

TESTING

TESTING
PROGRAMMING

NEEDS OF
SOFTWARE

BUGS

METHODS

DATABASE

USABILITY

TESTING

TESTING



Dinamičko testiranje softvera

Za razliku od statičkog, dinamičko testiranje softvera predstavlja proces provere, koji se sprovodi u toku izvršavanja programa.

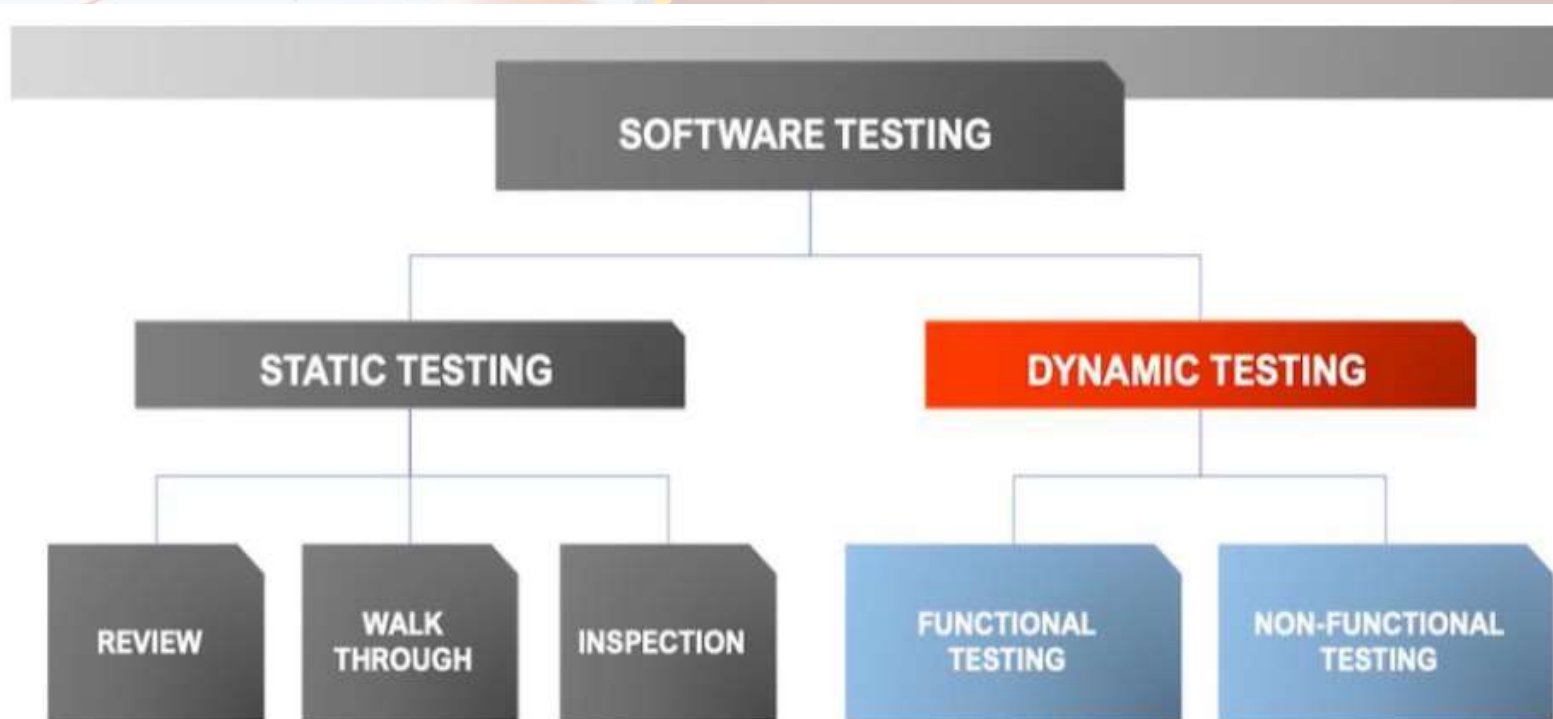
Dinamičkim testiranjem se vrši provera ponašanja sistema, tačnije upoređivanje dobijenog i očekivanog izlaza.

Testovima se teži da se pronađu slabe tačke softvera u stvarnom okruženju gde se nakon unošenja ulaznih podataka rezultati uspoređuju sa očekivanim rezultatima.

Za testiranje korektnosti softvera potrebni su **dobro formirani testni slučajevi**.

- ✓ Ulazni podaci treba da:
- ✓ Povećaju verovatnoću nalaženja greške
- ✓ Što manji broj test primera

Dinamičko Testiranje softvera



A close-up photograph of a yellow pen tip pointing at a piece of paper with handwritten math problems. One problem is $10 + 6 =$ and another is 16 . The background is a solid dark red color.

Intenzitet Testiranja softvera

Kada se govori o Testiranju softvera, ma koliko bili uspešni u otkrivanju potencijalnih grešaka, uvek postoji mogućnost da ipak postoji neka greška u softveru.

PREPORUKA:

- ☐ Testirati softver u svakoj mogućoj situaciji sa svim mogućim ulaznim podacima u svim mogućim kombinacijama

Testiranje softvera može da dokaže da ne postoje greške, ali sa njim se smanjuje mogućnost greške

Potpun test ne postoji ili bolje reći da nije izvodjiv

- ✓ Niko ne zna unapred ukupan broj testova (praktično on bi bio beskonačan), zbog broja mogućih kombinacija.



Dinamičko testiranja softvera

Postoje sledeći Tipovi dinamičkog testiranja softvera:

- ☐ Testiranje zasnovano na iskustvu
✓ *experience-based testing*
- ☐ *White-box testing*
- ☐ *Black-box testing*



Testiranje zasnovano na iskustvu i intuiciji

Znanje i veštine učesnika u projektu se koristi da bi se kreirali test slučajevi. Oni koji su najiskusniji testeri radiće na verifikaciji programa. Zbog velikog iskustva oni će ukazati na određene nedostatke u samom softveru.

Neke od karakteristika ovog vida testiranja softvera su:

- Tehnika testiranja softvera nije potpuno sistematična (bazira je na intuiciji)
- Oslanja se na znanje o tehnologiji
- Veoma važno iskustvo testera iz nekih ranijih projekata
- Da bi se otkrili bagovi i defekti iskustvo testera je neophodno



Pogadanje grešaka -error guessing

Data tehnika se oslanja na iskustvo i procenu samog testera.

To je takozvana umetnost pogađanja gde bi na osnovu iskustva i procene samog testera potencijalna greška mogla da bude sakrivena.

Za ovu tehniku ne postoje ni specifični alati ni uputstva, ni dokumentacija.



CILJEVI DINAMIČKOG TESTIRANJA SOFTVERA

Testirani objekat: Da li on zadovoljava zadatu funkcionalnost ?

- ❑ Da li se može pojaviti otkaz za neki testirani objekat pre svega u odnosu na njegovu funkcionalnost koja sadrži potencijalno neke nedostatke

Dizajn test slučaja

Test = koji su ciljevi testa

Da li postoje (i ako postoje moraju precizno biti navedeni) neki preduslov za testove

Definisati test slučajeve

- Implementirati test slučaj u određenom programskom jeziku
- Postupak izvršavanja test slučaja (redosled,.....

