano na tabelama odlučivanja
  - Tehnike objektno-orijentisanog testiranja
- Integraciono testiranje.
- Inkrementalno i neinkrementalno testiranje.
- Testiranja višeg reda.
- Upravljanje procesom testiranja.
  - Planiranje testiranja
  - Kriterijumi završetka procesa testiranja.
  - Izveštavanje o rezultatima testiranja
- Upotreba alata za testiranje na praktičnom projektu.