Black box testiranje softvera





Black box testiranje softvera

Postoje veli broj tipova testiranja sofvera koje se nalaze u strategiju crne kutije:

- ❖ Funkcionalno testiranje (functional testing)
- ❖ Alfa testiranje (alpha testing)
- ❖ Beta testiranje (beta testing)
- ❖ Ad-hoc testiranje (ad-hoc testing)
- ❖ Istraživačko testiranje (exploratory testing)
- ❖ Testiranje upotrebljivosti (usability testing)
- ❖ Testiranje opterećenja (load testing)
- ❖ Testiranje oporavka (recovery testing)
- ❖ Testiranje pod pritiskom (stress testing)
- ❖ Testiranje prihvatljivosti od strane korisnika (user acceptance testing)
- ❖ Testiranje opsega (volume testing)
- ❖ Testiranje sistema (system testing)



Black box testiranje softvera

Testiranje metodom crne kutije (black box testing) ili u nekoj literaturi se može naći kao pojam - funkcionalno testiranje.

To je postupak testiranja kod kojeg tester ne poznaje unutrašnju strukturu, dizajn ili implementaciju samog softvera.

Na softver se gleda kao na crnu kutiju koja obrađuje neke ulazne podatke i na kraju se dobijaju izlazne vrednosti.

Ulazni podaci treba da:

- ✓ povećaju verovatnoću nalaženja greške
- ✓ smanje veličinu skupa testova (što manji broj test primera)

Black box testiranje softvera

Klasifikacija testova

Prema kriterijumu dostupnosti koda

Princip "crne kutije" ("black-box")

- Kod se tretira kao crna kutija
- Porede se ulazne vrednosti sa očekivanim izlaznim vrednostima, a zanemaruje se interno funkcionisanje koda

-

U ovom pristupu, svi testovi potiču iz specifikacije programa i ne vrši se nikakvo razmatranje programskog koda

- Testiraju se samo javno dostupne klase, metode i neka svojsva
- Prednost - kada se promeni interni kod neke metode, svi testovi i dalje funkcionišu
- -Mana - zbog "nepoznavanja" unutrašnje strukture koda mogu se propustiti neki testovi (što je čest slučaj)



TESTIRANJE SOFTVERA

Testiranje metodom crne kutije tipično se izvršava kao životni ciklus razvoja kada je :

- kod kompletno integrisanog sistema
- tokom integracionog testiranja
- testiranja interfejsa
- sistemskog testiranja i
- testiranja prihvatljivosti.
- Na tom nivou, komponente sistema su dovoljno integrisane da pokažu da li su kompletni zahtevi ispunjeni skoro u potpunosti.

TESTIRANJE SOFTVERA



Po funkcionalnim zahtevima, pri čemu se svaki funkcionalni zahtev prevodi u kriterijum funkcionalnog testa.



TESTIRANJE SOFTVERA

- ✓ BlackBox testiranje je metod softverskog testiranja u kojem unutrašnja struktura sistema koji se testira nije poznata testeru (tester je vidi kao crnu kutiju i ne zna kako je softver implementiran).
- ✓ greške u netačnim ili nedostajućim funkcijama,
- ✓ greške u interfejsu,
- ✓ greške u pristupu bazi podataka,
- ✓ greške u performansama sistema, itd.



TESTIRANJE SOFTVERA

Sva testiranja vrše se sa aspekta korisnika, a tester prepostavlja šta bi softver trebao da radi.

Pre početka testa ne istražujemo unutrašnju struktura samog softvera.

Tester je tokom testa upoznat sa ulaznim vrednostima i očekivanim izlazim vrednostima.

Test objekat se posmatra kao crna kutija.



TESTIRANJE SOFTVERA

- Tester ne zna kako softver ili aplikacija obrađuje ulazne podatke i daje izlazne podatke.
- Testeri prolaze samo važeće i nevalidne (loše) unose podataka i utvrđuju ispravnost na osnovu očekivanih rezultata.
- Svi test slučajevi koji se testiraju takvom metodom kreiraju se na osnovu zahteva i specifikacija.



TESTIRANJE SOFTVERA

- Prednost ovog metoda testiranja je da se on veoma lako izvršava.
- Ne zahteva preterano znanje o programskim jezicima.
- Jedna od mana jeste da testovi mogu biti teški za dizajniranje i mogu da budu redundantni.
- Takođe ova tehnika ne može da testira sve moguće funkcionalnosti (detaljno testiranje)
- Problem koji se dosta često dešava je da jedan deo razvijenog koda može ostati ne testiran.

TESTIRANJE SOFTVERA



- Funkcionalno ili black-box, testiranje, gde su testovi definisani na osnovu specifikacije softvera.
- Sistem se testira kvalitetnim ulaznim i izlaznim vrednostima (očekivanim i neočekivanim)
 - ✓ Da se pri testiranju koriste i ispravne i neispravne vrednosti,
 - ✓ Zatim neki karakteristični i najčešće očekivani ulazni parametri

TESTIRANJE SOFTVERA

Black Box Testing





TESTIRANJE SOFTVERA

- ❖ Jedina poznata informacija koju tester ima za određivanje i dizajn testova je specifikacija zahteva programa.
- ❖ Metodom crne kutije se mogu otkriti nepostojeće funkcionalnosti ili pogrešne implementacije u softveru, greške u ponašanju softvera kao i problemi sa performansama samog sistema.



Black Box testiranje

Koji je razlog Black Box testiranja?

- Glavna razlog testiranja Black Box-a je da utvrdi da li softver zadovoljava očekivanja korisnika ili ne.

Potencijalni Bugovi (greške) :

- Greške inicijalizacije
- Neispravne ili neke funkcije nisu urađene
- Greške u strukturi podataka ili pristup eksternoj bazi podataka.
- Greške u samom izvršavanju

Black box testiranje softvera

Prednost metoda:

- testiranje ne vodi računa o ograničenjima koja proističu iz unutrašnje strukture komponente i logike njenog rada
- Štedi resurse

Nedostatak metoda:

- detaljno testiranje svih kombinacija različitih ulaznih podataka, odnosno možemo testirati samo mali broj mogućih test slučajeva
- Jedan od nedostatak ove metode testiranja je da uvek postoji mogućnost da pojedine funkcionalnosti ne budu pokrivene testom.
- U mnogim slučajevima zahtevi ili specifikacije nisu dovoljno jasni, što otežava izvođenje test slučajeva
- **Nemogućnost** pridržavanja u **standardu** kodiranja.
- Sa ovim testiranjem nećemo pronaći sve greške u programu.



Black Box testiranje

- Isprobavanja svih mogućih kombinacija ulaznih vrednosti - exhaustive input testing
- Za netrivialne programe nije moguće koristiti ovu tehniku.
- Ulazni podaci treba da: povećaju verovatnoću nalaženja greške smanje veličinu skupa testova (što manji broj test primera)
- Strategija potpuno fokusirana na funkcionalnostima rada softvera
- Specifikacija mora biti tačna, razumljiva i potpuna



Black Box testiranje

- Black Box testiranje alat su prvenstveno alati za snimanje i reprodukciju testova.

Alati se koriste za regresisko testiranje

Testiranje crne kutije je tehnika testiranja u kojoj se funkcionalnost aplikacije testira bez gledanja interne strukture koda, detalja implementacije i znanja o internim putanjama softvera.

Ova vrsta testiranja zasniva se u potpunosti na softverskim zahtevima i specifikaciji.

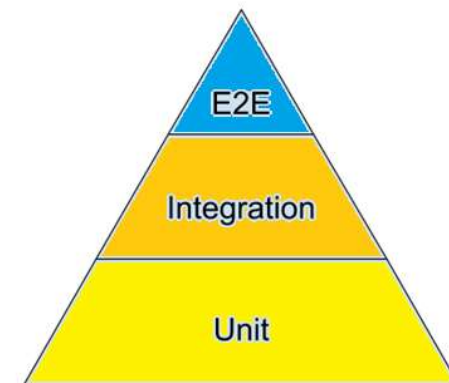


TESTIRANJE SOFTVERA

- Funkcionalno testiranje se može primeniti na svim nivoima testiranja
 - ✓ jediničnom,
 - ✓ integracijskom i
 - ✓ sistemskom nivou
- Uticaj nivoa testiranja na metod crne kutije se ogleda samo u kompleksnosti izvršavanja ove metode

Piramida testiranja softvera

- 2009 postavio Mike Cohn,,
- Piramida ukazuje na međusobni odnos broja različitih tipova testova u sistemu
- Najviše jediničnih testova
- Najmanje *end-to-end* testova



Piramida testiranja softvera

Posmatrano iz ugla project managera sistemski testovi imaju najveću važnost. Zbog toga je nastala i struktura obrnute piramide testiranja (reverse testing pyramid)





Piramida testiranja

Dokazano je da se koristi za kreiranje jediničnih testova

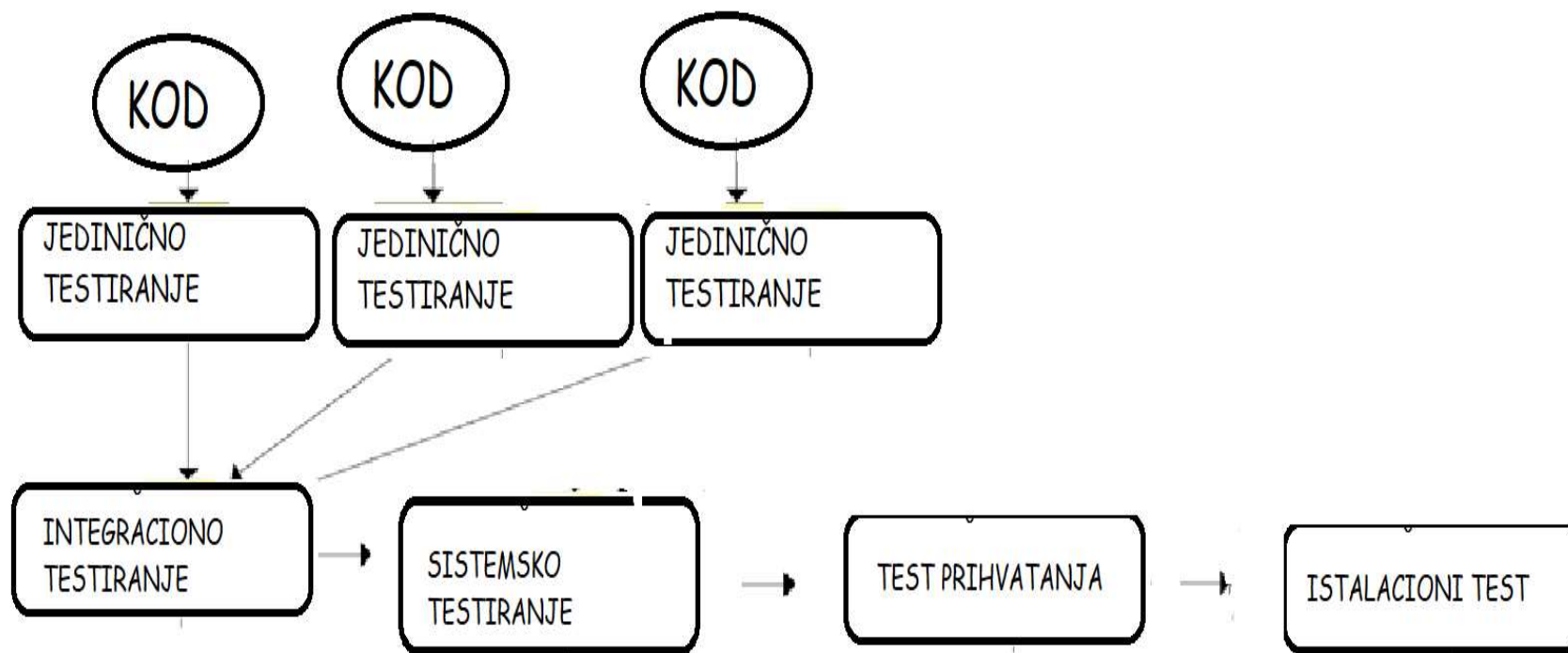
- jedinični testovi testiraju izolovano ponašanje što olakšava lociranje nedostatka u slučaju neuspešnog testa
 - kod e2e testa je teško lociranje problema
 - utvrdi se da sistem ne radi ali se ne zna zašto
 - pošto e2e test testira kompletan sistem, uzrok može biti bilo gde (hardver, softversko okruženje, neki kod u aplikaciji, ...)
- jedinični testovi su brzi za izvršavanje
- e2e testovi su zahtevni za održavanje
 - obzirom da simuliraju korisničku interakciju, neznatna promena grafičkog interfejsa zahteva promenu testa

iranje.

VRSTE TESTIRANJA SOFTVERA

FAZE TESTIRANJA	Unit	Integration	System
METODE	Black box	Black box	Black box
TESTIRANJA	White box	White box	White box
	Grey box	Grey box	Grey box

Proces testiranja softvera





Unit testing

Unit testing (jedinično testiranje) je osnovni nivo testiranja softvera.

Unit testovi (jedinični) služe za zasebno testiranje svake programske komponente tj. funkcionalnosti nezavisno od ostalih delova sistema

U literature se može naći termin testiranje komponenti, testiranje modula, testiranje klasa



Unit testing

- Analize funkcionalnosti
- Analiza izvornog koda komponente

Unit testovi su sastavni deo razvojnog procesa softvera i oni se koriste kako bi se izvršila validacija ispravnosti određenog dela koda

Softverska komponenta testira se nezavisno i izolovano od ostatka sistema.



Jedinično testiranje

- ❑ IEEE Standard for Software Unit Testing

Jedan *unit* test obično obuhvata tri aktivnosti:

- ❑ Priprema objekata, tj. njihovo kreiranje i prilagođavanje potrebama metoda
- ❑ Poziv metode nad nekim objektom i izvršavanje funkcionalnosti koje se žele testirati
- ❑ -Dokazivanje i potvrđivanje očekivanih akcija - ova aktivnost se izvršava korištenjem *Assert* klase

Cilj jediničnih testova je dokazivanje da komponenta ima predviđenu funkcionalnost.



JEDINIČNO TESTIRANJE

Ukoliko postoji napisan veliki broj testova za određeni kôd, potrebno je da programer na osnovu naziva testa može zaključiti koja metoda ili funkcionalnost se testira.

Previše vremena treba za pisanje testova

Mnogi tek na kraju projekta pišu testove

Ako programer nije siguran kako treba nazvati testnu metodu, potrebno je da prvo napiše taj test.

Ako su svi unit testovi uspešni, kod se prihvata i šalje QA inženjeru za inspekciju i testiranje.

Unit testiranje ne zamenjuje druge tipove testiranja

A yellow pen is pointing at a math problem on a piece of paper. The problem involves a triangle with sides labeled 10, 6, and 16. The text "JEDINIČNO TESTIRANJE" is written in red capital letters on a yellow background.

JEDINIČNO TESTIRANJE

Predstavlja testiranje izolovanih delova u sistemu.

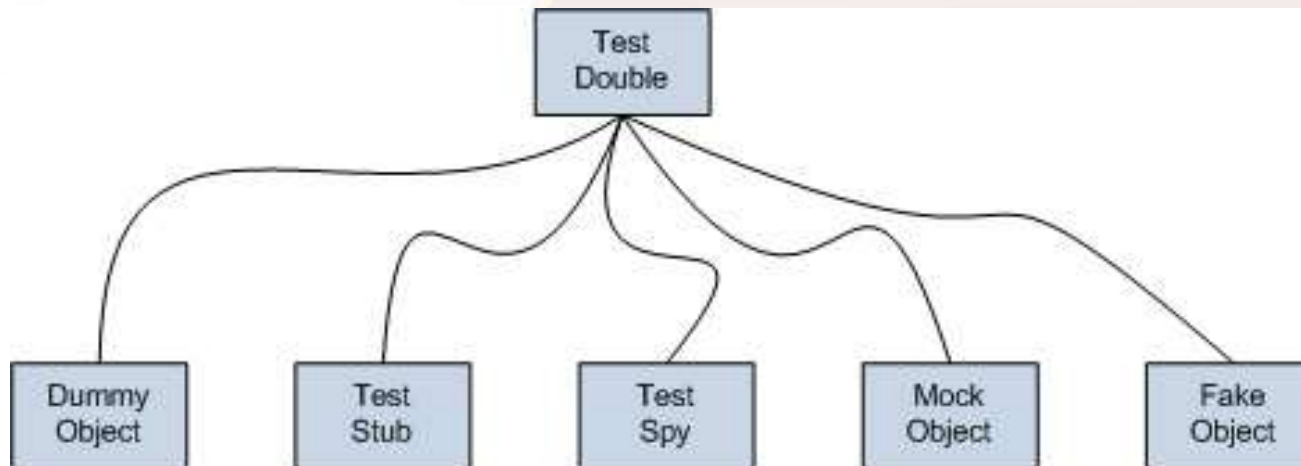
Jedan od uslova je da se komponenta koja se testira može posmatrati kao nezavisna celina koja se može izvući iz konteksta sistema i testirati izolovano od ostalih komponenti.

- Prednost ovakvog načina testiranja je što se greške mogu lokalizovati i ispraviti u elementarnim komponentama kako bi se obezbedilo da ispravna komponenta uđe u sistem.
- Kada se vrši jedinično testiranje, sistem se sastoji iz povezanih komponenti i testira se svaka komponenta posebno.
- Cilj jediničnog testiranja je da izoluje svaki deo programa i pokaže da individualni delovei rade u skladu sa zahtevima i funkcionalnostima.

Jedinično testiranje

Test dvojnik (*test double*)

- Postoje različiti tipovi dvojnika



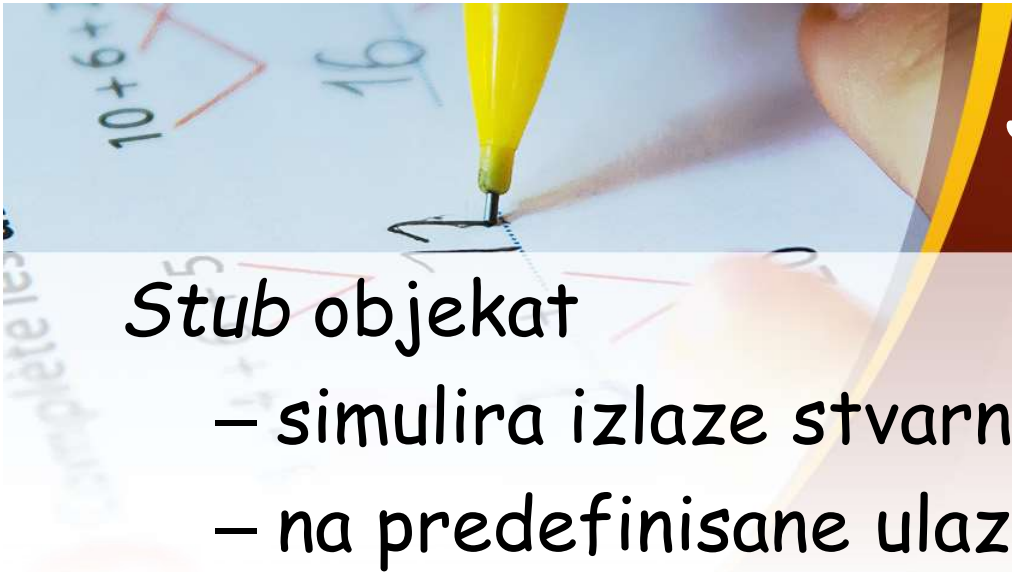


Jedinično testiranje

Tipovi test dvojnika

Dummy objekat

1. Objekat koji mora postojati u kodu, ali se nikad ne koristi
2. Najčešće postoji da popuni obaveznu listu parametara
3. Termin se koristi i za svaki objekat inicijalizovan podacima čiji sadržaj nije važan za test
 1. npr. ako testiramo kreiranje nekog entiteta, sami podaci koje entitet sadrži nam nisu važni




Jedinično testiranje

Tipovi test dvojnika

Stub objekat

- simulira izlaze stvarnog objekta
- na predefinisane ulaze daje predefinisane izlaze
- zna da reaguje samo na ulaze koje konkretan test predviđa




Jedinično testiranje

Tipovi test dvojnika

Mocking predstavlja jednu od tehnika. testiranja u objektnom programiranju koja uvodi pojam *mock objekata*.

Mock objekat

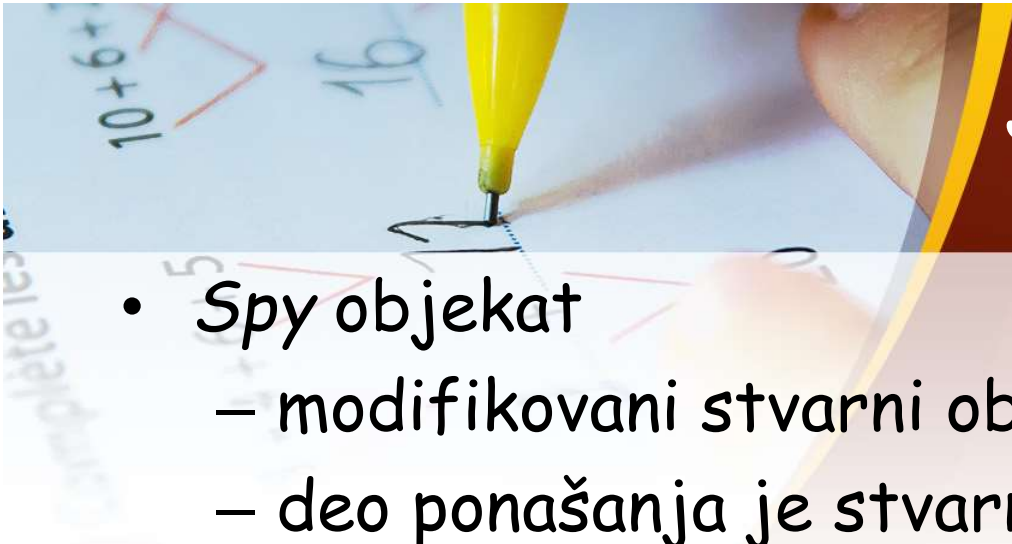
- simulira ponašanje stvarnog objekta
 - za razliku od *stub* objekta koji simulira samo izlaze
- Ako stvarni objekat sadrži sekvencu poziva određenih metoda, *mock objekat* ponavlja ovu sekvencu
 - na ovaj način dobijamo dvojnika koji se i ponaša kao stvarni objekat
 - sami izlazi koji su rezultat ponašanja su i dalje simulirani



Jedinično testiranje

Tipovi test dvojnika

- Lažni (*fake*) objekat
 - sadrži stvarni programski kod koji se izvršava
 - jedino je zbog jednostavnosti ili brzine implementacija drugačija u odnosu na aplikaciju u produkciji i prilagođena je testiranju
 - dobar primer je korišćenje *in-memory* baze podataka za potrebe testiranja



Jedinično testiranje

Tipovi test dvojnika

- *Spy* objekat
 - modifikovani stvarni objekat
 - deo ponašanja je stvarno ponašanje
 - deo ponašanja je simuliran zbog potreba testiranja
 - korišćenjem *spy* objekata test može da dobija stvarno ponašanje jednog dela, a simulirano ponašanje drugog dela funkcionalnosti



Integraciono testiranje

Jedan od najvažnijih aspekata softverskih razvojnih projekata je integraciona strategija.

Generalno, što su veći projekti, to je integraciona strategija važnija. Veoma mali sistemi često su sakupljeni i testirani u jednoj fazi. Za većinu realnih sistema, ovo je nepraktično iz dva bitna razloga.

Prvo, sistem bi pao na mnogo mesta odjednom tako da pokušaji debugovanja i retestiranja bi bili potpuno nepraktični.

Drugo, zadovoljavanje kriterijuma testiranja ''bela kutija'' bi bilo veoma teško, zbog ogromne količine detalja razjedinjavanjem ulaznih datoteka.

U stvari, većina integracionih testiranja je tradicionalno limitirano na tehnike funkcionalnog testiranja.



Integraciono testiranje

Veliki sistemi mogu zahtevati mnoge integracione faze, počevši od sakupljanja modula u nisko-rangiranim podsistemima, pa onda okupljanje podsistema u veće podsisteme i konačno sklapanje podsistema viših nivoa u celokupan sistem.

Da bi bile efikasnije, tehnike integracionog testiranja treba da se dobro uklope u celokupnu integracionu strategiju.

U višefaznoj integraciji, testiranje u svakoj fazi pomaže da se greška uoči ranije, a sistem drži pod kontrolom.

Izvodeći pojedinačno testiranje u ranoj integracionoj fazi, a zatim koristeći rigoroznije kriterijume u finalnoj etapi realna je samo varijanta visokog rizika "big bang" (veliki prasak) pristupa.

A close-up photograph of a yellow pen tip pointing at a math problem on a piece of paper. The problem involves a triangle with a vertical line from the top vertex to the base, and the numbers '10+6' and '16' are visible. The background of the slide is a solid dark red color.

Integraciono testiranje

Pritom, izvođenje rigoroznih testova celog softvera angažovanog u svakoj integracionoj fazi angažuje dosta nepotrebnog dupliranja truda kroz faze.

Rešenje je premostiti celokupne integracione sisteme, ostvariti strogo testiranje u svakoj fazi i smanjiti dupliciranje truda. Važno je razumeti odnose između modula testiranja i integracionog sistema.

Sa jedne strane, moduli su strogo testirani koristeći drajvere pre nego što je pokušana bilo koja integracija.

Zatim se integracioni sistem u potpunosti koncentriše na interakciju modula, smatrajući da su detalji u svakom modulu tačni.

Sa druge strane, moduli i integraciono testiranje mogu biti kombinovani, potvrđujući detalje izvršavanja svakog modula u integracionom kontekstu.



Integraciono testiranje

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.

- Testiranje metodama crne kutije



Integraciono testiranje

- Jedan od najvažnijih aspekata softverskih razvojnih projekata je integraciona strategija.
- Integracija može biti izvedena odjednom, od vrha ka dnu, od dna ka vrhu, kritični deo prvo ili prvim integracionim funkcionalnim podsistemom i tek onda integrisati podsisteme u odvojenim fazama koristeći bilo koju osnovnu strategiju.
- Generalno, što su veći projekti, to je integraciona strategija važnija. Veoma mali sistemi često su sakupljeni i testirani u jednoj fazi.
- Za većinu realnih sistema, ovo je nepraktično iz dva bitna razloga. Prvo, sistem bi pao na mnogo mesta odjednom tako da pokušaji debugovanja i retestiranja bi bili potpuno nepraktični



Integraciono testiranje

Drugo, zadovoljavanje kriterijuma testiranja „bela kutija“ bi bilo veoma teško, zbog ogromne količine detalja razjedinjavanjem ulaznih datoteka. U stvari, većina integracionih testiranja je tradicionalno limitirano na tehnike funkcionalnog testiranja (Hetzel.) Veliki sistemi mogu zahtevati mnoge integracione faze, počevši od sakupljanja modula u nisko-rangiranim podsistemima, pa onda okupljanje podsistema u veće podsisteme i konačno sklapanje podsistema viših nivoa u celokupan sistem. Da bi bile efikasnije, tehnike integracionog testiranja treba da se dobro uklope u celokupnu integracionu strategiju.



Integraciono testiranje

U višefaznoj integraciji, testiranje u svakoj fazi pomaže da se greška uoči ranije, a sistem drži pod kontrolom. Izvodeći pojedinačno testiranje u ranoj integracionoj fazi, a zatim koristeći rigoroznije kriterijume u finalnoj etapi realna je samo varijanta visokog rizika „big bang” (veliki prasak) pristupa. Pritom, izvođenje rigoroznih testova celog softvera angažovanog u svakoj integracionoj fazi angažuje dosta nepotrebnog dupliranja truda kroz faze.



Integraciono testiranje

Rešenje je premostiti celokupne integracione sisteme, ostvariti strogo testiranje u svakoj fazi i smanjiti dupliciranje truda.

Važno je razumeti odnose između modula testiranja i integracionog sistema.

Sa jedne strane, moduli su strogo testirani koristeći drajvere pre nego što je pokušana bilo koja integracija.

Zatim se integracioni sistem u potpunosti koncentriše na interakciju modula, smatrajući da su detalji u svakom modulu tačni.

Sa druge strane, moduli i integraciono testiranje mogu biti kombinovani, potvrđujući detalje izvršavanja svakog modula u integracionom kontekstu.



Integraciono testiranje

Oba gledišta integracionog testiranja mogu biti prikladna za bilo koji zadati projekat, tako da bi integracioni metod testiranja trebao biti dovoljno fleksibilan da se prilagodi svima. Najjednostavnija aplikacija strukturnog testiranja je kombinovanje testiranja modula sa integracionim testiranjem tako da osnovni setovi putanja kroz sve module su izvršeni u integracionom kontekstu.



Integraciono testiranje

Kriterijum testiranja modula često može biti uopšten na nekoliko mogućih načina

Apliciranjem na svaku fazu više-fazne integracione strategije, na primer, vodi ka prekomernoj količini suvišnih testiranja.

Korisnije uopštavanje, prilagođavanje kriterijuma testiranja modula, fokusiran na interakciju između modula pre nego pokušaj testiranja svih detalja svakog od izvršavanja modula u kontekstu integracije.

Izveštaj kriterijuma testiranja modula u kojem je svaka konstatacija potrebna da bude vežbana tokom testiranja modula, može biti uopštena da potražuje od svakog modula da pozove izveštaj vežbe tokom integracionog testiranja. Iako je specifičnost uopštavanja strukturnog testiranja detaljnija, pristup je isti.



Integraciono testiranje

se vrši kao nastavak jediničnog testiranja.

Integracionim testiranjem se testira više komponenti istovremeno.

Integracioni testovi koji proveravaju da li systemske komponente sarađuju, kao što je opisano u specifikacijama dizajna sistema i programa. Njima se testira sve, od korisničkog interfejsa do baze podataka i služe za simulaciju rada korisnika.

Kada su napravljene komponente i testirane svaka ponaosob potrebo ih je i testirati u radu sa ostalim komponentama i proveriti da li rade bez grešaka.

Iako svaki softverski modul prolazi kroz jedinični test, još uvek može da ima neke greške

- Testiranje koje rade testeri

A close-up photograph of a yellow pen tip pointing at a math problem on a piece of paper. The problem involves a triangle with sides labeled '10', '6', and '16'. The text '10+6+' is visible above the triangle. The background is a solid dark red color.

Integraciono testiranje

Jedan od najvećih problema je (kod integracije komponenti) u njihovom međusobnom povezivanju.

Dok prolaze kroz interfejs, podaci se mogu izgubiti;

- - jedna komponenta može imati nepovoljan uticaj na neku drugu;
- - nepreciznosti koje su prihvatljive u pojedinačnim komponentama, mogu u međusobnom povezivanju da narastu do neprihvatljivo velikih vrednosti;
- - globalne strukture podataka mogu da predstavljaju problem

Strategija integracionog testiranja koja će biti izabrana zavisi od

- arhitekture sistema
- plana projekta
- plana testiranja



Integraciono testiranje

Zašto je integraciono testiranje bitno?

Zar nisu sve jedinice i moduli već detaljno testirani na jediničnom nivou?

Integraciono testiranje polazi od komponenti koje su prošle jedinično testiranje.

Primenjuju se testovi definisani u planu integracionog testiranja

Kao izlaz se na kraju dobija integrisan sistem spreman za sistemsko testiranje

Neizbežno će se pojaviti novi problemi

Više komponenti koje sada prvi put rade zajedno

Ukoliko je loše urađena integracija, svi problemi će se pojaviti odjedanput

- - Teško za testiranje, debug, fix



Integraciono testiranje

Najočiglednije kombinovana strategija je čista "big bang" integracija referenci, u kojoj se celokupan sistem okuplja i testira u jednom koraku bez prethodnog testiranja modula.

Ovde nema dodatnih integraciono-specifičnih testiranja koji zahtevaju više od testiranja modula determinisano strukturnim testiranjem.

Takođe je moguće kombinovati integraciono i testiranje modula sa dno-vrh integracionom strategijom.

Kod ove strategije, korišćenjem testiranja drajverima, počinje izvođenje strukturnog testiranja modul-nivo na najnižim nivo modulima koristeći test drajvere.

Zatim, izvođenje modul-nivo strukturno testiranje u sličnom stilu na svakom sledećem nivou kroz dizajniranu hijerarhiju koristeći test drajvera za svaki novi modul koji je integraciono testiran sa svim nižim-nivo modulima.

-



Integraciono testiranje

- **Big bang** integracija (Run it and see pristup)
- Pojedinačni moduli programa se razvijaju i testiraju zasebno (na jediničnom nivou)
- Integracija ne počinje dok sve komponente nisu gotove i spremne
- Tada se sve spajaju odjedanput



Integraciono testiranje

- Big bang - nije način na koji treba integrirati i testirati softver

Integracija se vrši bez ikakvog formalnog integracionog testiranja.

Ideja je da će sve komponente raditi savršeno u kombinaciji sa drugima.

Primenljivo samo na jako male sisteme (od nekoliko komponenti), ne i za ozbiljnije aplikacije



TESTIRANJE SOFVERA

- Jedinično ili integraciono testiranje?
- Jedinično testiranje se odnosi na postupak kada se testiraju individualne jedinice izvornog koda
- Pod jedinicom se smatra najmanji deo aplikacije koji se može testirati
- U objektno orijentisanom programiranju, jedinica je najčešće metoda ili jedna klasa
- Najčešće izvode sami programeri



Black-box testiranja

- ☐ Podela na klase ekvivalencije
- ☐ Analiza graničnih vrednosti
- ☐ Testiranje prelaza stanja
- ☐ Testiranje zasnovano na slučajevima korišćenja
- ☐ Logički zasnovane tehnike



Klasa ekvivalencije

Podele na klase ekvivalencije polazi od ideje da se ulazni podaci mogu podeliti u reprezentativne klase, tako da se za sve pripadnike jedne klase program ponaša na sličan način.

- Te podele klase u reprezentativne klase su poznate pod pojmom klasa ekvivalencije



Klasa ekvivalencije

- ☐ Testiranje se vrši samo za jednu vrednost ulaza iz svake klase ekvivalencije
- ☐ (Može da se desi da bi se za sve ostale vrednosti pronašla (ista) greška u programu)
- ▶. Za svaki uslov se posmatraju dve grupe klasa :
 - ☐ legalne klase koje obuhvataju dozvoljene situacije
 - ☐ nelegalne klase koje obuhvataju sve nedozvoljene situacije



Klasa ekvivalencije

Ekvivalentna klasa predstavlja skup vrednosti ulaznih promenljivih koji proizvodi izlazne rezultate.

Ekvivalentna klasa koja sadrži samo **validna stanja** označava se kao **validna klasa**, dok ekvivalentna klasa koji sadrži **nevalidna stanja** označava se kao **nevalidna EC klasa**.

Validne i nevalidne ekvivalentne klase se kreiraju za svaku varijablu iz skupa vrednosti ulaznih promenljivih.



Kako odrediti klase ekvivalencije

Posmatraju se svi uslovi vezani za ulaze i izlaze vrednosti u softveru koji proizilaze iz specifikacije (tipovi promenljivih, eksplicitna ograničenja).

Za svaki uslov se posmatraju dva grupe:

- Legalne klase obuhvataju dozvoljene situacije.
- Nelegalne klase obuhvataju sve ostale situacije.

Prednost ovog metoda: vreme potrebno za testiranje

Klase ekvivalencije / VALIDNE VREDNOSTI

Metoda koja izračunava na primer ocenu đaka na osnovu broja bodova na pismenom iz matematike

KLASA EKVIVALENCIJE	BROJ BODOVA	REPREZENTATIVNA VREDNOST
Validna klasa ekvivalencije 1	$0 > A > 10$	7
Validna klasa ekvivalencije 2	$11 > A > 20$	13
Validna klasa ekvivalencije 3	$21 > A > 30$	24
Validna klasa ekvivalencije 4	$31 > A > 40$	36
Validna klasa ekvivalencije 5	$41 > A > 50$	47

Klase ekvivalencije / NEVALIDNE VREDNOSTI

Metoda koja izračunava na primer ocenu đaka na osnovu broja bodova na pismenom iz matematike

KLASA EKVIVALENCIJE	BROJ BODOVA	REPREZENTATIVNA VREDNOST
Validna klasa ekvivalencije 1	$0 > A > 10$	-7
Validna klasa ekvivalencije 2	$11 > A > 20$	101
Validna klasa ekvivalencije 3	$21 > A > 30$	16
Validna klasa ekvivalencije 4	$31 > A > 40$	100000
Validna klasa ekvivalencije 5	$41 > A > 50$	-200



Klase ekvivalencije

Upotrebom ekvivalentnih klasa nastoji se rešavanju problema redundantnosti i nepotpuno testiranog softvera koji postoje kod testiranja graničnih vrednosti.

Kada se radi metoda klasa ekvivalencije, neophodno je da se formiraju **test primeri** za:

- ❑ **Sve legalne klase ekvivalencije** (cilj je da se jedan test primer primeni na što više legalnih klasa)
- ❑ **Sve nelegalne klase ekvivalencije** (za svaku nelegalnu klasu mora se napisati poseban test primer, kako bi se izbeglo da jedan neregularan ulazni podatak maskira neki drugi, takođe neregularan, zbog proveru unetih u kod)



Klase ekvivalencije

- Grupe međusobno ekvivalentnih ulaznih podataka proizvode isti rezultat iz kojih se bira po jedan predstavnik grupe koji se koristi u tokom testiranja.
- Ulazne vrednosti programa se dele u klase ekvivalencije.
- Program se ponaša na sličan način za sve ulazne vrednosti koje pripadaju istoj klasi ekvivalencije



Zašto definišemo klase ekvivalencije?

Ideja:

Testirati program sa po jednom reprezentativnom vrednošću ulaza iz svake klase ekvivalencije.

Ova metoda se može koristiti u modulskom testu.

Posebno voditi računa o vrednostima na granici klase ekvivalencije.

Česta greška je da se za određenu vrednost ne identifikuje da pripada posebnoj klasi ekvivalencije

Pogrešno se pretpostavi da se sistem za tu vrednost ponaša na isti način kao za druge vrednosti



Metoda graničnih vrednosti

Metoda je dopuna metode klase ekvivalencije.

Bavi se vrednostima koje su na granicama klasa ekvivalencije - drugim rečima na samoj granici

- Otkazi su vrlo često uzrokovani graničnim vrednostima
 - razlog je što često granice nisu jasno definisane ili su pogrešno interpretirane

Tehnika se primenjuje kada su vrednosti u klasi ekvivalencije uređene i imaju jasno određene granice



Metoda graničnih vrednosti

Pripada testiranju crne kutije.

Skoro uvek se koristi zajedno sa klasama ekvivalencije.

Kada se radi o metodi testiranja graničnih vrednosti akcenat je dat na granicnim vrednostima jer tu potencijalno može da bude puno grešaka.

- Greška koju programeri često čine je pogrešno kodiranje testova nejednakosti.
- Primer toga je pisanje $>$ znaka umesto $<$ znaka.



Metoda graničnih vrednosti

Analiza graničnih vrednosti:

- Izabrati test primere na granicama različitih klasa ekvivalencije (jedno ili multidimenzionalnih), ili na osnovu specificiranih veza među ulazima

Po jedan test primer za svaku graničnu vrednost

- Testove pravimo tako što za svaku prethodno određenu graničnu vrednost definišemo jednu tačku ispod i jednu tačku iznad granice.



Metoda graničnih vrednosti

Postoji više vrsta testiranja metodom graničnih vrednosti, neke od njih su:

- ☐ Testiranje slabih unutrašnjih graničnih vrednosti
- ☐ Testiranje slabih spoljašnjih graničnih vrednosti
- ☐ Robusno testiranje slabih graničnih vrednosti
- ☐ Robusno testiranje jednostavnih slabih graničnih vrednosti
- ☐ Testiranje najgoreg slučaja granične vrednosti
- ☐ Testiranje slabih jednostavnih spoljašnjih graničnih vrednosti



Metoda graničnih vrednosti

Za svaku klasu ekvivalencije testira se

- vrednost koja je tačno na granici klase
- najbliža vrednost graničnoj koja se nalazi izvan klase ekvivalencije
- najbliža vrednost graničnoj koja se nalazi unutar klase ekvivalencije

Susedna vrednost je vrednost sa minimalnim pomerajem u odnosu na graničnu vrednost

- kod decimalnih brojeva, izabere se određena tolerancija



Metoda graničnih vrednosti

PREDNOSTI

- Vrlo jednostavna za korišćenje
- Test slučajevi koji ova metoda generiše su mali

MANE

- ✓ Sistem zanemaruje testiranje svih mogućih ulaznih vrednosti, pa su izlazi nesigurni
- ✓ Metoda se ne uklapa dobro sa bool promenljivama



Kriterijum završetka testiranja graničnih vrednosti

Postavlja se pitanje da li postoji neki kriterijum za završetak ovog vida testiranja softvera:

KRITERIJUM

Postignuti nivo pokrivenosti svih graničnih vrednosti

Izračunava se kao odnos testiranih i ukupnih graničnih vrednosti

$$coverage = \frac{testedBV}{totalBV} * 100\%$$

U ukupnom broju graničnih vrednosti se nalaze i ulazne vrednosti kao i najbliži susedi graničnih vrednosti.

Ako se desi slučaj da se vrednosti preklapaju za dve klase ekvivalencije, vrednost se računa samo jednom.



Testiranje bazirano na tabeli odluke

Tabele odluke su decenijama unazad korišćene kako bi predstavile i analizirale složene logičke veze. Pripada grupi „uzrok-efekat” analiza

One su idealne za opisivanje situacija u kojima se koristi više kombinacija akcija pod različitim setovima uslova.

Da bi se identifikovali test slučajevi pomoću tabele odluke, uslovi se koriste kao ulazi dok akcije predstavljaju izlaze.

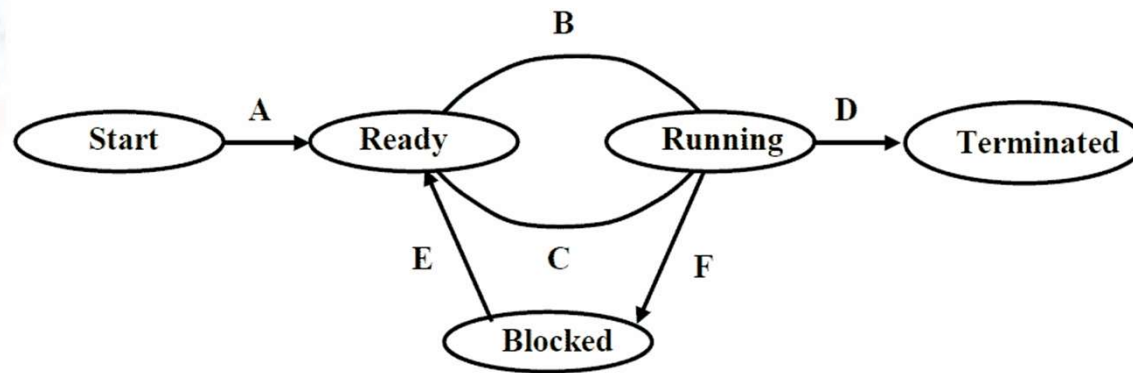
Testiranje bazirano na tabeli odluke

C1: a,b,c form a triangle	F	T	T	T	T	T	T	T	T
C2: a=b ?	-	T	T	T	T	F	F	F	F
C3: a=c ?	-	T	T	F	F	T	T	F	F
C4: b=c ?	-	T	F	T	F	T	F	T	F
A1: nije trougao	X								
A2: raznostranični									X
A3: jednakokraki					X		X	X	
A4: jednakostranični		X							
A5: nemoguć			X	X		X			

Na primeru sa trouglovima, ulazne vrednosti su uslovi C1, C2, C3, C4, i oni sadrže različite kombinacije true(T) ili false(F) vrednosti u kolonama sa desne strane. Očekivane izlazne vrednosti su predstavljene akcijama A1, A2, A3, A4 i A5 i sa njihove desne strane je označena izlazna vrednost (akcija) koja se dobija prethodnim kombinacijama uslova.

Na primer, ako svi uslovi imaju vrednost T, što znači da a, b i c formiraju trougao da je a=b, b=c i a=c, slede da je trougao jednakostranični i tako dalje.

Testiranje prelaza 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 (finite state machine).

Dijagram stanja je jedan od načina prikaza takvog modela



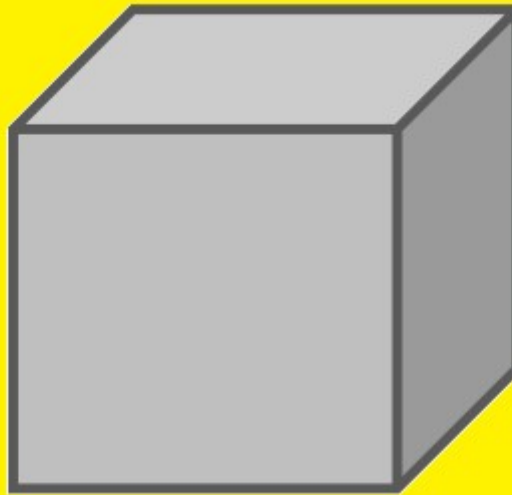
Logički zasnovane tehnike

- ✓ Posmatraju i međusobne zavisnosti između ulaznih parametara.
- ✓ Za razliku od drugih tehnika koje pri kreiranju test slučajeva svaki parametar nezavisno posmatraju

Postoje različite logički zasnovane tehnike:

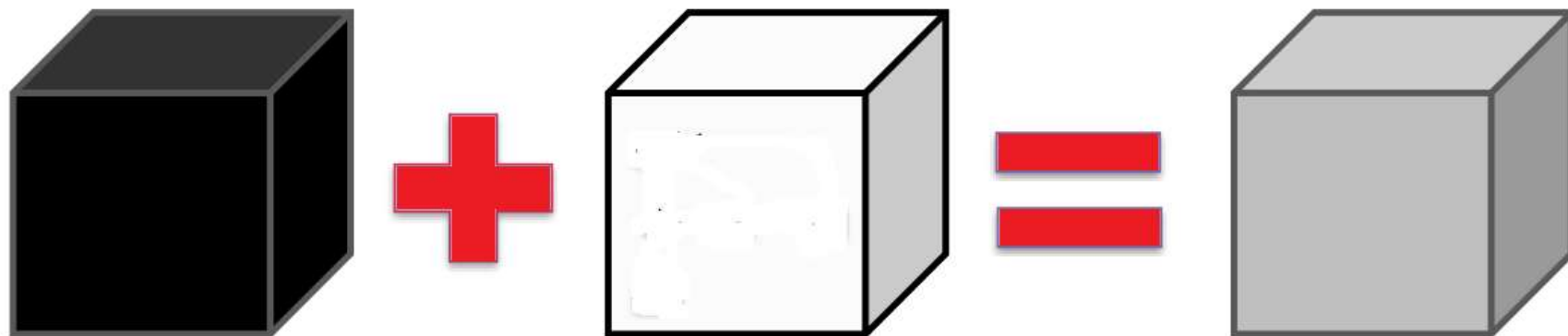
- ✓ Tehnika uzročno-posledičnih grafova
- ✓ Tehnika parova podataka

Grey Box testiranje softvera





Grey Box testiranje softvera





Grey Box testiranje softvera

“Svrha testiranja sive kutije je pretraživanje i prepoznavanje nedostataka zbog nepravilne strukture koda i nepravilnog korištenja aplikacije” (Guru99).

Za razliku od testiranja crne kutije, kod ove tehnike testerima imaju više uvida u mehanizma softvera nad kojim vrše testiranje.

Daje veću mogućnost za kreiranje boljih testnih scenarija.

- Za dobijanje informacija o unutrašnjim strukturama koriste se razni procesi analiza strukture, funkcija i operacija ili drugih metoda obrnutog inženjerstva - reverse engineering



Grey Box testiranje softvera

Metoda testiranja sive kutije je kombinacija metoda testiranja crne kutije i testiranja bele kutije.

Ovaj metoda testiranja je korisna u slučajevima testiranja integracije različitih delova koda.

Testiranje sive kutije predstavlja metod za testiranje aplikacije sa ograničenim ponavljanjem internog rada aplikacije.

Tester je delimično upoznat sa internom strukturom.



Grey Box testiranje softvera

PREDNOSTI:

Testiranje se sprovodi iz perspektive samog korisnika

Koristi kombinaciju prednosti testiranja crne kutije i testiranja bele kutije

Dizajn testnih slučajeva može se obaviti u kratkom vremenskom periodu.

MANE

- ✓ Ograničena je mogućnost proučavanja programskog koda.
- ✓ Testerima nemaju mogućnost uvida u izvorni kod
- ✓ Postoji mogućnost ponavljanja testnih scenarija
- ✓ Tester vrši testiranje nad istim delovima koda koje je već testirao programer



Grey Box testiranje softvera

Drugi izraz za sivu kutiju je prozirna kutija gde tester ima ograničeno znanje o unutrašnjosti same aplikacije.

Testiranje se vrši na osnovu visokog nivoa dijagrama toka podataka i dijagrama baze podataka.

Jedna od korištenih informacija o programu koji se testira je pokrivenost koda.

Pokrivenost koda predstavlja informaciju o određenim putanjama i koliki deo od ukupnih putanja je otkriven.

Ne uzima se u obzir testiranje algoritama.



Revizijsko (klase) testiranja softvera

Kako da vidimo da li je neki softver uspešno testiran ili ne?

Revizija softvera je jedan od bitnih uslova kada se govori o testiranju softvera

Testovi koji pripadaju ovoj klasi su:

- ☐ **Testovi održavanja** (maintainability tests)
- ☐ **Testovi fleksibilnost** (flexibility tests)
- ☐ **Testovi određivanja pogodnosti testiranja softvera** (testability tests).



Testovi održavanja (maintainability tests)

Testovi se odnose na:

- ✓ **Strukturu sistema** - da li su podržani standardi i procedure .
- ✓ **Dokumentaciju** koji treba biti urađena odnosno pripremljena u skladu sa određenim standardima.
- ✓ **Internu dokumentaciju** koja treba biti pripremljena u skladu sa određenim procedurama.



Testovi određivanja pogodnosti testiranja softvera (testability tests).

Testability

dodatne karakteristike programa koje pomažu testerima u njihovom testiranju softvera kao što je mogućnost formiranja međurezultata za pojedine tačke, formiranje log fajlova.

Jedan od potencijalnih problema u testiranju softvera je koliko ulaznih vrednosti je dovoljno testirati i kada se može reći da je testiranje završeno.

Jako je bitno propusti određeni set ulaznih podataka nekoliko puta da se ne bi desilo da dobijamo različite rezultate.